Del 1: Beskrivelse 150-200 ord

Applikasjonen jeg har laget heter «Airline Manager». Det er et spill der man oppretter et flyselskap og oppgaven er å utvide selskapet med nye fly og reise rundt hele verden. Spillet er inspirert av «Pocket Planes» (Wikipedia, 2022). Man starter med en kapital som tillater at man kan kjøpe sitt første fly og begynne å frakte passasjerer til de nærmeste flyplassene. Hver passasjer betaler en viss sum for reisen. Denne er beregnet basert på avstanden til destinasjonen. Etter hvert som man tjener penger, kan man kjøpe seg flere fly. Det finnes ulike flytyper som kan ha med flere passasjerer, fly lengre, kjappere, og mere effektivt. Når man starter en flytur beregnes flytiden basert på avstand og hvor fort flyet kan fly. Når flyet har landet mottar man pengene fra passasjerene som har ankommet riktig flyplass. Har man med passasjerer som skal videre blir de værende i flyet.

På hver flyplass er det et maks antall reisende. De skal til forskjellige destinasjoner så det kan være at man ikke klarer å fylle opp flyet med passasjerer for å maksimere profitt. Ønsker man å fylle opp flyet med passasjerer som skal til samme destinasjon, vil flyplassene oppdatere reisende det hvert femte minutt.

Del 2: Kun ett diagram

Del 3:

1. Arv er implementert i klassene Airport og City. Tanken bak det er at en by kan ha flere flyplasser, og at det derfor er smart at flyplassen kan utvide byen. En slik løsning gjør det enkelt å utvide klasseforholdet med mere informasjon om byen eksempelvis byen. Jeg har blant annet ikke lagret hvilket land byen ligger i, men dette kan legges til på en smart måte senere. I tillegg har flyplassene ulike koordinater, og arv gjør det mulig å lagre ulike koordinatverdier. Foreløpig gir det ingen mening å opprette en instans av City klassen, derfor har jeg gjort den abstrakt.

I prosjektet jeg har prøvd å dele opp koden i så mange klasser jeg føler jeg nødvendig, for å konkretisere hva klassene gjør. Det medfører og jeg jevnt over har delegert mye. Delegeringen som brukes mest er beregningen av avstanden mellom flyplasser. Jeg har laget en egen klasse (CalculateFlightDistance) med en statisk funksjon for å regne ut avstanden mellom to flyplasser med «Haversine»-formelen. Flyplassene implementerer grensesnittet «Comparable» og inneholder metoden «compareTo(Airport)», som delegerer utregningen til CalculateFlightDistance. Denne delegeringen brukes også i «AirportDistanceComparator».

Timingen i spillet er delegert til klassen SecondClock. Spillet kan startes og stoppes ved at AirlineManagerGame kaller på SecondClock. Foreløpig brukes start og stopp metodene kun ved innlasting av fil. Viktigere er det at SecondClock er observerbar, og har ansvaret for å kalle på observende klasser. Observerende klasser (Kontrolleren, Spill-klassen og Plane-klassen) implementerer SecondClockListener-grensesnittet og inneholder tick() metode.

Jeg har valgt å lage to grensesnitt for filbehandling. «InterfaceGameSaveHandler» beskriver hvordan en filbehandlingsklasse som skal lagre og laste opp tilstanden til spillet, skal implementeres. «InterfaceGameFileLoader» beskriver hvordan klasser som skal lese inn spillinformasjon skal fungere. Data for flyplasser og flytyper er lagret i csv-filer og er viktig at blir lest inn riktig. Grensesnittet hjelper i tilfelle disse klassene må oppdateres.

Selv om iteratoren ikke brukes mye i prosjektet, har jeg laget «PlaneIterator». Den gir en enkel metode å behandle alle Plane-objektene i Airline-klassen, spesielt hvis jeg utvider med mulighet for å behandle flere Airline-objekter. Airline-klassen implementerer «Iterable» som gjør at man kan skrive en «for-each» løkke for å iterere over flyene i flyselskapet.

1. Dersom deler av pensum ikke er dekket i prosjektet deres, hvordan kunne dere brukt disse delene av pensum i appen?
2. Koden forholder seg bra til Model-View-Controller prinsippet. Jeg har strippet kontrollen for så mye tilstand og logikk jeg kan. Den har kun ett felt med binding til «AirlineManagerGame», og konstanter for hvordan knappene i brukergrensesnittet skal se ut. Til å begynne med tok jeg vare på hvilket fly som var i fokus, i kontrollen, men skjønte etter hvert at jeg hadde mulighet til å flytte den logikken inn i spill-klassen. (SE OM DET ER NOE SOM LIGGER IGJEN I KONTROLLER)

Det jeg er mest usikker på er implementasjonen av klokka i applikasjonen. For å holde brukergrensesnittet oppdatert med tidsbestemte tilstander i spiller måtte jeg sette kontrollen som en observerende med SecondClockListener-grensesnittet. Jeg vet ikke om det er riktig implementasjon av en timer i en applikasjon, men det er det nærmeste jeg kommer å fjerne logikk fra kontrolleren. Det holdt ikke mål å kun oppdatere brukergrensesnittet når en bruker benytter funksjoner i applikasjonen.

1. Hvordan har dere gått frem når dere skulle teste appen deres, og hvorfor har dere valgt de testene dere har? Har dere testet alle deler av koden? Hvis ikke, hvordan har dere prioritert hvilke deler som testes og ikke? (Her er tanken at dere skal reflektere rundt egen bruk av tester)
   1. Ikke valgt å teste gettere og settere da disse er enkle.
   2. Tester de viktigste delene av spillet. Passe på at interne regler følges, så ikke et flyselskap kan kjøpe fly uten penger, ta av uten penger, at reisende oppdateres, at userInput ikke er ulovlige, at posisjonen til objekter er riktig (Passasjer), regler for takeoff og landing for å unngå at andre deler av koden feiler.
   3. Har ikke klart å få en test til å funke. Får visuelt riktig resultat, men ikke kodevis.
2. Har dere møtt på noen utfordringer i løpet av prosjektet? Hva ville dere gjort annerledes en annen gang?
   1. Jeg har valgt å ikke la Plane-klassen utvide Aircraft for å kapsle inn tilstanden best mulig. Jeg ønsker at alle fly som kjøpes skal referere til samme instans
   2. Tanken var å opprette flyene og airline ved load samtidig, og kun tillate at kontrukøren kunne ta inn fly uten betaling, men det var ingen god måte å gjøre det på med restriksjonene rundt filskriving og lesing. Derfor la jeg på at metodene for å reopprette data måtte ha med signaturen til en gamesavehandler for å fungere.
   3. Koden er preget av at jeg ikke har mye erfaring med javaFX, og at noe funksjonalitet ville vært skrevet på en helt annen måte hadde jeg visst mulighetene til JavaFX. Visste f.eks ikke at «button» kunne ta vare på objektet i seg.
   4. Burde definitivt ha valgt et mindre prosjekt. Ble massivt, med mange filer, og koblinger som gjorde at filbehandlingen er knotete. Har gjort klart for funksjonalitet jeg ikke rekker å lage, som liveries til flyene. Men det var noe jeg var interessert i å få til å funke, og ikke noe som var kjedelig å jobbe med.

Referanser

Wikipedia. (2022, April 20). *Haversine Formula*. Hentet fra Wikipedia.org: https://en.wikipedia.org/wiki/Haversine\_formula

Wikipedia. (2022, April 20). *Pocket Planes*. Hentet fra Wikipedia.org: https://en.wikipedia.org/wiki/Pocket\_Planes