3.sem DW (OO-Programming) E17

Obligatorisk afleveringsopgave 1 *Erik Sørensen 30. august 2017*
Opdateret 19. September

Som bekendt skal der være et antal porteføljeopgaver i kurset. I OOP delen vil der være to obligatoriske programmeringsopgaver, som skal løses, for at kunne deltage i den afsluttende eksamen:

Opg.1: En simpel udgave af det velkendte, gamle brætspil *Matador (Monopoly* på engelsk).
Opg.2: En simpel TCP/IP server, med adgang til databaser (jdbc og NoSql). Denne opgave kobles sammen med en klient fra Andrea's del af kurset.

Opgaverne løses i grupper på 2-3 studerende og afleveres på BlackBoard hhv. **25/10 under lektion 7** og **29/11 under lektion 12**. Rettidig aflevering er en betingelse for at kunne deltage i eksamen.

Tidsplan

Følgende vejledende tidsplan er lagt for projektet. Undervisningen under lektionerne er tilrettelagt, så de nogenlunde følger planen.

Lektion 2: Gruppedannelse og start på opgave 1.

Lektion 3: Kast med terninger og flytning af spillere rundt på spillepladen.

Lektion 4 - 5: Arv og Polymorphism på felterne.
Køb af Grunde/Bryggerier/Dampskibsselskaber
samt betaling for at lande på anden spillers ejendom.

Lektion 6: File I/0. Informationer skrives i / læses fra filer.

Lektion 7: Aflevering af NetBeans projekt på BlackBoard,
Fremlæggelse for klassen og (evt.) godkendelse.

Lektion 8: Start på opgave 2.
Simpel, trådet TCP/IP server

Lektion 9 - 10: JDBC: Informationer skrives i / læses fra SQL-database

Lektion 11: NoSQL: Informationer skrives i / læses fra NoSQL-database

Lektion 12: Færdiggørelse og aflevering.

Porteføljeopgave 1: Matador

Der skal udvikles en simpel udgave af det velkendte, gamle brætspil *Matador (Monopoly* på engelsk).

Herunder gives et forslag til arbejdet fra lektion til lektion, indtil det samlede projekt afleveres under lektion 7.

Opgave til Lektion 3

Efter lektion 3 bør (mindst) følgende være implementeret:

- En klasse `DiceCup` indeholdende to instancer af `Die`, hver repræsenterende en terning med 6 sider. `DiceCup` skal have en metode, `public int throwDice()`, som kaster de to terninger og returnerer det samlede antal øjne. Desuden en metode som, når den kaldes, fortæller om der blev slået to ens.
- En klasse `Field`, repræsenterende felterne på brættet. I første omgang behøver `Field` kun at indeholde disse indkapslede variable:
  - `String name` // et kort navn på feltet
  - `int number` // et tal i intervallet [0..39]

  Begge variable skal initialiseres i en constructor og der må kun være `getter()`-*metoder* på dem, da de aldrig skal ændres efter oprettelsen.
  Desuden skal der være en metode med signaturen `public String toString()`, så det er nemt at udskrive hvilket `Field` en spiller er landet på.
  NB: *Klassen er foreløbig og skal erstattes af et interface og et arve-hierarki senere i forløbet.*
- En klasse `Player`, som (mindst) indeholder variablene:
  - `String name` // et kort navn på spilleren
  - `Field currentField` // En reference til det felt spilleren står på

  `Player` skal indeholde en metode, `public int move(DiceCup cup)`, så det simuleres at spilleren får raflebægeret og selv slår med terningerne.

- Simulering af spillet skal implementeres i en driver-klasse, som opretter et statisk array med 40 pladser, hver indeholdende en reference til en instance af klassen `Field`. Desuden oprettes mindst 2 spillere, som placeres på "Start"-feltet, (indeks 0 i arrayet). I et loop skal spillerne kaste terninger efter tur og det skal skrives på `System.out`, hvor de lander efter hvert slag. Find selv på en betingelse for at slutte spillet, fx kan vinderen være den der først når et antal omgange rundt.

Opgave til lektion 4 og 5

- *Arv, Interfaces og polymorphism på felter.*
- *Metode til consequence() implementeres. (Defineres i et Interface eller abstrakt klasse og implementeres i de konkrete klasser)*
- *player tilføjes en liste til købte grunde mv.*

Udleveret kode

Pakken `dw_monopoly_2` indeholder 4 javafiler som kan benyttes som udgangspunkt for et arve heiraki for Felterne i spillet. Det er helt frivilligt hvordan i vil bruge koden (slet ikke, som inspiration, bruge den som den er eller ændre i den):

- `public interface MonopolyConstants:` Indeholder nogle definerede konstanter (`public final static`), som kan benyttes fx under opstart af programmet. Bla. er der et array af 40 String, indeholdende feltnavnene fra en gammel udgave af Matador (ca. 1960). Det er ikke nødvendigt at implementerer dette interface. Konstanterne kan tilgås statisk, fx:
  `MonopolyConstants.FIELD_NAMES[10]` giver navnet på felt nr. 10: "Fængsel".

- `public interface FieldInterface:` Dette interface definerer de grundlæggende operationer et Felt har:
  - `String getName();`
  - `int getNumber();`
  - `void consequense(Player poorPlayer);`, hvor `poorPlayer` er den stakkels spiller som er havnet på et felt.

- `public abstract class OwnebleField implements FieldInterface`: Denne klasse er fælles for de 3 typer af felter, som en spiller kan købe og måske tjene penge på: *Grunde, Dampskibs selskaber/Færgeruter* og *Bryggerier/Sodavandfabrikker (i disse politisk korrekte tider)* : Der er disse variable:
  - `private String name;`
  - `private int number;`
  - `private int price;`
  - `private Player owner = null;`

  De to metoder `getName()` og `getNumber()` fra interfacet implementeres, samt `getPrice()` –, og `getOwner()`-metoder. Der er desuden en `setOwner(Player owner);` som benyttes når feltet bliver købt (`owner == null` indtil det er solgt).

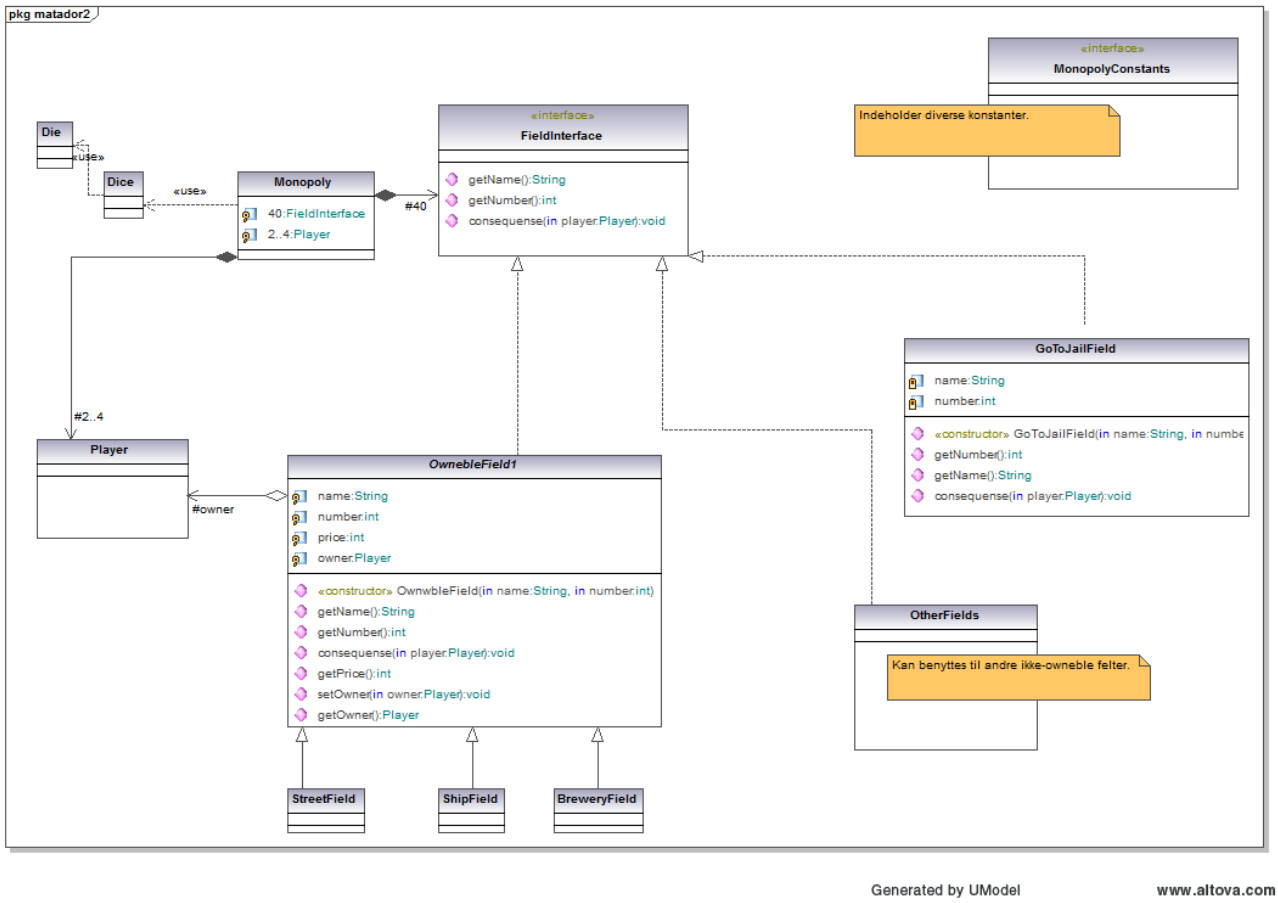
  Metoden `void consequense(Player poorPlayer)`, er *ikke* implementeret her, da konsekvenserne er forskellige i de 3 klasser, der extender `OwnebleField`. Den manglende

implementation er grunden til at klassen er `abstract`. Fælles for implementationerne af `consequense()` vil være:

- Hvis feltet ikke ejes (`owner == null`) kan det købes. Det er op til jer at belutte om det altid sker, eller om der skal træffes en beslutning om at købe (fx med en Random-generator eller prompt for input fra tastaturet)
- Hvis feltet ejes af den player, som netop er landet på det, udskrives en meddelelse herom.
- Hvis feltet ejes af en anden player, betales et beløb til `owner`.

- `public class GoToJailField implements FieldInterface`: Et eksempel på en klasse som *ikke* kan købes og derfor heller ikke extender `OwnebleField`. I stedet implementeres alle metoder fra `FieldInterface`.

Herunder ses et klassediagram:



De fire udleverede klasser/interfaces er her (næsten) fuldt designet og sat i sammenhæng med spillets øvrige klasser.

Minimumskrav til opgaven

Implementer mindst én af klasserne, som extender `OwnebleField`, således at en spiller, som rammer et felt af denne type får konsekvenserne at føle, som beskrevet i spillets regler.

Opdater `Player`-klassen (fra 1. del af opgaven), så den får metoder, der kan kaldes fra de forskellige `consequense()`-implementationer.

Implementer klassen `OtherField implements FieldInterface`. Implementationen af `consequense(Player poorPlayer)` kan blot være noget i retning af :

```
System.out.println(poorPlayer.getName() + " landed on " + this.getName());
```

Opdatering af driverklassen, så den kan håndtere ovenstående. Det er et krav at felterne i driverklassen indlæses i et array af interfacetypen `FieldInterface`. Dvs der skal benyttes polymorphism.
Hint: oprettelse af felterne kan fx ske med denne metode, hvor `plate` er `FieldInterface`-arrayet:

```
public void fillFields(){
     for(int i = 0; i< plate.length; i++){
         switch(i+1){
              // Andre felter:
              case 1:
              case 3:
              case 5:
              case 8:
              case 11:
              case 18:
              case 21:
              case 23:
              case 34:
              case 37:
              case 39:
                  plate[i] = new OtherField(MonopolyConstants.FIELD_NAMES[i], i+1);
                  break;
              // Gå i spjældet:
              case 31:
                  plate[i] = new GoToJailField(MonopolyConstants.FIELD_NAMES[i], i+1);
                  break;
              // Rederier:
              case 6:
              case 16:
              case 26:
              case 36:
                  plate[i] = new ShippingCompany(50, MonopolyConstants.FIELD_NAMES[i],i+1,200);
                  break;
              //Bryggerier:
              case 13:
              case 29:
                          // Her er dice en parameter, da betaling afhænger af antal øjne der er slået
                  plate[i] = new Brewery(dice, MonopolyConstants.FIELD_NAMES[i], i+1, 150);
                  break;
              // Gader:
              default:
                          // Afgift for at lande på feltet og pris for at købe feltet er her simuleret
                          // med hhv 3*i og 10*i. I er velkomne til at finde på noget smartere.
                  plate[i] = new StreetField(3*i, MonopolyConstants.FIELD_NAMES[i], i+1, 10*i);
         }
     }
```