Indledende Programmering - Hjemmeopgave 3

Asbjørn Kjær Olling S163615
Oliver Sander Poulsen S174122

Arbejdsdelingen

Rapport

Oliver Sander Poulsen og Asbjørn Kjær Olling.

Problem 1 - Inheritance

Oliver Sander Poulsen

Problem 2 - Completing a program

Asbjørn Kjær Olling

Problem 1 - Inheritance:

I del a skulle vi færdigegøre boolean metoden `equals`, således at den kunne undersøge om to fly havde samme id. Dette har vi løst ved at lave et if statement, der sammenligner objektet, der er et plane, med et id-nummer fra et andet plane. Hvis id-nummeret er det samme som objektets id-nummer, vil der returneres true, ellers returneres false. Herudover har vi skrevet `toString`, så den printer en `String`, som består af Plane+id-nummer, manufacturer og type.

I del b skulle vi designe 2 subklasser - `PassengerPlane` og `FreightPlane`. De er begge opbygget med samme struktur, men med den forskel at, `PassengerPlane` har variablen `Passengers`, mens `FreightPlane` har variablen `payload`. Da de begge er subklasser af `Plane`, har vi lavet en `extends Plane` og lavet en konstruktør med en `super`, der refererer til `Planes manufacturer` og `type`. Til sidst er der en toString, der returnerer en `string` med `Plane+id`, `manufacturer`, `type` og henholdsvis `seats` eller `payload`.

I den sidste del, skulle vi lave en klasse kaldt `Airport`. Her startede vi med at oprette en `ArrayList` for `Plane`, der kan holde styr på hvilke fly der er i denne lufthavn.

Metoden land tjekker om et fly kan lande i lufthavnen, dette gøres ved et for loop, der går alle index i ArrayListen igennem, og sammenligner dem med flyet,

der skal lande. Hvis flyet ikke står på listen over fly i lufthavnen, vil det blive tilføjet til listen (i forlængelse af listen). Hvis det allerede står på listen, vil metoden stoppe via return.

`start` tjekker ligeledes listen igennem via id, men her fjernes flyet fra listen, hvis den står på. I tilfælde af at id-nummeret ikke findes på listen vil der ikke yderligere ske noget.

Den sidste metode `toString` laver en string af listen. Dette sker i form af et for loop, der løber fra index `0`, der er det første index på listen, til og med det sidste index, hvilket vil være størrelsen af Arraylisten-1 (deraf `k < planeList.size()`, i stedet for `<=`). For hver gennemgang omskrives index `k` fra listen til en string, og tilføjes til strengen `stringPlaneList`, med et linjeskift som afslutning. Efter for loopet er færdigt, vil metoden returnerer strengen `stringPlaneList`, der altså er en liste over alle fly fra vores `ArrayList`.

Problem 2 - Completing a program

Filerne `make_plants.sh` og `plant_template.java` udgør løsningen på Problem 2. Opgaven gik ud på at færdiggøre programmet, som udfører en simulation af plantevækst på Øresundsøen Peberholm.

Opgaven løstes ved at implementere en klasse for hver af de forskellige plantetyper. Plantetyperne viste sig at være identiske, bortset fra hvilke af `PeberholmConstantsAndUtilities`' konstanter der blev brugt. Det ville være redundant at implementere `spreadSeeds` metoden i hvaer af de fire forskellige `Plant`-subklasser. Det ville altså være idéelt at have én implementering af metoden i f.eks. `PeberholmConstantsAndUtilities`.

Vi strødte dog på ent kritisk problem: at hver plantes `spreadSeeds` skal generere objekter af forskellige klasser. Vi forsøgte vha. reflection i java at bruge en klasse som parameter til en metode i `PeberholmConstantsAndUtilities`. Dog havde vi problemer med `getNewInstance` metoden til formålet, og måtte gå til en anden løsning:

`plane_template.java` er en skabelon for de fire `Plant`-subklasser: `Bush.java`, `Tree.java`, `Flower.java` og `Moss.java`, hvor navnet på planten er erstattet med en placeholdertekst `<Type>`. `make_plants.sh` er et script, som bruger GNU `coreutils` og `sed` til at generere de fire nye (nær-identiske) klasser. Det giver udvikleren fordelen af kun at skulle redigere i én fil, for at ændre i planternes opførsel eller at tilføje en ny plante-type. (Kildekoden til de fire resulterende klasser er inkluderet i afleveringen).

Selve metoden `spreadSeeds` udgør hoveddelen af opgaven, og er i og for sig ikke særligt kompliceret. Rækkevidden for den idividuelle plante er udregnet i konstruktøren, som bruges til at skabe et nyt punkt til en plantes afkom. Der genereres planter i en `for`-løkke, som returneres i et `Plant[]` array.