Danmarks Tekniske Universitet DTU

Kursus: 02101: Indledende programmering

Institut: IMM, Danmarks Tekniske Universitet.

Titel: Læg en studieplan

Dato: 7/12-2007

Gruppenummer: 20

Studienummer: s936818 s073035

Databar: 43

Forfattere: Michael Thomas Holm Mikael Andersen

Indholdsfortegnelse:

1. Resume

Til en DTU-studerende er der udviklet et anvendelsesprogram, som kan lægge en **studieplan**, der semester for semester viser de kurser man ønsker at følge. Brugeren fører en **dialog** med programmet via et *terminal window*, hvori brugeren indtaster ordrer til programmet, og programmet udskriver svar til brugeren. De kurser, der kan indplaceres i en studieplan, står i en **kursusbase** hvis data stammer fra **tre filer**.

Overordnet etablerer programmet **kursusbasen** med de kurser der kan indgå i en studieplan, dernæst kan brugeren lægge en **studieplan** i **dialog** med programmet. Brugeren kan gemme sin profil, så brugeren senere kan arbejde videre hvor brugeren slap sidst.

Programmet kan håndtere at flere brugere gemmer deres studieplan, men programmet kan kun behandle en bruger ad gangen.

Denne rapport incl. aktuelle filer er elektronisk tilgængelige på http://www.holm-teknik.dk/02101/

1.1. Bemærkelsesværdige ting

Programmet starter op med en default bruger ved navn ”Peter Pedal” med studienummer ”s012345”. Brugeren kan så oprette sig selv som bruger med kommandoen ’opretbruger <studienummer>’, eller brugeren kan køre videre med profilen for Peter Pedal. (Se evt. forbedringsforslag kapitel 6.1 side 40).

Brugeren skal huske at slå ”Unlimited buffering” til i BlueJ’s terminalvindue. Ellers vil brugeren ikke kunne scrolle op til de første kurser i listen efter kørsel af kommandoen ’udskrivbase’.

Se i øvrigt listen over forbedringsforslag i kapitel 6 startende på side 40.

1.2. Ikke implementeret

Der er kun begrænset JUnit test af klasserne, som håndterer filer. Der er ingen JUnit test af klassen ’Kontrol’. Se uddybende forklaringer på dette i kapitel 4 side 28.

Se i øvrigt kapitel 6 side 40 for forbedringsforslag.

2. Kravanalyse

Den komplette kravbeskrivelse kan læses her: http://www2.imm.dtu.dk/courses/02101/opgave.html
Det vigtigste behandles herefter i resten af dette kapitel.

Til en DTU-studerende skal der udvikles et anvendelsesprogram som kan lægge en **studieplan**, der semester for semester viser de kurser man ønsker at følge. Brugeren fører en **dialog** med programmet via et *terminal window*, hvori brugeren indtaster ordrer til programmet, og programmet udskriver svar til brugeren. De kurser der kan indplaceres i en studieplan står i en **kursusbase**, hvis data stammer fra **tre filer**.

Overordnet skal programmet først etablere **kursusbasen** med de kurser der kan indgå i en studieplan, dernæst kunne lægge en **studieplan** i **dialog** med brugeren af programmet, og til sidst kunne gemme den lagte studieplan i en fil.

Programmet skal leve op til følgende krav:

1. Kunne fortælle brugeren navnene på de ordrer programmet kender.
2. Have en hjælpefacilitet der i terminalvinduet kan oplyse detaljer om ordrerne.
3. Danne programmets kursusbase ud fra de tre filer:

Fil med kursusnavne (filnavn navne.txt)	Fil med skemagrupper (filnavn: skemagrp.txt)	Fil med forudsætningskurser (filnavn: forud.txt)
01005 Matematik 1 01017 Diskret matematik og databaser 02101 Indledende programmering 02105 Algoritmer og Datastrukturer I 02121 Ingeniørarbejde 02161 Software Engineering 1	01005-e E5A E5B E3B 01005-f F5A F5B F3B 01017 E1B 02101 E3A 02105 F2B 02121 E4B	02105 01017 02101 02161 02101 .... (se kapitel 2.1.2)

4. For hvert kursus rummer kursusbasen oplysninger om kursets nummer, kursusnavnet, skemagruppeplaceringen og forudsætningskurserne. Disse data stammer som tidligere nævnt fra tre filer.
5. Kunne give en tekstrepræsentation af kursusbasen - `toString()`.
6. Vise i terminalvinduet den aktuelle studieplan som den ser ud her og nu.
7. Kunne indsætte i studieplanen på et angivet semester et kursus der står i kursusbasen.
8. Kunne fjerne et tidligere indsat kursus fra studieplanen.
9. Afslutte programkørslen med mulighed for at man i en efterfølgende kørsel (*program run*) kan arbejde videre på en tidligere udarbejdet studieplan - benyt *Object serialization.*

10. Kunne arbejde videre på en tidligere udarbejdet studieplan.

11. Give fejlmeldinger når brugeren beder om noget der ikke kan lade sig gøre.

12. Skemalokation:

		Mandag	Tirsdag	Onsdag	Torsdag	Fredag
Efterår	8-12	E1A	E3A	E5A	E2B	E4B
	13-17	E2A	E4A	E5B	E1B	E3B
Forår	8-12	F1A	F3A	F5A	F2B	F4B
	13-17	F2A	F4A	F5B	F1B	F3B

13. Første semester er et efterår.

14. For kurser på ulige semestre angives kursernes placering i DTU kursusbasen med bogstavet E (efterår), og på lige semestre med bogstavet F (forår) foran skemagruppen.

15. Hvert kursus forekommer kun én gang (se kapitel 2.1).

16. For hvert semester er der kun ét kursus i hver skemagruppe.

17. Alle forudsætningskurser for et vilkårligt kursus på 2. eller efterfølgende semestre er placeret på tidligere semestre (se kapitel 2.1).

18. Foretag en fuldstændig JUnit test af alle public metoder, med testdatavalg der omfatter såvel positiv som negativ test.

Herudover er 3 tillægskrav (fra 3-gruppers kravene) indirekte også blevet opfyldt, dog uden at vi har haft fokus på dette, da vi er en 2-personers gruppe:

19. Ved definitionen af skemagrupper for de enkelte kurser forenklede vi modellen ved at tilføje -e eller -f til kursusnummeret for at lette håndteringen af kurser over to semestre og af kurser der blev udbudt såvel forår som efterår. Foretag de nødvendige rettelser så skemagruppefiler kan se sådan ud:

```
01005 E5A E5B E3B F5A F5B F3B
01017 E1B
02101 E3A
.....
```

20. Alle fejlmeldinger skal fortælle årsagen til at det gik galt. Fejlmeldingsteksterne i afsnittet **Eksempel på en dialog** angiver årsager, fx:

```
... manglende forudsætningskurser
```

Men det kan gøres endnu bedre:

```
*** Kursus 02105 ej tilføjet - forudsætningskurset 01017 mangler ***
```

21. Indfør ordrer, så programmet kan håndtere flere studieplaner, enten for samme person, eller for flere personer. Dog naturligvis kun én studieplan ad gangen. Man skal i samme kørsel kunne afslutte en plan og skifte til en anden. Og kunne skifte mellem personerne.

2.1. Konflikter i kravanalysen

2.1.1. Kurser, der strækker sig over 2 semestre, henholdsvis kurser, der udbydes både forår og efterår.

Der er modstridende krav i kravspecifikationen. Der står:

- For kurser der strækker sig over to semestre, som fx `01005`, har vi valgt en semesteropdeling, så kursusbasen internt indeholder kurserne `01005-e` og `01005-f`. Brugeren må så *tilføje* kurset `01005` af to gange, nemlig som `01005-e` og som `01005-f`. Og når kurset ønskes fjernet/flyttet må brugeren også gøre dette af to gange.
- For kurser der som `02402` udbydes på to semestre har vi også internt valgt at supplere kursusnummeret med `-e` og `-f`. Herved forenkles søgningen i kursusbasen: Enten forekommer kurset netop én gang eller også findes det ikke i basen. Brugeren må så selv sørge for kun at placere kurset én gang i sin studieplan."

Disse 2 ovenstående punkter konflikter med kravanalysen for forudsætningerne, fordi:

Som filerne i kravspecifikationen er lavet, er det ikke muligt at skelne mellem hvorvidt kurset er et dobbeltkursus (eks 01005) eller kurset er et enkeltkursus der udbydes både forår og efterår (eks 02402). Derfor er det ikke muligt at tjekke på om forudsætningerne for et dobbeltkursus er 100% opfyldt. Det er dog muligt at tjekke, at dobbeltkurser forekommer på mindst 1 semester, hvis andre kurser afhænger af disse. Man kan derfor sige, at det er muligt at tjekke "50%". Af denne årsag er programmet valgt at lavet således:

For kurser, som både udbydes om foråret og om efteråret, indeholder programmet ikke noget tjek af at disse starter om efteråret. Det indeholder heller ikke noget tjek af at kurset ikke optræder mere end 2 gange i skemaet. (Sidstnævnte kunne godt have ladet sig gøre, men det er nedprioriteret, da brugeren alligevel selv manuelt skal ind og tjekke de øvrige ting for netop denne slags kurser).
Denne slags kurser har brugeren frit mulighed for at placere så mange gange, som brugeren ønsker det. Ved tjek på kurser, som afhænger af et dobbeltkursus, tjekkes blot at kurset forekommer mindst 1 gang før det kursus, som afhænger af dobbeltkurset. Det kan således lade sig gøre at programmet accepterer et kursus, som afhænger af et dobbeltkursus, trods dobbeltkurset kun er sat på et enkelt semester.

2.1.2. Kursus 2105 – forudsætter 01017 eller ej.

I kravspecifikationen står der, at kursus 02105 forudsætter kursus 01017 samt 02101. Men i filen forud.txt står der at 02105 kun forudsætter 02101.

Vi har valgt at lade forud.txt være 'master' og antage at teksten i kravformuleringen er forkert. Men vi har ladet det forkerte eksempel stå under krav #3, da det er godt til at illustrere filens principielle opbygning.

2.2. Løsninger til kravene

2.2.1. Løsning til krav #1:

Programmet fortæller brugeren navnene på de ordre programmet kender i følgende situationer:

- Når programmet startes første gang.
- Når brugeren skriver en forkert kommando
- Når man skifter bruger og en ny bruger overtager programmet.

2.2.2. Løsning til krav #2:

Kommandoen ’hjælp’ er implementeret således, at hvis man skriver ’hjælp’, så viser programmet de ordre programmet kender. Den viser også, at kommandoen ’hjælp’ kan bruges i formen ’hjælp kommando’.
Hvis kommandoen bruges i den udvidedet for med eks. ’hjælp tilføj’, så viser programmet hvordan kommandoen ’tilføj’ skal bruges. Kommandoen ’hjælp’ har således 2 forskellige funktioner afhængigt af hvordan den bruges.
Den nuværende løsning kan formentligt forbedres, se kapitel 6.1.2 side 40.

2.2.3. Løsning til krav #3 + #4:

Programmet indlæser først kursusnavne-filen (navne.txt) hvor alle kurserne bliver oprettet i kursusbasen. Dernæst bliver skemagrupperne indlæst for alle kurserne (fra filen skemagrp.txt), og der tjekkes på at:

- hver skemagruppe har en lovlig lokation (jvf. krav #12) før at den bliver tilføjet det aktuelle kursus.
- Kursusnummeret allerede er oprettet (da lokationer ikke skal tilføjes kurser, som ikke indgår i navne.txt filen..

Kursusforudsætningerne indlæses (fra filen forud.txt), og det kontrolleres:

- at kursusnummeret er oprettet.
- at forudsætningskurset er oprettet.

Herefter bliver forudsætningerne tilføjet til det aktuelle kursus i kursusbasen i form af kursusnumre (dvs. ikke referencer til kursusobjekter, hvilket også kunne have været valgt. Se forbedringsforslag kapitel 6.2 side 41).

2.2.4. Løsning til krav #5:

Programmet indeholder kommandoen ’udskrivbase’, som via Kontrol-objektet udprinter teksten genereret i en toString() metode fra Kursusbase-klassen, som igen genereres fra en toString() metode fra hvert kursus-objekt. Således bliver der gjort brug af toString() metoder hele vejen ned gennem objekt-hierakiet.

2.2.5. Løsning til krav #6:

Programmet indeholder kommandoen ’visplan’, som udskriver studieplanen for den aktuelle bruger. Den viser både brugerens navn, studienummer, samt alle semestre og deres tilhørende valgte kurser. Semester-nummeret samt hvorvidt det er et forår eller efterår vises, og tilspunktet (klokkeslettet) for kurset vises også. Se udskrivt fra programkørsel i kapitel 5.1 side 34.

2.2.6. Løsning til krav #7:

Programmet indeholder kommandoen 'tilføj <heltal>', som herefter beder brugeren om at indtaste det semesternummer, som brugeren ønsker kurset placeret på. Herefter laver programmet et tjek på at:

- det ønskede semester eksisterer – og hvis ikke at det (incl. mellemliggende) oprettes.
- skemaet for det ønskede semester er ledigt i den/de skemaplaceringer, som kurset ligger på.
- forudsætningerne er opfyldt (evt. forudsætningskurser forefindes på tidligere semestre).
- at kurset udbydes det på det ønskede semester (forår eller efterår).

2.2.7. Løsning til krav #8:

Krav-specifikationen siger vedr. dobbelt-kurser: "For kurser der strækker sig over to semestre, som fx 01005, har vi valgt en semesteropdeling, så kursusbasen internt indeholder kurserne 01005-e og 01005-f. Brugeren må så tilføje kurset 01005 af to gange, nemlig som 01005-e og som 01005-f. Og når kurset ønskes fjernet/flyttet må brugeren også gøre dette af to gange."

Vi har valgt at gøre fjernelse af dobbeltkurser lettere for brugeren ved at programmet sletter kurset alle steder i kursusplanen i et hug.

Der er 2 måder at vælge imellem for håndtering af afhængigheder ved fjernelse af et kursus:

1. Fjerne alle efterfølgende kurser, som afhænger af det kurus man ønsker at fjerne (og helst gøre brugeren opmærksom på hvad der er blevet fjernet.
2. Ikke tillade at fjerne et kursus så længe at efterfølgende kurser afhænger af dette. Dvs man skal i så fald først fjerne de kurser, som afhænger af det man ønsker at fjerne. Brugeren skal så helst få en liste over de kurser, som først skal fjernes.

Løsning #2 er nok nemmest at implementere, da løsning #1 kan give rekursiv fjernelse (et kursus, som fjerner et andet, som igen fjerner et tredie, etc. Løsning #2 er ofte også det man ønsker, da man så kan overskue konsekvensen af at fjerne et kursus (så ikke man ved en fejl fjerner/flytter et kursus, hvilket fører til at en masse andet bliver slettet fra ens plan).

Hvis man brugte løsning #1, så vil det også være ulogisk i forhold til den nuværende funktionalitet, når man tilføjer et kursus. Får så burde tilføjelse af et kursus ligeledes automatisk tilføje afhængigheds-kurser (ala automatisk fjernelse af afhængigheds-kurser) og det ønsker vi i hvert fald ikke!

Derfor er løsning #2 valgt.

2.2.8. Løsning til krav #9:

Programmet giver mulighed for at brugeren kan gemme sin studieplan og senere hente den igen. Programmet gemmer ikke automatisk studieplanen, når programmet afsluttes, da det ikke er sikkert at den studerende vil have overskrevet eventuelle tidligere gemte data. Det kan tænkes, at den studerende blot er inde og prøve nogle nye muligheder af.
Vi har derfor lagt det op til brugeren selv at gemme sin studieplan ved manuelt at køre kommandoen 'gem'.
Når brugeren kører kommandoen 'afslut', vil programmet således afslutte uden at gemme. Se evt. forbedringsforslag kapitel 6.1.5 side 41.

2.2.9. Løsning til krav #10:

Når programmet køres, så kan en tidligere bruger (incl. tidligere studieplan) hentes ved at skrive 'hent <studienummer>'. Herved skifter programmet til den hentede bruger (hvis brugeren tidligere har haft gemt sin profil) og den tidligere studieplan er nu klar til at fortsætte på.

2.2.10. Løsning til krav #11:

Alle kommandoer evalueres, og hvis en kommando ikke er gyldig enten fordi:

1. Kommandoen ikke eksisterer
2. Kommandoen bruges med forkerte parametre

, så gives der besked til brugeren.
Hvis brugeren har fejltype 1 (jvf. ovenfor), så får brugeren en liste med samtlige gyldige kommandoer.
Hvis brugeren har fejltype 2(jvf. ovenfor), så får brugeren detaljeret information om hvordan den givne kommando bruges.
Dette er implementeret i klassen 'Kontrol'.

2.2.11. Løsning til krav #12:

Skemaplaceringer er hard-kodet ind i klassen 'Kontrol' i en ArrayListe. Denne liste overføres til de metoder, som omhandler skemalokationer, således at det bliver tjekket at en lokation er en gyldig lokation. Den bruges eksempelvis:

- under indlæsning af filen skemagrp.txt
- i forbindelse med udskrift af studieplanen med kommandoen 'visplan'.

Skemaplaceringer bruges også i kursus-klassen og når et kursus skal adderes til studieplanen.

2.2.12. Løsning til krav #13+14:

Dette krav er hardkodet ind i klassen 'Studerende', hvor et kursus ønskes tilføjet et semesternummer. Det er defineret, at hvis (semesternummeret modulus 2 == 0), så er det et forårs-semester.

2.2.13. Løsning til krav #15:

Dette krav opfyldes, idet klassen 'Studerende' har metoden 'addCourse', som tjekker om kurset allerede er valgt i forvejen (hvis ellers det ikke er et dobbelt-semester kursus. For ved dobbelt-semester-kurser er der ingen begrænsning, se kapitel 2.1.1 side 8).

2.2.14. Løsning til krav #16:

Dette krav opfyldes, idet klassen 'Studerende' har metoden 'addCourse', som tjekker om det valgte semesters skema er ledigt på den/de skemaplacering(er), som det ønskede kursus er tilknyttet.

2.2.15. Løsning til krav #17:

Dette krav opfyldes, idet klassen 'Studerende' har metoden 'addCourse', som tjekker om det valgte kursus's forudsætninger alle er opfyldte. Metoden går for hvert forudsætningskursus gennem alle tidligere semestre og verificerer at kurset optræder mindst 1 gang.

2.2.16. Løsning til krav #18:

En fuld JUnit test af samtlige Public-metoder har ikke kunnet lade sig gøre, da vi ikke har fundet nogen passende løsning til at teste metoder, som har med filhåndtering at gøre. Vi har heller ikke kunnet lave JUnit test af selve 'Kontrol'-klassen, da den kun har én Public metode – nemlig main-metoden, og da denne ikke laver et objekt på objekt-bench'en.
De øvrige klasser er der i vid udstrækning lavet JUnit tests for deres Public metoder med både positive og negative tests.
Metoder, som kunne laves Private, er naturligvis lavet Private for at give størst mulig uafhængighed mellem klasserne. Private-metoder er der ikke lavet JUnit tests på. Se endvidere kapitel 4 side 28.

2.2.17. Løsning til krav #19

Vores program har hele tiden været opbygget ud fra princippet i krav #19. Princippet går ud på:

- Hvis blot én af skemaplaceringerne starter med "F", så udbydes kurset om foråret.
- Hvis blot én af skemaplaceringerne starter med "E", så udbydes kurset om efteråret.
- Et kursus kan således godt udbydes både forår og efterår.

2.2.18. Løsning til krav #20

Fejlmeddelelser af denne slags sker gennem metoden 'addCourse', som ligger i klassen 'Studerende':

- Hvis der – som kravet giver som eksempel – mangler en forudsætning, så udskrives en fejlbesked indeholdende kursusnummeret for det kursus, som mangler i et tidligere semester.
- Hvis kurset ikke kan placeres på et givent semester, fordi skemagruppen er optaget af et andet kursus (eller samme kursus, se kapitel 6.1.4 side 40), så udskrives beskeden inklusive kursusnummeret for det andet kursus, som konflikter.
- Hvis kurset ikke kan placeres på det ønskede semester, fordi at kurset ikke udbydes det halvår, så udskrives fejlbeskeden incl. besked for hvilket halvår, semesteret som ønskes brugt, er.
- Hvis et kursus ønskes slettet, men et senere kursus er afhængig af det kursus som ønskes slettet, så udskrives en besked til brugeren om først at slette det kursus (kursusnummer udskrives), som blokerer for sletning af det ønskede kursus.

2.2.19. Løsning til krav #21

Programmet benytter et objekt af typen Studerende, som kaldes activeUser. Denne kan ændres til en anden (ny) ved brug af kommandoen 'opretbruger <studienummer>'. Eller man kan skifte til en tidligere oprettet bruger, hvis den tidligere bruger har gemt sine data med kommandoen 'gem'. Man kan så skifte mellem brugerne ved at skrive 'hent <studienummer>' for alle de brugere, som har haft gemt deres data.

3. Programdesign og implementering

3.1. Overordnet opbygning

Vi har taget udgangspunkt i *Model-View-Control*-løsningen, hvor vi dog har smeltet View og Control klasserne sammen til klassen 'Kontrol'. Men bortset fra det har vi forsøgt så svag kobling mellem klasserne som muligt.

Klassen 'Kontrol' er det centrale led i programmet. Klassen udfører i store træk:

1) Opretter kursusbasen (Figure 1) med de kurser, der kan indgå i en studieplan.
2) Opretter den aktive bruger med mulighed for en række semestre med plads til kurser på (Figure 2).
3) Startes en dialog (Figure 3) med brugeren.

Klassen 'Kontrol' styrer herefter input fra brugeren og initialiserer løbende programmets handlinger.

Kursusbasen (bestående af objekter af klassen Kurser) opbygges ud fra klasserne Datafiles, Kursusnavne, SkemaGrupper samt Forudsaetningskurser:

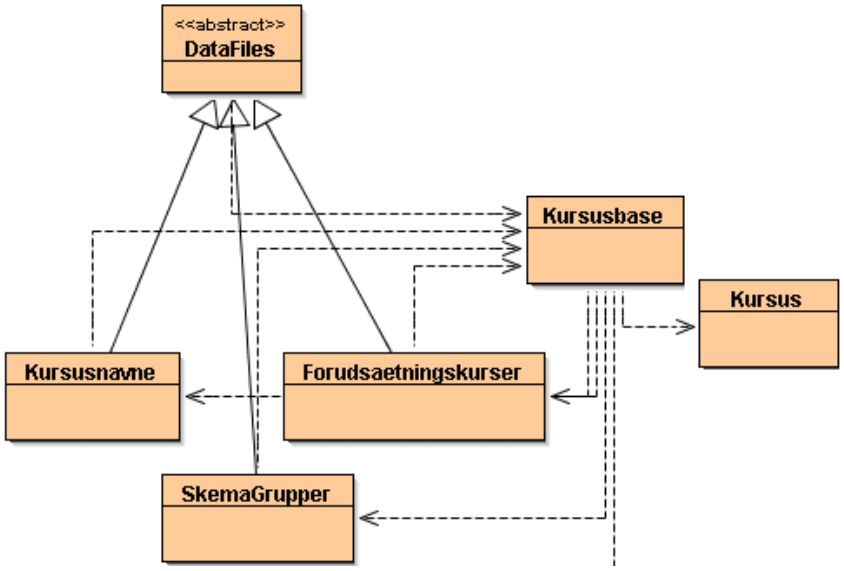

Figure 1: Opbygning af databasen (udsnit af klassediagram).

Et objekt af typen Studerende har mulighed for at få en række semestre, hvorpå der kan knyttes en række kurser:

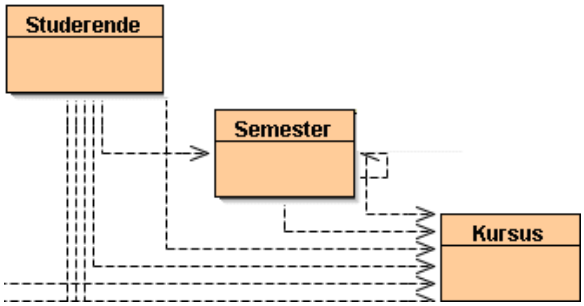

Figure 2: Opbygning af Studerende med studieplan (udsnit af klassediagram).

Vi har benyttet samme struktur m.h.t. kommandofortolker som i eksemplet World-of-Zuul fra lærerbogen:

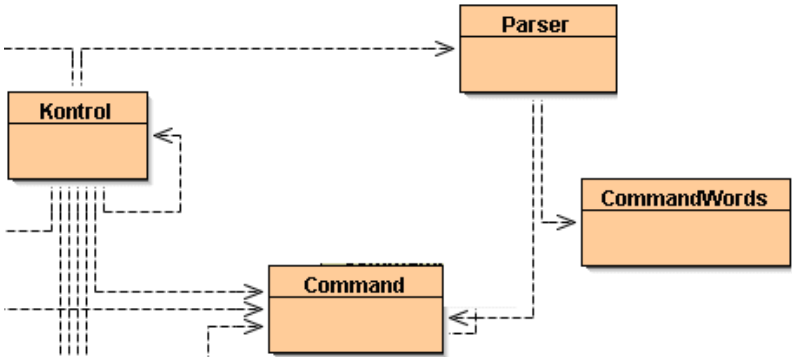

Figure 3: Opbygning af input-parser (udsnit af klassediagram).

Vi har tilstræbt kun at udskrive tekst til brugeren fra klassen 'Kontrol'. Der er dog valgt også at udskrive til brugeren fra øvrige klasser, hvis der er tale om fejlbeskeder, som ikke er blevet implementeret som exceptions.

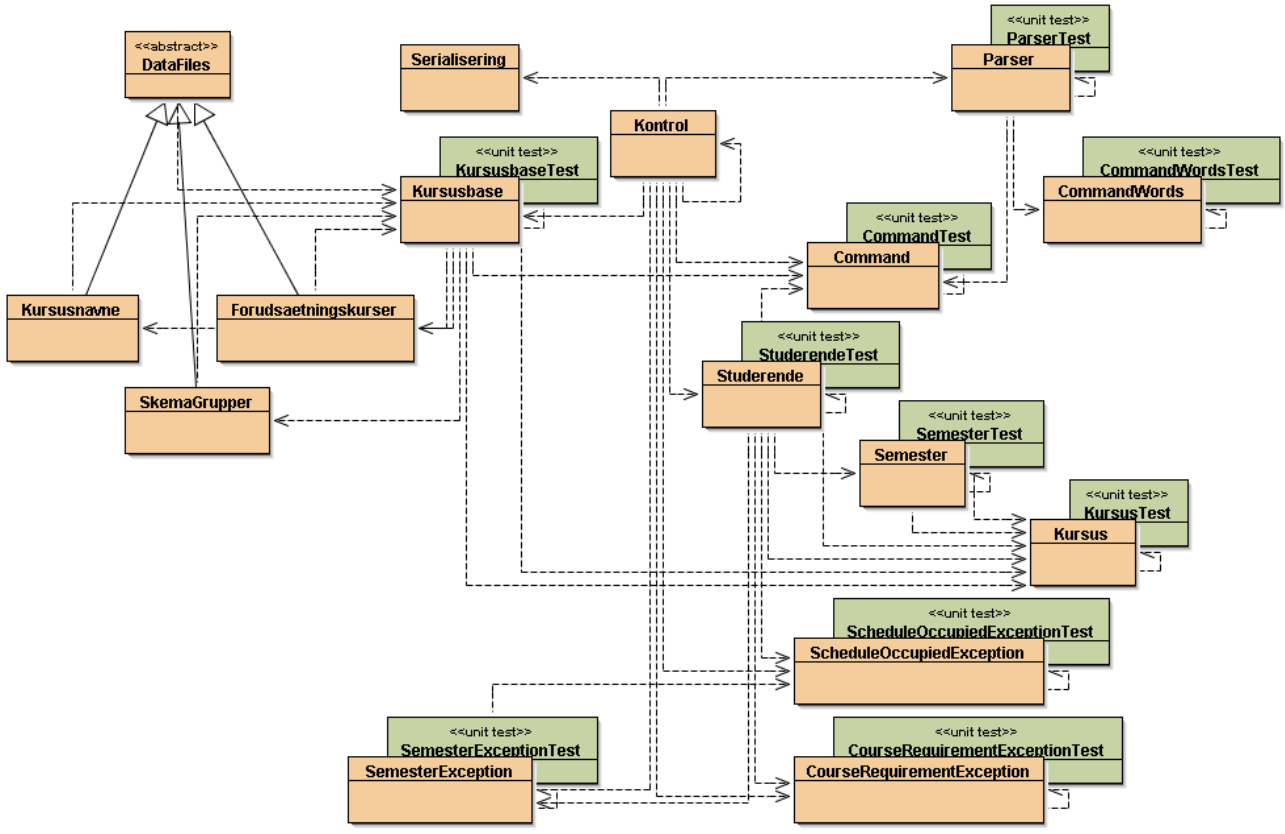

Figure 4: Komplet klassediagram

3.2. Gennemgang af klassen: Kontrol

Konstruktøren:

1) Konstuktøren kræver ingen parametre.
2) Den benytter streng-array'et *legalLocation* og lægger værdierne over i en ArrayList *legalLoc*. ArrayListen indeholder nu alle lovlige skemaplaceringer samt deres position i en ugeplan. Se tabel under krav #12 for skemaposition definition.
3) Herefter oprettes et kursusbase-objekt, hvorved alle datafilerne bliver læst ind. Se mere herom i klassen Kursusbase.
4) Et *activeUser*-objekt af typen Studerende bliver oprettet med parametrene navn="Peter Pedal " og studienummer="s012345".
5) Til sidst bliver der oprettet et *parser*-objekt af typen Parser, som bruges til bruger-interface.

Vi har generelt i Kontrol-koden brugt debug-udskrifter flere steder i metoderne. For nemt at kunne slå dem til og fra, bliver de aktiveret med feltet boolean *debugging = true/false*.

Kontrol indeholder en række metoder, som kan udføre følgende:

⇒ "main" – Starter programmet. Et objekt af typen Kontrol oprettes, og dette objekts metode start() bliver kaldt.

⇒ "start" – Udskriver en velkomstbesked med *printWelcome* metoden og herefter startes en while-løkke, der står og venter på at en *command* af typen Command bliver returneret fra *parser*. Med *processCommand*-metoden bliver command analyseret og behandlet. Hvis der bliver returneret boolean false, vil while-løkken stoppe og en afslutningsbesked vil blive udskrevet inden programmet afsluttes.

⇒ "printWelcome" – Udskriver en velkomstbesked med *activeUser* navn og studienummer i. Alle kommandoerne bliver skrevet ud med *printHelp()* metoden.

⇒ "processCommand" – Analyserer *command*. Første ord checkes for om det er en lovlig kommando. Hvis det ikke er det, bliver en fejlbesked udskrevet, ellers bliver første ord sammenlignet med de lovlige kommandoerne og en specifik metode bliver kaldt.

⇒ "addCourse" – Tilføjer et kursus, hvis alle krav er opfyldt. Først kontrolleres om der er indtastet et ord efter "tilføj", og dernæst om det er en integer. Hvis ikke disse forudsætninger er opfyldt bliver en fejlbesked med hjælpetekst udskrevet. Herefter bliver bruger bedt om at indtaste et semester nr. hvorfor der oprettes et *integerReader*-objekt af typen Scanner, hvori det indtastede semester nr. bliver returneret. Hvis det ikke er en interger eller tallet ligger udenfor MAX_SEMESTRE bliver en fejlbesked udskrevet.
Nu bliver det undersøgt om kursusnummeret er oprettet i kursusbasen. Hvis det er det, bliver *activeUser.addCourse(Course, semesterNo)* metoden kaldt, der efterfølgende tager sig af de praktiske kontrol, såsom om activeUser opfylder kursus-forudsætningerne eller om der evt. skulle være skema-overlab, osv. Se mere herom i klassen Studerende for denne metode – *addCourse*.
Da metoden *activeUser.addCourse* kan kaste en exception under disse tjek, bliver de fanget i Kontrol-klassen, og en fejlbesked bliver udskrevet for den specifikke fejl.

⇒ "printHelp" - uden parameter : Udskriver alle kommandonavnene.
- eller med et kommandonavn som parameter : Udskriver en specifik hjælpetekst for den kommando der er efter fulgt af hjælp. F.eks. "hjælp tilføj", udskriver hjælpetekst for "tilføj".

⇒ "removeCourse" – Undersøger først om kursusnummeret er en integer, og herefter bliver *activeUser.removeCourse(requestedCourseNo)* metoden kaldt, der tager sig af de praktiske tjek, såsom om det i det hele taget er et kursus som *activeUser* har, eller om der er andre kurser der har kurset som forudsætning. Metoden returnerer en status-besked.

⇒ ”gemPersonData” – Gemmer *activeUser*, som er et objekt af typen Studerende, med metoden *gem* fra klassen Serialisering. Men inden da bliver filnavnet lavet ud fra studienummer for *activeUser* med tilføjelse af extention .ser. Et filnavn kan eks. se således ud: ”s012345.ser”. Det hele foregår inde i en try-løkke, da *gem* kan kaste en exception. Herved er det klassen Kontrol, der tager sig af en evt. fejlbesked.

⇒ ”hentPersonData” – Kan hente et objekt af typen Studerende, og sætte *activeUser* til at pege på det indlæste objekt. Herefter bliver der udskrevet en velkomstbesked for den nye bruger. Det hele foregår inde i en try-løkke, så evt. fejl ved fil-indlæsningen kan blive håndteret korrekt. Det kunne f.eks. være, at en studieplan for det opgivne studienummer ikke findes, eller filen ikke kan åbnes/indlæses p.g.a. fejl på harddisken, eller at filen ikke kan lukkes. Ved en af disse situationer kaster *gem*-metoden en exception tilbage til Kontrol-klassen.

⇒ ”viskursus” – Udskriver kursus-beskrivelse (Kursusnummer, kursusnavn, skemaplacering og forudsætninger). Der laves først en verificering af om det ønskede kursusnummer er en integer, og ved fejl udskrives en fejlbesked i form af en hjælpetekst. Ved korrekt indtastet kursusnummer bruges metoden fra *kursusbase.viskursus(kursusnummer)*. Denne metode returnerer en status-streng enten med kursusbeskrivelsen, eller en fejlbesked hvis kursusnummeret ikke findes i kursusbasen.

⇒ ”opretbruger” – Giver mulighed for at oprette en ny bruger. Der laves først en kontrol af om kommandoen er indtastet korrekt (der er dog ingen tjek på om studienummer-syntaksen er korrekt). Herefter bliver bruger bedt om at indtaste brugernavn hvorfor der oprettes en *nameReader* objekt af typen Scanner, hvori det indtastede navn bliver returneret. Nu kan et nyt Studerende-objekt oprettes og activeUser sættes til at pege på det nye objekt. Til sidst udskrives en velkomst-besked til den nye bruger.

⇒ ”quit” – Afslutter programmet med ’afslut’-kommandoen. Returnerer i princippet kun en boolean true, hvilket får while-løkken i start()-metoden til at stoppe. Hvis der fejlagtig er ekstra kommandonavne efter afslut, vil programmet ikke stoppe, men bruger vil få en fejlbesked.

⇒ ”secondWordIsInteger” – Prøver at konvertere ”secondWord” fra *command* til en integer. Det hele gøres i en try-løkke, og hvis der ikke sker fejl, så returneres *true*. Hvis konverteringen derimod ikke er mulig, så opstår en konverteringsfejl, som bliver fanget i catch, der udskriver en fejlbesked og returnere *false*.

3.3. Gennemgang af klassen: Kursusbase

Konstruktøren:

1. Konstuktøren kræver som parameter en ArrayList med informationer om lovlige skemaplaceringer. Feltet *legalLocations* sættes til at pege på dennee ArrayList.
2. Opretter en collection kaldet *kurser* af typen TreeMap<K,V>. Her indeholder K(key) et kursusnummer og V(Value) et Kursusobjekt. Vi har valgt at lade key være en Integer, da kursusnummer altid er et heltal.
3. Opretter et Kursusnavne-objekt *kursusnavne*. Herved startes indlæsning af filen med kursusnavne. Se mere herom i klassen Kursusnavne.
4. Opretter et SkemaGrupper-objekt *skemaGrupper*. Herved startes indlæsning af filen med skemagrupper. Se mere herom i klassen SkemaGrupper.
5. Opretter et Forudsaetningskurser-objekt *forudsaetningskurser*. Herved startes indlæsning af filen med forudsætninger. Se mere herom i klassen Forudsaetningskurser.

Ved at bruge en TreeMap collection, fås en meget hurtig søgning på et bestemt kursus ud fra et givent kursusnummer, da elementerne i TreeMap'en ligger i en sorteret rækkefølge i forhold til deres key. Dette giver også en hurtig måde at få udskrevet alle kurserne i en ordnet rækkefølge ud fra deres kursusnummer. En HashMap kunne også fint havde været brugt, men den ville ikke være lige så hurtig ved en søgning, da elementerne så ikke ligger i en ordnet rækkefølge. Under beslutningstidspunktet for Collection type faldt valget på TreeMap, ud fra ovenstående betragtning, men her var kursusnavnefilen langt fra færdig. Nu hvor den er færdig kan vi se at alle kurserne ligger i ordnet rækkefølge ud fra deres kursusnumre, hvilket betyder at man praktisk taget blot kunne indlæse alle kurserne i en ArrayList, og få dem listet ud. Men når man skal finde et specifikt kursus ud fra kursusnummer, så er en TreeMap at foretrække.
Man kan jo altid diskutere hvor hurtig det skal være, hvilket jo også er i forhold til størrelsen på kursusbasen, cpu kraft, memory, osv.

Kursusbase indeholder en række metoder, som kan udføre følgende:

⇒ "getCourse" – Returnerer et kursusobjekt ud fra kursusnummer (key). Her bliver returneret et "null" hvis det aktuelle kursus ikke findes i *kurser*.

⇒ "addCourseNumberAndName" – Opretter et kursusobjekt og tilføjer samtidig kursusnummer og navn i objektes konstruktør. Herefter bliver kursusobjektet tilføjet *kurser,* med kursusnummer som key.

⇒ "addScheduleLocation" – tilføjer en skemalokation til et kursusobjekt. Først verificeres om kursusnummer findes i *kurser*. Derefter verificeres det, om det er en lovlig lokation ved at slå op i ArrayList *legalLocations*. Først herefter kan det aktuelle kursusobjekt få tilføjet en lokation med *addLocation()*-metoden fra *Kursus*. Hvis ovenstående verificering ikke kan opfyldes, bliver en specifik fejlmelding udskrevet. Dette var brugbart under udarbejdelsen af programmet, da der var en del fejl i de første versioner af skemaplaceringsfilen "skema.txt".

⇒ "addCourseRequirement" – tilføjer et forudsætningskursus for et kursusobjekt. Først verificeres om kursusnummer findes i *kurser*. Derefter verificeres det, om forudsætningskurset er oprettet i *kurser*. Først herefter kan det aktuelle kursusobjekt få tilføjet et forudsætningskursus med *addRequirement()*-metoden fra *Kursus.* Hvis ovenstående verificering ikke kan opfyldes, bliver en specifik fejlmelding udskrevet. Dette var brugbart under udarbejdelsen af programmet, da der var en del fejl i de første versioner af forudsætningsfilen "forud.txt".

⇒ "toString" – Returnere en komplet streng for hele databasen (*kurser*) med alle kursus-informationerne i (kursusnummer, navn, skemalokationer samt forudsætninger).
Der bliver lavet en ny collection af typen Set<Kursus> med navnet *sortedDetails*, der indeholder alle kursusobjekterne fra kurser i en ordnet rækkefølge. Her gøres brug at metoden

values() fra TreeMap, som returnerer kursusobjekterne i den rækkefølge deres key ligger i. For at afgøre rækkefølgen bruger TreeMap klassen interfacet *Comparable*, som skal bruges i *Kursus*, der gør brug af metoden compareTo(Kursus …) (Se mere herom i klassen *Kursus*). Nu kan *sortedDetails* nemt blive tilgået én for én i en for-each løkke. Under hver cyklus bliver toString()-metoden brugt fra hvert kursusobjekt, der returnere detaljerne for kurset.

⇒ ”viskursus” – Returnerer en streng med komplet kursusinformation (kursusnummer, navn, skemalokationer samt forudsætninger) for et specifikt kursus.

3.4. Gennemgang af klassen: DataFiles

Konstruktøren:

1) Der er ingen konstruktør til denne superklasse.

Formålet med klassen DataFiles er udelukkende at være superklasse for klasserne Kursusnavne, SkemaGrupper og Forudsaetningskurser, hvilke gør den til en abstrakt klasse.
Formålet er at håndtere indlæsningen af tekst datafilerne, og lade de tre sub-klasser bearbejde datafilerne på hver deres måde linie for linie. Da det er tekstfiler, bliver filerne læst ind med klassen *reader*, der kan håndtere ”character streams”.
For hver linie der er indlæst, bliver ”addLineToMap”-metoden brugt fra sub-klasserne, hvor desikrering af den enkelte linie foregår samt tjek om det første ord er en integer. Dette tjek kunne i princippet havde været lavet i superklassen, da det er en fælles ting for alle sub-klasserne, men så ville det ikke længere være en specifik fejlmelding man ville få, ved en evt. fejl i én af datafilerne.

DataFiles indeholder en række metoder, som kan udføre følgende:

⇒ ”readFile” – læser alle linier i den specifikke fil. For hver linie der læses, bliver linier kortere end fem karakterer ignoreret. Dette er for at håndtere filer, der evt. ikke overholder datastrukturen i filen. (Det er set under dette projektforløb, at de tre datafiler havde syntaksfejl). Derefter bliver en metode addLineToMap(line) fra sub-klassen brugt.

⇒ ”addLineToMap” – er en abstrakt metode. Den bruges i sub klasserne og er derfor erklæret her som abstrakt.

3.5. Gennemgang af klassen: Kursusnavne

Denne klasse er en sub-klasse, der nedarver fra klassen DataFiles.

Konstruktøren:

1) Konstuktøren kræver som parameter et Kursusbase-objekt *kursusbase* samt et filnavn *filename* for filen med kursusnavne.
2) Feltet med kursusbasen bliver sat lig referencen til den parameter-overførte kursusbase, og derefter bliver *readFile*-metoden i superklassen DataFiles kaldt, og indlæsningen af filen med kursusnavne begynder – linie for linie.

Kursusbase-objektet bliver brugt her for at give adgang til metoden *addCourseNumberAndName*, der tilføjer kursusnummer og navn til kursusdatabasen.

Kursusnavne indeholder en metode, som kan udføre følgende:

⇒ ”addLineToMap” – får som inputparameter en tekststreng, der er blevet læst ind fra superklassen DataFiles. Strengen bliver desikeret, hvor først kursusnummer bliver taget ud og lavet om til en integer, og dernæst bliver kursusnavnet gemt. Disse to parametre bliver derefter tilføjet Kursusbase med *addCourseNumberAndName(kursusnummer, kursusnavn)*-metoden.
Konvertering fra streng til integer foregår inde i en try-løkke, for at undgå programmet i at gå ned, hvis der er en fejl i kursusnavne-filen. Ved en fejl, vil det pågældende kursus ikke blive tilføjet, og der vil komme en specifik fejlbesked.

3.6. Gennemgang af klassen: Skemagrupper

Denne klasse er en sub-klasse, der nedarver fra klassen DataFiles.

Konstruktøren:

3) Konstuktøren kræver som parameter et Kursusbase-objekt *kursusbase* samt et filnavn *filename* for filen med skemagrupper.
4) Feltet med kursusbasen bliver sat lig referencen til den parameter-overførte kursusbase, og derefter bliver *readFile*-metoden i superklassen DataFiles kaldt, og indlæsningen af filen med skemagrupper begynder – linie for linie.

Kursusbase-objektet bliver brugt her for at give adgang til metoden *addScheduleLocation*, der tilføjer en kursuslokation for et specifikt kursusobjekt i kursusdatabasen *kurser*.

SkemaGrupper indeholder en metode, som kan udføre følgende:

⇒ ”addLineToMap” – får som inputparameter en tekststreng, der er blevet læst ind fra superklassen DataFiles. Her bruger vi Scanner-klassen, som er en simpel tekst scanner, der som default deler en streng op for hvert mellemrum i strengen. Scanner giver frie muligheder for forskellige streng-metoder. Her bruger vi dog kun *next()* og *hasNext()*. Strengen bliver desikeret hvor først kursusnummer bliver taget ud og lavet om til en integer. Her bliver ”-e” og ”-f” foran kursusnummeret slettet og ikke brugt mere, da det er dobbelt kontekst, idet det allerede kan ses ud fra kursuslokation, om det er et efterårs- eller forårs-semester ud fra begyndelsesbogstavet ”E” eller ”F”.
Dernæst bliver det undersøgt, om der er en skemalokation for det aktuelle kursus, og hvis der er det, bliver kursusnummer samt lokation overført til kursusbasens metode *addScheduleLocation(kursusnummer, lokation)*. Dette bliver gentaget, indtil at der ikke er flere skemalokationer tilbage.
Konvertering fra streng til integer foregår inde i en try-løkke, for at undgå programmet i at gå ned, hvis der er en fejl i skemagruppefilen. Ved en fejl, vil det pågældende kursus’s skemaplasering ikke blive tilføjet, og der vil komme en specifik fejlbesked.

3.7. Gennemgang af klassen: Forudsaetningskurser

Denne klasse er en sub klasse der nedarver fra DataFiles klassen.

Konstruktøren:

1) Konstuktøren kræver som parameter et Kursusbase-objekt *kursusbase* samt et filnavn *filename* for forudsætningsfilen.
2) Feltet med kursusbasen bliver sat lig referencen til den parameter-overførte kursusbase, og derefter bliver *readFile*-metoden i superklassen DataFiles kaldt, og indlæsningen af filen med forudsætningskurser begynder – linie for linie.

Kursusbase-objektet bliver brugt her for at give adgang til metoden *addCourseRequirement*, der tilføjer et forudsætningskursus for et specifikt kursusobjekt i kursusdatabasen *kurser*.

Forudsaetningskurser indeholder en metode, som kan udføre følgende:

⇒ ”addLineToMap” – får som inputparameter en tekststreng, der er blevet læst ind fra superklassen DataFiles. Her bruger vi igen Scanner klassen, som er en simpel tekst scanner, der som default deler en streng op for hvert mellemrum i strengen. Strengen bliver desikeret, hvor først kursusnummer bliver taget ud og lavet om til integer.
Dernæst bliver det undersøgt, om der er et forudsætningskursus for det aktuelle kursus, og hvis der er det, bliver kursusnummeret samt forudsætningskurset tilføjet kursusbasen med *addCourseRequirement(kursusnummer, forudsætningskursus)* metoden. Dette bliver gentaget, indtil at der ikke er flere forudsætningskurser tilbage.
Konvertering fra streng til integer foregår inde i en try-løkke, for at undgå programmet i at gå ned, hvis der er en fejl i forudsætningsfilen. Ved en fejl vil den pågældende kursusforudsætning ikke blive tilføjet, og der vil komme en specifik fejlbesked.

3.8. Gennemgang af klassen: Serialisering

Konstruktøren:

1) Der er ingen konstruktør til denne klasse.

Formålet med klassen Serialisering er at kunne gemme eller hente en studieplan for en studerende. Alt information for en studerende (studienummer, navn og kursusplan) ligger gemt i klassen *Studerende*. Derfor er det oplagt at bruge serialization, der tillader at gemme eller hente hele objekter. Da vi her arbejder med *activeUser* (et objekt af typen studerende), er det en simpel sag blot at gemme activeUser eller at hente et specifikt objekt og sætte *activeUser* til at referere til det. For at kunne gemme Studerende må denne klasse, samt de klasser Studerende bruger (såsom Semester og Kursus), alle have implementeret interfacet Serialization.

Serialisering indeholder to metoder, som kan udføre følgende:

⇒ ”gem” – får som input parameter et objekt *obj* samt et filnavn *filename*. Der åbnes en fil med navnet *filename*, hvorefter *obj* bliver gemt. Til slut bliver filen lukket. Hvis der opstår fejl, kaster metoden en IOException tilbage.

⇒ ”hent” – får som input parameter et filnavn *filename*. Der åbnes en fil med navnet *filename*, hvorefter *obj* bliver læst. Til slut bliver filen lukket og *obj* bliver returneret. Hvis der opstår fejl, kaster metoden en Exception tilbage. Årsagen til at ”hent” kaster en ”Exception” og ikke kun en ”IOException” er, at der kan opstå en fejl under typecasting til typen Studerende, hvis det hentede objekt ikke matcher denne klasse.

3.9. Gennemgang af klassen: Studerende

Konstruktøren:

1) Konstuktøren kræver som inputparametre navn og studienummer på den studerende.
2) De to input parametre bliver lagt ind i objektets felter *navn* og *studieNr*.
3) Der bliver også oprettet en *studieplan* at typen *HashMap<Integer, Semester>* hvor HashMap key er en Integer (semesternummer) og værdien er et objekt af typen Semester.

Klassen Studerende indeholder alt information om en given studerende. Den indeholder den studerendes navn, studienummer samt alle semestre hvorpå den studerende har valgt kurser. Hvert af disse semestre indeholder de kurser, som den studerende har valgt for det givne semester. Kort sagt, den indeholder den studerendes studieplan.
Studieplanen ligger i *studieplan* og er lavet som collection af typen HashMap<Integer,Semester>. Den er valgt som HashMap, fordi det er nemt at holde styr på semester-rækkefølgen, da semesternummeret angives med Integer som key, dvs Integer=1 er lig første semester, hvilket giver en nem og hurtig adgang til et bestemt semester.
En alternativ collection er en ArrayListe, hvor man skal løbe igennem hele array'et (eller store dele af det), hver gang man skulle finde et givent semester med det semesternummer, som man ønskede. Disse årsager har gjort, at der er valgt en HashMap.
Mellemliggende semestre, som er tomme, genereres også. Men tomme semestre i slutningen af studieplanen fjernes.
Studerende implementerer interfacet Serializable, da et objekt af denne klasse skal kunne gemmes i en enkelt fil. Se mere herom i klassen Serialisering.

Studerende indeholder en række metoder, som kan udføre følgende:

⇒ "getName" – Returnerer *navn* for den aktuelle studerende.

⇒ "getStudieNr" – Returnerer *studieNr* for den aktuelle studerende.

⇒ "addCourse" – skal have et objekt af typen Kursus kaldet *course*, og en integer *semesterNo* som inputparameter. Metoden tilføjer et eller flere Semester objekt/er, hvis de ikke allerede er oprettet i *studieplan*, og den tilføjer et Kursus *course* i semesternummer *semesterNo* ind i *studieplan* og returnerer en status besked til brugeren.
Metode gennemgang:
   1) Opretter op til *semesterNo* nye semester objekter, hvis ikke de allerede findes i *studieplan*.
   2) Henter skemaplasering(erne) for *course*, og lægger dem over i *wantedScheduleLocations* som er en arrayListe.
   3) Undersøger om semestret (*semesterNo*) er ledigt på den givne skemaplacering. Det foregår ved at hente allerede valgte kurser for det ønskede semester ind i previousSelectedeCourses som er en arrayListe.
   4) I to for-each løkker undersøges det om der er sammenfald mellem det nye *course* skema-plasering og tidligere valgte kurser skemaplaseringer. Her kastes en *ScheduleOccupiedException* hvis der er sammenfald.
   5) Undersøger om forudsætningerne er opfyldt. Henter forudsætningerne for course og lægger dem over i *forudsaetninger* som er en arrayList.
   6) I en for-each løkke undersøges det om kurserne i *forudsaetninger* er med i et tidligere semester med metoden – *courseIsPriorInPlan*. Her kastes en *CourseRequirementException* hvis der mangler et forudsætningskursus.
   7) Undersøger om *semesterNo* er et forårs- eller efterårs-semester, og gennemgår *wantedScheduleLocations* for det valgte *course* og ser om det indeholder mindst én skemaplacering på det korrekte halvår. Her kastes en *SemesterException*, hvis kurset ikke udbydes for det specifikke halvår.

8) Undersøger om kurset er et enkeltsemester-kursus, hvilket i bekræftende fald betyder at det ikke må ligge andre steder i *studieplan*. Hvis kurset er et enkeltsemester-kusus og ligger på et tidligere semester, så kastes en *SemesterException*. Hvis det strækker sig over flere semestre, bliver bruger gjort opmærksom på selv at indsætte det de relevante steder.
9) Kurset *course* kan nu blive tilføjet *studieplan* i semester *semesterNo*.

Hvis kurset ikke kunne tilføjes (af de forskellige mulige årsager nævnt ovenfor), så sørger metoden for at slette eventuelle tomme semestre, som blev genereret i starten af metoden. Således undgår der at ligge tomme semestre i enden af studieplanen.

⇒ ”removeCourse” – skal have et kursusnummer *requestedCourseNo* som inputparameter. Metoden fjerner et specifikt kursus fra *studieplan* (i alle semestre hvor det optræder). Fjerner også ubrugte semestre i *studieplan* hvis de sidste skulle været blevet tomme. Dog bliver kurset ikke fjernet, hvis der er andre kurser, der har det som forudsætning. Dette vil brugeren i så fald blive gjort opmærksom på i retur-strengen.
Metode gennemgang:

1) Opretter en collection arrayList af typen *Integer* kaldet *isOnSemester*. Her bliver semesternumre, hvor kursus *requestedCourseNo* ligger, lagt over i.
2) I en for-each løkke gennemgås de fundne semestre hver især, hvor det for hvert semester undersøges, om der er andre efterfølgende kurser på senere semestre, der har *requestedCourseNo* som et forudsætningskursus. Her bruges metoden *findDependsOn*.
3) Hvis der ikke er nogen kurser på et senere semester, som har kursus *requestedCourseNo* som forudsætning, bliver kurset fjernet i det aktuelle semester fra *studieplan* med *Semester* metoden *removeCourse(requestedCourseNo)*. Til sidst bliver der returneret en streng med information om at kurset er fjernet fra studieplanen.
4) Hvis der derimod blev fundet kurser på et senere semester, som har kursus *requestedCourseNo* som forudsætning, bliver kurset ikke fjernet, og en returstreng med information om hvilket kursus der er afhængig af *requestedCourseNo* bliver returneret.

⇒ ”cleanUpEmtySemestersAtEnd” – fjerner tomme semestre i enden af studieplanen, som ikke længere indeholder et kursus. I en for-løkke bliver semestrene undersøgt (startende med det sidste semester) m.h.t. om det er et tomt semester. Det sker med Semester-metoden *isEmpty*. Hvis semesteret er tomt, bliver det fjernet. Dette fortsætter, indtil at der kommer et semester med et kursus. Her springer programmet ud af metoden, da der så ikke er flere tomme semestre i enden af studieplanen.

⇒ ”courseIsPriorInPlan” – får som input parametre *neededCourse* og *currentSemesterNo*. Returnerer true hvis kurset *neededCourse* er fundet på et tidligere semester end *currentSemesterNo*.

⇒ ”courseIsDualSemester” – får som input parameter et Kursusobjekt *course*. Returnerer true hvis kursus har skemalokation både forår og efterår. I en for-each løkke bliver alle skemalokationer for *course* gennemlæst og verificeret om der både er forår og efterår-semestre.

⇒ ”findDependsOn” – får som input parametre Semester *semester* og *requestedCourseNo*. Undersøger om andre senere valgte kurser afhænger af *requestedCourseNo*. Returnerer det først fundne kursusnummer, som har *requestedCourseNo* som forudsætning.

⇒ ”visplanHeader” – er en hjælpemetode for visplan. Den returnerer en header-streng med alle ugens dage.

⇒ ”visplanRow” – er en hjælpemetode for visplan. Den returnerer en tekststreng med en række til skemaet, med enten formiddags- eller eftermiddags-kurserne for det specifikke semester.

⇒ ”visplan” – får som inputparameter en arrayList *legalLoc*, der indeholder alle skemaplaceringer i en bestemt rækkefølge ud fra skemaposition definitionen i henhold til krav #12. Den returnerer en komplet streng med alle semestrene og de kurser der findes i *studiplan*.

3.10. Gennemgang af klassen: Semester

Konstruktøren:

1) Konstuktøren kræver som input parameter et semesternummer.
2) Den opretter en ny ArrayListe af typen *Kursus* kaldet *kurser*. Her skal senere placeres de kurser, som activeUser vælger for det specifikke semester.
3) Semesternummeret fra inputparameteren bliver lagt i objektets *semesterNo* felt.

Der er ikke nogen (set fra brugeren) brugbar fejl-håndtering i denne klasse, da parametrene allerede er kontrolleret i klasserne *Kontrol* eller *Studerende*.
Semester implementerer interfacet Serializable, da klassen bliver brugt i Studerende og et objekt af denne klasse skal kunne gemmes i en enkelt fil. Se mere herom i klassen Serialisering.

Semester indeholder en række metoder, som kan udføre følgende:

⇒ ”getCourses” – Returnerer en ArrayListe *kurser*, som indeholder de kurser, der er tilføjet for det aktuelle semester.
⇒ ”addCourse” – Tilføjer et kursus til *kurser* for det aktuelle semester. Metoden melder om programfejl, hvis kurset som ønskes tilføjet er ’null’. Dette bør ikke kunne ske. Derfor er der ikke lavet yderligere fejlhåndtering.
⇒ ”removeCourse” – Fjerner et kursus fra *kurser* for det aktuelle semester.
⇒ ”containsCourse” – Returnerer boolean true hvis det forespurgte *courseNo* ligger i *kurser*.
⇒ ”isEmty” – Returnere boolean true, hvis der ingen kurser ligger i *kurser* for det aktuelle semester.
⇒ ”getSemesterNo” – Returnere det aktuelle semesternummer.

3.11. Gennemgang af klassen: Kursus

Konstruktøren:

1) Konstuktøren kræver som inputparametre et kursusnummer samt et kursusnavn.
2) De to inputparametre bliver lagt ind i objektets felter *kursusnummer* og *kursusnavn*.
3) Der bliver oprettet en ArrayListe af typen String kaldet *skemaPlaceringer*. Her kan alle skemaplaseringerne blive lagt ind for det specifikke kursus.
4) Der bliver oprettet en ArrayListe af typen Integer kaldet *forudsætninger*. Her kan alle forudsætningskursernes kursusnumre blive lagt ind for det specifikke kursus.

Kursus implementerer Serializable interfacet, da klassen bliver brugt i Semester, som igen bliver brugt i Studerende, og da et objekt af denne klasse skal kunne gemmes i en enkelt fil. Se mere herom i klassen Serialisering.
Kursus implementerer Comparable interfacet, da man i Kursusbase bruger treeMap metoden *values()*, som gør brug af Comparable interfacet. Den bruges for at få en ny collection med alle kurserne i en ordnet rækkefølge, hvor de er sorteret efter kursusnummer. Comparable gør brug at metoden CompareTo, som vi har lavet her i Kursus-klassen. Se beskrivelse af denne metode nedenfor.

Kursus indeholder en række metoder, som kan udføre følgende:

⇒ ”addLocation” – tilføjer en lokation til *skemaPlaceringer* for det aktuelle kursus.
⇒ ”addRequirement” – tilføjer forudsætnings-kursusnummer til *forudsætninger* for det aktuelle kursus.
⇒ ”getName” – Returnerer *kursusnavn* for det aktuelle Kursus.
⇒ ”getForudsaetninger” – Returnerer *forudsætninger* for det aktuelle Kursus.
⇒ ”getSkemaPlaceringer” – Returnerer *skemaPlaceringer* for det aktuelle Kursus.
⇒ ”getCourseNo” – Returnerer *kursusnummer* for det aktuelle Kursus.
⇒ ”toString” – Returnerer en streng med *kursusnavn, kursusnummer*, skemaplaceringer, samt evt. forudsætningskurser for det aktuelle Kursus.
⇒ ”CompareTo” – får som input parameter et Kursusobjekt, som sammenlignes med det aktuelle Kursus. Metoden returnerer -1, 0 eller 1 afhængig af om det aktuelle kursus kommer før, er magen til, eller kommer efter det kursusobjekt, som kom fra input-parameteren. Her sammenlignes med *kursusnummer* (Integer), men i princippet kunne det også have være en tekststreng, der kunne sammenlignes med, f.eks. *kursusnavn*. Kurser ville så blive skrevet ud sorteret efter deres *kursusnavn*. Dette kunne f.eks. være implementeret som en valgbar parameter med ”udskrivbase” kommandoen. Se evt. forbedringsforslag kapitel 6.2.2 side 42.

3.12. Gennemgang af klassen: Parser

Konstruktøren:

1) Konstuktøren kræver ingen parametre.
2) Der bliver oprette et objekt af typen CommandWords kaldet *commands*.
3) Der oprettes et objekt af typen Scanner(System.in) kaldet *reader*. Dette skal bruges til indtastninger fra brugeren (fra keyboard).

Parser indeholder en række metoder, som kan udføre følgende:

⇒ ”getCommand” – Når en kommando er blever eksekveret fra kommando-vinduet, vil *reader.nextLine()* lægge indholdet over i *inputLine*. Dernæst oprettes en *tokenizer* objekt af typen Scanner, hvori den indtastede linie bliver lagt ind i. Nu bliver linien desikeret og første ord bliver lagt over i *word1* og resten af linien bliver lagt over i *word2*.
Herefter bliver det undersøgt om word1 er en lovlig kommando (med metoden *isCommand* fra *Command*), og hvis det er det, bliver de to ord lagt over i et nyt Command objekt kaldet *command(word1, word2)* og returneret. Hvis det ikke var en lovlig kommando i første ord, vil det samme ske, bare hvor *word1=null*.
Metoden returnerer objektet af typen *Command*, hvor det første ord i den indtastede linie ligger i *word1* og resten af linien ligger i *word2*.

⇒ ”getCommandList” – Returnerer en streng med alle lovlige kommandoer via metoden *commands.showAll()* fra klassen CommandWords.

⇒ ”getAllCommandsAsArray” – Returnerer en ArrayList af typen String med alle lovlige kommandoer via metoden *commands.getAllCommandsAsArray()* fra klassen CommandWords.

3.13. Gennemgang af klassen: CommandWords

Konstruktøren:

1) Konstuktøren kræver ingen parametre og den laver ikke noget.

Der bliver erklæret et felt *validCommands*, som består af et string-array hvori alle lovlige kommandoer er oprettet.

CommandsWords indeholder en række metoder, som kan udføre følgende:

⇒ ”isCommand” – Returnerer true hvis input-parameteren *aString* er en lovlig kommando, der ligger i *validCommands*. Ellers bliver der returneret false.

⇒ ”showAll” – Returnerer en streng med alle navne på lovlige kommandoer.

⇒ ”getAllCommandsAsArray” – Returnerer et arrayList med alle lovlige kommandoer.

3.14. Gennemgang af klassen: Command

Konstruktøren:
1) Konstuktøren kræver som input parametre to strenge – *firstWord* og *secondWord*.
2) De to input-parametre bliver lagt ind i objektets felter *commandWord* henholdsvis *secondWord*.

Command indeholder en række metoder, som kan udføre følgende:
⇒ ”getCommandWord” – Returnerer *commandWord*.
⇒ ”getSecondWord” – Returnerer *secondWord*.
⇒ ”isUnknown” – Returnerer true hvis *commandWord=null*, dvs. hvis det ikke er en lovlig kommando. Dette sker under oprettelsen af et Command objekt i Parser-klassen, hvis første ord ikke er en lovlig kommando.
⇒ ”hasSecondWord” – Returnerer true hvis *secondWord* indeholder noget.

3.15. Gennemgang af klassen: CourseRequirementException

Denne klasse er en sub klasse der nedarver fra Exception klassen.

Konstruktøren:
1) Konstuktøren kræver som input parameter en streng – *key*.
2) Input parameteren bliver lagt ind i objektets felt *key*.

CourseRequirementException indeholder en række metoder, som kan udføre følgende:
⇒ ”getKey” – Returnerer *key*.
⇒ ”toString” – Returnerer en streng, der indeholder *key*.

3.16. Gennemgang af klassen: ScheduleOccupiedException

Denne klasse er magen til klassen CourseRequirementException, bortset fra klassens navn. Se derfor kapitel 3.15 side 27.

3.17. Gennemgang af klassen: SemesterException

Denne klasse er magen til klassen CourseRequirementException, bortset fra klassens navn. Se derfor kapitel 3.15 side 27.

4. Afprøvning / test

Der er 3 testklasser, som tester den studerende og dens tilhørende studieplan:
- StuderendeTest
- SemesterTest
- KursusTest

Der er 3 testklasser, som tester kommando-interfacet:
- ParserTest
- CommandTest
- CommandWordsTest

Der er 3 testklasser, som tester user-defined exceptions:
- SemesterExceptionTest
- CourseRequirementExceptionTest
- ScheduleOccupiedExceptionTest

Der er lavet 1 testklasse, som tester kursusbasen:
- KursusbaseTest

Der er ikke lavet JUnit tests af klassen Kontrol, da denne klasse ikke opretter et objekt på objekt-bench'en (da den kun har main-metoden som public), og vi derfor ikke har kunnet lave en JUnit test på samme måde som for de øvrige JUnit testklasser. Se evt. forbedringsforslag i kapitel 6.1.7 side 41.

Metoder, som har kunnet blive lavet Private, er naturligvis lavet Private for at give størst mulig uafhængighed mellem klasserne. Private-metoder er der ikke lavet JUnit tests på.

I de efterfølgende kapitler gennemgås de ovenfor omtalte testklasser.

4.1. StuderendeTest

Denne testklasse indeholder:

Metode 'SetUp', som opretter en default studerende ved navn "Peter Pedal" med studienummer "s012345".

Der er herudover 3 tests:

- testConstructor: Tester konstruktøren ved at udlæse navnet og studienummeret, som skal stemme med det der blev genereret i 'SetUp'.
- testNegativeTesting: Tester at et nyt kursus kan oprettes incl. en skemaplacering for kurset. Herefter tilføjes kurset til den studerendes studieplan på semester 1. Kurset 1017 forsøges herefter fjernet, og det verificeres at der kommer en fejlbesked med at kurset 1017 ikke kan fjernes, da det ikke indgår i studieplanen.
- testAddRemove: Tester at et nyt kursus kan oprettes incl. en skemaplacering for kurset. Herefter fjernes kurset igen, og det verificeres at der kommer besked til brugeren om at kurset er fjernet.

Koden til testklassen ses i Appendix B, kapitel A.26 side 79.

4.2. SemesterTest

Denne testklasse indeholder:

Metode ’SetUp’, som opretter et tomt semester-objekt med semesternummer 1. Metoden opretter også et kursus-objekt med kursusnummer 1017 samt titel ”Diskret matematik og databaser".

Der er herudover 2 tests:

- test_semesterNo_isEmpty_containsCourse: Tester at konstruktøren af semestret (fra SetUp) har virket samt at metoderne ’getSemesterNo’, ’isEmpty’, ’addCourse’ samt ’containsCourse’ alle virker (positiv test).
- testNegativeTesting: Tester at metoden ’containsCourse’ returnerer false for et kursus, som ikke er på semestret. Tester også, at metoden ’isEmpty’ returnerer false når et kursus er adderet til semestret.

Koden til testklassen ses i Appendix B kapitel A.22 side 71.

4.3. KursusTest

Denne testklasse indeholder:

Metode ’SetUp’, som opretter et kursus-objekt med kursusnummer 2105 samt titel ”Algoritmer og Datastrukturer I". Herefter tillægges kurset forudsætningskurserne 1017 samt 2101. Herefter sættes kursets skemaplacering til ”F2B”. Kursus 1017 oprettes også som selvstændigt objekt.

Der er herudover 2 tests:

- testConstructor: Verificerer at konstruktøren samt metoderne ’getCourseNo’ og ’getName’ virker (positiv test).
- test_Requirement_Location_Size: Tester at forudsætningerne er lagt korrekt ind, dvs tester metoderne ’addRequirement’ samt ’getForudsaetninger’ (positiv test). Tester også at kurset har præcis 2 forudsætninger.
  Tester også at skemaplaceringerne er lagt korrekt ind, dvs tester metoderne ’addLocation’ samt ’getSkemaPlaceringer’.
  Tester til sidst, at metoden ’compareTo’ også virker (ved at teste om kurset er magen til sig selv).
- testNegativeTesting: Tester negativ test på metoden ’getCourseNo’ ved at kurset oprettet under ’SetUp’ ikke har kursusnummer 1234.
  Tester negativ test på metoden ’getName’ ved at kurset oprettet under ’SetUp’ ikke har navnet ”Hjemkundskab II”.
  Tester negativ test på at kurset 2405 ikke er en del af forudsætningerne samt at der ikke kun er ét forudsætningskursus (da der er indsat både 1017 samt 2101).
  Tester at skemaplaceringen ”E1A” ikke er med i kursets skemaplaceringer.
  Tester at antallet af skemaplaceringer er forskelligt fra 2 (da det bør være 1).
  Tester at metoden ’compareTo’ giver false ved sammenligning af 2 forskellige kurser.

Koden til testklassen ses i Appendix B, kapitel A.15 side 63.

4.4. ParserTest

Denne testklasse indeholder metoden ’testAll’, som tester.

- Kontruktøren af Parseren.
- At accessor getCommandList virker (positiv test).
- At accessor getAllCommandsAsArray indeholder en kendt kommando (positiv test)
- At accessor getAllCommandAsArray ikke indeholder en (specifik) ukendt kommando (negativ test).
- At antallet af kendte kommandoer er som forventet (i alt 10 kommandoer).

Metoden getCommand er det ikke lykkedes at lave test for, da BlueJ ikke giver et inputvidue under generering af testmetoder.

Koden til testklassen ses i Appendix B, kapitel A.17 side 66.

4.5. CommandTest

Denne testklasse indeholder 2 tests:

- testPositivTest: Tester konstruktøren ved at oprette et kommando-objekt med ord1=”viskursus” og ord2=”2101”.
  Tester, at accessor ’getCommandWord’ returnerer ”viskursus” (positiv test).
  Tester, at accessor ’getSecondWord’ returnerer ”2101” (positiv test).
  Tester, at metoden ’hasSecordWord’ returnerer ’true’ (positiv test).
  Tester, at metoden ’isUnknown’ returnerer ’false’ (negativ test).
- testNegativTest: Tester at konstruktøren kan håndtere ord2 sat til ’null’. Først oprettes et kommando-objekt med ord1=”Peter” og ord2=’null’.
  Tester, at accessor ’get.CommandWord’ returnerer ”Peter” (positiv test).
  Tester, at accessor ’getSecondWord’ returnerer ‘null’ (positiv test).
  Tester, at metoden ’hasSecordWord’ returnerer ’false’ (negativ test).
  Tester, at metoden ’isUnknown’ returnerer ’true’ (positiv test).

Koden til testklassen ses i Appendix B, kapitel A.3 side 45.

4.6. CommandWordsTest

Denne testklasse tester:

- Konstruktøren til klassen.
- Metoden ’isCommand’ med negativ test (en ukendt kommando).
- Metoden ’is Command’ med positiv test (en kendt kommando).
- Metoden ’showAll’, som udskriver samtlige kendte kommandoer til en streng.

Koden til testklassen ses i Appendix B, kapitel A.5 side 48.

4.7. SemesterExceptionTest

Denne testklasse tester:

- Konstruktøren til klassen med tilhørende streng ”Teststreng”.
- Metoden ’getKey’ med positiv test (matcher ”Teststreng”).
- Metoden ’toString’ med positiv test (matcher ”Teststreng”).

Koden til testklassen ses i Appendix B, kapitel A.21 side 70.

4.8. CourseRequirementExceptionTest

Denne testklasse tester magen til testen i kapitel 4.7 side 31.

4.9. ScheduleOccupiedExceptionTest

Denne testklasse tester magen til testen i kapitel 4.7 side 31.

4.10. KursusbaseTest

Denne test benytter en ArrayList med skemalokationer til at køre konstruktøren til klassen ’Kursusbase’. I denne forbindelse testes også indirekte positiv test af klasserne DataFiles, Kursusnavne, SkemaGrupper samt Forudsaetningskurser, da disse er med til at danne kursusbasen.

Det tjekkes, at kursus 02105 er blevet indlæst. Følgende tjekkes for kurset:

- kursusnummer (=2105)
- kursusnavn (=”Algoritmer og Datastrukturer I”)
- skemalokationen (=”F2B”)
- at der kun er én lokation for dette kursus (metoden size=1)
- forudsætningskurset (=2101)
- at der kun er ét forudsætningskursus for dette kursus (metoden size=1)

Det tjekkes endvidere, at et fiktivt kursus med kursusnummer 99999 kan tilføjes incl. kursusnummer, kursusnavn, en skemaplacering samt en forudsætning. Herefter testes på tilsvarende vis for kursus 99999:

- kursusnummer (=99999)
- kursusnavn (=”Testkursus”)
- skemalokationen (=”E3A”)
- at der kun er én lokation for dette kursus (metoden size=1)
- forudsætningskurset (=2101)
- at der kun er ét forudsætningskursus for dette kursus (metoden size=1)

Koden til testklassen ses i Appendix B, kapitel A.13 side 61.

4.11. Kørsel af alle JUnit tests

.Alle JUnit tests kan køres på én gang ved brug af BlueJ miljøet ved tryk på knappen ”Run tests”. Resultatet ses af Figure 5.

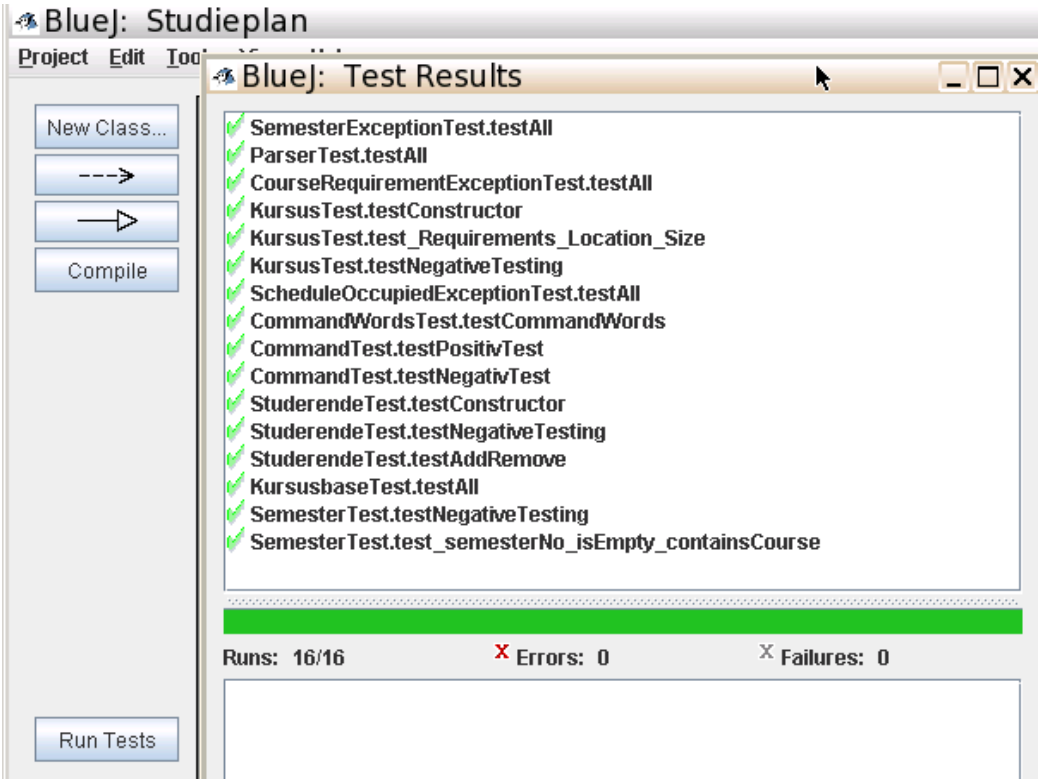

Figure 5: Output ved valg af "Run Tests" fra BlueJ.

5. Kørselsdokumentation

Man starter projektet via BlueJ-miljøet, hvor man højreklikker på ”Kontrol” og vælger "void main(String args[])" og tryk Enter. Studieplan er herved startet.
Programmet vil altid starte med en default bruger/studerende her – Peter Pedal med studienummer s012345, se nedenstående opstartsvindue:

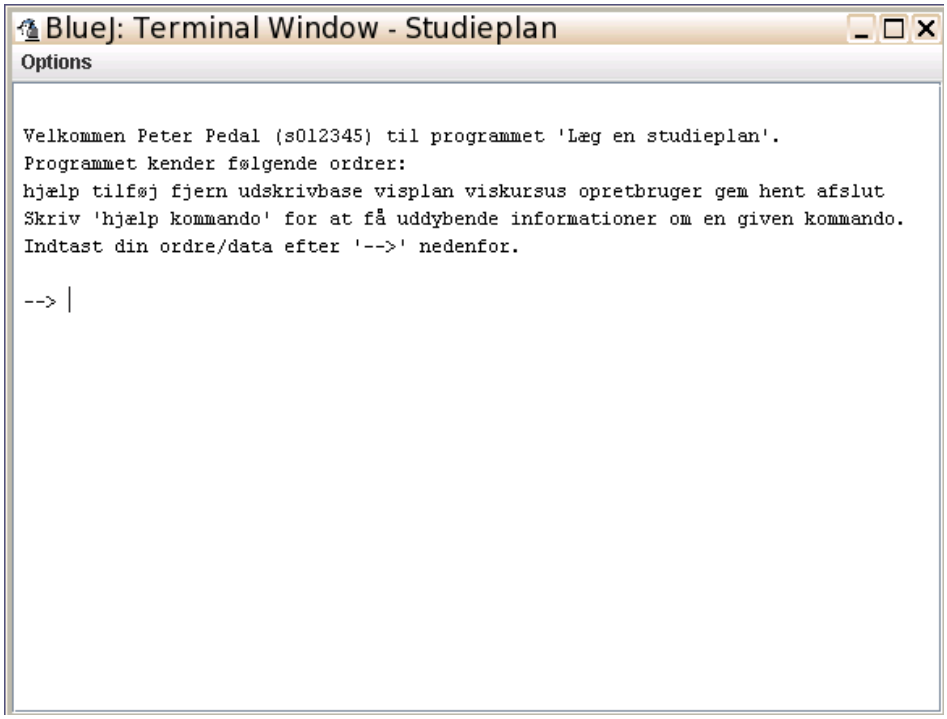

Figure 6: Opstartsvindue programmet.

5.1. Programkørsel demo

Her er en programkørsel, hvor vi kommer igennem de fleste bruger-operationer samt nogle fejl-operationer, for at vise hvordan programmet håndterer det:

Følgende punkter er med i eksemplet:

- Peter Pedal (s012345) er default bruger.
- Tilføje forskellige lovlige kurser på flere semestre.
- Tilføje lovlige dobbelt kurser om efterår og forår.
- Viser kursusplan
- Tilføje et kursus hvis forudsætning ikke er opfyldt.
- Tilføje et efterårskursus om foråret og få en fejl besked.
- Fjern et kursus som andre tilføjet kurser har som forudsætning og få en fejl besked.
- Vis kursusdata for et enkelt kursus.
- Fjern et kursus og se at den viste kursus plan bliver opdateret.
- Gemmer Peter Pedals studieplan.
- Opretter ny bruger: s543210 – Manden med den gule hat.
- Tilføjer et kursus til den nye bruger.
- Viser kursusplan for Manden med den gule hat.
- Gemmer Manden med den gule hat's studieplan.
- Fjerner et kursus som ikke findes i studieplanen.
- Tilføjer et kursus som ikke findes i kursusdatabasen.
- Tilføjer et kursus som Manden med den gule hat ikke har kursus forudsætninger til.
- Henter en studieplan for en der ikke er lavet (filen findes ikke).
- Henter s012345 Peter Pedals kursus plan.
- Viser Peter Pedals kursusplan.
- Afprøver hjælp – til alle kommandoer.
- Viser hele kursusdatabasen. (har ikke medtaget alle kurser da det ville fylde for meget. Ca. 25 sider)

```
Velkommen Peter Pedal (s012345) til programmet 'Læg en studieplan'.
Programmet kender følgende ordrer:
hjælp tilføj fjern udskrivbase visplan viskursus opretbruger gem hent afslut
Skriv 'hjælp kommando' for at få uddybende informationer om en given kommando.
Indtast din ordre/data efter '-->' nedenfor.

--> tilføj 1005
Angiv semesternummer
--> 1
Vær opmærksom på at kursus 1005 strækker sig over flere semestre,
og at du selv skal indsætte det de relevante steder.
Dette program tilføjer ikke automatisk i flere semestre.
Kurset 1005 er blevet tilføjet.
--> tilføj 1005
Angiv semesternummer
--> 2
Vær opmærksom på at kursus 1005 strækker sig over flere semestre,
og at du selv skal indsætte det de relevante steder.
Dette program tilføjer ikke automatisk i flere semestre.
Kurset 1005 er blevet tilføjet.
--> tilføj 1931
Angiv semesternummer
--> 1

Kurset 1931 er blevet tilføjet.
--> tilføj 2101
```

```
Angiv semesternummer
--> 1

Kurset 2101 er blevet tilføjet.
--> tilføj 10020
Angiv semesternummer
--> 1
Vær opmærksom på at kursus 10020 strækker sig over flere semestre,
og at du selv skal indsætte det de relevante steder.
Dette program tilføjer ikke automatisk i flere semestre.
Kurset 10020 er blevet tilføjet.
--> tilføj 10020
Angiv semesternummer
--> 2
Vær opmærksom på at kursus 10020 strækker sig over flere semestre,
og at du selv skal indsætte det de relevante steder.
Dette program tilføjer ikke automatisk i flere semestre.
Kurset 10020 er blevet tilføjet.
--> tilføj 2131
Angiv semesternummer
--> 3

Kurset 2131 er blevet tilføjet.
--> tilføj 2151
Angiv semesternummer
--> 3

Kurset 2151 er blevet tilføjet.
--> visplan
Studerende: Peter Pedal
Studienr. : s012345
-------------------------------------------------------------------------
|  Semester |  Tid  |  Mandag | Tirsdag |  Onsdag | Torsdag |  Fredag |
-------------------------------------------------------------------------
|     1     |  8-12 |  -----  |   2101  |   1005  |  -----  |   1931  |
| (Efterår) | 13-17 |  -----  |  -----  |   1005  |  10020  |   1005  |
-------------------------------------------------------------------------
|     2     |  8-12 |  -----  |  -----  |   1005  |  -----  |  -----  |
|  (Forår)  | 13-17 |  -----  |  -----  |   1005  |  10020  |   1005  |
-------------------------------------------------------------------------
|     3     |  8-12 |  -----  |  -----  |   2131  |   2151  |  -----  |
| (Efterår) | 13-17 |   2151  |  -----  |   2131  |  -----  |  -----  |
-------------------------------------------------------------------------

--> tilføj 2152
Angiv semesternummer
--> 3
Fejl: Kurset mangler forudsaetningskursus: 2131.
--> tilføj 2152
Angiv semesternummer
--> 4
Fejl: Kurset udbydes ikke om foråret.
--> tilføj 2152
Angiv semesternummer
--> 5

Kurset 2152 er blevet tilføjet.
--> visplan
Studerende: Peter Pedal
Studienr. : s012345
-------------------------------------------------------------------------
|  Semester |  Tid  |  Mandag | Tirsdag |  Onsdag | Torsdag |  Fredag |
-------------------------------------------------------------------------
|     1     |  8-12 |  -----  |   2101  |   1005  |  -----  |   1931  |
| (Efterår) | 13-17 |  -----  |  -----  |   1005  |  10020  |   1005  |
-------------------------------------------------------------------------
|     2     |  8-12 |  -----  |  -----  |   1005  |  -----  |  -----  |
|  (Forår)  | 13-17 |  -----  |  -----  |   1005  |  10020  |   1005  |
-------------------------------------------------------------------------
|     3     |  8-12 |  -----  |  -----  |   2131  |   2151  |  -----  |
| (Efterår) | 13-17 |   2151  |  -----  |   2131  |  -----  |  -----  |
```

```
-------------------------------------------------------------------------
|     4     |  8-12 |  -----  |  -----  |  -----  |  -----  |  -----  |
|  (Forår)  | 13-17 |  -----  |  -----  |  -----  |  -----  |  -----  |
-------------------------------------------------------------------------
|     5     |  8-12 |   2152  |  -----  |  -----  |  -----  |  -----  |
| (Efterår) | 13-17 |  -----  |  -----  |  -----  |   2152  |  -----  |
-------------------------------------------------------------------------

--> fjern 2101
Kurset 2101 kan ikke fjernes, da 2131 er i planen og har 2101 som forudsætning.
Fjern først 2131.

--> fjern 2131
Kurset 2131 kan ikke fjernes, da 2152 er i planen og har 2131 som forudsætning.
Fjern først 2152.

--> viskursus 2152
Nr:  2152: Navn          : Parallelle systemer
           Skemaplacering: E1A E1B
           Forudsætninger: 2101 2131
--> fjern 2152
Kurset 2152 er fjernet fra semester 5.
--> visplan
Studerende: Peter Pedal
Studienr. : s012345
-------------------------------------------------------------------------
|  Semester |  Tid  |  Mandag | Tirsdag |  Onsdag | Torsdag |  Fredag |
-------------------------------------------------------------------------
|     1     |  8-12 |  -----  |   2101  |   1005  |  -----  |   1931  |
| (Efterår) | 13-17 |  -----  |  -----  |   1005  |  10020  |   1005  |
-------------------------------------------------------------------------
|     2     |  8-12 |  -----  |  -----  |   1005  |  -----  |  -----  |
|  (Forår)  | 13-17 |  -----  |  -----  |   1005  |  10020  |   1005  |
-------------------------------------------------------------------------
|     3     |  8-12 |  -----  |  -----  |   2131  |   2151  |  -----  |
| (Efterår) | 13-17 |   2151  |  -----  |   2131  |  -----  |  -----  |
-------------------------------------------------------------------------

--> gem
Gemmer studieplan for en specifik studerende (bruger studienummer som filnavn) på
harddisk.
Person data gemt.
--> opretbruger s543210
Indtast dit navn: Manden med den gule hat

Velkommen Manden med den gule hat (s543210) til programmet 'Læg en studieplan'.
Programmet kender følgende ordrer:
hjælp tilføj fjern udskrivbase visplan viskursus opretbruger gem hent afslut
Skriv 'hjælp kommando' for at få uddybende informationer om en given kommando.
Indtast din ordre/data efter '-->' nedenfor.

--> tilføj 1815
Angiv semesternummer
--> 1

Kurset 1815 er blevet tilføjet.
--> visplan
Studerende: Manden med den gule hat
Studienr. : s543210
-------------------------------------------------------------------------
|  Semester |  Tid  |  Mandag | Tirsdag |  Onsdag | Torsdag |  Fredag |
-------------------------------------------------------------------------
|     1     |  8-12 |  -----  |  -----  |  -----  |  -----  |  -----  |
| (Efterår) | 13-17 |  -----  |  -----  |  -----  |   1815  |  -----  |
-------------------------------------------------------------------------

--> gem
Gemmer studieplan for en specifik studerende (bruger studienummer som filnavn) på
harddisk.
Person data gemt.
--> fjern 2141
```

```
Kurset 2141 kan ikke fjernes, da det ikke eksisterer i studieplanen.
--> tilføj 1900
Angiv semesternummer
--> 1
Fejl: Kurset '1900' eksisterer ikke.
--> tilføj 1932
Angiv semesternummer
--> 2
Fejl: Kurset mangler forudsaetningskursus: 1931.
--> hent s567890
Kan ikke åbne fil: s567890.ser . Gad vide om du ikke har skrevet forkert :-)
--> hent s012345
Person data hentet.

Velkommen Peter Pedal (s012345) til programmet 'Læg en studieplan'.
Programmet kender følgende ordrer:
hjælp tilføj fjern udskrivbase visplan viskursus opretbruger gem hent afslut
Skriv 'hjælp kommando' for at få uddybende informationer om en given kommando.
Indtast din ordre/data efter '-->' nedenfor.

--> visplan
Studerende: Peter Pedal
Studienr. : s012345
-------------------------------------------------------------------------
|  Semester |  Tid  |  Mandag | Tirsdag |  Onsdag | Torsdag |  Fredag |
-------------------------------------------------------------------------
|     1     |  8-12 |  -----  |   2101  |   1005  |  -----  |   1931  |
| (Efterår) | 13-17 |  -----  |  -----  |   1005  |  10020  |   1005  |
-------------------------------------------------------------------------
|     2     |  8-12 |  -----  |  -----  |   1005  |  -----  |  -----  |
|  (Forår)  | 13-17 |  -----  |  -----  |   1005  |  10020  |   1005  |
-------------------------------------------------------------------------
|     3     |  8-12 |  -----  |  -----  |   2131  |   2151  |  -----  |
| (Efterår) | 13-17 |   2151  |  -----  |   2131  |  -----  |  -----  |
-------------------------------------------------------------------------

--> hjælp tilføj
Kommandoen 'tilføj' skal bruges med 1 parameter, som skal være et heltal.
Kommandoen tilføjer kurset med det angivne kursusnummer, eks 'tilføj 2101'.
Hereftert skal man indtaste det semesternummer (heltal), hvorpå man ønsker kurset.

--> hjælp fjern
Kommandoen 'fjern' skal bruges med 1 parameter, som skal være et heltal.
Kommandoen fjerner kurset med det angivne kursusnummer, eks 'fjern 2101'.
Kurset fjernes, hvis ikke øvrige efterfølgende kurser afhænger af kurset.

--> hjælp udskrivbase
Kommandoen 'udskrivbase' udskriver alle kurser i databasen.

--> hjælp visplan
Kommandoen 'visplan' viser den aktuelle studieplan for den aktuelle elev.

--> hjælp viskursus
Kommandoen 'viskursus' skal bruges med 1 parameter, som skal være et heltal.
Kommandoen viser detaljer om kurset med det angivne kursusnummer, eks 'viskursus 2101'.

--> hjælp opretbruger
Kommandoen 'opretbruger' skal bruges med 1 parameter, som skal være et ord (en streng).
Kommandoen opretter en ny studerende i systemet med
det angivne ord som studienummer, eks 'opretbruger s012345'.
Kommandoen skifter samtidigt automatisk til denne nye bruger.

--> hjælp gem
Kommandoen 'gem' gemmer den aktive bruger incl. dens studieplan til en fil.
Filen placeres samme sted som programmet køres fra,
og filens navn bliver brugerens studienummer med extension '.ser'.

--> hjælp hent
Kommandoen 'hent' skal bruges med 1 parameter, som skal være et ord (et studienummer).
Kommandoen henter en tidligere gemt studerende incl. dens tidligere valgte studieplan.
Kommandoen skifter samtidigt automatisk til denne bruger.
```

```
--> hjælp afslut
Kommandoen 'afslut' afslutter programmet.

--> udskrivbase
Nr:  1005: Navn          : Matematik 1
           Skemaplacering: E5A E5B E3B F5A F5B F3B
           Forudsætninger: Ingen
Nr:  1007: Navn          : Lineære og differentiable matematiske modeller
           Skemaplacering: E2A F2A
           Forudsætninger: Ingen
Nr:  1009: Navn          : Matematik 1 - for sundhed og produktion
           Skemaplacering: E1A E2A E2B F1A F2A F2B
           Forudsætninger: Ingen
Nr:  1017: Navn          : Diskret matematik og databaser
           Skemaplacering: E1B
           Forudsætninger: Ingen
Nr:  1035: Navn          : Matematik 2
           Skemaplacering: E1A E2B F2B
           Forudsætninger: Ingen
Nr:  1141: Navn          : Kompleks funktionsteori
           Skemaplacering: E2B
           Forudsætninger: 1005
Nr:  1227: Navn          : Grafteori
           Skemaplacering: F1B
           Forudsætninger: 1005
Nr:  1234: Navn          : Differentialgeometri med Anvendelser
           Skemaplacering: F2B
           Forudsætninger: 1005
Nr:  1246: Navn          : Partielle differentialligninger - anvendt matematik
           Skemaplacering: E5A E5B
           Forudsætninger: 1035
Nr:  1250: Navn          : Funktionalanalyse og global analyse
           Skemaplacering: F4A F4B
           Forudsætninger: 1035
Nr:  1259: Navn          : Fejlrettende koder
           Skemaplacering: F1A F1B
           Forudsætninger: 1017
Nr:  1319: Navn          : Seminar course on Mathematical Biology: Models based on
Differential Equations
           Skemaplacering: E4A
           Forudsætninger: Ingen
Nr:  1410: Navn          : Kryptologi 1
           Skemaplacering: E3A
           Forudsætninger: 1017
Nr:  1415: Navn          : Computational Discrete Mathematics
           Skemaplacering: E2B
           Forudsætninger: Ingen
Nr:  1426: Navn          : Kryptologi 2
           Skemaplacering: F2A
           Forudsætninger: Ingen
Nr:  1450: Navn          : Dynamiske systemer, ikke-lineariteter og solitoner
           Skemaplacering: E1B
           Forudsætninger: 1035 1246
```

OSV, OSV, OSV.....

```
Nr: 88394: Navn          : Ingeniørkunstens historie og kultur
           Skemaplacering: E5A E5B
           Forudsætninger: Ingen
Nr: 88395: Navn          : Ingeniørkunstens historie og kultur
           Skemaplacering: E5A E5B
           Forudsætninger: Ingen
Nr: 88401: Navn          : Patentkursus
           Skemaplacering: F2A
           Forudsætninger: Ingen
Nr: 88533: Navn          : Filosofisk etik
           Skemaplacering: E5B F5B
           Forudsætninger: Ingen
Nr: 88890: Navn          : Engelsk 1
           Skemaplacering: E4B E3A F4B F3A
```

```
           Forudsætninger: Ingen
Nr: 88891: Navn          : Engelsk 2
           Skemaplacering: F4B
           Forudsætninger: 88890

--> afslut
Farvel Peter Pedal.
```

6. Forbedringsforslag

6.1. Klassen Kontrol

6.1.1. Metoden ’main’

Programmet starter op med en default bruger ved navn ”Peter Pedal” med studienummer ”s012345”. Brugeren kan så oprette sig selv som bruger med kommandoen ’opretbruger <studienummer>’, eller brugeren kan køre videre med profilen for Peter Pedal.
Ét af følgende forbedringsforslag ville være passende:

1. Programmet starter med at bede om studienummer, og hvis studienummeret ikke findes som gemt data, så oprettes brugeren som en ny bruger.
2. Den sidste bruger af systemet er default næste gang programmet startes op. Dvs programmet husker hvilken bruger der sidst kørte programmet.

6.1.2. Metoden ’hjælp kommando’

Klassen ’Kontrol’ indeholder metoden ’hjælp’, som er implementeret med headeren
`Private String printHelp(Command command)`

.Denne metode har et stykke nede i koden følgende kode:

```
} else if (word2.equals("tilføj")){
                    returnStr += "Kommandoen 'tilføj' skal bruges med 1 parameter, som skal være et heltal.\n";
                    returnStr += "Kommandoen tilføjer kurset med det angivne kursusnummer, eks 'tilføj 2101'.\n";
                    returnStr += "Hereftert skal man indtaste det semesternummer (heltal), hvorpå man ønsker
kurset.\n";
```

Det er upraktisk, at hjælpeteksten for en given kommando ligger i klassen ’Kontrol’, da kommandoerne selv ligger i enum-klassen ’CommandWords’. Det havde været pænere at få hjælpeteksten linket sammen med kommandonavnet inde i enum-klassen ’CommandWords’, så det var samlet ét sted.

Et andet forslag er at lave hver kommando til et selvstændigt objekt med en kommandonavn og en hjælpebeskrivelse. Således vil det være kædet sammen i samme objekt.

6.1.3. Metoden ’opretbruger’

Klassen ’Kontrol’ indeholder metoden ’opretbruger’, som opretter en ny bruger ud fra studienummer og navn. Der er p.t. intet tjek af om brugeren tidligere har brugt systemet og gemt sin tidligere studieplan. Det ville være pænt, hvis programmet derfor fortalte brugeren, at brugeren (ud fra studienummeret) allerede havde en profil liggende.
På nuværende tidspunkt skal man selv vide (eller kikke via en fil-browser), om man har sine data gemt fra en tidligere programkørsel.

6.1.4. Metoden ’tilføj’

Ved mislykket forsøg på at tilføje et kursus til studieplanen, uskrives kun første fundne fejl. Hvis eks. kursus 02101 allerede er placeret på semester 1 i forvejen, og brugeren forsøger at placere kurset endnu engang på samme semester, så udskrives fejlen om at skemaplaceringen er optaget. Dette er for så vidt ikke forkert, men det ville være pænere hvis samtlige fejl/konflikter blev udskrevet (dvs at kurset også allerede er i studieplanen). Årsagen til den nuværende funktionalitet er, at metoden afsluttes så snart første fejl opstår. Dette kan laves om, så samtlige konflikter undersøges før at programmet returneres fra kodeblokken ’tilføj’.

6.1.5. Metoden 'afslut'

Klassen 'Kontrol' indeholder metoden 'afslut', som blot afslutter programmet. Det vil være pænt, hvis programmet spørger om brugeren i den forbindelse ønsker at gemme eventuelle ændringer.

6.1.6. Metoden 'udskrivbase'

Klassen 'Kontrol' indeholder metoden 'udskrivbase', som lister samtlige kurser, som en studerende kan vælge imellem. Det kunne overvejes at liste en begrænset mængde kurser, hvor brugeren så skal trykke eks. "space" for at se næste portion af kurser. Herved vil brugeren have mulighed for at læse om kurserne uden først at skulle scrolle op.

Brugeren skal p.t. huske at slå "Unlimited buffering" til i BlueJ's terminalvindue. Ellers vil brugeren ikke kunne scrolle op til de første kurser i listen.

6.1.7. JUnit test

Klassen 'Kontrol' indeholder p.t. kun én Public metode – nemlig main-metoden, som starter programmet. Ved forsøg på at lave JUnit tests på dette har vi ikke kunnet få et objekt frem på objekt bench'en, hvilket gør at vi umiddelbart ikke kan lave JUnit tests på den sædvanlige måde i BlueJ. En JUnit test vil formentlig kunne lade sig gøre ved at addere endnu en konstruktør til klassen 'Kontrol', som evt. tager en parameter, som så blot ikke benyttes i koden efterfølgende. Således vil man formentligt kunne skabe et objekt på objekt bench'en og derved kunne lave JUnit tests. Men at tilføje kode til klassen 'Kontrol' alene med formålet at kunne lave JUnit test er nedprioriteret i vores projekt.

6.2. Klassen Kursus

6.2.1. Collection-typen for forudsætninger

Klassen 'Kursus' indeholder en en ArrayListe<Integer> med kursusnumrene på kurser, som er en forudsætning. Det ville have været bedre at have ArrayListe<Kursus>, da man så kunne hente oplysninger om forudsætningskursernes navne, m.v.
Det ville eksempelvis være praktisk, at man under 'viskursus xxxxx' ville få vist både kursusnummer og kursusnavn på de kurser, som var forudsætninger for det aktuelle kursus. Dette kan ikke lade sig gøre nu, da toString() køres fra et kursus-objekt, som ikke har adgang til hele kursusbasen, og som derfor ikke på en nem og pæn programmeringsmåde kan få information om et kursusnavn ud fra et kursusnummer. Havde man i stedet valgt forudsætnings-collection som ArrayList<Kursus>, så ville det have været nemt at få information om kursusnavnene.

6.2.2. Metoden 'compareTo'

Klassen 'Kursus' indeholder metoden 'compareTo', som p.t. får et kursusobjekt som input-parameter. Den sammenligner så det overførte kursus' kursusnummer (heltal) med objektets eget kursusnummer. Metoden returnere så -1, 0 eller 1 afhængig om det aktuelle kursus kommer før, er ens, eller kommer efter det Kursusobjekt som kom fra input parameteren.
Dette bevirker, at kurserne bliver udskrevet sorteret efter kursusnummeret.

Hvis man i stedet sammenlignede kursernes kursusnavne (dvs String-sammenligning), så ville alle kurserne i stedet blive udskrevet sorteret efter kursusnavnet.

Det ville være smart, hvis metoden compareTo havde en parameter mere, som fastsatte om der skulle sorteres efter kursusnummer eller kursusnavn. Kommandoen 'udskrivbase' kunne så have mulighed for en 2. parameter, som specificerede hvad brugeren ønskede.
Eksempel på ny metode-header: compareTo(Kursus kursus, boolean sortByName)
Eksempel på brugen af ny kommando:
'udskrivbase' (udskriver sorteret ud fra default kursusnummer)
'udskrivbase kursusnumre' (udskriver sorteret ud fra kursusnummer)
'udskrivbase kursusnavne' (udskriver sorteret ud fra kursusnavn)

6.3. Klassen Kursusbase

6.3.1. Konstruktøren

Konstruktøren af klassen 'Kursusbase' bør kalde en metode, som verificerer at samtlige kurser i kursusbasen har fået tildelt mindst én skemaplacering. Dette er ikke implementeret.

6.4. User-defined Exception-klasser

Der er tre user-defined Exception-klasser:

- SemesterExceptionTest
- CourseRequirementExceptionTest
- ScheduleOccupiedExceptionTest

De er alle ens på nær deres navn. Det virker overdrevet at gøre dette. Den eneste årsag har været, at navnet på exception'en så bliver mere sigende.
Den vigtige information genereres i throw-øjeblikket, hvor alle detaljerne til brugeren lægges i strengen. Selve exception-klassen gør ikke andet end at vise den allerede overførte streng.
En forbedring ville derfor være enten at bruge en allerede defineret klasse for disse 3 exceptions, eller alternativt blot have én user-defineret exception-klasse, eks kaldet "AddCourseException".

A. Kildekode

A.1. Readme

```
Project: Læg en studieplan.
Authors: Michael Holm & Mikael Andersen.
Version: 1.0

Sådan startes programmet: Programmet startes ved at højreklikke i BlueJ på
                          klassen 'Kontrol' og vælge metoden 'main'.

Instruktioner: Der er løbende dialog, som hjælper brugeren. Kommandoen 'hjælp' lister
               alle kommandoer, og hvis der tilføjes en kommando efter 'hjælp', så
               kommer der uddybende forklaring for den pågædende kommando.
```

A.2.Command.class

```
/**
 * Dette er en kopi af klassen 'Command' fra bogens eksempel "World of Zuul".
 * Metoderne er ikke ændret. Kommentarerne er ændret til Dansk, så JavaDoc
 * dokumentationen bliver ensartet.
 *
 * @author  Michael Kolling and David J. Barnes, kommentarer oversat af MH og MAN.
 * @version 2007-12-02
 */

public class Command
{
    private String commandWord;     //Første ord i linien, som brugeren indtaster.
    private String secondWord;      //Andet ord (resten af linien), som brugeren indtaster.

    /**
     * Konstruktør. Første og andet ord skal bruges, men de kan godt være sat til 'null'.
     * @param firstWord Det første ord i kommandoen. 'null' hvis kommandoen ikke kunne genkendes.
     * @param secondWord Resten af linien i kommandoen.
     */
    public Command(String firstWord, String secondWord)
    {
        commandWord = firstWord;
        this.secondWord = secondWord;
    }

    /**
     * Accessor, som returnerer første ord i kommandoen. Hvis kommandoen ikke er en gyldig kommando,
     * så vil denne være sat til 'null'.
     * @return Returnerer det første ord i kommandoen.
     */
    public String getCommandWord()
    {
        return commandWord;
    }

    /**
     * Accessor, som returnerer andet ord (resten af linien) i kommandoen.  Hvis kommandoen ikke har 2 ord, så vil
     * andet ord være sat til 'null', og metoden returnerer i så fald 'null'.
     * @return Returnerer det andet ord (incl. resten af linien) i kommandoen.
     */
    public String getSecondWord()
    {
        return secondWord;
    }

    /**
     * Metode, som undersøger om første ord er en ukendt kommando.
     * @return Returnerer 'true' hvis kommandoen ikke kendes.
     */
    public boolean isUnknown()
    {
        return (commandWord == null);
    }

    /**
     * Metode, som undersøger om kommandoen har mere end 1 ord.
     * @return Returnerer 'true' hvis kommandoen har mere end 1 ord.
     */
    public boolean hasSecondWord()
    {
        return (secondWord != null);
    }
}
```

A.3.CommandTest.class

```
/**
 * JUnit test af klassen CommandTest.
 *
 * @Michael Holm & Mikael Andersen.
 * @2007-12-02
 */
public class CommandTest extends junit.framework.TestCase
{
    /**
     * Default konstruktør for klassen CommandTest.
     */
    public CommandTest()
    {
    }

    /**
     * Metoden køres automatisk før de andre test-metoder i denne klasse.
     */
    protected void setUp()
    {
    }

    /**
     * Afslutter tests.
     * Køres efter hver test-metode.
     */
    protected void tearDown()
    {
    }

    /**
     * JUnit test, som tester at et objekt af typen Command kan oprettes, og at
     * parametrene bliver gemt korrekt i objektet.
     *
     * Tester konstruktøren ved at oprette et kommando-objekt med ord1="viskursus" og ord2="2101".
     * Tester, at accessor 'getCommandWord' returnerer "viskursus" (positiv test).
     * Tester, at accessor 'getSecondWord' returnerer "2101" (positiv test).
     * Tester, at metoden 'hasSecordWord' returnerer 'true' (positiv test).
     * Tester, at metoden 'isUnknown' returnerer 'false' (negativ test).
     */
            public void testPositivTest()
            {
            Command command1 = new Command("viskursus", "2101");
                    assertEquals("viskursus", command1.getCommandWord());
                    assertEquals("2101", command1.getSecondWord());
                    assertEquals(true, command1.hasSecondWord());
                    assertEquals(false, command1.isUnknown());
            }

    /**
     * JUnit test, som tester at et objekt af typen Command kan oprettes, og at
     * konstruktøren kan håndtere hvis 2. ord er sat til 'null'.
     * Opretter et kommando-objekt med ord1="Peter" og ord2='null'.
     * Tester, at accessor 'get.CommandWord' returnerer "Peter" (positiv test).
     * Tester, at accessor 'getSecondWord' returnerer 'null' (positiv test).
     * Tester, at metoden 'hasSecordWord' returnerer 'false' (negativ test).
     * Tester, at metoden 'isUnknown' returnerer 'true' (positiv test).
     */
            public void testNegativTest()
            {
                    Command command1 = new Command("Peter", null);
                    assertEquals("Peter", command1.getCommandWord());
                    assertEquals(null, command1.getSecondWord());  //Genereret manuelt, da BlueJ miløjet ikke returnerede noget på
dette sted.
                    assertEquals(false, command1.hasSecondWord());
                    assertEquals(false, command1.isUnknown());
            }
}
```

A.4.CommandWords.class

```
import java.util.ArrayList;

/**
 * Denne klasse indeholder en 'enumeration' af alle kendte kommandonavne.
 * Den bruges til at genkende kommandoer.
 *
 * Kommandoer: hjælp, tilføj, fjern, udskrivbase, visplan, viskursus, opretbruger, gem, hent samt afslut.
 * hjælp: Udskriver listen af mulige kommandoer.
 * hjælp 'kommando' : Udskriver detaljer om hvordan kommandoen 'kommando' bruges.
 * tilføj xxxxx : Tilføjer kurset med kursusnummer xxxxx til studieplanen.
 * fjern xxxxx: Fjerner kursus med kursusnummer xxxxx fra studieplanen.
 * udskrivbase : Udskriver alle kurser som programmet kender.
 * visplan : Udskriver studieplanen.
 * viskursus xxxxx : Viser kursusdetaljer for kurset med kursusnummer xxxxx. (Kursusnummer, kursusnavn, skemaplaceringer samt
forudsætninger.
 * opretbruger 'studienummer' : Opretter en ny bruger med studienummer 'studienummer' samt navn, som brugeren bliver bedt om at
indtaste.
 * hent 'studienummer' : Skifter til en tidligere gemt bruger med studienummeret 'studienummer'
 * gem : Gemmer den studerendes studieplan incl. den studerendes navn og studienummer.
 * afslut: Afslutter programmet.
 *
 * @author  Michael Holm & Mikael Andersen
 * @version 2007.11.21
 */

public class CommandWords
{
    // Et statisk fast array, som indeholder alle valide kommandoer.
    private static final String[] validCommands = {
        "hjælp", "tilføj", "fjern", "udskrivbase", "visplan", "viskursus", "opretbruger", "gem", "hent", "afslut"
    };

    /**
     * Konstruktør.
     */
    public CommandWords()
    {
    }

    /**
     * Undersøger hvorvidt en given streng er en korrekt kommando.
     * @param aString Streng, som ønskes undersøgt.
     * @return Returnerer'true' hvis strengen er en korrekt kommando.
     */
    public boolean isCommand(String aString)
    {
        for(int i = 0; i < validCommands.length; i++) {
            if(validCommands[i].equals(aString))
                return true;
        }
        return false;  // hvis programmet når hertil, så var strengen ikke fundet blandt korrekte kommandoer.
    }

    /**
     * Returnerer en streng med alle tilladte kommandoer.
     * @return Returnerer en streng med alle tilladte kommandoer.
     */
    public String showAll()
    {
      String returnStr = "";
      for (String command : validCommands) {
         returnStr += command + " ";
        }
      return returnStr;
    }

    /**
     * Returnerer alle tilladte kommandoer i en ArrayListe af strenge.
     * @return Returnerer alle tilladte kommandoer i en ArrayListe af strenge.
     */
    public ArrayList<String> getAllCommandsAsArray()
    {
      ArrayList<String> returnArray = new ArrayList<String>();
      for (String command : validCommands) {
```

```
        returnArray.add(command);
       }
     return returnArray;
   }
}
```

A.5.CommandWordsTest.class

```
/**
 * JUnit test af klassen CommandWordsTest.
 *
 * @Michael Holm & Mikael Andersen
 * @2007-12-01
 */
public class CommandWordsTest extends junit.framework.TestCase
{
    /**
     * Default konstruktør for klassen CommandWordsTest
     */
    public CommandWordsTest()
    {
    }

    /**
     * Metoden køres automatisk før de andre test-metoder i denne klasse.
     */
    protected void setUp()
    {
    }

    /**
     * Afslutter tests.
     * Køres efter hver test-metode.
     */
    protected void tearDown()
    {
    }

    /**
     * Tester:
     * Konstruktøren til klassen.
     * Metoden 'isCommand' med negativ test (en ukendt kommando).
     * Metoden 'is Command' med positiv test (en kendt kommando).
     * Metoden 'showAll', som udskriver samtlige kendte kommandoer til en streng.
     */
    public void testCommandWords()
    {
        CommandWords commandW1 = new CommandWords();
        assertEquals(false, commandW1.isCommand("hej"));
        assertEquals(true, commandW1.isCommand("tilføj"));
        assertEquals("hjælp tilføj fjern udskrivbase visplan viskursus opretbruger gem hent afslut ", commandW1.showAll());
    }
}
```

A.6.CourseRequirementException.class

```
/**
 *
 * @Michael Holm & Mikael Andersen
 * @2007-12-03
 */
public class CourseRequirementException extends Exception
{
    private String key;


    public CourseRequirementException(String key)
    {
     this.key=key;
    }

    public String getKey()
    {
     return key;
    }

    public String toString()
    {
     return key;
    }
}
```

A.7.CourseRequirementExceptionTest.class

```
/**
 * JUnit test af klassen CourseRequirementExceptionTest.
 *
 * @Michael Holm & Mikael Andersen
 * @2007-12-03
 */
public class CourseRequirementExceptionTest extends junit.framework.TestCase
{
    /**
     * Default konstruktør for klassen CourseRequirementExceptionTest
     */
    public CourseRequirementExceptionTest()
    {
    }

    /**
     * Metoden køres automatisk før de andre test-metoder i denne klasse.
     */
    protected void setUp()
    {
    }

    /**
     * Afslutter tests.
     * Køres efter hver test-metode.
     */
    protected void tearDown()
    {
    }

    /**
     * Tester:
     * Konstruktøren til klassen med tilhørende streng "Teststreng".
     * Metoden 'getKey' med positiv test (matcher "Teststreng").
     * Metoden 'toString' med positiv test (matcher "Teststreng").
     */
            public void testAll()
            {
                    CourseRequirementException courseRe1 = new CourseRequirementException("Teststreng");
                    assertEquals("Teststreng", courseRe1.getKey());
                    assertEquals("Teststreng", courseRe1.toString());
            }
}
```

A.8.DataFiles.class

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileNotFoundException;
import java.io.IOException;

/**
 * dataFiles er en superklasse der læser en datafil fra harddisk.
 * Det er en Abstrakt klasse da den kun bliver brugt som superklasse
 * for Kursusnavne, SkemaGrupper og Forudsaetningskurser klasserne.
 * @author Michael Holm & Mikael Andersen
 * @version 2007-11-22
 */
public abstract class DataFiles
{
    public Kursusbase kursusbase;

    /**
     * Constructor for objekter af klassen dataFiles
     * Der er ingen initialisering for denne klasse.
     */
    public DataFiles()
    {
    }

    /**
     * Mutator, som læser en hel fil, linie for linie.
     * Hvis filen ikke kan åbnes, bliver der sendt en fejlmeddelelse.
     * Ligeledes sker der fejl under læsning af filen bliver der
     * sendt en fejlmeddelelse.
     * Endelig bliver der sendt en meddelelse hvis  filen ikke
     * kan lukkes korrekt.
     * @param filename Filnavn for den pågældende fil
     */
    protected void readFile(String fileName)
    {
        BufferedReader reader = null;
        try {
            File resultsFile = new File(fileName);
            reader = new BufferedReader(new FileReader(resultsFile));
            String line = reader.readLine();
            while(line != null) {
                if (line.length() > 5) addLineToMap(line); // linier mindre end 5 char bliver ignoreret.
                line = reader.readLine();
            }
        }
        catch(FileNotFoundException e) {
            System.out.println("Kan ikke åbne fil: " + fileName);
        }
        catch(IOException e) {
            System.out.println("Fejl i læsning af fil: " +fileName);
        }
        finally {
            if(reader != null) {
                try {
                    reader.close();
                }
                catch(IOException e) {
                    System.out.println("Fejl i lukning af fil: " + fileName);
                }
            }
        }
    }

    /**
     * Abstrakt metode:
     * Mutator, som desikere en linie og hiver kursusnummer ud
     * som integer og navn ud som en streng. Disse to parametre bliver
     * tilføjet kursusbasen med addCourseNumberAndName - metoden fra
     * kursusbase.
     * @param line En enkelt linie fra en datafil
     */
    abstract protected void addLineToMap(String line);
}
```

A.9.Forudsaetningskurser.class

```
import java.util.Scanner;

/**
 * Forudsaetningskurser er en en sub klasse der nedarver fra dataFiles.
 * Den behandler en indlæst linie en af gangen og tilføjer
 * forudsætningerne for det aktuelle kursusnummer til Kursusbase
 *
 * @author Michael Holm & Mikael Andersen
 * @version 2007-11-22
 */
public class Forudsaetningskurser extends DataFiles
{

    /**
     * Constructor for objekter af klassen Forudsaetningskurser.
     * @param kursusbase En reference til Objektet kursusbase.
     * @param filename Filnavnet for forudsætnings filen.
     */
    public Forudsaetningskurser(Kursusbase kursusbase, String filename)
    {
        this.kursusbase = kursusbase;
        readFile(filename);
    }

    /**
     * Mutator, som desikerer en linie og hiver kursusnummer ud
     * som integer og forudsætningerne ud som en streng. Disse to parametre bliver
     * tilføjet kursusbasen én for én med addCourseRequirement - metoden fra
     * kursusbase.
     * Hvis der ikke kan laves et interger kursusnummer fra linien
     * bliver linien ignoreret og en fejlbesked bliver udskrevet.
     * @param line En enkelt linie fra kursusnavnefilen
     */
    protected void addLineToMap(String line)
    {
        try {
            Scanner scanner = new Scanner(line);
            String tmpStr = scanner.next().substring(0,5);  // Crop kursusnummer
            int CourseNo=Integer.parseInt(tmpStr);          // Konverter nummer til en integer
            String requirement;
            while (scanner.hasNext()) {
                requirement = scanner.next();               // get schedule requirement/s
                int CourseReq=Integer.parseInt(requirement);
                kursusbase.addCourseRequirement(CourseNo, CourseReq);
            }
        }
        catch (Exception e){
            System.out.println("Der har været en fejl i forudsætnings filen"+e);
        }
    }
}
```

A.10.Kontrol.class

```
import java.util.ArrayList;
import java.util.TreeMap;
import java.util.Scanner;
import java.io.FileNotFoundException;

/**
 * 'Læg en studieplan' er et program, hvor studerende kan opbygge deres individuelle studieplan.
 * Programmet læser fra 3 filer, som danner grunlag for en kursusbase med tilhørende skemaplacering og kursus-
 * forudsætninger. Den studerende kan tilføje og fjerne kurser, få vist samtlige kurser, få vist sin opbyggede
 * studieplan, skifte til en anden bruger (andet navn og studienummer), hente en tidligere gemt studieplan samt
 * gemme sin nuværende studieplan. Hver studerende kan kun have 1 studieplan gemt ad gangen (det gemmes på
 * studienummeret).
 * @author  Michael Holm & Mikael Andersen
 * @version 2007-12-02
 */
public class Kontrol
{
    private Parser parser;
    private Kursusbase kursusbase;
    private Studerende activeUser;
    private static final int MAX_SEMESTRE = 20;  //Det maksimale antal semestre, som kan oprettes.
    private static final int MAX_LOCATIONS = 20;  //Antallet af skemalokationer i legalLocations array'et.
    private final String[] legalLocations = {"E1A","E3A","E5A","E2B","E4B","E2A","E4A","E5B","E1B","E3B",
                                             "F1A","F3A","F5A","F2B","F4B","F2A","F4A","F5B","F1B","F3B"};
    private ArrayList<String> legalLoc;
    private static final boolean debugging = false;

    /**
     * Konstruktør, som genererer lovlige skemaplaceringer som en ArrayListe, samt genererer kursusbasen samt sætter den aktive
     * bruger til en default studerende (Peter Pedal). Feltet (objektet) parser oprettes.
     */
    private Kontrol()
    {
        legalLoc = new ArrayList<String>();
        int index=0;
        while(index<MAX_LOCATIONS) {
            legalLoc.add(legalLocations[index]);
            index++;
        }
        kursusbase = new Kursusbase(legalLoc); //indlæser alle kurser fra filer til TreeMap'en kurser.
        activeUser = new Studerende("Peter Pedal", "s012345");
        parser = new Parser();
    }


    /**
     * Starter programmet.
     */
    public static void main(String args[])
    {
     Kontrol kontrol = new Kontrol();
     kontrol.start();
    }


    /**
     *  Her køres hovedløkken indtil at brugeren ønsker at afslutte programmet.
     */
    private void start()
    {
        printWelcome(activeUser);
        boolean finished = false;
        while (! finished) {
            System.out.print("--> ");     // udskriver prompt'en.
            Command command = parser.getCommand();
            if (debugging) {System.out.println("Debug Kontrol: Er nu klar til at køre kommandoen ");}
            finished = processCommand(command, activeUser);
        }
        System.out.println("Farvel "+activeUser.getName()+".");
    }

    /**
     * Metode, som genererer en velkomst-besked til brugeren samt viser mulige kommandoer.
     */
```

```
    private void printWelcome(Studerende activeUser)
    {
        System.out.println("\nVelkommen "+ activeUser.getName() +" ("+activeUser.getStudieNr()+") til programmet 'Læg en studieplan'.");
        System.out.println(printHelp());
    }

    /**
     * Metode, som eksekverer en given kommando fra en given bruger.
     * @param command Kommandoen, som ønskes udført.
     * @param activeUser Brugeren, som ønsker at udføre den pågældende kommando.
     * @return Returnerer 'true' hvis brugeren ønsker at afslutte programmet.
     */
    private boolean processCommand(Command command, Studerende activeUser)
    {
        if (debugging) {System.out.println("Debug Kontrol: Er inde i processCommand.");}
        boolean wantToQuit = false;
        if(command.isUnknown()) {
            System.out.println("Du har skrevet en ugyldig kommando "+activeUser.getName()+". Skriv 'hjælp' for at få en kommando-
oversigt.");
            return false;
        }
        String commandWord = command.getCommandWord();
        if (commandWord.equals("hjælp"))
            System.out.println(printHelp(command));
        else if (commandWord.equals("tilføj"))
            addCourse(command);
        else if (commandWord.equals("fjern"))
            removeCourse(command);
        else if (commandWord.equals("udskrivbase"))
            System.out.println(kursusbase);  //bruger toString() i Kursusbase som returnere hele databasen
        else if (commandWord.equals("visplan"))
            System.out.println(activeUser.visplan(legalLoc));
        else if (commandWord.equals("viskursus"))
            viskursus(command);  //Skal vise kurset, hvor SecondWord indeholder kursusnummret.
        else if (commandWord.equals("opretbruger"))
            opretbruger(command);
        else if (commandWord.equals("gem"))
            gemPersonData();
        else if (commandWord.equals("hent"))
            hentPersonData(command);
        else if (commandWord.equals("afslut"))
            wantToQuit = quit(command);
        return wantToQuit;
    }

    /**
     * Mutator, som tilføjer et kursus til den aktive brugers kursusplan.
     * @param command Kommando, som ønskes udført. Indeholder "tilføj" som første ord. Andet ord bør indeholde et integer med
kurusnummeret, som ønskes tilføjet.
     */
    private void addCourse(Command command)
    {
        if(!command.hasSecondWord()){
            System.out.println("Du bruger kommandoen forkert!\n"+printHelp(new Command("hjælp","tilføj")));
            return;  //Hvis kommandoen er brugt forkert, så afslut metoden addCourse nu ved at gå ud af metoden.
        }
        if (!secondWordIsInteger(command) ){
            return; //Gå ud af metoden, da 2. ord i kommandoen ikke kan fortolkes som et kursusnummer. (En fejlbesked er allerede blevet
kastet fra metoden secondWordIsInteger).
        }
        String tmpStr = command.getSecondWord();
        int requestedCourseNo = Integer.valueOf(command.getSecondWord()).intValue();
        //Beder nu brugeren om at indtaste semesternummer (integer).
        Boolean semesterNoValid = false;
        int semesterNo = 0;
        while (!semesterNoValid){
            System.out.print("Angiv semesternummer\n--> ");
            try {
                Scanner integerReader = new Scanner(System.in);; //bruges til at indlæse semesternr. Nødvendigt at lave nyt scanner-objekt
hver gang. Ellers virker nextInt() ikke i løkken.
                semesterNo = integerReader.nextInt();
            } catch (Exception e) {
                System.out.println("Forkert angivelse af semesternummer (skal være heltal).");
            }
            if (!(semesterNo>0 && semesterNo<=MAX_SEMESTRE)){
                System.out.println("Semesternummer skal være mellem 1 og "+MAX_SEMESTRE+".");
            } else {
                if (debugging) {System.out.println("Debug Kontrol: gyldigt semesternummer '"+semesterNo+"' indtastet. Fortsætter...");}
                semesterNoValid = true;
```

```
            }
        }
        //Nu haves ønsket kursusnr samt ønsket semesternummer.
        if(kursusbase.getCourse(requestedCourseNo)!=null){  //Hvis kurset eksisterer i kursusbasen
            if (debugging) System.out.println("Debug Kontrol: Kurset "+requestedCourseNo+" eksisterer.");
            try {
                System.out.println(activeUser.addCourse(kursusbase.getCourse(requestedCourseNo),semesterNo));
                System.out.println("Kurset "+requestedCourseNo+" er blevet tilføjet.");
            }
            catch(ScheduleOccupiedException e){  //Det giver ikke så meget mening at have 3 forskellige exceptions, som gør det samme.
                System.out.println("Fejl: "+e);  //Men det er valgt for at vise at man kan catch'e forskellige slags exceptions.
            }
            catch(CourseRequirementException e){
                System.out.println("Fejl: "+e);
            }
            catch(SemesterException e){
                System.out.println("Fejl: "+e);
            }
        } else {
            System.out.println("Fejl: Kurset '"+requestedCourseNo+"' eksisterer ikke.");
        }
    }

    /**
     * Accessor, som genererer en streng med hjælpeinformation om mulige kommandoer.
     * @return Returnerer en streng med hjælpeinformation om mulige kommandoer.
     */
    private String printHelp()
    {
        String returnStr = "";
        returnStr += "Programmet kender følgende ordrer:\n";
        returnStr += parser.getCommandList();  //henter liste med kommandoerne.
        returnStr += "\nSkriv 'hjælp kommando' for at få uddybende informationer om en given kommando.\n";
        returnStr += "Indtast din ordre/data efter '-->' nedenfor.\n";
        return returnStr;
    }

    /**
     * Accesssor, som returnerer en streng med hjælpeinformation.
     * Hvis kommandoen som forespørges på kun indeholder 1 ord, så genereres en liste over alle kendte kommandoer.
     * Hvis kommandoen som forespørges på derimod indeholder 2 ord, så kommer der uddybende hjælp til netop den kommando.
     * @param command Kommadoens første ord skal være 'hjælp' og et eventuelt 2. ord som parameter.
     * @return Returnerer en streng med hjælpeinformation (detaljeret eller generelt, afhængig af parameteren command).
     */
    private String printHelp(Command command)
    {
        String returnStr = "";
        if (command==null){
            System.out.println("Der er opstået en fejl i programmet i klassen 'Kontrol' og metoden 'printHelp(Command command)'.\n");
        }
        if (debugging) System.out.println("Debug: Skal nu tjekke om !command.hasSecondWord");
         if(!command.hasSecondWord())
         {
           // Hvis der ikke er et 2. ord, så skal listen over samtlige kommandoer vises.
            if (debugging) {System.out.println("Debug: Udprinter nu listen for samtlige kommandoer.\n");}
            returnStr += printHelp();
        } else {
            String word2 = command.getSecondWord();
            ArrayList<String> commandArray = parser.getAllCommandsAsArray();
            if (commandArray.contains(word2)){ //"hjælp", "tilføj", "fjern", "udskrivbase", "visplan", "viskursus", "opretbruger", "gem", "hent",
"afslut"
                if (word2.equals("hjælp")){
                    returnStr += "Kommandoen 'hjælp' kan bruges uden parameter til at liste samtlige kommandoer.\n";
                    returnStr += "Hvis den bruges med 1 parameter, så viser den detaljeret hjælp for den angivne kommando.";
                } else if (word2.equals("tilføj")){
                    returnStr += "Kommandoen 'tilføj' skal bruges med 1 parameter, som skal være et heltal.\n";
                    returnStr += "Kommandoen tilføjer kurset med det angivne kursusnummer, eks 'tilføj 2101'.\n";
                    returnStr += "Hereftert skal man indtaste det semesternummer (heltal), hvorpå man ønsker kurset.\n";
                } else if (word2.equals("fjern")){
                    returnStr += "Kommandoen 'fjern' skal bruges med 1 parameter, som skal være et heltal.\n";
                    returnStr += "Kommandoen fjerner kurset med det angivne kursusnummer, eks 'fjern 2101'.\n";
                    returnStr += "Kurset fjernes, hvis ikke øvrige efterfølgende kurser afhænger af kurset.\n";
                } else if (word2.equals("udskrivbase")){
                    returnStr += "Kommandoen 'udskrivbase' udskriver alle kurser i databasen.\n";
                } else if (word2.equals("visplan")){
                    returnStr += "Kommandoen 'visplan' viser den aktuelle studieplan for den aktuelle elev.\n";
                } else if (word2.equals("viskursus")){
                    returnStr += "Kommandoen 'viskursus' skal bruges med 1 parameter, som skal være et heltal.\n";
                    returnStr += "Kommandoen viser detaljer om kurset med det angivne kursusnummer, eks 'viskursus 2101'.\n";
```

```
            } else if (word2.equals("opretbruger")){
                returnStr += "Kommandoen 'opretbruger' skal bruges med 1 parameter, som skal være et ord (en streng).\n";
                returnStr += "Kommandoen opretter en ny studerende i systemet med\ndet angivne ord som studienummer, eks
'opretbruger s012345'.\n";
                returnStr += "Kommandoen skifter samtidigt automatisk til denne nye bruger.\n";
            } else if (word2.equals("gem")){
                returnStr += "Kommandoen 'gem' gemmer den aktive bruger incl. dens studieplan til en fil.\n";
                returnStr += "Filen placeres samme sted som programmet køres fra,\nog filens navn bliver brugerens studienummer med
extension '.ser'.\n";
            } else if (word2.equals("hent")){
                returnStr += "Kommandoen 'hent' skal bruges med 1 parameter, som skal være et ord (et studienummer).\n";
                returnStr += "Kommandoen henter en tidligere gemt studerende incl. dens tidligere valgte studieplan.\n";
                returnStr += "Kommandoen skifter samtidigt automatisk til denne bruger.\n";
            } else if (word2.equals("afslut")){
                returnStr += "Kommandoen 'afslut' afslutter programmet.\n";
            } else {
                returnStr += "Kontrol: printHelp: Denne kommando er desværre ukendt for hjælpe-systemet. Bed programmøren om at
opdatere klassen 'Kontrol'.\n";
            }
        } else {
            returnStr += "Kommandoen '" + word2 + "' eksisterer ikke.\n";
            return returnStr;
        }
    }
    return returnStr;
}

/**
 * Mutator, som fjerner et valgt kursus fra studieplanen for den aktive user (elev).
 * Metoden skriver selv output til system out.
 */
private void removeCourse(Command command)
{
    if (debugging) System.out.println("Debug: Kontrol: removeCourse: Skal nu til at fjerne kursus "+command.getSecondWord());
    if (!secondWordIsInteger(command) ){  //Udskriver en fejlmeddelelse til brugeren, hvis ikke der er indtastet en integer.
        System.out.println("Du bruger kommandoen forkert!\n"+printHelp(new Command("hjælp","fjern")));
        return;
    }
    //Går gennem alle semstre for at se hvor kurset eksisterer. Selv hvis kurset findes på flere semestre (eks. hvis
    //kurset er et to-semester kursus), slettes kurset fra alle semestre.
    int requestedCourseNo = Integer.valueOf(command.getSecondWord()).intValue();
    System.out.println(activeUser.removeCourse(requestedCourseNo)); //Udskriver resultatet af forsøget på at fjerne kurset.
}

/**
 * Metode, som gemmer alt information om den givne bruger incl. valgte kurser.
 */
private void gemPersonData()
{
    System.out.println("Gemmer studieplan for en specifik studerende (bruger studienummer som filnavn) på harddisk.");
    try {
        String fileName=activeUser.getStudieNr()+".ser";
        Serialisering.gem(activeUser,fileName);
        System.out.println("Person data gemt.");
    }
    catch (Exception e) {
        System.out.println("Der er opstået en fejl da studie data var forsøgt gemt." +e);
    }
}

/**
 * Metode, som henter alt information om en given bruger incl. valgte kurser.
 * @param command Kommandoen indeholdende studienummer som 2. ord for den studerende, som ønskes hentet.
 */
private void hentPersonData(Command command)
{
    if(!command.hasSecondWord()){
        System.out.println("Du bruger kommandoen forkert!\n"+printHelp(new Command("hjælp","hent")));
        return;
    }
    if (debugging) System.out.println("Henter nu studieplanen for en specifik studerende (bruger studienummer) på harddisk og
systemet skifter til denne bruger.");
    String fileName="";
    try {
        fileName = command.getSecondWord()+".ser";
        activeUser = (Studerende) Serialisering.hent(fileName);
        System.out.println("Person data hentet.");
        printWelcome(activeUser);
    }
```

```
        catch(FileNotFoundException e) {
            System.out.println("Kan ikke åbne fil: " + fileName +" . Gad vide om du ikke har skrevet forkert :-)");
        }
        catch (Exception e) {
            System.out.println("Der er opstået en fejl da studie data var forsøgt hentet.");
        }
    }

    /**
     * Accessor, som udskriver detaljeret information om et kursus.
     * @param command Kommando indeholdende kursusnummer som 2. ord i kommandoen.
     */
    private void viskursus(Command command)
    {
        if (debugging) System.out.println("Debug Kontrol: viskursus skal nu vise kursusnr "+command.getSecondWord());
        if (!secondWordIsInteger(command) ){
            System.out.println("Du bruger kommandoen forkert!\n"+printHelp(new Command("hjælp","viskursus")));
            return;  //Går ud af metoden viskursus, da kursusnummeret er forkert.
        }
        System.out.println(kursusbase.viskursus(Integer.valueOf(command.getSecondWord()).intValue()));
    }

    /**
     * Mutator, som opretter en nu studerende.
     * @param command Kommando, som indeholder det nye studienummer som 2. ord i kommandoen.
     */
    private void opretbruger(Command command)
    {
        if(!command.hasSecondWord()){
         System.out.println("Du bruger kommandoen forkert!\n"+printHelp(new Command("hjælp","opretbruger")));
         return;
        }
        System.out.print("Indtast dit navn: ");
        String newName="";
        try {
            Scanner nameReader = new Scanner(System.in);; //bruges til at indlæse navnet på den nye bruger.
            newName = nameReader.nextLine();
        } catch (Exception e) {
            System.out.println("Der skete en fejl under indlæsning af navn!");
        }
        activeUser = new Studerende(newName,command.getSecondWord());
        printWelcome(activeUser);
    }

    /**
     * Metode, som indikerer at brugeren har valgt at afslutte programmet.
     * Denne metode kan evt. udvides til at spørge brugeren om den nuværende studieplan ønskes gemt før der afsluttes.
     * @return Returnerer 'true' hvis brugeren har skrevet 'afslut' uden yderligere parametre.
     */
    private boolean quit(Command command)
    {
        if(command.hasSecondWord()) {
            System.out.println("kan ikke afslutte "+command.getSecondWord());
            return false;
        }
        else {
            return true;  // signalerer at vi ønsker at afslutte programmet.
        }
    }

    /**
     * Accessor, som undersøger om kommandoens 2. ord er et heltal.
     * @param command Kommando, som ønskes undersøgt.
     * @return Returnerer 'true' hvis kommandoens 2. ord er et heltal.
     */
    private boolean secondWordIsInteger(Command command)
    {
        String parameter="";
        try {
          int myint = Integer.valueOf(command.getSecondWord()).intValue();
          return true;
        }
        catch (NumberFormatException e)
        {
            System.out.println("Kommandoens parameter '"+command.getSecondWord()+"' er ikke et heltal, hvilket det skal være!");
            return false;
        }
    }
}
```

A.11.Kursus.class

```
import java.util.ArrayList;
import java.lang.Comparable;
import java.io.Serializable;

/**
 * Klassen Kursus indeholder kursusnavn, kursusnummer, en liste over hvilke kurser dette kursus forudsætter,
 * samt en liste over hvor i skemaet dette kursus ligger. Skemaplaceringen indeholder også information om hvorvidt
 * det er et efterårskursus eller et forårskursus.
 *
 * @Michael Holm & Mikael Andersen.
 * @2007-11-29
 */
public class Kursus implements Comparable<Kursus>, Serializable
{
    private int kursusnummer;   //Indeholder kursusnummeret.
    private String kursusnavn;  //Indeholder kursusnavnet.
    private ArrayList<String> skemaPlaceringer;  //Skal indeholde skemaplaceringerne for kurset.
    private ArrayList<Integer> forudsaetninger;  //Skal indeholde numrene på de kurser, som er en forudsætning for dette kursus.

    /**
     * Konstruktør for objekter af klassen Kursus
     * @param kursusnummer Kursets nummer.
     * @param kursusnavn Kursets navn.
     */
    public Kursus(int kursusnummer, String kursusnavn)
    {
        this.kursusnummer = kursusnummer;
        this.kursusnavn = kursusnavn;
        skemaPlaceringer = new ArrayList<String>();
        forudsaetninger = new ArrayList<Integer>();
    }

    /**
     * Mutator, som tilføjer en skemaplacering for kurset.
     * @param location Lokationen, som ønskes tilføjet til skemaplaceringen for kurset.
     */
    public void addLocation(String location) {
            skemaPlaceringer.add(location);
    }

    /**
     * Mutator, som tilføjer en forudsætning til kurset.
     * @param requirement Forudsætningskursets kursusnummer.
     */
    public void addRequirement(int requirement){
            forudsaetninger.add(requirement);
    }

    /**
     * Accessor, som henter kursets navn.
     * @return Returnerer kursets navn.
     */
    public String getName()
    {
        return kursusnavn;
    }

    /**
     * Accessor, som henter et array af kursusnumre indeholdende de kurser, som dette kursus forudsætter.
     * @return Returnerer et array af kursusnumre indeholdende de kurser, som dette kursus forudsætter.
     */
    public ArrayList<Integer> getForudsaetninger()
    {
        return forudsaetninger;
    }

    /**
     * Accessor, som henter et array af skemaplaceringer.
     * @return Returnerer et array med kursets skemaplaceringer.
     */
    public ArrayList<String> getSkemaPlaceringer()
    {
        return skemaPlaceringer;
    }

    /**
     * Accessor, som henter kursusnummeret.
```

```
     * @return Returnerer kursusnummeret.
     */
    public int getCourseNo()
    {
     return kursusnummer;
    }

    /**
     * Accessor, som henter en streng med alt information om det aktuelle kursus.
     * @return Returnerer en steng med kursets nummer, navn, skemaplacering(er) samt eventuelle forudsætninger.
     */
    public String toString()
    {
        String CourseNo ="";
        CourseNo = CourseNo.format("%5d", kursusnummer); // ensure to always have 5 chars
        String returnStr = "Nr: " + CourseNo + ": Navn          : "+kursusnavn;
        //Goes through all skemaPlacering elements:
        returnStr += "\n          Skemaplacering:";
        for(String skemaplacering : skemaPlaceringer){
          returnStr += " " + skemaplacering;
        }
        //Goes through all forudsaetninger elements:
        returnStr += "\n          Forudsætninger:";
        boolean forudsaetningerFound = false;
        for(Integer forudsaetningsKursus : forudsaetninger){
            returnStr += " " +forudsaetningsKursus;
            forudsaetningerFound = true;
        }
        if (!forudsaetningerFound) returnStr +=" Ingen";
        return returnStr;
    }

    /**
     * Denne metode er nødvendig for at klassen kan virke med "implements Comparable".
     * @param otherCourse Et andet kursus, som skal sammenlignes med.
     * @return Returnerer et negativt heltal, nul, eller et positivt heltal hvis dette objekt er mindre end, ens med, eller større end det
specificerede objekt.
     */
    public int compareTo(Kursus otherCourse)
    {
        return ((Integer)kursusnummer).compareTo((Integer)otherCourse.getCourseNo());
    }
}
```

A.12.Kursusbase.class

```
import java.util.TreeMap;
import java.util.ArrayList;
import java.util.TreeSet;
import java.util.SortedMap;
import java.util.Set;

/**
 * Kursusbase holder styr på alle kurserne i en TreeMap, hvor
 * kurserne er gemt som et objekt på hver plads, om med
 * kursusnummer som key (Integer).
 * @author Michael Holm & Mikael Andersen
 * @version 2007-11-22
 */
public class Kursusbase
{
    private TreeMap<Integer,Kursus> kurser;  //Kursusnr som key.
    private static final boolean debugging = false;
    private Kursus kursus;
    private Kursusnavne kursusnavne;
    private SkemaGrupper skemaGrupper;
    private Forudsaetningskurser forudsaetningskurser;
    private ArrayList<String> legalLocations;

    /**
     * Konstruktør for objekter af klassen Kursusbase.
     * Her bliver følgende ting initialiseret:
     * 1) TreeMap - kurser der kan indeholde alle kursusobjekterne.
     * 2) kursusnavne - læser kursusnavnefilen og gemmer alle kurserne i kurser.
     * 3) skemagruppe - læser skemagruppefilen og gemmer alle lokationer i kurser.
     * 4) forudsaetningskurser - læser forudsætningsfilen og gemmer alle forudsætningerne i kurser.
     * @param legalLoc En ArrayList med alle lovlige skemalokationer.
     */
    public Kursusbase(ArrayList<String> legalLoc)
    {
        legalLocations = legalLoc;
        kurser = new TreeMap<Integer,Kursus>();
        //læser kursusnavne og gemmer dem i en TreeMap
        kursusnavne = new Kursusnavne(this, "navne.txt");
        //læser alle skemalokationer og gemmer det i det aktuelle kursus elementet
        skemaGrupper = new SkemaGrupper(this, "skemagrp.txt");
        //læser alle forudsætningerne og gemmer det i det aktuelle kursus elementet
        forudsaetningskurser = new Forudsaetningskurser(this, "forud.txt");
    }

    /**
     * Denne metode returnerer et kursusobjekt, for det specifikke kursusnr.
     * @return Returnerer kursusobjektet for kursusnr.
     */
    public Kursus getCourse(int kursusnr)
    {
        if (debugging) System.out.println("Debug Kursusbase: Har ønsket at modtage kursusobjekt med kursusnr "+kursusnr);
        return kurser.get(kursusnr);
    }

    /**
     * Tilføjer et kursusobjekt til TreeMap - kurser.
     * Kursusnummer bliver brugt som key (Integer) i TreeMap.
     * @param CourseName Kursusnavn for det pågældende kursus.
     * @param CourseNumber Kursusnummer for det pågældende kursus.
     */
    public void addCourseNumberAndName(int CourseNumber,String CourseName) {
        Kursus kursus = new Kursus(CourseNumber, CourseName);
        if (debugging) System.out.println("Debug Kursusbase: addCourseNumberAndName: Coursenumber="+CourseNumber+" and
CourseName="+CourseName);
        kurser.put(CourseNumber,kursus);
    }

    /**
     * Tilføjer skemalokation i et kursusobjekt i TreeMap kurser.
     * Undersøger først om det pågældende kursus findes i kurser,
     * dernæst undersøges om det er en lovlig lokation. Hvis begge
     * ting er opfyldt bliver skamelokationen tilføjet det
     * pågældende kursus. Men hvis ikke begge ting er opfyldt
     * bliver der skrevet en fejl besked ud.
     * @param location Kursuslokation for det pågældende kursus.
```

```
     * @param CourseNumber Kursusnummer for det pågældende kursus.
     */
    public void addScheduleLocation(int CourseNumber, String location){
        if (kurser.containsKey(CourseNumber)) { //Verifcer at kursus nr. er oprettet
            boolean locationFound=false;
            for(String legalLoc : legalLocations) { //Verificer at lokation er lovlig
                if (legalLoc.equals(location)) locationFound = true;
            }
            if (!locationFound) {
                System.out.println("Lokation:" + location +" for kursus:"+CourseNumber+" er ikke en lovlig lokation og derfor ikke tilføjet");
            } else {
                kurser.get(CourseNumber).addLocation(location);
            }
        } else {
            System.out.println("Kursusnummer:"+CourseNumber+" fra skemagruppe filen findes ikke i kursusnavne filen!!!");
        }
    }

     /**
     * Tilføjer forudsætninger i et kursusobjekt i TreeMap kurser.
     * Undersøger først om det pågældende kursus findes i kurser,
     * dernæst undersøges om forudsætningskursuset er oprettet. Hvis begge
     * ting er opfyldt bliver forudsætningskursuset tilføjet det
     * pågældende kursus. Men hvis ikke begge ting er opfyldt
     * bliver der skrevet en fejl besked ud.
     * @param requirement Kursusforudsætning for det pågældende kursus.
     * @param CourseNumber Kursusnummer for det pågældende kursus.
     */
    public void addCourseRequirement(int CourseNumber, int requirement){
        //verificer at kursusnummer er oprettet
        if (kurser.containsKey(CourseNumber)) {
            //verificer at forudsætningskursusnummer er oprettet
            if (kurser.containsKey(requirement)) {
                kurser.get(CourseNumber).addRequirement(requirement);
            }
            else {
                System.out.println("Forudsætningskursus:"+requirement+" er ikke med i kursusnavne filen! og derfor ikke tilføjet");
            }
        }
        else {
            System.out.println("Kursusnummer:"+CourseNumber+" fra forudsætningskursus filen findes ikke i kursusnavne filen!!!");
        }
    }

     /**
     * Genererer en streng indeholdende hele kursusbasen.
     * Metoden values() returnere hele kurser Collection i en
     * sorteret Collection i en rækkefølge sorteret efter key (kursusnummer).
     * Herved kan kursus detaljer findes i en sorteret rækkefølge.
     * @return Returnerer hele kursusbasen i en streng, sorteret efter kursusnummer.
     */
    public String toString()
    {
        Set<Kursus> sortedDetails = new TreeSet<Kursus>(kurser.values());
        String database="";
        for(Kursus course : sortedDetails) {
            database += course + "\n";  //Bruger toString()-metoden i course.
        }
        return database;
    }

    /**
     * Genererer en streng der indeholder kursus information for et kursus.
     * Der kontrolleres om kursusnummer eksisterer og hvis ikke skrives
     * en fejl besked
     * @param courseNo Det aktuelle kursusnummer der skal returneres
     * @return Returnere en streng med kursus detaljer.
     */
    public String viskursus(int courseNo)
    {
        if (kurser.get(courseNo)!=null){
            return kurser.get(courseNo).toString();
        } else {
            return "Kurset "+courseNo+ " eksisterer ikke.";
        }
    }
}
```

A.13.KursusbaseTest.class

```
import java.util.ArrayList;

/**
 * The test class KursusbaseTest.
 *
 * @Michael Holm & Mikael Andersen
 * @2007-12-03
 */
public class KursusbaseTest extends junit.framework.TestCase
{
   private final String[] legalLocations = {"E1A","E3A","E5A","E2B","E4B","E2A","E4A","E5B","E1B","E3B",
                                            "F1A","F3A","F5A","F2B","F4B","F2A","F4A","F5B","F1B","F3B"};
   private ArrayList<String> legalLoc;

   /**
    * Default constructor for test class KursusbaseTest
    */
   public KursusbaseTest()
   {
   }

   /**
    * Sets up the test fixture.
    *
    * Called before every test case method.
    */
   protected void setUp()
   {
      legalLoc = new ArrayList<String>();
      int index=0;
      while(index<20) {
         legalLoc.add(legalLocations[index]);
         index++;
      }
   }


   /**
    * Tears down the test fixture.
    *
    * Called after every test case method.
    */
   protected void tearDown()
   {
   }

   public void testAll()
   {
      //Tester konstruktøren og indirekte også klasserne DataFiles,
      //Kursusnavne, SkemaGrupper samt Forudsaetningskurser.
      Kursusbase kursusba1 = new Kursusbase(legalLoc);
      //Tester at kursusnummer 2105 er læst korrekt ind.
      assertEquals(2105, kursusba1.getCourse(2105).getCourseNo());
      assertEquals("Algoritmer og Datastrukturer I", kursusba1.getCourse(2105).getName());
      ArrayList<String> tempLocations = kursusba1.getCourse(2105).getSkemaPlaceringer();
      assertEquals(true, tempLocations.contains("F2B"));
      assertEquals(1, tempLocations.size());
      ArrayList<Integer> tempForuds = kursusba1.getCourse(2105).getForudsaetninger();
      assertEquals(true, tempForuds.contains(2101));
      assertEquals(1, tempForuds.size());

      //Tester at man kan tilføje et kursus, og at det er lagt korrekt ind.
      kursusba1.addCourseNumberAndName(99999, "Testkursus");
      kursusba1.addScheduleLocation(99999, "E3A");
      kursusba1.addCourseRequirement(99999,2101);
      assertEquals("Testkursus", kursusba1.getCourse(99999).getName());
      assertEquals(99999, kursusba1.getCourse(99999).getCourseNo());
      tempLocations = kursusba1.getCourse(99999).getSkemaPlaceringer();
      assertEquals(true, tempLocations.contains("E3A"));
      assertEquals(1, tempLocations.size());
      tempForuds = kursusba1.getCourse(99999).getForudsaetninger();
      assertEquals(true, tempForuds.contains(2101));
      assertEquals(1, tempForuds.size());
   }
}
```

A.14.Kursusnavne.class

```
/**
 * Kursusnavne er en en sub klasse der nedarver fra dataFiles.
 * Den behandler en indlæst linie en af gangen og tilføjer
 * kursusnummer og kursus navn den til Kursusbase
 *
 * @author Michael Holm & Mikael Andersen
 * @version 2007-11-22
 */
public class Kursusnavne extends DataFiles
{
    /**
     * Constructor for objekter af klassen Kursusnavne.
     * @param kursusbase En reference til objektet kursusbase.
     * @param filename Filnavnet for kursusnavne filen.
     */
    public Kursusnavne(Kursusbase kursusbase, String filename)
    {
        this.kursusbase = kursusbase;
        readFile(filename);
    }


    /**
     * Mutator, som desikerer en linie og hiver kursusnummer ud
     * som integer og navn ud som en streng. Disse to parametre bliver
     * tilføjet kursusbasen med addCourseNumberAndName - metoden fra
     * kursusbase.
     * Hvis der ikke kan laves et interger kursusnummer fra linien
     * bliver linien ignoreret og en fejlbesked bliver udskrevet.
     * @param line En enkelt linie fra kursusnavnefilen
     */
    protected void addLineToMap(String line)
    {
        try {
            String tmpStr = line.substring(0,5);   // Crop kursusnummer
            int CourseNo=Integer.parseInt(tmpStr); // Konverter nummer til en integer
            tmpStr = line.substring(6);            // Fjerner de første 6 char i strengen
            kursusbase.addCourseNumberAndName(CourseNo, tmpStr);
        }
        catch (Exception e){
            System.out.println("Der har været en fejl i kursusnavne filen"+e);
        }
    }
}
```

A.15.KursusTest.class

```
import java.util.ArrayList;

/**
 * The test class KursusbaseTest.
 *
 * @Michael Holm & Mikael Andersen
 * @2007-12-03
 */
public class KursusbaseTest extends junit.framework.TestCase
{
    private final String[] legalLocations = {"E1A","E3A","E5A","E2B","E4B","E2A","E4A","E5B","E1B","E3B",
                                             "F1A","F3A","F5A","F2B","F4B","F2A","F4A","F5B","F1B","F3B"};
    private ArrayList<String> legalLoc;

    /**
     * Default constructor for test class KursusbaseTest
     */
    public KursusbaseTest()
    {
    }

    /**
     * Sets up the test fixture.
     *
     * Called before every test case method.
     */
    protected void setUp()
    {
        legalLoc = new ArrayList<String>();
        int index=0;
        while(index<20) {
            legalLoc.add(legalLocations[index]);
            index++;
        }
    }

    /**
     * Tears down the test fixture.
     *
     * Called after every test case method.
     */
    protected void tearDown()
    {
    }

    public void testAll()
    {
        //Tester konstruktøren og indirekte også klasserne DataFiles,
        //Kursusnavne, SkemaGrupper samt Forudsaetningskurser.
        Kursusbase kursusba1 = new Kursusbase(legalLoc);
        //Tester at kursusnummer 2105 er læst korrekt ind.
        assertEquals(2105, kursusba1.getCourse(2105).getCourseNo());
        assertEquals("Algoritmer og Datastrukturer I", kursusba1.getCourse(2105).getName());
        ArrayList<String> tempLocations = kursusba1.getCourse(2105).getSkemaPlaceringer();
        assertEquals(true, tempLocations.contains("F2B"));
        assertEquals(1, tempLocations.size());
        ArrayList<Integer> tempForuds = kursusba1.getCourse(2105).getForudsaetninger();
        assertEquals(true, tempForuds.contains(2101));
        assertEquals(1, tempForuds.size());

        //Tester at man kan tilføje et kursus, og at det er lagt korrekt ind.
        kursusba1.addCourseNumberAndName(99999, "Testkursus");
        kursusba1.addScheduleLocation(99999, "E3A");
        kursusba1.addCourseRequirement(99999,2101);
        assertEquals("Testkursus", kursusba1.getCourse(99999).getName());
        assertEquals(99999, kursusba1.getCourse(99999).getCourseNo());
        tempLocations = kursusba1.getCourse(99999).getSkemaPlaceringer();
        assertEquals(true, tempLocations.contains("E3A"));
        assertEquals(1, tempLocations.size());
        tempForuds = kursusba1.getCourse(99999).getForudsaetninger();
        assertEquals(true, tempForuds.contains(2101));
        assertEquals(1, tempForuds.size());
    }
}
```

A.16.Parser.class

```
import java.util.Scanner;
import java.util.StringTokenizer;
import java.util.ArrayList;

/**
 * Denne klasse læser input fra brugeren og fortolker det til kommandoer.
 * Der kan være op til 2 kommando-ord pr. input.
 * Klassen returnerer resultatet som et command-objekt.
 *
 * Parseren kender til et sæt af lovlige kommandoer.
 * Hvis den indtastede tekst matcher en kendt kommando i første ord, så
 * returneres et command objekt med første ord samt resten af linien.
 * Hvis den indtastede tekst ikke matcher en kendt kommando i første ord, så
 * returneres et command objekt med første kommando sat til 'null'.
 *
 * Koden er modificeret ud fra Michael Kolling and David J. Barnes's World-of-zuul eksempel.
 * @author Michael Holm & Mikael Andersen, 2007-12-01 (orig. Michael Kolling and David J. Barnes)
 * @version 2007-12-02
 */
public class Parser
{
    private CommandWords commands;  // Indeholder alle kendte kommandoer
    private Scanner reader;         // Kilden til input fra brugeren.
    private boolean debugging = false;

    /**
     * Kontruktør til at læse fra terminal-vindue.
     */
    public Parser()
    {
        commands = new CommandWords();
        reader = new Scanner(System.in);
    }

    /**
     * Metode, som læser en streng fra brugeren og fortolker den til en kommando.
     * @return Returnerer en kommando (består af 2 commando-ord, hvoraf ord nummer 2 kan være en hel streng).
     */
    public Command getCommand()
    {
        String inputLine;   // Indeholder hele den indtastede linie.
        String word1 = null;
        String word2 = null;
        inputLine = reader.nextLine();

        // Deler input op i det første ord henholdsvis resten af linien.
        Scanner tokenizer = new Scanner(inputLine);
        if(tokenizer.hasNext()) {
            word1 = tokenizer.next();      // henter første ord.
            if (debugging) System.out.println("Debug parser: word1='"+word1+"'");
            if(tokenizer.hasNext()) {
                word2 = tokenizer.nextLine(); //Henter resten af linien.
                word2 = word2.trim();  //Fjerner white space i starten af linien.
                if (debugging) System.out.println("Debug parser: word1='"+word1+"' and word2='"+word2+"'");
            }
        }

        // Tjekker nu om det er en kendt kommando. Hvis det er det, så laves et Command-objekt med det. Ellers så lav et
        // Command-objekt med 'null' sat på pladsen for 1. kommando-ord (for ukendt kommando).
        if(commands.isCommand(word1)) {
            if (debugging) System.out.println("Debug parser: returnerer nu et nyt command-objekt");
            return new Command(word1, word2);
        }
        else {
            return new Command(null, word2);
        }
    }

    /**
     * Accessor, som genererer en streng med samtlige kendte kommandoer.
     * @return Returnerer en streng med samtlige kendte kommandoer.
     */
    public String getCommandList()
    {
        return commands.showAll();
    }
```

```
    /**
     * Accessor, som returnerer en ArrayListe med samtlige kendte kommandoer.
     * @return Returnerer en ArrayListe med samtlige kendte kommandoer.
     */
    public ArrayList<String> getAllCommandsAsArray()
    {
        return commands.getAllCommandsAsArray();
    }
}
```

A.17.ParserTest.class

```
/**
 * JUnit test af klassen Parser.
 * Metoden getCommand er det ikke lykkedes at lave test for, da BlueJ ikke giver et inputvidue under generering af testmetoder.
 *
 * @Michael Holm & Mikael Andersen
 * @2007-12-02
 */
public class ParserTest extends junit.framework.TestCase
{
    /**
     * Default konstruktør for testklassen ParserTest.
     */
    public ParserTest()
    {
    }

    /**
     * Sætter testen op.
     * Metoden køres automatisk før de andre test-metoder i denne klasse.
     */
    protected void setUp()
    {
    }

    /**
     * Afslutter tests.
     * Køres efter hver test-metode.
     */
    protected void tearDown()
    {
    }

    /**
     * JUnit test af ParserTest, som tester kontruktøren af Parseren, at accessor getCommandList virker,
     * at accessor getAllCommandsAsArray indeholder en kendt kommando og ikke indeholder en (specifik)
     * ukendt kommando, samt at antallet af kendte kommandoer er som forventet.
     */
    public void testAll()
    {
        Parser p1 = new Parser();
        assertEquals("hjælp tilføj fjern udskrivbase visplan viskursus opretbruger gem hent afslut ", p1.getCommandList());
        java.util.ArrayList arrayLis1 = p1.getAllCommandsAsArray();
        assertEquals(true,arrayLis1.contains("tilføj"));
        assertEquals(false,arrayLis1.contains("ukendt_kommando"));
        assertEquals(10,arrayLis1.size());
    }
}ScheduleOccupiedException.class
/**
 *
 * @Michael Holm & Mikael Andersen
 * @2007-12-03
 */
public class ScheduleOccupiedException extends Exception
{
    private String key;


    public ScheduleOccupiedException(String key)
    {
     this.key=key;
    }

    public String getKey()
    {
     return key;
    }

    public String toString()
    {
     return key;
    }
}
```

A.18.ScheduleOccupiedExceptionTest.class

```
/**
 * JUnit test af klassen ScheduleOccupiedExceptionTest.
 *
 * @Michael Holm & Mikael Andersen
 * @2007-12-03
 */
public class ScheduleOccupiedExceptionTest extends junit.framework.TestCase
{
    /**
     * Default konstruktør for klassen ScheduleOccupiedExceptionTest
     */
    public ScheduleOccupiedExceptionTest()
    {
    }

    /**
     * Metoden køres automatisk før de andre test-metoder i denne klasse.
     */
    protected void setUp()
    {
    }

    /**
     * Afslutter tests.
     * Køres efter hver test-metode.
     */
    protected void tearDown()
    {
    }

    /**
     * Tester:
     * Konstruktøren til klassen med tilhørende streng "Teststreng".
     * Metoden 'getKey' med positiv test (matcher "Teststreng").
     * Metoden 'toString' med positiv test (matcher "Teststreng").
     */
            public void testAll()
            {
                    ScheduleOccupiedException schedule1 = new ScheduleOccupiedException("Teststreng");
                    assertEquals("Teststreng", schedule1.getKey());
                    assertEquals("Teststreng", schedule1.toString());
            }
}
```

A.19.Semester.class

```
import java.util.ArrayList;
import java.lang.Comparable;
import java.io.Serializable;

/**
 * Et objekt af klassen Semester indeholder et givent semester for en studerende.
 * Det indeholder valgte kurser for semestret samt semesternummeret.
 *
 * @Michael Holm & Mikael Andersen
 * @2007-11-30
 */
public class Semester implements Serializable
{
    private ArrayList<Kursus> kurser;   //Valgte kurser.
    private int semesterNo;             //Semestrets nummer.

    /**
     * Konstruktør for objekter af klassen Semester
     * @param semesterNo Semesternummeret angivet som heltal.
     */
    public Semester(int semesterNo)
    {
        kurser = new ArrayList<Kursus>();
        this.semesterNo=semesterNo;
    }

    /**
     * Accessor, som henter listen af kurser på det givne semester.
     * @return Returnerer en ArrayListe med alle kurser på semestret.
     */
    public ArrayList<Kursus> getCourses()
    {
     return kurser;
    }

    /**
     * Mutator, som tilføjer et kursus til semestret.
     * @param course Kursus, som ønskes tilføjet.
     */
    public void addCourse(Kursus course)
    {
        if (course!=null) {
            kurser.add(course);
        } else {
            System.out.println("Programfejl: En intern fejl er opstået i programmet.\nMetoden 'addCourse' i klassen 'Semester' har fået
nullpointer\n som reference til et kursus!");
        }
    }

    /**
     * Mutator, som fjerner et kursus fra semestret.
     * @param courseNo Kursusnummeret for det kursus, som ønskes fjernet.
     */
    public void removeCourse(int courseNo)
    {
        Kursus courseToRemove = null;
        for (Kursus course : kurser){
            if (course.getCourseNo()==courseNo)
                courseToRemove = course;
        }
        kurser.remove(courseToRemove); //Slettes udenfor foreach-løkken, da man ikke
        //må slette elementer fra en ArrayListe, som man er ved at gennemløbe.
    }

    /**
     * Accessor, som undersøger om semestret indeholder et givent kursus.
     * @param courseNo Kursusnummeret for det kursus, som der undersøges for.
     * @return Returnerer 'true', hvis kurset eksisterer på semestret.
     */
    public boolean containsCourse(int courseNo)
    {
        for (Kursus course : kurser){
            if (course.getCourseNo()==courseNo)
                return true;
```

```
        }
        return false;
    }

    /**
     * Accessor, som undersøger om semestret er tomt, dvs ikke indeholder nogle kurser.
     * @return Returnerer 'true' hvis semestret ikke indeholder nogle kurser.
     */
    public boolean isEmpty()
    {
        if (kurser.size()==0){
            return true;
        }
        return false;
    }

    /**
     * Accessor, som henter semesternummeret.
     * @return Returnerer semesternummeret (heltal).
     */
    public int getSemesterNo()
    {
     return semesterNo;
    }
}
```

A.20.SemesterException.class

```
/**
 *
 * @Michael Holm & Mikael Andersen
 * @2007-12-03
 */
public class SemesterException extends Exception
{
    private String key;


    public SemesterException(String key)
    {
     this.key=key;
    }

    public String getKey()
    {
     return key;
    }

    public String toString()
    {
     return key;
    }
}
```

A.21.SemesterExceptionTest.class

```
/**
 * JUnit test af klassen SemesterExceptionTest.
 *
 * @Michael Holm & Mikael Andersen
 * @2007-12-03
 */
public class SemesterExceptionTest extends junit.framework.TestCase
{
    /**
     * Default konstruktør for klassen SemesterExceptionTest
     */
    public SemesterExceptionTest()
    {
    }

    /**
     * Metoden køres automatisk før de andre test-metoder i denne klasse.
     */
    protected void setUp()
    {
    }

    /**
     * Afslutter tests.
     * Køres efter hver test-metode.
     */
    protected void tearDown()
    {
    }

    /**
     * Tester:
     * Konstruktøren til klassen med tilhørende streng "Teststreng".
     * Metoden 'getKey' med positiv test (matcher "Teststreng").
     * Metoden 'toString' med positiv test (matcher "Teststreng").
     */
            public void testAll()
            {
                    SemesterException semester1 = new SemesterException("Teststreng");
                    assertEquals("Teststreng", semester1.getKey());
                    assertEquals("Teststreng", semester1.toString());
            }
}
```

A.22.SemesterTest.class

```
/**
 * Testklassen SemesterTest.
 *
 * @Michael Holm & Mikael Andersen
 * @2007-12-03
 */
public class SemesterTest extends junit.framework.TestCase
{
    Semester semester1;
    Kursus kursus1;

    /**
     * Default konstruktør
     */
    public SemesterTest()
    {
    }

    /**
     * Opretter et tomt semester-objekt med semesternummer 1. Metoden opretter også et kursus-objekt med
     * kursusnummer 1017 samt titel "Diskret matematik og databaser".
     *
     * Metoden køres automatisk før de andre test-metoder i denne klasse.
     */
    protected void setUp()
    {
        semester1 = new Semester(1);
        kursus1 = new Kursus(1017, "Diskret matematik og databaser");
    }

    /**
     * Afslutter tests.
     * Køres efter hver test-metode.
     */
    protected void tearDown()
    {
    }

    /**
     * Tester at konstruktøren af semestret (fra SetUp) har virket samt at metoderne 'getSemesterNo',
     * 'isEmpty', 'addCourse' samt 'containsCourse' alle virker (positiv test).
     */
    public void test_semesterNo_isEmpty_containsCourse()
    {
        assertEquals(1, semester1.getSemesterNo());
        assertEquals(true, semester1.isEmpty());
        semester1.addCourse(kursus1);
        assertEquals(true, semester1.containsCourse(1017));
    }

    /**
     * Tester at metoden 'containsCourse' returnerer false for et kursus, som ikke er på semestret.
     * Tester også, at metoden 'isEmpty' returnerer false når et kursus er adderet til semestret.
     */
    public void testNegativeTesting()
    {
        assertEquals(false, semester1.containsCourse(2101));
        semester1.addCourse(kursus1);
        assertEquals(false, semester1.isEmpty());
    }
}
```

A.23.Serialisering.class

```
import java.io.IOException;
import java.io.FileOutputStream;
import java.io.FileInputStream;
import java.io.ObjectOutputStream;
import java.io.ObjectInputStream;
import java.lang.Comparable;
import java.io.Serializable;

/**
 * Serialisering kan gemme eller hente objecter i en fil.
 * Ved at gemme activeUser (Objektet - Studerende) bliver
 * alt information for den activeUser gemt på en gang.
 * Ligeledes kan alt information (et objekt) hentes ind igen.
 *
 * @author Michael Holm & Mikael Andersen
 * @version 2007-11-22
 */
public class Serialisering
{
    /**
     * Gemmer et object i en fil med navnet: filnavn.
     * @param filename filnavn for den pågældende fil.
     * @param obj Det aktuelle objekt der skal gemmes.
     * @throws IOException ved fejl under oprettelse af fil, skrivning eller lukning af fil.
     */
    public static void gem(Object obj, String filename) throws IOException
    {
        FileOutputStream data = new FileOutputStream(filename);
        ObjectOutputStream object = new ObjectOutputStream(data);
        object.writeObject(obj);
        object.close();
    }

    /**
     * Henter et object fra en bestemt fil.
     * @param filename filnavn for den pågældende fil.
     * @param obj Det aktuelle objekt der skal hentes.
     * @return Returnerer det indlæste objekt.
     * @throws Exception ved fejl under oprettelse af fil, hentning eller lukning af fil.
     */
    public static Object hent(String filename) throws Exception
    {
        FileInputStream data = new FileInputStream(filename);
        ObjectInputStream object = new ObjectInputStream(data);
        Object obj = object.readObject();
        object.close();
        return obj;
    }
}
```

A.24.Skemagrupper.class

```
import java.util.Scanner;

/**
 * Skemagrupper er en en sub klasse der nedarver fra dataFiles.
 * Den behandler en indlæst linie en af gangen og tilføjer
 * skemaplacering for det aktuelle kursusnummer til Kursusbase
 *
 * @author Michael Holm & Mikael Andersen
 * @version 2007-11-22
 */
public class SkemaGrupper extends DataFiles
{
    /**
     * Constructor for objekter af klassen SkemaGrupper.
     * @param kursusbase En reference til objektet kursusbase.
     * @param filename Filnavnet for skemagruppe filen.
     */
    public SkemaGrupper(Kursusbase kursusbase, String filename)
    {
        this.kursusbase = kursusbase;
        readFile(filename);
    }

    /**
     * Mutator, som desikerer en linie og hiver kursusnummer ud
     * som integer og skemagruppe ud som en streng. Disse to parametre bliver
     * tilføjet kursusbasen én for én med addScheduleLocation - metoden fra
     * kursusbase.
     * Hvis der ikke kan laves et interger kursusnummer fra linien
     * bliver linien ignoreret og en fejlbesked bliver udskrevet.
     * @param line En enkelt linie fra kursusnavnefilen
     */
    protected void addLineToMap(String line)
    {
        try {
            Scanner scanner = new Scanner(line);
            String tmpStr = scanner.next().substring(0,5); // Crop kursusnummer (fjerner -e / -f)
            int CourseNo=Integer.parseInt(tmpStr);         // Konverter nummer til en integer
            String location;
            while (scanner.hasNext()) {
                location = scanner.next();                 // Henter skemalokation/er
                kursusbase.addScheduleLocation(CourseNo, location);
            }
        }
        catch (Exception e){
            System.out.println("Der har været en fejl i skemagruppe filen"+e);
        }
    }
}
```

A.25.Studerende.class

```
import java.util.HashMap;
import java.util.ArrayList;
import java.lang.Comparable;
import java.io.Serializable;

/**
 * Klassen Studerende indeholder alt information om en given studerende. Den indeholder den studerendes navn, studienummer samt
alle
 * semestre hvorpå den studrende har valgt kurser. Hvert af disse semestre indeholder de kurser, som den studerende har valgt for det
 * givne semester. Kort sagt, den indeholder den studerendes studieplan.
 *
 * @author Michael Holm & Mikael Andersen
 * @version 2007-11-30
 */
public class Studerende implements Serializable     //Serializable er valgt for at kunne gemme objektet til en fil.
{
   private String navn;  //Den studerendes navn.
   private String studieNr;  //Den studerendes studienummer.
   private HashMap<Integer,Semester> studieplan;  //Indeholder studieplanen for den studerende.
                                       //Key er semesternummer, dvs '1' = første semester.
                                       //Mellemliggende semestre, som er tomme, genereres også. Men tomme semestre
                                       //i slutningen af studieplanen fjernes.
   private boolean debugging = false;

   /**
    * Konstruktør for klassen Studerende.
    * Opretter en studerende med navn og studienummer samt en tom studieplan.
    * @param navn Den studerendes navn, eks 'Peter Pedal'.
    * @param studieNr Den studerendes studienummer, eks 's012345'.
    */
   public Studerende(String navn, String studieNr)
   {
      studieplan = new HashMap<Integer, Semester>();
      this.navn = navn;
      this.studieNr=studieNr;
   }

   /**
    * Accessor, som returnerer den studerendes navn.
    * @return Returnerer den studerendes navn, eks 'Peter Pedal'.
    */
   public String getName()
   {
      return navn;
   }

   /**
    * Accessor, som returnerer den studerendes studienummer.
    * @return Returnerer den studerendes studienummer, eks 's012345'.
    */
   public String getStudieNr()
   {
      return studieNr;
   }

  /**
   * Mutator, som tilføjer et kursus til studieplanen.
   * Denne metode bør måske gøres mindre, så del-opgaver lægges ud i hjælpe-metoder (så den bliver lettere at overskue).
   * @param course Kurset, som ønskes tilføjet.
   * @param semesterNo Semesternummeret, hvorpå kurset ønskes.
   * @return Returnerer en streng med information til brugeren.
   * @throws Kan kaste exceptions: ScheduleOccupiedException, CourseRequirementException, SemesterException.
   */
   public String addCourse(Kursus course, int semesterNo) throws ScheduleOccupiedException, CourseRequirementException,
SemesterException
   {
      String returnStr = "";
      //Her skal tjekkes om det ønskede semester er oprettet for den studerende.
      //Hvis ikke, så skal semestret (samt eventuel mellemliggende, som også mangler) oprettes.
      if (debugging) System.out.println("Debug Studerende: AddCourse method: studieplan.size="+studieplan.size()+" og
semesterNo="+semesterNo);
      for (int i = 1; i<=semesterNo;i++){
         if (studieplan.get(i)==null){
            studieplan.put(i, new Semester(i));
```

```
            if (debugging) System.out.println("Debug Studerende: AddCourse: adderer nu nyt semester nr "+i+" til studieplanen:");
        }
    }
    if (debugging) System.out.println("Debug Studerende: AddCourse: Har nu oprettet alle nødvendige semestre.");
    //Henter nu de nødvendige skemaplaceringer (som skal være frie) ud fra det ønskede kursus:
    ArrayList<String> wantedScheduleLocations = course.getSkemaPlaceringer();
    //Skal nu tjekke om skemaet er ledigt for det givne semester. Henter de allerede valgte kurser for det ønskede semester.
    ArrayList<Kursus> previousSelectedCourses = studieplan.get(semesterNo).getCourses();
    if (debugging) System.out.println("Debug Studerende: AddCourse: Har nu hentet skemaplaceringer samt semestrets øvrige valgte
kurser.");
    // Går nu gennem alle de tidligere valgte kurser for at se om skemaet er frit på pladser hvor det nye kursus skal placeres.
    if (previousSelectedCourses != null){
        if (debugging) System.out.println("Debug Studerende: Der eksisterer previousSelectedCourses i samme semester.");
        for(Kursus prevSelCourse : previousSelectedCourses) {
            ArrayList<String> occupiedScheduleLocations = prevSelCourse.getSkemaPlaceringer();
            //Ser nu om den ønskede placering i dette tidligere valgte kursus har skemasammenfald med det ønskede kursus.
            for(String occupiedScheduleLocation : occupiedScheduleLocations) {
                if (wantedScheduleLocations.contains(occupiedScheduleLocation)){
                    if (debugging) System.out.println("Debug Studerende: addCourse: Kurset "+prevSelCourse.getCourseNo()+" optager
skemagruppen "+occupiedScheduleLocation+". Derfor kan kurset ikke oprettes!");
                    throw new ScheduleOccupiedException("Skemaplacering "+occupiedScheduleLocation+" er optaget af kurset
"+prevSelCourse.getCourseNo()+".");
                }
            }
        }
    }
    //Hvis rutinen når hertil, så betyder det at der ikke var konflikt med skemaplaceringen.
    //Programmet fortsætter nu med at tjekke om forudsætningerne er opfyldt.
    if (debugging) System.out.println("Debug Studerende: addCourse: Tjekker nu forudsætningskrav.");
    if(course.getForudsaetninger()!=null){
        if (debugging) System.out.println("Debug Studerende: addCourse: Der er forudsætningskrav, som nu undersøges.");
        ArrayList<Integer> forudsaetninger = course.getForudsaetninger();
        //Går nu gennem listen af forudsætninger
        for(int neededCourse : forudsaetninger){
            if (debugging) System.out.println("Debug Studerende: addCourse: Ser nu på forudsætningskursusnr: "+neededCourse);
            if (!courseIsPriorInPlan(neededCourse, semesterNo)){
                cleanUpEmptySemestersAtEnd();  //Needs this, as the method might have created a semester that is not used anyway.
                throw new CourseRequirementException("Kurset mangler forudsaetningskursus: "+neededCourse+".");
            }
        }
    }
    if (debugging) System.out.println("Debug Studerende: addCourse: Forudsætningskrav er opfyldt (eller ikke-eksisterende).");
    //Hvis programmet er nået hertil, så betyder det at de nødvendige forudsætninger er tilstede.
    //Tjekker nu at kurset ligger på det korrekte semester. (Første semester er defineret til at være efterår).
    boolean foundSemesterLocation=false; //er falsk indtil at kurset er set liggende i et skemafelt for det korrekte semester
                                         //(F eller E afhængig af semesternummer).
    if(semesterNo%2==1){ //Det er et efterårskursus.
        if (debugging) System.out.println("Debug Studerende: addCourse: Du har valgt et semester, som ligger om efteråret");
        for (String location : wantedScheduleLocations){
            if (location.substring(0, 1).equals("E")){
                if (debugging) System.out.println("Debug Studerende: addCourse: Dit ønskede kursus ligger om efteråret.");
                foundSemesterLocation=true;
            }
        }
    } else { //Det er et forårskursus
        if (debugging) System.out.println("Debug Studerende: addCourse: Du har valgt et semester, som ligger om foråret");
        for (String location : wantedScheduleLocations){
            if (location.substring(0, 1).equals("F")){
                if (debugging) System.out.println("Debug Studerende: addCourse: Dit ønskede kursus ligger om foråret.");
                foundSemesterLocation=true;
            }
        }
    }
    if (!foundSemesterLocation){  //Kurset udbydes ikke på det pågældende semesterhalvår.
        if(semesterNo%2==1){
            cleanUpEmptySemestersAtEnd();  //Fjerner nu eventuelle nye semestre, som blev genereret i starten af denne metode, men
som
                                           //ikke skal bruges alligevel, da kurset ikke skal placeres.
            throw new SemesterException("Kurset udbydes ikke om efteråret.");
        } else {
            cleanUpEmptySemestersAtEnd();  //Fjerner nu eventuelle nye semestre, som blev genereret i starten af denne metode, men
som
                                           //ikke skal bruges alligevel, da kurset ikke skal placeres.
            throw new SemesterException("Kurset udbydes ikke om foråret.");
        }
    }
    //Hvis programmet når hertil, så ligger kurset på det korrekte semester, der er ledigt i skemaet, og forudsætningerne er opfyldt.
    //Tjekker nu om kurset er et enkeltsemester-kursus. Hvis det er det, så må det ikke optræde andre steder i kursusplanen.
    if (!courseIsDualSemester(course) && courseIsPriorInPlan(course.getCourseNo(), semesterNo)){
```

```
            cleanUpEmptySemestersAtEnd();  //Fjerner nu eventuelle nye semestre, som blev genereret i starten af denne metode, men som
                                           //ikke skal bruges alligevel, da kurset ikke skal placeres.
            throw new SemesterException("Kurset er allerede valgt på et tidligere semester, og det strækker sig ikke over 2 semestre.");
        }
        //Hvis det er et flersemester-kursus, så bliver brugeren gjort opmærksom på det:
        if (courseIsDualSemester(course)) {
            returnStr += "Vær opmærksom på at kursus "+course.getCourseNo()+" strækker sig over flere semestre,\nog at du selv skal ";
            returnStr += "indsætte det de relevante steder.\nDette program tilføjer ikke automatisk i flere semestre.";
        }
        //Kurset adderes nu til semestret.
        if (debugging) System.out.println("Debug Studerende: addCourse: Skal nu til at tilføje kurset: "+course.getCourseNo()+" med kursusnavn: "+course.getName()+" på semesterNo: "+semesterNo);
        studieplan.get(semesterNo).addCourse(course);
        return returnStr;
    }

    /** Mutator, som fjerner et specifikt kursus fra kursusplanen (på alle semestre, hvor det optræder).
     * Fjerner også ubrugte semestre i slutningen (i enden) af studieplanen.
     * @param requestedCourseNo Kursusnummeret på det kursus, som ønskes fjernet.
     * @return Returnerer en streng med information til brugeren.
     */
    public String removeCourse(int requestedCourseNo)
    {
        String returnStr = "";
        if (debugging) System.out.println("Debug: Studerende: removeCourse method entered.");
        ArrayList<Integer> isOnSemesters = new ArrayList<Integer>();
        for (int i = 1; i<=studieplan.size();i++){        //Løber gennem alle semstret for at se om kurset optræder på hvert af dem.
            if (debugging) System.out.println("Debug Studerende: removeCourse: int="+i+", studieplan.size()="+studieplan.size());
            if (studieplan.get(i).containsCourse(requestedCourseNo)){
                if (debugging) System.out.println("Kurset er fundet på semesternummer: "+i+".");
                isOnSemesters.add(i);  //genererer en separat liste med semestre, hvorpå kurset skal fjernes.
            }                          //Årsagen er, at man ikke bør ændre elementer i en liste, som man er ved at gennemløbe.
        }
        //Array'et isOnSemsters indeholder nu de semesternumre, hvorpå kurset forefindes.
        //Går nu igennem de fundne semestre hver især og undersøger om andre senere valgte kurser afhænger af det kursus som ønskes slettet.
        for (int semesterNo : isOnSemesters)
        {
            int dependentCourse = 0;
            dependentCourse = findDependsOn(studieplan.get(semesterNo), requestedCourseNo);  //returnerer første kursus, som afh. af det som ønskes fjernet.
            if (debugging) System.out.println("Debug: Studerende: Har nu undersøgt om senere kurser afhænger af dette. dependentCourse="+dependentCourse);
            if (dependentCourse == 0) {  //Der er ingen kurser på senere semestre, som afhænger af det kursus som ønskes slettet.
                if (debugging) System.out.println("Debug: Studerende: removeCourse: Skal nu fjerne kursusnummer "+requestedCourseNo+" med studieplan.get(semesterNo).removeCourse(requestedCourseNo).");
                studieplan.get(semesterNo).removeCourse(requestedCourseNo); //Fjerner nu kurset fra det fundne semester.
                returnStr += "Kurset "+requestedCourseNo+" er fjernet fra semester "+semesterNo+".\n";
            } else {
                returnStr += "Kurset "+requestedCourseNo+" kan ikke fjernes, da "+dependentCourse+" er i planen og har "+requestedCourseNo+" som forudsætning.\nFjern først "+dependentCourse+".\n";
            }
        }
        cleanUpEmptySemestersAtEnd(); //Fjerner nu eventuelle nye semestre, som blev genereret i starten af denne metode,
                                      //men som ikke skal bruges alligevel, da kurset ikke skal placeres.
        if (isOnSemesters.size()==0) {
            returnStr += "Kurset "+requestedCourseNo+" kan ikke fjernes, da det ikke eksisterer i studieplanen.";
        }
        return returnStr;
    }

    /**
     * Mutator, som fjerner tomme semestre i enden af studieplanen.
     */
    private void cleanUpEmptySemestersAtEnd()
    {
        if (debugging) System.out.println("Debug Studerende: cleanUpEmptySemestersAtEnd: starting the method.");
        for (int semesterNo=studieplan.size(); semesterNo>=1;semesterNo--){  //Går gennem alle semestre (sidste semester først).
            if (studieplan.get(semesterNo).isEmpty()){
                if (debugging) System.out.println("Debug: Studerende: cleanUpEmptySemestersAtEnd: Semesternummer "+semesterNo+" er tomt. Det fjernes nu fra studieplanen.");
                studieplan.remove(semesterNo);
            } else {
                return;  //Der er fundet et semester med kurser. Der eksisterer således ikke flere tomme semestre i enden af studieplanen.
            }
        }
    }
```

```
/**
 * Accessor, som undersøger om et givent kursus eksisterer på et tidligere semester end det angivne semester.
 * @param neededCourse Kurset, som ønskes undersøgt om eksisterer.
 * @param currentSemesterNo Alle semestre frem til (eksklusiv) semesternummeret 'currentSemesterNo', undersøges.
 * @return Returnerer 'true' hvis kurset er fundet på et tidligere semester i studieplanen.
 */
private boolean courseIsPriorInPlan(int neededCourse, int currentSemesterNo)
{
   //Går gennem alle tidligere semestre og verificerer at det nødvendige kursus findes..
   if (currentSemesterNo < 2)
      return false;
   for (int semesterNo=1; semesterNo < currentSemesterNo; semesterNo++){  //Går gennem alle semestre
      //Går gennem alle kurser på det givne semester.
      ArrayList<Kursus> prevCoursesOnSemester = studieplan.get(semesterNo).getCourses();
      for (Kursus kursus : prevCoursesOnSemester) {
         if (kursus.getCourseNo() ==  neededCourse)
            return true;
      }
   }
   return false;
}

/**
 * Accessor, som undersøger om et kursus er et 'dobbeltkursus', dvs om det udbydes både forår og efterår.
 * @param course Kurset, som ønskes undersøgt.
 * @return Returnerer 'true' hvis kurset har skemalokationer både forår og efterår.
 */
private boolean courseIsDualSemester(Kursus course)
{
   boolean spring = false;
   boolean fall = false;
   ArrayList<String> ScheduleLocations = course.getSkemaPlaceringer();
   if (ScheduleLocations != null){
      for(String ScheduleLocation : ScheduleLocations) {
         if (ScheduleLocation.substring(0,1).equals("F")){
            spring = true;
         } else {
            fall = true;
         }
      }
   }
   if (spring && fall)
      return true;
   return false;
}

/**
 * Accessor, som undersøger om andre senere valgte kurser afhænger af dette kursus.
 * @param semester Semestret, hvorpå det pågældende kursus er placeret.
 * @param requestedCourseNo Kursusnummeret, som det undersøges om andre valgte kurser afhænger af.
 * @return Returnerer det først fundne kursusnummer, som afhænger af det undersøgte kursus.
 */
private int findDependsOn(Semester semester, int requestedCourseNo)
{
   //Går gemmen alle semestre, som kommer efter det specificerede semester, da disse er de eneste
   //semestre, som kan indeholde kurser, som afhænger af det forespurgte kursusnummer (requestedCourseNo).
   int semesterNo=semester.getSemesterNo()+1;
   while (semesterNo <= studieplan.size()){  //Løber gennem semestre
      ArrayList<Kursus> prevCoursesOnSemester = studieplan.get(semesterNo).getCourses();
      for (Kursus kursus : prevCoursesOnSemester) {  //Løber gennem alle kurserne på det givne semester.
         //Går gennem afhængighederne for hver enkelt kursus.
         ArrayList<Integer> forudsaetninger = kursus.getForudsaetninger();
         for (int courseNoDependent : forudsaetninger){
            if (courseNoDependent ==  requestedCourseNo) {
               return kursus.getCourseNo();
            }
         }
      }
      semesterNo++;
   }
   return 0; //der blev ikke fundet kurser i studieplanen, som afhænger af det angivne kursus.
}

/**
 * Hjælpemetode, som genererer en streng som header til visplan-udskriften.
 * @return Returnerer en header-streng.
 */
private String visplanHeader()
{
```

```
        String returnStr = "";
        returnStr += "-------------------------------------------------------------------\n";
        returnStr += "|  Semester |  Tid  |  Mandag | Tirsdag |  Onsdag | Torsdag |  Fredag |\n";
        returnStr += "-------------------------------------------------------------------\n";
        return returnStr;
    }

    /**
     * Accessor, som genererer en række til at indsætte i skemaet for visplan-metoden.
     * @param semester Det aktuelle semester.
     * @param isMorningClass 'True' hvis tidsrummet er fra 8-12, og 'false' hvis det er fra 13-17.
     * @param legalLoc Listen over mulige skemaplaceringer.
     * @Return Returnerer en streng i formatet "| SemesterNo/SemesterTekst | tidsrum | kursusnummer | kursusnummer | kursusnummer
| kursusnummer | kursusnummer |".
     */
    private String visplanRow(Semester semester, boolean isMorningClass, ArrayList<String> legalLoc)
    {
        String returnStr = "";
        int semesterNo = semester.getSemesterNo();
        int locationShift = 0;  //Bruges til at forskyde skemaet (locationLoc) 10 pladser hvis det er et forårs-semester.
        if (semesterNo%2==0){   //'true' hvis det er et forårs-semester.
            locationShift = 10;
            if (debugging) System.out.println("Studerende visplan: Det er et forårs-semester.");
        }
        //Genererer kolonne 1. Hvis det er første række for semestret, så inkluderes semesternummeret.
        //Hvis det er anden række for semestret, så inkluderes teksten "(Forår(" eller "(Efterår)".
        String str = "";
        if (isMorningClass) {
            returnStr += "|    "+str.format("%2d", semesterNo)+"     |";
        } else {
            if (semesterNo%2==0) {returnStr = "|  (Forår)  |";} else {returnStr= "| (Efterår) |";}
        }
        //Genererer kolonne 2:
        if (isMorningClass) {
            returnStr += "  8-12 |";
        } else {
            returnStr += " 13-17 |";
            locationShift += 5;
        }  //Genererer kolonne 3 til 7:
        for (int colonne=0; colonne<5;colonne++){
            boolean courseFound = false;
            for (Kursus course : semester.getCourses()) {
            //Går gennem alle lokationer for det givne kursus for at se om der er en match.
                for (String location : course.getSkemaPlaceringer()){
                    if (debugging) System.out.println("Debug Studerende: visplan: location="+location);
                    if (location.equals(legalLoc.get(locationShift+colonne))){
                        returnStr += "  "+str.format("%5d", course.getCourseNo())+"  |";
                        courseFound = true;
                    }
                }
            }
            if (!courseFound)
                returnStr += "  -----  |";  // udskriver '-----' hvis der ikke er noget kursus i det aktuelle felt.
        }
        returnStr += "\n";
        return returnStr;
    }

    /**
     * Accessor 'visplan' genererere en streng med studieplanen for den aktive bruger. Hvis der er tomme semestre imellem ikke-tomme
     * semestre, så vises disse også. Men tomme semestre i slutningen af planen vises ikke (de eksisterer faktisk slet ikke).
     * @param legalLoc Array med lovlige skemaplaceringer.
     * @Return Returnerere en streng med den komplette studieplan for den aktive bruger.
     */
    public String visplan(ArrayList<String> legalLoc)
    {
        if (studieplan.size()==0)
            return "Studieplanen er tom!";
        String returnStr = "Studerende: " + navn +"\n"
                    + "Studienr. : " + studieNr +"\n";
        returnStr += visplanHeader();
        for (int semesterNo=1; semesterNo<=studieplan.size();semesterNo++){  //Går gennem alle semestre.
            returnStr += visplanRow(studieplan.get(semesterNo),true, legalLoc);  //Parametre er semestret + morgenklasse (8-12) eller ej +
lovlige skemaplaceringer.
            returnStr += visplanRow(studieplan.get(semesterNo),false, legalLoc);  //Parametre er semestret + morgenklasse (8-12) eller ej +
lovlige skemaplaceringer.
            returnStr += "-------------------------------------------------------------------\n";
        }
        return returnStr;
```

```
    }
}
```

A.26.StuderendeTest.class

```
/**
 * Testklassen StuderendeTest.
 *
 * @Michael Holm & Mikael Andersen
 * @2007-12-03
 */
public class StuderendeTest extends junit.framework.TestCase
{
    Studerende studeren1;

    /**
     * Default konstruktør.
     */
    public StuderendeTest()
    {
    }

    /**
     * Metode 'SetUp', som opretter en default studerende ved navn "Peter Pedal" med studienummer "s012345".
     * Kaldes før alle test-metoder.
     */
    protected void setUp()
    {
        studeren1 = new Studerende("Peter Pedal", "s012345");
    }

    /**
     * Afslutter tests.
     * Køres efter hver test-metode.
     */
    protected void tearDown()
    {
    }

    /**
     * Tester konstruktøren ved at udlæse navnet og studienummeret, som skal stemme med det der blev genereret i 'SetUp'.
     */
    public void testConstructor()
    {
        assertEquals("Peter Pedal", studeren1.getName());
        assertEquals("s012345", studeren1.getStudieNr());
    }

    /**
     * Tester at et nyt kursus kan oprettes incl. en skemaplacering for kurset. Herefter tilføjes kurset til den studerendes
     * studieplan på semester 1. Kurset 1017 forsøges herefter fjernet, og det verificeres at der kommer en fejlbesked med at
     * kurset 1017 ikke kan fjernes, da det ikke indgår i studieplanen.
     */
    public void testNegativeTesting() throws ScheduleOccupiedException, CourseRequirementException, SemesterException
    {
        Kursus kursus1 = new Kursus(2101, "Indledende programmering");
        kursus1.addLocation("E3A");
        assertEquals("", studeren1.addCourse(kursus1, 1));
        java.lang.String string1 = studeren1.removeCourse(1017);
        assertEquals("Kurset 1017 kan ikke fjernes, da det ikke eksisterer i studieplanen.", string1);
    }

    /**
     * Tester at et nyt kursus kan oprettes incl. en skemaplacering for kurset. Herefter fjernes kurset igen, og det
     * verificeres at der kommer besked til brugeren om at kurset er fjernet.
     */
    public void testAddRemove() throws ScheduleOccupiedException, CourseRequirementException, SemesterException
    {
        Kursus kursus1 = new Kursus(2101, "Indledende programmering");
        kursus1.addLocation("E3A");
        assertEquals("", studeren1.addCourse(kursus1, 1));
        assertEquals("Kurset 2101 er fjernet fra semester 1.\n", studeren1.removeCourse(2101));
    }
}
```

B. JavaDoc output.

skip-navbar_top

Package Class Tree Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

B.1.Class Command

```
java.lang.Object
  └Command
```

```
public class Command
extends Object
```

Dette er en kopi af klassen 'Command' fra bogens eksempel "World of Zuul". Metoderne er ikke ændret. Kommentarerne er ændret til Dansk, så JavaDoc dokumentationen bliver ensartet.

Version:
2007-12-02

Author:
Michael Kolling and David J. Barnes, kommentarer oversat af MH og MAN.

Constructor Summary

Command**(String firstWord,** String secondWord) Konstruktør.	

Method Summary

String	getCommandWord**()** Accessor, som returnerer første ord i kommandoen.
String	getSecondWord**()** Accessor, som returnerer andet ord (resten af linien) i kommandoen.
`boolean`	hasSecondWord**()** Metode, som undersøger om kommandoen har mere end 1 ord.
`boolean`	isUnknown**()** Metode, som undersøger om første ord er en ukendt kommando.

Methods inherited from class java.lang. Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Command

```
public Command(String firstWord,
               String secondWord)
```

Konstruktør. Første og andet ord skal bruges, men de kan godt være sat til 'null'.
Parameters:
`firstWord` - Det første ord i kommandoen. 'null' hvis kommandoen ikke kunne genkendes.
`secondWord` - Resten af linien i kommandoen.

Method Detail

getCommandWord

`public String` **`getCommandWord()`**

Accessor, som returnerer første ord i kommandoen. Hvis kommandoen ikke er en gyldig kommando, så vil denne være sat til 'null'.
Returns:
Returnerer det første ord i kommandoen.

---

getSecondWord

`public String` **`getSecondWord()`**

Accessor, som returnerer andet ord (resten af linien) i kommandoen. Hvis kommandoen ikke har 2 ord, så vil andet ord være sat til 'null', og metoden returnerer i så fald 'null'.
Returns:
Returnerer det andet ord (incl. resten af linien) i kommandoen.

---

hasSecondWord

`public boolean` **`hasSecondWord()`**

Metode, som undersøger om kommandoen har mere end 1 ord.
Returns:
Returnerer 'true' hvis kommandoen har mere end 1 ord.

---

isUnknown

`public boolean` **`isUnknown()`**

Metode, som undersøger om første ord er en ukendt kommando.
Returns:
Returnerer 'true' hvis kommandoen ikke kendes.

---

skip-navbar_top
Package Class Tree Index Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

B.2.Class CommandTest

```
java.lang.Object
  └junit.framework.Assert
      └junit.framework.TestCase
          └CommandTest
```

All Implemented Interfaces:
junit.framework.Test

```
public class CommandTest
extends junit.framework.TestCase
```

JUnit test af klassen CommandTest.

Constructor Summary

CommandTest**()** Default konstruktør for klassen CommandTest.	

Method Summary

protected void	setUp**()** Metoden køres automatisk før de andre test-metoder i denne klasse.
protected void	tearDown**()** Afslutter tests.
void	testNegativTest**()** JUnit test, som tester at et objekt af typen Command kan oprettes, og at konstruktøren kan håndtere hvis 2.
void	testPositivTest**()** JUnit test, som tester at et objekt af typen Command kan oprettes, og at parametrene bliver gemt korrekt i objektet.

Methods inherited from class junit.framework.TestCase
countTestCases, createResult, getName, run, run, runBare, runTest, setName, toString

Methods inherited from class junit.framework.Assert
assertEquals, assertFalse, assertFalse, assertNotNull, assertNotNull, assertNotSame, assertNotSame, assertNull, assertNull, assertSame, assertSame, assertTrue, assertTrue, fail, fail

Methods inherited from class java.lang. Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

CommandTest

`public` **`CommandTest()`**
Default konstruktør for klassen CommandTest.

Method Detail

setUp

`protected void` **`setUp()`**
Metoden køres automatisk før de andre test-metoder i denne klasse.
Overrides:
`setUp` in class `junit.framework.TestCase`

---

tearDown

`protected void` **`tearDown()`**
Afslutter tests. Køres efter hver test-metode.
Overrides:
`tearDown` in class `junit.framework.TestCase`

---

testNegativTest

`public void` **`testNegativTest()`**
JUnit test, som tester at et objekt af typen Command kan oprettes, og at konstruktøren kan håndtere hvis 2. ord er sat til 'null'. Opretter et kommando-objekt med ord1="Peter" og ord2='null'. Tester, at accessor 'get.CommandWord' returnerer "Peter" (positiv test). Tester, at accessor 'getSecondWord' returnerer 'null' (positiv test). Tester, at metoden 'hasSecordWord' returnerer 'false' (negativ test). Tester, at metoden 'isUnknown' returnerer 'true' (positiv test).

---

testPositivTest

`public void` **`testPositivTest()`**
JUnit test, som tester at et objekt af typen Command kan oprettes, og at parametrene bliver gemt korrekt i objektet. Tester konstruktøren ved at oprette et kommando-objekt med ord1="viskursus" og ord2="2101". Tester, at accessor 'getCommandWord' returnerer "viskursus" (positiv test). Tester, at accessor 'getSecondWord' returnerer "2101" (positiv test). Tester, at metoden 'hasSecordWord' returnerer 'true' (positiv test). Tester, at metoden 'isUnknown' returnerer 'false' (negativ test).

B.3.Class CommandWords

```
java.lang.Object
  └CommandWords
```

```
public class CommandWords
extends Object
```

Denne klasse indeholder en 'enumeration' af alle kendte kommandonavne. Den bruges til at genkende kommandoer. Kommandoer: hjælp, tilføj, fjern, udskrivbase, visplan, viskursus, opretbruger, gem, hent samt afslut. hjælp: Udskriver listen af mulige kommandoer. hjælp 'kommando' : Udskriver detaljer om hvordan kommandoen 'kommando' bruges. tilføj xxxxx : Tilføjer kurset med kursusnummer xxxxx til studieplanen. fjern xxxxx: Fjerner kursus med kursusnummer xxxxx fra studieplanen. udskrivbase : Udskriver alle kurser som programmet kender. visplan : Udskriver studieplanen. viskursus xxxxx : Viser kursusdetaljer for kurset med kursusnummer xxxxx. (Kursusnummer, kursusnavn, skemaplaceringer samt forudsætninger. opretbruger 'studienummer' : Opretter en ny bruger med studienummer 'studienummer' samt navn, som brugeren bliver bedt om at indtaste. hent 'studienummer' : Skifter til en tidligere gemt bruger med studienummeret 'studienummer' gem : Gemmer den studerendes studieplan incl. den studerendes navn og studienummer. afslut: Afslutter programmet.

Version:
2007.11.21

Author:
Michael Holm & Mikael Andersen

Constructor Summary

CommandWords**()** Konstruktør.

Method Summary

ArrayList<String>	getAllCommandsAsArray**()** Returnerer alle tilladte kommandoer i en ArrayListe af strenge.
`boolean`	isCommand**(String aString)** Undersøger hvorvidt en given streng er en korrekt kommando.
String	showAll**()** Returnerer en streng med alle tilladte kommandoer.

Methods inherited from class java.lang. Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

CommandWords

```
public CommandWords()
```

Konstruktør.

Method Detail

getAllCommandsAsArray

```
public ArrayList<String> getAllCommandsAsArray()
```

Returnerer alle tilladte kommandoer i en ArrayListe af strenge.

Returns:

Returnerer alle tilladte kommandoer i en ArrayListe af strenge.

---

isCommand

```
public boolean isCommand(String aString)
```

Undersøger hvorvidt en given streng er en korrekt kommando.

Parameters:

`aString` - Streng, som ønskes undersøgt.

Returns:

Returnerer'true' hvis strengen er en korrekt kommando.

---

showAll

```
public String showAll()
```

Returnerer en streng med alle tilladte kommandoer.

Returns:

Returnerer en streng med alle tilladte kommandoer.

---

skip-navbar_top
Package Class Tree Index Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

B.4.Class CommandWordsTest

```
java.lang.Object
  └junit.framework.Assert
      └junit.framework.TestCase
          └CommandWordsTest
```

All Implemented Interfaces:
junit.framework.Test

```
public class CommandWordsTest
extends junit.framework.TestCase
```

JUnit test af klassen CommandWordsTest.

Constructor Summary

CommandWordsTest**()** Default konstruktør for klassen CommandWordsTest

Method Summary

protected void	setUp**()** Metoden køres automatisk før de andre test-metoder i denne klasse.
protected void	tearDown**()** Afslutter tests.
void	testCommandWords**()** Tester: Konstruktøren til klassen.

Methods inherited from class junit.framework.TestCase

countTestCases, createResult, getName, run, run, runBare, runTest, setName, toString

Methods inherited from class junit.framework.Assert

assertEquals, assertFalse, assertFalse, assertNotNull, assertNotNull, assertNotSame, assertNotSame, assertNull, assertNull, assertSame, assertSame, assertTrue, assertTrue, fail, fail

Methods inherited from class java.lang. Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

CommandWordsTest

`public` **`CommandWordsTest()`**

Default konstruktør for klassen CommandWordsTest

Method Detail

setUp

`protected void` **`setUp()`**

Metoden køres automatisk før de andre test-metoder i denne klasse.

Overrides:

`setUp` in class `junit.framework.TestCase`

---

tearDown

`protected void` **`tearDown()`**

Afslutter tests. Køres efter hver test-metode.

Overrides:

`tearDown` in class `junit.framework.TestCase`

---

testCommandWords

`public void` **`testCommandWords()`**

Tester: Konstruktøren til klassen. Metoden 'isCommand' med negativ test (en ukendt kommando). Metoden 'is Command' med positiv test (en kendt kommando). Metoden 'showAll', som udskriver samtlige kendte kommandoer til en streng.

B.5.Class CourseRequirementException

```
java.lang.Object
  └java.lang.Throwable
      └java.lang.Exception
          └CourseRequirementException
```

All Implemented Interfaces:
Serializable

```
public class CourseRequirementException
extends Exception
```

See Also:
Serialized Form

Constructor Summary

CourseRequirementException**(String key)**	

Method Summary

String	getKey**()**
String	toString**()**

Methods inherited from class java.lang. Throwable
fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace

Methods inherited from class java.lang. Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

CourseRequirementException

```
public CourseRequirementException(String key)
```

Method Detail

getKey

```
public String getKey()
```

toString

```
public String toString()
```

Overrides:
toString in class Throwable

skip-navbar_bottom
Package Class Tree Index Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

skip-navbar_top
Package Class Tree Index Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

B.6.Class CourseRequirementExceptionTest

```
java.lang.Object
  └junit.framework.Assert
      └junit.framework.TestCase
          └CourseRequirementExceptionTest
```

All Implemented Interfaces:
junit.framework.Test

```
public class CourseRequirementExceptionTest
extends junit.framework.TestCase
```

JUnit test af klassen CourseRequirementExceptionTest.

Constructor Summary

CourseRequirementExceptionTest**()** Default konstruktør for klassen CourseRequirementExceptionTest

Method Summary

protected void	setUp**()** Metoden køres automatisk før de andre test-metoder i denne klasse.
protected void	tearDown**()** Afslutter tests.
void	testAll**()** Tester: Konstruktøren til klassen med tilhørende streng "Teststreng".

Methods inherited from class junit.framework.TestCase
countTestCases, createResult, getName, run, run, runBare, runTest, setName, toString

Methods inherited from class junit.framework.Assert
assertEquals, assertFalse, assertFalse, assertNotNull, assertNotNull, assertNotSame, assertNotSame, assertNull, assertNull, assertSame, assertSame, assertTrue, assertTrue, fail, fail

Methods inherited from class java.lang. Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

CourseRequirementExceptionTest

`public` **`CourseRequirementExceptionTest()`**

Default konstruktør for klassen CourseRequirementExceptionTest

Method Detail

setUp

`protected void` **`setUp()`**

Metoden køres automatisk før de andre test-metoder i denne klasse.

Overrides:

`setUp` in class `junit.framework.TestCase`

---

tearDown

`protected void` **`tearDown()`**

Afslutter tests. Køres efter hver test-metode.

Overrides:

`tearDown` in class `junit.framework.TestCase`

---

testAll

`public void` **`testAll()`**

Tester: Konstruktøren til klassen med tilhørende streng "Teststreng". Metoden 'getKey' med positiv test (matcher "Teststreng"). Metoden 'toString' med positiv test (matcher "Teststreng").

---

skip-navbar_top
Package Class Tree Index Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

B.7.Class DataFiles

```
java.lang.Object
  └DataFiles
```

Direct Known Subclasses:
Forudsaetningskurser, Kursusnavne, SkemaGrupper

```
public abstract class DataFiles
extends Object
```

dataFiles er en super klasse der læser en datafil fra harddisk. Det er en Abstract klasse da den kun bliver brugt som super klasse for Kursusnavne, SkemaGrupper og Forudsaetningskurser klasserne.

Version:
2007-11-22

Author:
Michael Holm & Mikael Andersen

Field Summary

Kursusbase	kursusbase

Constructor Summary

DataFiles**()** Constructor for objekter af klassen dataFiles Der er ingen initialisering for denne klasse.

Method Summary

`protected abstract void`	addLineToMap**(String line)** Abstract metode: Mutator, som desikrere en linie og hiver kursus nummer ud som integer og navn ud som en streng.
`protected void`	readFile**(String fileName)** Mutator, som læser en hel fil, linie for linie.

Methods inherited from class java.lang. Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

kursusbase

```
public Kursusbase kursusbase
```

Constructor Detail

DataFiles

`public` **`DataFiles()`**

Constructor for objekter af klassen dataFiles Der er ingen initialisering for denne klasse.

Method Detail

addLineToMap

`protected abstract void` **`addLineToMap(String line)`**

Abstract metode: Mutator, som desikrere en linie og hiver kursus nummer ud som integer og navn ud som en streng. Disse to parametre bliver tilføjet kursusbasen med addCourseNumberAndName - metoden fra kursusbase.

Parameters:

`line` - En enkelt linie fra en datafil

---

readFile

`protected void` **`readFile(String fileName)`**

Mutator, som læser en hel fil, linie for linie. Hvis filen ikke kan åbnes, bliver der sendt en fejlmeddelelse. Ligeledes sker der fejl under læsning af filen bliver der sendt en fejlmeddelelse. Endelig bliver der sendt en meddelelse hvis filen ikke kan lukkes korrekt.

Parameters:

`filename` - Filnavn for den pågældende fil

---

skip-navbar_top
Package Class Tree Index Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

B.8.Class Forudsaetningskurser

```
java.lang.Object
  └DataFiles
      └Forudsaetningskurser
```

```
public class Forudsaetningskurser
extends DataFiles
```

Forudsaetningskurser er en en sub klasse der nedarver fra dataFiles. Den behandler en indlæst linie en af gangen og tilføjer forudsætningerne for det aktuelle kursus nummer til Kursusbase

Version:
2007-11-22
Author:
Michael Holm & Mikael Andersen

Field Summary

Fields inherited from class DataFiles
kursusbase

Constructor Summary

Forudsaetningskurser**(Kursusbase kursusbase,** String filename) Constructor for objekter af klassen Forudsaetningskurser.

Method Summary

`protected void`	addLineToMap**(String line)** Mutator, som desikrerer en linie og hiver kursus nummer ud som integer og forudsætningerne ud som en streng.

Methods inherited from class DataFiles
readFile

Methods inherited from class java.lang. Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Forudsaetningskurser

```
public Forudsaetningskurser(Kursusbase kursusbase,
                            String filename)
```

Constructor for objekter af klassen Forudsaetningskurser.

Parameters:

`kursusbase` - Et link til Objektet kursusbase. (MAN en det et Link ?)

`filename` - Filnavnet for forudsætnings filen.

Method Detail

addLineToMap

```
protected void addLineToMap(String line)
```

Mutator, som desikrerer en linie og hiver kursus nummer ud som integer og forudsætningerne ud som en streng. Disse to parametre bliver tilføjet kursusbasen en for en med addCourseRequirement - metoden fra kursusbase. Hvis der ikke kan laves et interger kursus nummer fra linien bliver linien ignoreret og en fejlbesked bliver udskrevet.

Specified by:

addLineToMap in class DataFiles

Parameters:

`line` - En enkelt linie fra kursusnavnefilen

skip-navbar_top
Package Class Tree Index Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

B.9.Class Kontrol

```
java.lang.Object
  └Kontrol
```

```
public class Kontrol
extends Object
```

'Læg en studieplan' er et program, hvor studerende kan opbygge deres individuelle studieplan. Programmet læser fra 3 filer, som danner grunlag for en kursusbase med tilhørende skemaplacering og kursus- forudsætninger. Den studerende kan tilføje og fjerne kurser, få vist samtlige kurser, få vist sin opbyggede studieplan, skifte til en anden bruger (andet navn og studienummer), hente en tidligere gemt studieplan samt gemme sin nuværende studieplan. Hver studerende kan kun have 1 studieplan gemt ad gangen (det gemmes på studienummeret).

Version:
2007-12-02

Author:
Michael Holm & Mikael Andersen

Method Summary

`static void`	main**(String args[])** Starter programmet.

Methods inherited from class java.lang. Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Method Detail

main

```
public static void main(String args[])
```

Starter programmet.

skip-navbar_bottom
Package Class Tree Index Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

skip-navbar_top
Package Class Tree Index Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

B.10.Class Kursus

```
java.lang.Object
  └Kursus
```

All Implemented Interfaces:
Serializable, Comparable<Kursus>

```
public class Kursus
extends Object
implements Comparable<Kursus>, Serializable
```

Klassen Kursus indeholder kursusnavn, kursusnummer, en liste over hvilke kurser dette kursus forudsætter, samt en liste over hvor i skemaet dette kursus ligger. Skemaplaceringen indeholder også information om hvorvidt det er et efterårskursus eller et forårskursus.

See Also:
Serialized Form

Constructor Summary

Kursus**(int kursusnummer,** String kursusnavn) Konstruktør for objekter af klassen Kursus	

Method Summary

`void`	addLocation**(String location)** Mutator, som tilføjer en skemaplacering for kurset.
`void`	addRequirement**(int requirement)** Mutator, som tilføjer en forudsætning til kurset.
`int`	compareTo**(Kursus otherCourse)** Denne metode er nødvendig for at klassen kan virke med "implements Comparable".
`int`	getCourseNo**()** Accessor, som henter kursusnummeret.
ArrayList<Integer>	getForudsaetninger**()** Accessor, som henter et array af kursusnumre indeholdende de kurser, som dette kursus forudsætter.
String	getName**()** Accessor, som henter kursets navn.
ArrayList<String>	getSkemaPlaceringer**()** Accessor, som henter et array af skemaplaceringer.
String	toString**()** Accessor, som henter en streng med alt information om det aktuelle kursus.

Methods inherited from class java.lang. Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

Kursus

```
public Kursus(int kursusnummer,
              String kursusnavn)
```

Konstruktør for objekter af klassen Kursus

Parameters:

`kursusnummer` - Kursets nummer.

`kursusnavn` - Kursets navn.

Method Detail

addLocation

```
public void addLocation(String location)
```

Mutator, som tilføjer en skemaplacering for kurset.

Parameters:

`location` - Lokationen, som ønskes tilføjet til skemaplaceringen for kurset.

---

addRequirement

```
public void addRequirement(int requirement)
```

Mutator, som tilføjer en forudsætning til kurset.

Parameters:

`requirement` - Forudsætningskursets kursusnummer.

---

compareTo

```
public int compareTo(Kursus otherCourse)
```

Denne metode er nødvendig for at klassen kan virke med "implements Comparable".

Specified by:

compareTo in interface Comparable<Kursus>

Parameters:

`otherCourse` - Et andet kursus, som skal sammenlignes med.

Returns:

Returnerer et negativt heltal, nul, eller et positivt heltal hvis dette objekt er mindre end, ens med, eller større end det specificerede objekt.

---

getCourseNo

```
public int getCourseNo()
```

Accessor, som henter kursusnummeret.

Returns:

Returnerer kursusnummeret.

---

getForudsaetninger

```
public ArrayList<Integer> getForudsaetninger()
```

Accessor, som henter et array af kursusnumre indeholdende de kurser, som dette kursus forudsætter.

Returns:

Returnerer et array af kursusnumre indeholdende de kurser, som dette kursus forudsætter.

---

getName

```
public String getName()
```

Accessor, som henter kursets navn.
Returns:
Returnerer kursets navn.

---

getSkemaPlaceringer

```
public ArrayList<String> getSkemaPlaceringer()
```

Accessor, som henter et array af skemaplaceringer.
Returns:
Returnerer et array med kursets skemaplaceringer.

---

toString

```
public String toString()
```

Accessor, som henter en streng med alt information om det aktuelle kursus.
Overrides:
toString in class Object
Returns:
Returnerer en steng med kursets nummer, navn, skemaplacering(er) samt eventuelle forudsætninger.

B.11.Class Kursusbase

```
java.lang.Object
  └Kursusbase
```

```
public class Kursusbase
extends Object
```

Kursusbase holder styr på alle kurserne i en TreeMap, hvor kurserne er gemt som et objekt på hver plads, om med kursusnummer som key (Integer).

Version:
2007-11-22

Author:
Michael Holm & Mikael Andersen

Constructor Summary

Kursusbase**(ArrayList<String> legalLoc)** Konstruktor for objeker af klassen Kursusbase.

Method Summary

`void`	addCourseNumberAndName**(int CourseNumber,** String CourseName) Tilføjer et kursus objekt til TreeMap - kurser.
`void`	addCourseRequirement**(int CourseNumber,** `int requirement)` Tilføjer forudsætninger i et kursus objekt i TreeMap kurser.
`void`	addScheduleLocation**(int CourseNumber,** String location) Tilføjer skemalokation i et kursus objekt i TreeMap kurser.
Kursus	getCourse**(int kursusnr)** Denne metode returnerer et kursus objekt, for det specifikke kursusnr.
String	toString**()** Genererer en streng indeholdende hele kursusbasen.
String	viskursus**(int courseNo)** Genererer en streng der indeholder kursus information for et kursus.

Methods inherited from class java.lang. Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

Kursusbase

```
public Kursusbase(ArrayList<String> legalLoc)
```

Konstruktor for objeker af klassen Kursusbase. Her bliver følgende ting initialiseret: 1) TreeMap - kurser der kan indeholde alle kursus objekterne. 2) kursusnavne - læser kursusnavnefilen og gemmer alle kurserne i kurser. 3) skemagruppe - læser skemagruppefilen og gemmer alle lokationer i kurser. 4) forudsaetningskurser - læser forudsætningsfilen og gemmer alle forudsætningerne i kurser.

Parameters:

`legalLoc` - En ArrayList med alle lovlige skemalokationer.

Method Detail

addCourseNumberAndName

```
public void addCourseNumberAndName(int CourseNumber,
                                   String CourseName)
```

Tilføjer et kursus objekt til TreeMap - kurser. Kursusnummer bliver brugt som key (Integer) i TreeMap.

Parameters:

`CourseName` - Kursusnavn for det pågældende kursus.

`CourseNumber` - Kursusnummer for det pågældende kursus.

---

addCourseRequirement

```
public void addCourseRequirement(int CourseNumber,
                                 int requirement)
```

Tilføjer forudsætninger i et kursus objekt i TreeMap kurser. Undersøger først om det pågældende kursus findes i kurser, dernæst undersøges om forudsætningskursuset er oprettet. Hvis begge ting er opfyldt bliver forudsætningskursuset tilføjet det pågældende kursus. Men hvis ikke begge ting er opfyldt bliver der skrevet en fejl besked ud.

Parameters:

`requirement` - Kursusforudsætning for det pågældende kursus.

`CourseNumber` - Kursusnummer for det pågældende kursus.

---

addScheduleLocation

```
public void addScheduleLocation(int CourseNumber,
                                String location)
```

Tilføjer skemalokation i et kursus objekt i TreeMap kurser. Undersøger først om det pågældende kursus findes i kurser, dernæst undersøges om det er en lovlig lokation. Hvis begge ting er opfyldt bliver skamelokationen tilføjet det pågældende kursus. Men hvis ikke begge ting er opfyldt bliver der skrevet en fejl besked ud.

Parameters:

`location` - Kursuslokation for det pågældende kursus.

`CourseNumber` - Kursusnummer for det pågældende kursus.

---

getCourse

```
public Kursus getCourse(int kursusnr)
```

Denne metode returnerer et kursus objekt, for det specifikke kursusnr.

Returns:

Returnerer kursus objekted for kursusnr.

---

toString

```
public String toString()
```

Genererer en streng indeholdende hele kursusbasen. Metoden values() returnere hele kurser Collection i en sorteret Collection i en rækkefølge sorteret efter key (kursus nummer). Herved kan kursus detaljer findes i en sorteret rækkefølge.

Overrides:

toString in class Object

Returns:

Returnerer hele kursusbasen i en streng, sorteret efter kursus nummer.

---

viskursus

`public String` **`viskursus(int courseNo)`**

Genererer en streng der indeholder kursus information for et kursus. Der kontrolleres om kursus nummer eksisterer og hvis ikke skrives en fejl besked

Parameters:

`courseNo` - Det aktuelle kursus nummer der skal returneres

Returns:

Returnere en streng med kursus detaljer.

---

skip-navbar bottom

Package Class Tree Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

skip-navbar_top
Package Class Tree Index Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

B.12.Class KursusbaseTest

```
java.lang.Object
  └junit.framework.Assert
      └junit.framework.TestCase
          └KursusbaseTest
```

All Implemented Interfaces:
junit.framework.Test

```
public class KursusbaseTest
extends junit.framework.TestCase
```

The test class KursusbaseTest.

Constructor Summary

KursusbaseTest**()** Default constructor for test class KursusbaseTest

Method Summary

protected void	setUp**()** Sets up the test fixture.
protected void	tearDown**()** Tears down the test fixture.
void	testAll**()**

Methods inherited from class junit.framework.TestCase
countTestCases, createResult, getName, run, run, runBare, runTest, setName, toString

Methods inherited from class junit.framework.Assert
assertEquals, assertFalse, assertFalse, assertNotNull, assertNotNull, assertNotSame, assertNotSame, assertNull, assertNull, assertSame, assertSame, assertTrue, assertTrue, fail, fail

Methods inherited from class java.lang. Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

KursusbaseTest

`public KursusbaseTest()`

Default constructor for test class KursusbaseTest

Method Detail

setUp

`protected void setUp()`

Sets up the test fixture. Called before every test case method.

Overrides:

`setUp` in class `junit.framework.TestCase`

---

tearDown

`protected void tearDown()`

Tears down the test fixture. Called after every test case method.

Overrides:

`tearDown` in class `junit.framework.TestCase`

---

testAll

`public void testAll()`

B.13.Class Kursusnavne

```
java.lang.Object
  └DataFiles
      └Kursusnavne
```

```
public class Kursusnavne
extends DataFiles
```

Kursusnavne er en en sub klasse der nedarver fra dataFiles. Den behandler en indlæst linie en af gangen og tilføjer kursus nummer og kursus navn den til Kursusbase

Version:
2007-11-22

Author:
Michael Holm & Mikael Andersen

Field Summary

Fields inherited from class DataFiles
kursusbase

Constructor Summary

Kursusnavne**(Kursusbase kursusbase,** String filename) Constructor for objekter af klassen Kursusnavne.

Method Summary

`protected void`	addLineToMap**(String line)** Mutator, som desikrere en linie og hiver kursus nummer ud som integer og navn ud som en streng.

Methods inherited from class DataFiles
readFile

Methods inherited from class java.lang. Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Kursusnavne

```
public Kursusnavne(Kursusbase kursusbase,
                   String filename)
```

Constructor for objekter af klassen Kursusnavne.

Parameters:

`kursusbase` - Et link til Objektet kursusbase. (MAN en det et Link ?)

`filename` - Filnavnet for kursusnavne filen.

Method Detail

addLineToMap

```
protected void addLineToMap(String line)
```

Mutator, som desikrere en linie og hiver kursus nummer ud som integer og navn ud som en streng. Disse to parametre bliver tilføjet kursusbasen med addCourseNumberAndName - metoden fra kursusbase. Hvis der ikke kan laves et interger kursus nummer fra linien bliver linien ignoreret og en fejlbesked bliver udskrevet.

Specified by:

addLineToMap in class DataFiles

Parameters:

`line` - En enkelt linie fra kursusnavnefilen

B.14.Class KursusTest

```
java.lang.Object
  └junit.framework.Assert
      └junit.framework.TestCase
          └KursusTest
```

All Implemented Interfaces:
junit.framework.Test

```
public class KursusTest
extends junit.framework.TestCase
```

Testklassen KursusTest tester klassen Kursus. Alle public metoder i klassen Kursus bliver testet med både positiv test og negativ test.

Field Summary	
(package private) Kursus	kursus1
(package private) Kursus	kursus2

Constructor Summary
KursusTest() Default konstruktør for klassen KursusTest

Method Summary	
protected void	setUp() Opretter et kursus-objekt med kursusnummer 2105 samt titel "Algoritmer og Datastrukturer I".
protected void	tearDown() Afslutter tests.
void	test_Requirements_Location_Size() Tester at forudsætningerne er lagt korrekt ind, dvs tester metoderne 'addRequirement' samt 'getForudsaetninger' (positiv test).
void	testConstructor() Verificerer at konstruktøren samt metoderne 'getCourseNo' og 'getName' virker (positiv test).
void	testNegativeTesting() Tester negativ test på metoden 'getCourseNo' ved at kurset oprettet under 'SetUp' ikke har kursusnummer 1234.

Methods inherited from class junit.framework.TestCase
`countTestCases, createResult, getName, run, run, runBare, runTest, setName, toString`

Methods inherited from class junit.framework.Assert
`assertEquals, assertFalse, assertFalse, assertNotNull, assertNotNull, assertNotSame, assertNotSame, assertNull, assertNull, assertSame, assertSame, assertTrue, assertTrue, fail, fail`

Methods inherited from class java.lang. Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

kursus1

`Kursus` **`kursus1`**

---

kursus2

`Kursus` **`kursus2`**

Constructor Detail

KursusTest

`public` **`KursusTest()`**

Default konstruktør for klassen KursusTest

Method Detail

setUp

`protected void` **`setUp()`**

Opretter et kursus-objekt med kursusnummer 2105 samt titel "Algoritmer og Datastrukturer I". Herefter tillægges kurset forudsætningskurserne 1017 samt 2101. Herefter sættes kursets skemaplacering til "F2B". Kursus 1017 oprettes også som selvstændigt objekt. Metoden køres automatisk før de andre test-metoder i denne klasse.

Overrides:

`setUp` in class `junit.framework.TestCase`

---

tearDown

`protected void` **`tearDown()`**

Afslutter tests. Køres efter hver test-metode.

Overrides:

`tearDown` in class `junit.framework.TestCase`

---

test_Requirements_Location_Size

`public void` **`test_Requirements_Location_Size()`**

Tester at forudsætningerne er lagt korrekt ind, dvs tester metoderne 'addRequirement' samt 'getForudsaetninger' (positiv test). Tester også at kurset har præcis 2 forudsætninger. Tester også at skemaplaceringerne er lagt korrekt ind, dvs tester metoderne 'addLocation' samt 'getSkemaPlaceringer'. Tester til sidst, at metoden 'compareTo' også virker (ved at teste om kurset er magen til sig selv).

testConstructor

`public void` **`testConstructor()`**

Verificerer at konstruktøren samt metoderne 'getCourseNo' og 'getName' virker (positiv test).

testNegativeTesting

`public void` **`testNegativeTesting()`**

Tester negativ test på metoden 'getCourseNo' ved at kurset oprettet under 'SetUp' ikke har kursusnummer 1234. Tester negativ test på metoden 'getName' ved at kurset oprettet under 'SetUp' ikke har navnet "Hjemkundskab II". Tester negativ test på at kurset 2405 ikke er en del af forudsætningerne samt at der ikke kun er ét forudsætningskursus (da der er indsat både 1017 samt 2101). Tester at skemaplaceringen "E1A" ikke er med i kursets skemaplaceringer. Tester at antallet af skemaplaceringer er forskelligt fra 2 (da det bør være 1). Tester at metoden 'compareTo' giver false ved sammenligning af 2 forskellige kurser

skip-navbar_top
Package Class Tree Index Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

B.15.Class Parser

```
java.lang.Object
  └Parser
```

```
public class Parser
extends Object
```

Denne klasse læser input fra brugeren og fortolker det til kommandoer. Der kan være op til 2 kommando-ord pr. input. Klassen returnerer resultatet som et command-objekt. Parseren kender til et sæt af lovlige kommandoer. Hvis den indtastede tekst matcher en kendt kommando i første ord, så returneres et command objekt med første ord samt resten af linien. Hvis den indtastede tekst ikke matcher en kendt kommando i første ord, så returneres et command objekt med første kommando sat til 'null'. Koden er modificeret ud fra Michael Kolling and David J. Barnes's World-of-zuul eksempel.

Version:
2007-12-02

Author:
Michael Holm & Mikael Andersen, 2007-12-01 (orig. Michael Kolling and David J. Barnes)

Constructor Summary

Parser**()** Kontruktør til at læse fra terminal-vindue.

Method Summary

ArrayList<String>	getAllCommandsAsArray**()** Accessor, som returnerer en ArrayListe med samtlige kendte kommandoer.
Command	getCommand**()** Metode, som læser en streng fra brugeren og fortolker den til en kommando.
String	getCommandList**()** Accessor, som genererer en streng med samtlige kendte kommandoer.

Methods inherited from class java.lang. Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Parser

```
public Parser()
```

Kontruktør til at læse fra terminal-vindue.

Method Detail

getAllCommandsAsArray

```
public ArrayList<String> getAllCommandsAsArray()
```

Accessor, som returnerer en ArrayListe med samtlige kendte kommandoer.

Returns:

Returnerer en ArrayListe med samtlige kendte kommandoer.

---

getCommand

```
public Command getCommand()
```

Metode, som læser en streng fra brugeren og fortolker den til en kommando.

Returns:

Returnerer en kommando (består af 2 commando-ord, hvoraf ord nummer 2 kan være en hel streng).

---

getCommandList

```
public String getCommandList()
```

Accessor, som genererer en streng med samtlige kendte kommandoer.

Returns:

Returnerer en streng med samtlige kendte kommandoer.

---

skip-navbar_top
Package Class Tree Index Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

B.16.Class ParserTest

```
java.lang.Object
  └junit.framework.Assert
      └junit.framework.TestCase
          └ParserTest
```

All Implemented Interfaces:
junit.framework.Test

```
public class ParserTest
extends junit.framework.TestCase
```

JUnit test af klassen Parser. Metoden getCommand er det ikke lykkedes at lave test for, da BlueJ ikke giver et inputvidue under generering af testmetoder.

Constructor Summary

ParserTest**()** Default konstruktør for testklassen ParserTest.

Method Summary

protected void	setUp**()** Sætter testen op.
protected void	tearDown**()** Afslutter tests.
void	testAll**()** JUnit test af ParserTest, som tester kontruktøren af Parseren, at accessor getCommandList virker, at accessor getAllCommandsAsArray indeholder en kendt kommando og ikke indeholder en (specifik) ukendt kommando, samt at antallet af kendte kommandoer er som forventet.

Methods inherited from class junit.framework.TestCase
countTestCases, createResult, getName, run, run, runBare, runTest, setName, toString

Methods inherited from class junit.framework.Assert
assertEquals, assertFalse, assertFalse, assertNotNull, assertNotNull, assertNotSame, assertNotSame, assertNull, assertNull, assertSame, assertSame, assertTrue, assertTrue, fail, fail

Methods inherited from class java.lang. Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

ParserTest

`public` **`ParserTest()`**

Default konstruktør for testklassen ParserTest.

Method Detail

setUp

`protected void` **`setUp()`**

Sætter testen op. Metoden køres automatisk før de andre test-metoder i denne klasse.

Overrides:

`setUp` in class `junit.framework.TestCase`

---

tearDown

`protected void` **`tearDown()`**

Afslutter tests. Køres efter hver test-metode.

Overrides:

`tearDown` in class `junit.framework.TestCase`

---

testAll

`public void` **`testAll()`**

JUnit test af ParserTest, som tester kontruktøren af Parseren, at accessor getCommandList virker, at accessor getAllCommandsAsArray indeholder en kendt kommando og ikke indeholder en (specifik) ukendt kommando, samt at antallet af kendte kommandoer er som forventet.

B.17.Class ScheduleOccupiedException

```
java.lang.Object
  └java.lang.Throwable
      └java.lang.Exception
          └ScheduleOccupiedException
```

All Implemented Interfaces:
Serializable

```
public class ScheduleOccupiedException
extends Exception
```

See Also:
Serialized Form

Constructor Summary

ScheduleOccupiedException**(String key)**

Method Summary

String	getKey**()**
String	toString**()**

Methods inherited from class java.lang. Throwable
fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace

Methods inherited from class java.lang. Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

ScheduleOccupiedException

```
public ScheduleOccupiedException(String key)
```

Method Detail

getKey

```
public String getKey()
```

---

toString

```
public String toString()
```

Overrides:
toString in class Throwable

---

skip-navbar_bottom

Package Class Tree Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

skip-navbar_top
Package Class Tree Index Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

B.18.Class ScheduleOccupiedExceptionTest

```
java.lang.Object
  └junit.framework.Assert
      └junit.framework.TestCase
          └ScheduleOccupiedExceptionTest
```

All Implemented Interfaces:
junit.framework.Test

```
public class ScheduleOccupiedExceptionTest
extends junit.framework.TestCase
```

JUnit test af klassen ScheduleOccupiedExceptionTest.

Constructor Summary

ScheduleOccupiedExceptionTest**()** Default konstruktør for klassen ScheduleOccupiedExceptionTest

Method Summary

`protected void`	setUp**()** Metoden køres automatisk før de andre test-metoder i denne klasse.
`protected void`	tearDown**()** Afslutter tests.
`void`	testAll**()** Tester: Konstruktøren til klassen med tilhørende streng "Teststreng".

Methods inherited from class junit.framework.TestCase

`countTestCases, createResult, getName, run, run, runBare, runTest, setName, toString`

Methods inherited from class junit.framework.Assert

`assertEquals, assertFalse, assertFalse, assertNotNull, assertNotNull, assertNotSame, assertNotSame, assertNull, assertNull, assertSame, assertSame, assertTrue, assertTrue, fail, fail`

Methods inherited from class java.lang. Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

ScheduleOccupiedExceptionTest

`public` **`ScheduleOccupiedExceptionTest()`**

Default konstruktør for klassen ScheduleOccupiedExceptionTest

Method Detail

setUp

`protected void` **`setUp()`**

Metoden køres automatisk før de andre test-metoder i denne klasse.

Overrides:

`setUp` in class `junit.framework.TestCase`

---

tearDown

`protected void` **`tearDown()`**

Afslutter tests. Køres efter hver test-metode.

Overrides:

`tearDown` in class `junit.framework.TestCase`

---

testAll

`public void` **`testAll()`**

Tester: Konstruktøren til klassen med tilhørende streng "Teststreng". Metoden 'getKey' med positiv test (matcher "Teststreng"). Metoden 'toString' med positiv test (matcher "Teststreng").

---

skip-navbar_top
Package Class Tree Index Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

B.19.Class Semester

```
java.lang.Object
  └Semester
```

All Implemented Interfaces:
Serializable

```
public class Semester
extends Object
implements Serializable
```

Et objekt af klassen Semester indeholder et givent semester for en studerende. Det indeholder valgte kurser for semestret samt semesternummeret.

See Also:
Serialized Form

Constructor Summary

Semester**(int semesterNo)** Konstruktør for objekter af klassen Semester	

Method Summary

`void`	addCourse**(Kursus course)** Mutator, som tilføjer et kursus til semestret.
`boolean`	containsCourse**(int courseNo)** Accessor, som undersøger om semestret indeholder et givent kursus.
ArrayList<Kursus>	getCourses**()** Accessor, som henter listen af kurser på det givne semester.
`int`	getSemesterNo**()** Accessor, som henter semesternummeret.
`boolean`	isEmpty**()** Accessor, som undersøger om semestret er tomt, dvs ikke indeholder nogle kurser.
`void`	removeCourse**(int courseNo)** Mutator, som fjerner et kursus fra semestret.

Methods inherited from class java.lang. Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Semester

`public` **`Semester(int semesterNo)`**

Konstruktør for objekter af klassen Semester

Parameters:

`semesterNo` - Semesternummeret angivet som heltal.

Method Detail

addCourse

`public void` **`addCourse(Kursus course)`**

Mutator, som tilføjer et kursus til semestret.

Parameters:

`course` - Kursus, som ønskes tilføjet.

---

containsCourse

`public boolean` **`containsCourse(int courseNo)`**

Accessor, som undersøger om semestret indeholder et givent kursus.

Parameters:

`courseNo` - Kursusnummeret for det kursus, som der undersøges for.

Returns:

Returnerer 'true', hvis kurset eksisterer på semestret.

---

getCourses

`public ArrayList<Kursus>` **`getCourses()`**

Accessor, som henter listen af kurser på det givne semester.

Returns:

Returnerer en ArrayListe med alle kurser på semestret.

---

getSemesterNo

`public int` **`getSemesterNo()`**

Accessor, som henter semesternummeret.

Returns:

Returnerer semesternummeret (heltal).

---

isEmpty

`public boolean` **`isEmpty()`**

Accessor, som undersøger om semestret er tomt, dvs ikke indeholder nogle kurser.

Returns:

Returnerer 'true' hvis semestret ikke indeholder nogle kurser.

---

removeCourse

`public void` **`removeCourse(int courseNo)`**

Mutator, som fjerner et kursus fra semestret.

Parameters:

`courseNo` - Kursusnummeret for det kursus, som ønskes fjernet.

---

skip-navbar_top
Package Class Tree Index Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

B.20.Class SemesterException

```
java.lang.Object
  └java.lang.Throwable
      └java.lang.Exception
          └SemesterException
```

All Implemented Interfaces:
Serializable

```
public class SemesterException
extends Exception
```

See Also:
Serialized Form

Constructor Summary

SemesterException**(String key)**

Method Summary

String	getKey**()**
String	toString**()**

Methods inherited from class java.lang. Throwable
fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace

Methods inherited from class java.lang. Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

SemesterException

```
public SemesterException(String key)
```

Method Detail

getKey

```
public String getKey()
```

---

toString

```
public String toString()
```

Overrides:
toString in class Throwable

---

skip-navbar_bottom

Package Class Tree Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

skip-navbar_top
Package Class Tree Index Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

B.21.Class SemesterExceptionTest

```
java.lang.Object
  └junit.framework.Assert
      └junit.framework.TestCase
          └SemesterExceptionTest
```

All Implemented Interfaces:
junit.framework.Test

```
public class SemesterExceptionTest
extends junit.framework.TestCase
```

JUnit test af klassen SemesterExceptionTest.

Constructor Summary

SemesterExceptionTest**()** Default konstruktør for klassen SemesterExceptionTest

Method Summary

protected void	setUp**()** Metoden køres automatisk før de andre test-metoder i denne klasse.
protected void	tearDown**()** Afslutter tests.
void	testAll**()** Tester: Konstruktøren til klassen med tilhørende streng "Teststreng".

Methods inherited from class junit.framework.TestCase
countTestCases, createResult, getName, run, run, runBare, runTest, setName, toString

Methods inherited from class junit.framework.Assert
assertEquals, assertFalse, assertFalse, assertNotNull, assertNotNull, assertNotSame, assertNotSame, assertNull, assertNull, assertSame, assertSame, assertTrue, assertTrue, fail, fail

Methods inherited from class java.lang. Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

SemesterExceptionTest

`public` **`SemesterExceptionTest()`**

Default konstruktør for klassen SemesterExceptionTest

Method Detail

setUp

`protected void` **`setUp()`**

Metoden køres automatisk før de andre test-metoder i denne klasse.

Overrides:

`setUp` in class `junit.framework.TestCase`

---

tearDown

`protected void` **`tearDown()`**

Afslutter tests. Køres efter hver test-metode.

Overrides:

`tearDown` in class `junit.framework.TestCase`

---

testAll

`public void` **`testAll()`**

Tester: Konstruktøren til klassen med tilhørende streng "Teststreng". Metoden 'getKey' med positiv test (matcher "Teststreng"). Metoden 'toString' med positiv test (matcher "Teststreng").

---

skip-navbar_top
Package Class Tree Index Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

B.22.Class SemesterTest

```
java.lang.Object
  └junit.framework.Assert
      └junit.framework.TestCase
          └SemesterTest
```

All Implemented Interfaces:
junit.framework.Test

```
public class SemesterTest
extends junit.framework.TestCase
```

Testklassen SemesterTest.

Field Summary	
(package private) Kursus	kursus1
(package private) Semester	semester1

Constructor Summary	
SemesterTest() Default konstruktør	

Method Summary	
protected void	setUp() Opretter et tomt semester-objekt med semesternummer 1.
protected void	tearDown() Afslutter tests.
void	test_semesterNo_isEmpty_containsCourse() Tester at konstruktøren af semestret (fra SetUp) har virket samt at metoderne 'getSemesterNo', 'isEmpty', 'addCourse' samt 'containsCourse' alle virker (positiv test).
void	testNegativeTesting() Tester at metoden 'containsCourse' returnerer false for et kursus, som ikke er på semestret.

Methods inherited from class junit.framework.TestCase
countTestCases, createResult, getName, run, run, runBare, runTest, setName, toString

Methods inherited from class junit.framework.Assert
`assertEquals, assertFalse, assertFalse, assertNotNull, assertNotNull, assertNotSame, assertNotSame, assertNull, assertNull, assertSame, assertSame, assertTrue, assertTrue, fail, fail`

Methods inherited from class java.lang. Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

kursus1

`Kursus` **`kursus1`**

---

semester1

`Semester` **`semester1`**

Constructor Detail

SemesterTest

`public` **`SemesterTest()`**

Default konstruktør

Method Detail

setUp

`protected void` **`setUp()`**

Opretter et tomt semester-objekt med semesternummer 1. Metoden opretter også et kursus-objekt med kursusnummer 1017 samt titel "Diskret matematik og databaser". Metoden køres automatisk før de andre test-metoder i denne klasse.

Overrides:

`setUp` in class `junit.framework.TestCase`

---

tearDown

`protected void` **`tearDown()`**

Afslutter tests. Køres efter hver test-metode.

Overrides:

`tearDown` in class `junit.framework.TestCase`

---

test_semesterNo_isEmpty_containsCourse

`public void` **`test_semesterNo_isEmpty_containsCourse()`**

Tester at konstruktøren af semestret (fra SetUp) har virket samt at metoderne 'getSemesterNo', 'isEmpty', 'addCourse' samt 'containsCourse' alle virker (positiv test).

---

testNegativeTesting

`public void` **`testNegativeTesting()`**

Tester at metoden 'containsCourse' returnerer false for et kursus, som ikke er på semestret. Tester også, at metoden 'isEmpty' returnerer false når et kursus er adderet til semestret.

skip-navbar_bottom

Package Class Tree Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

B.23.Class Serialisering

```
java.lang.Object
  └Serialisering
```

```
public class Serialisering
extends Object
```

Serialisering kan gemme eller hente objecter i en fil. Ved at gemme activeUser (Objektet - Studerende) bliver alt information for den activeUser gemt på en gang. Ligeledes kan alt information (et objekt) hentes ind igen.

Version:
2007-11-22

Author:
Michael Holm & Mikael Andersen

Constructor Summary

Serialisering**()**

Method Summary

`static void`	gem**(Object obj,** String filename) Gemmer et object i en fil med navnet: filnavn.
`static` Object	hent**(String filename)** Henter et object fra en bestemt fil.

Methods inherited from class java.lang. Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Serialisering

```
public Serialisering()
```

Method Detail

gem

```
public static void gem(Object obj,
                       String filename)
                throws IOException
```

Gemmer et object i en fil med navnet: filnavn.

Parameters:
`filename` - filnavn for den pågældende fil.
`obj` - Det aktuelle objekt der skal gemmes.
Throws:
IOException - ved fejl under oprettelse af fil, skrivning eller lukning af fil.

---

hent

```
public static Object hent(String filename)
                   throws Exception
```

Henter et object fra en bestemt fil.
Parameters:
`filename` - filnavn for den pågældende fil.
`obj` - Det aktuelle objekt der skal hentes.
Returns:
Returnerer det indlæste objekt.
Throws:
Exception - ved fejl under oprettelse af fil, hentning eller lukning af fil.

---

skip-navbar_bottom

Package Class Tree Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

skip-navbar_top
Package Class Tree Index Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

B.24.Class SkemaGrupper

```
java.lang.Object
  └DataFiles
      └SkemaGrupper
```

```
public class SkemaGrupper
extends DataFiles
```

Skemagrupper er en en sub klasse der nedarver fra dataFiles. Den behandler en indlæst linie en af gangen og tilføjer skemaplacering for det aktuelle kursus nummer til Kursusbase

Version:
2007-11-22
Author:
Michael Holm & Mikael Andersen

Field Summary

Fields inherited from class DataFiles
kursusbase

Constructor Summary

SkemaGrupper**(Kursusbase kursusbase,** String filename) Constructor for objekter af klassen SkemaGrupper.

Method Summary

`protected void`	addLineToMap**(String line)** Mutator, som desikrere en linie og hiver kursus nummer ud som integer og skemagruppe ud som en streng.

Methods inherited from class DataFiles
readFile

Methods inherited from class java.lang. Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

SkemaGrupper

```
public SkemaGrupper(Kursusbase kursusbase,
                    String filename)
```

Constructor for objekter af klassen SkemaGrupper.

Parameters:

`kursusbase` - Et link til Objektet kursusbase. (MAN en det et Link ?)

`filename` - Filnavnet for skemagruppe filen.

Method Detail

addLineToMap

```
protected void addLineToMap(String line)
```

Mutator, som desikrere en linie og hiver kursus nummer ud som integer og skemagruppe ud som en streng. Disse to parametre bliver tilføjet kursusbasen en for en med addScheduleLocation - metoden fra kursusbase. Hvis der ikke kan laves et interger kursus nummer fra linien bliver linien ignoreret og en fejlbesked bliver udskrevet.

Specified by:

addLineToMap in class DataFiles

Parameters:

`line` - En enkelt linie fra kursusnavnefilen

skip-navbar_top
Package Class Tree Index Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

B.25.Class Studerende

```
java.lang.Object
  └Studerende
```

All Implemented Interfaces:
Serializable

```
public class Studerende
extends Object
implements Serializable
```

Klassen Studerende indeholder alt information om en given studerende. Den indeholder den studerendes navn, studienummer samt alle semestre hvorpå den studrende har valgt kurser. Hvert af disse semestre indeholder de kurser, som den studerende har valgt for det givne semester. Kort sagt, den indeholder den studerendes studieplan.

Version:
2007-11-30
Author:
Michael Holm & Mikael Andersen
See Also:
Serialized Form

Constructor Summary

Studerende**(String navn,** String studieNr) Konstruktør for klassen Studerende.	

Method Summary

String	addCourse**(Kursus course,** `int semesterNo`) Mutator, som tilføjer et kursus til studieplanen.
String	getName**()** Accessor, som returnerer den studerendes navn.
String	getStudieNr**()** Accessor, som returnerer den studerendes studienummer.
String	removeCourse**(int requestedCourseNo)** Mutator, som fjerner et specifikt kursus fra kursusplanen (på alle semestre, hvor det optræder).
String	visplan**(ArrayList**<String> legalLoc) Accessor 'visplan' genererere en streng med studieplanen for den aktive bruger.

Methods inherited from class java.lang. Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Studerende

```
public Studerende(String navn,
                  String studieNr)
```

Konstruktør for klassen Studerende. Opretter en studerende med navn og studienummer samt en tom studieplan.

Parameters:

`navn` - Den studerendes navn, eks 'Peter Pedal'.

`studieNr` - Den studerendes studienummer, eks 's012345'.

Method Detail

addCourse

```
public String addCourse(Kursus course,
                        int semesterNo)
                 throws ScheduleOccupiedException,
                        CourseRequirementException,
                        SemesterException
```

Mutator, som tilføjer et kursus til studieplanen. Denne metode bør måske gøres mindre, så del-opgaver lægges ud i hjælpe-metoder (så den bliver lettere at overskue).

Parameters:

`course` - Kurset, som ønskes tilføjet.

`semesterNo` - Semesternummeret, hvorpå kurset ønskes.

Returns:

Returnerer en streng med information til brugeren.

Throws:

`Kan` - kaste exceptions: ScheduleOccupiedException, CourseRequirementException, SemesterException.
ScheduleOccupiedException
CourseRequirementException
SemesterException

---

getName

```
public String getName()
```

Accessor, som returnerer den studerendes navn.

Returns:

Returnerer den studerendes navn, eks 'Peter Pedal'.

---

getStudieNr

```
public String getStudieNr()
```

Accessor, som returnerer den studerendes studienummer.

Returns:

Returnerer den studerendes studienummer, eks 's012345'.

---

removeCourse

```
public String removeCourse(int requestedCourseNo)
```

Mutator, som fjerner et specifikt kursus fra kursusplanen (på alle semestre, hvor det optræder). Fjerner også ubrugte semestre i slutningen (i enden) af studieplanen.

Parameters:

`requestedCourseNo` - Kursusnummeret på det kursus, som ønskes fjernet.

Returns:

Returnerer en streng med information til brugeren.

---

visplan

`public String` **`visplan(ArrayList<`**`String> legalLoc)`

Accessor 'visplan' genererere en streng med studieplanen for den aktive bruger. Hvis der er tomme semestre imellem ikke-tomme semestre, så vises disse også. Men tomme semestre i slutningen af planen vises ikke (de eksisterer faktisk slet ikke).

Parameters:

`legalLoc` - Array med lovlige skemaplaceringer.

---

skip-navbar bottom

Package Class Tree Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

skip-navbar_top
Package Class Tree Index Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

B.26.Class StuderendeTest

```
java.lang.Object
  └junit.framework.Assert
      └junit.framework.TestCase
          └StuderendeTest
```

All Implemented Interfaces:
junit.framework.Test

```
public class StuderendeTest
extends junit.framework.TestCase
```

Testklassen StuderendeTest.

Field Summary

(package private) Studerende	studeren1

Constructor Summary

StuderendeTest() Default konstruktør.	

Method Summary

protected void	setUp() Metode 'SetUp', som opretter en default studerende ved navn "Peter Pedal" med studienummer "s012345".
protected void	tearDown() Afslutter tests.
void	testAddRemove() Tester at et nyt kursus kan oprettes incl.
void	testConstructor() Tester konstruktøren ved at udlæse navnet og studienummeret, som skal stemme med det der blev genereret i 'SetUp'.
void	testNegativeTesting() Tester at et nyt kursus kan oprettes incl.

Methods inherited from class junit.framework.TestCase
countTestCases, createResult, getName, run, run, runBare, runTest, setName, toString

Methods inherited from class junit.framework.Assert
`assertEquals, assertFalse, assertFalse, assertNotNull, assertNotNull, assertNotSame, assertNotSame, assertNull, assertNull, assertSame, assertSame, assertTrue, assertTrue, fail, fail`

Methods inherited from class java.lang. Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

studeren1

`Studerende` **`studeren1`**

Constructor Detail

StuderendeTest

`public` **`StuderendeTest()`**

Default konstruktør.

Method Detail

setUp

`protected void` **`setUp()`**

Metode 'SetUp', som opretter en default studerende ved navn "Peter Pedal" med studienummer "s012345". Kaldes før alle test-metoder.

Overrides:

`setUp` in class `junit.framework.TestCase`

---

tearDown

`protected void` **`tearDown()`**

Afslutter tests. Køres efter hver test-metode.

Overrides:

`tearDown` in class `junit.framework.TestCase`

---

testAddRemove

```
public void testAddRemove()
                    throws ScheduleOccupiedException,
                           CourseRequirementException,
                           SemesterException
```

Tester at et nyt kursus kan oprettes incl. en skemaplacering for kurset. Herefter fjernes kurset igen, og det verificeres at der kommer besked til brugeren om at kurset er fjernet.

Throws:

ScheduleOccupiedException
CourseRequirementException
SemesterException

testConstructor

```
public void testConstructor()
```

Tester konstruktøren ved at udlæse navnet og studienummeret, som skal stemme med det der blev genereret i 'SetUp'.

testNegativeTesting

```
public void testNegativeTesting()
                        throws ScheduleOccupiedException,
                               CourseRequirementException,
                               SemesterException
```

Tester at et nyt kursus kan oprettes incl. en skemaplacering for kurset. Herefter tilføjes kurset til den studerendes studieplan på semester 1. Kurset 1017 forsøges herefter fjernet, og det verificeres at der kommer en fejlbesked med at kurset 1017 ikke kan fjernes, da det ikke indgår i studieplanen.

Throws:
ScheduleOccupiedException
CourseRequirementException
SemesterException