Sprint4: Field klasse

Developer A

Lav en feature branch: `sprint4_field_[par]`

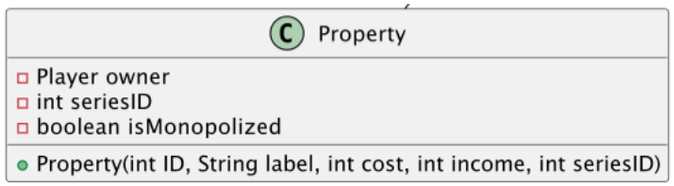
-Opret `Field` klassen med konstruktor og metoder der følger klassediagrammet *doc/plan/static/classdiagram_v2.puml*

Implementer en `toString()` metode, der returnerer feltets ID og label.

Implementer `onLand()` så den returnerer, hvem der er landet og hvor.

Eksempel:
“`Egon er landet på felt 4, Hvidovrevej`”

- Opret klassen `Property` som nedarver fra `Field`, og har to ekstra attributter: `int serieID, boolean isMonopolized`.

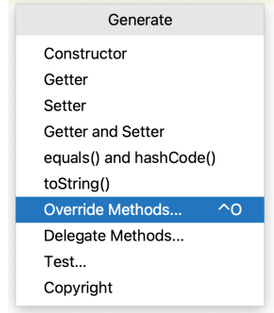


- Opret på samme måde klassen `Plot` som nedarver fra `Property`. Generer kode til at override metoderne.

- I `onLand` metoden skal der **på sigt** skrives kode, der bygger på algoritmen vist i diagrammet. *Se doc/plan/dynamic/activity_uc4_landAndAct_plot.puml*
Skriv pseudokoden som kommentar inde i metoden.

Developer B

I det følgende skal du i Property klassen overskrive metoderne fra superklassen. Du må du gerne generere denne kode:
command+n (el. ctrl+n) og vælg Override Methods



Metoderne vil blive genereret sådan at de blot returnerer resultatet af et kald til.

- Ret lidt i `toString`, så den returnerer resultatet af et kald til `super.toString()` men med `serieID` hæftet på.

- I `onLand`: I metoden skal der **på sigt** skrives kode der bygger på algoritmen vist i diagrammet: *doc/plan/dynamic/activity_uc4_landAndAct_property.puml*
Skriv pseudokoden som kommentar inde i metoden.

- Opret på samme måde klasserne `Brewery` og `ShippingLine` som nedarver fra `Property`

- Lav en ny version af klassediagrammet *doc/plan/classdiagram_v2.puml* hvor alle subklasserne til `Field` klassen indgår.
Gem diagrammet som *doc/plan/classdiagram_v3.puml*

Sprint4: FileIO klasse

Developer A

Lav en feature branch: `sprint4_fileIO_[par]`

– I skal sørge for indlæsning af data til at bygge spillepladen.

- I `Game` klassens `startSession` metode, nederst, indsættes nu flg. linjer. (Ignorer compilefejl)

```
String[] carddata = io.readData(“data/carddata.csv", 10);
String[] fielddata = io.readData("data/boarddata.csv", 10);
```

Evt: udskift de to stier med instansvariable som følger navnekonventionen for `playerDataPath`

- Lige efter instantiering af `File`, tilføj en tom `try-catch` blok som fanger `FileNotFoundException`

- Brug Scanner klassen i kombination med et loop for at læse hver enkelt linje i filen. For hver iteration I loop’et skal linjen lægges ind i et `String[]`.
(Dette kræver at du initialiserer en counter udenfor loop’et, som tælles op med en inde i loopet)

Developer B

Tilføj en metode til `FileIO` klassen,
`String[] readData(String path, int length).`
(`path` er stien til filen der skal loades og `length` er antallet af linjer i filen)

Bemærk at metoden skal være næsten identisk med den anden `readData` metode i klassen. Forskellen er at denne metode returnerer en `String[]` i stedet for en `ArrayList<String>`

– I den nye `readData`, start med at initialisere en String array, `data`, med plads til `length` elementer.

- Lav en instans af `File` til at loade den fil der er angivet i `path`.

- I bunden af metoden, returner `data`

- Bed data gruppen om “sample data” til at teste med.

- Placer deres sample data det rigtige sted (se stien i den kode I indsatte i første task).

Test
I bunden af `Game` klassens `startSession`, lige efter de to kald til readData, tilføj test kode:

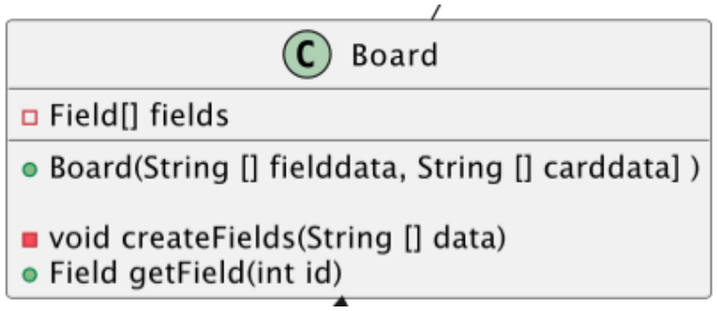
```
System.out.println(“tester card data: “+carddata[0]);
System.out.println(“tester field data: “+fielddata[0]);
```

Sprint4: Board klasse

Developer A

Lav en feature branch: `sprint4_board_[par]`

- Opret klassen `Board` med attributter, konstruktør og metoder der afspejler klassediagrammet:



- I konstruktoren, initialiser `fields`, så der er plads til ligeså mange elementer som `fielddata` er lang.

Metoden `createFields` skal generere `Field` objekter ud fra indholdet af array, `data[]` der modtages som argument. Hvert element i dette array beskriver et felt som en lang string. F.eks:
`"13, brewery, Coca Cola, 3000, 500, 7"`

- Lav et for-loop der kører så mange gange som `data[]` er lang.
- brug `split()` metoden til at ændre hvert String-element til et String-array:

```
String[] values = data[i].split(",");
```

`values` ser derefter sådan ud:
`["13", "brewery", "Coca Cola", "3000", "500", "7"]`

- Gem nu værdierne i `values` i passende variable.

Eksempel:
```
int cost = Integer.parseInt(values[3].trim());
```
- instansiér `Field` med værdierne i `values` som argumenter og placer instansen i `fields` arrayet.

Developer B

I er afhængige af Field teamet og CardDeck teamet for at få jeres kode til at compile, og er nødt til at koordinere med dem for at teste. Alternativt kan I lave dummy klasser mens i arbejder, sådan at jeres kode compiler, men undlad at commit'e dummy klasser.

Fra `Board` konstruktoren:

- Kald `createFields` fra konstruktoren med `fieldata` som argument
- Sæt denne linje sidst i konstruktoren (evt. som kommentar)

```
Chance.cardDeck = new CardDeck(carddata)
```

Denne linje sørger for at kortene bliver lavet og er tilgængelige for chancefelterne. CardDeck teamet, laver Chance klassen.

Tilføj metoden `Field getField(int i)` som returnerer en instans fra index `i` i `fields[]`.

Et kald til metoden fra `Game` klassen vil se sådan ud:
`board.getField(40)`
Metoden skal returnere det felt der repræsenterer Rådhuspladsen.
`board.getField(1)`, skal returnere det felt der repræsenterer Start.

Test
I kan først teste når det hele bliver samlet. Når det sker vil vi fra Game klassen nu kunne indsætte flg.:

```
//test Board
this.board = new Board(fielddata, carddata);
Field f = this.board.getField(11);
```

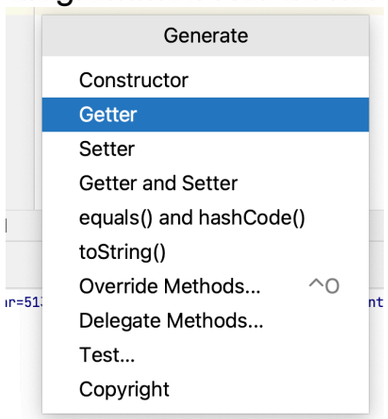
Sprint4: CardDeck klasse

Developer A

Lav en feature branch: `sprint4_carddeck_[par]`

Find klassediagrammet frem:
(doc/Plan/static/classdiagram_v2.puml)

Opret `Card` klassen således at den afspejler diagrammet. Når du har tilføjet de rigtige attributter kan du bruge autogenerering til at tilføje konstruktør og gettere:
command+n (el. ctrl+n)



I `createCards` metoden, skal parameteren `data` gennemløbes:

- Lav et `for-` loop, der kører så mange gange som `data` er lang.

- Inde i loop' et, brug `split()` til at ændre hvert element i `data` til et String-array:

```
String[] values = data[i].split(",");
```

- Gem værdierne i `values` i passende variable.
Eksempel: `String message = values[0];`

- Stadig i for-loop'et, lav en ny `Card` instans, med de variable du lige har lavet:

```
Card c = new Card(message, income, cost, event)
```

- Placer hvert kort i `cards` array'et

Developer B

Opret `CardDeck` klassen således at den afspejler diagrammet

I konstruktoren

- Initialiser `counter` attributten med værdien 0;
- Initialiser `cards` arrayet så der er plads til ligeså mange kort som parameteren `carddata[]` er lang.
- kald metoden `createCards` med `carddata` som argument.

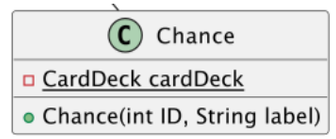
Bemærk at der er forskel på carddata og cards. Det første er `String` objekter det andet er `Card` objekter

- Bland kortene ved at indsætte denne kommando når alle kortene er blevet lagt i array'et (Kræver et par imports):

```
Collections.shuffle(Arrays.asList(cards));
```

- I `CardDeck`'s `getNext` metode, skriv kode der finder og returnerer det næste kort i stakken. (Brug `counter`, til at holde styr på hvilken plads i stakken vi er nået til)

For at vi til review kan teste jeres kode, skal I gøre følgende:



- Opret `Chance` klassen. Den skal nedarve fra Field
- Giv klassen attribut og konstruktor. I kaldet til super skal `cost` og `income` være 0,

Board teamet vil sørge for at den bliver tildelt en værdi. Koordiner evt. med dem.

Sprint4: Data

Developer A

Lav en feature branch: `sprint4_data_[par]`

I er afdelingens datateam, og skal klargøre 2 datasæt:
- data om spillepladens felter
- data om chancekort

- Opret en en tekstfil, `fielddata.csv`, placer den i `data` folderen med flg. header:

```
ID, FieldType, Label, Cost, Income, SeriesID.
```

Læs en beskrivelse her: *data/metadata.md*

- Skriv data til **kun de første 10** felter. Dette er sample data

- Del sample datasættet med team FileIO, så de kan bruge det til at teste deres kode.

- Færdiggør datasættet for felter. **Tip:** Fordel arbejdet imellem jer.
- Del igen med team FileIO
- Vær klar til at modtage rettelser fra team FileIO

Developer B

HVIS DER ER TID:

- Opret endnu en tekstfil, `carddata.csv`, placer den i `data` folderen med flg. header:

```
Message, Income, Cost, event
```

Læs beskrivelse her: *data/metadata.md*

- Skriv data til de **første 10** felter.

Noter undervejs i *data/metadata.md* hvis I opdager noget vores datastruktur ikke tager højde for.

- Del sample datasættet med team FileIO.

- Fortsæt analyse datasættet for chancekort.

    - Del igen med team FileIO
    - Vær klar til at modtage rettelser fra team FileIO