Adventure del 2 review

kea
KØBENHAVNS ERHVERVSAKADEMI

Læringsmål

- Forstå coupling som koncept for hvor tæt klasser er forbundet.
- Kunne refaktorere kode til at opnå lav kobling

Indhold

- Introduktion af lav kobling
- Introduktion af Demeters Lov
- Review af Adventure del II med fokus på information expert og coupling.

- Refactor / Ryd op i Adventure del 2 så vi er klar til del 3

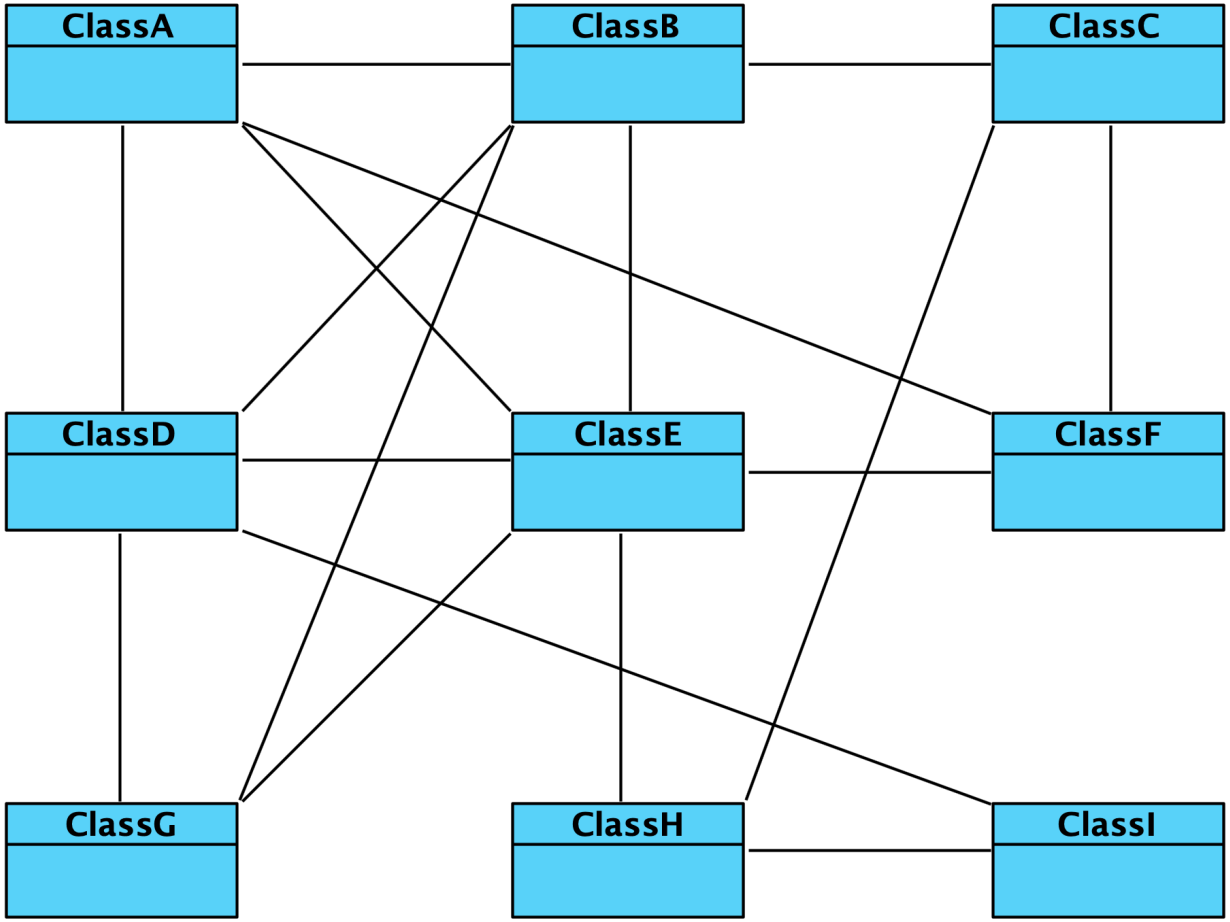
Lav Kobling (Low coupling)

- Et godt design er:
  - let at forstå
  - let at teste
  - let at vedligehold
- Opnås bl.a. med lav kobling (low coupling) og høj samhørighed (high cohesion)

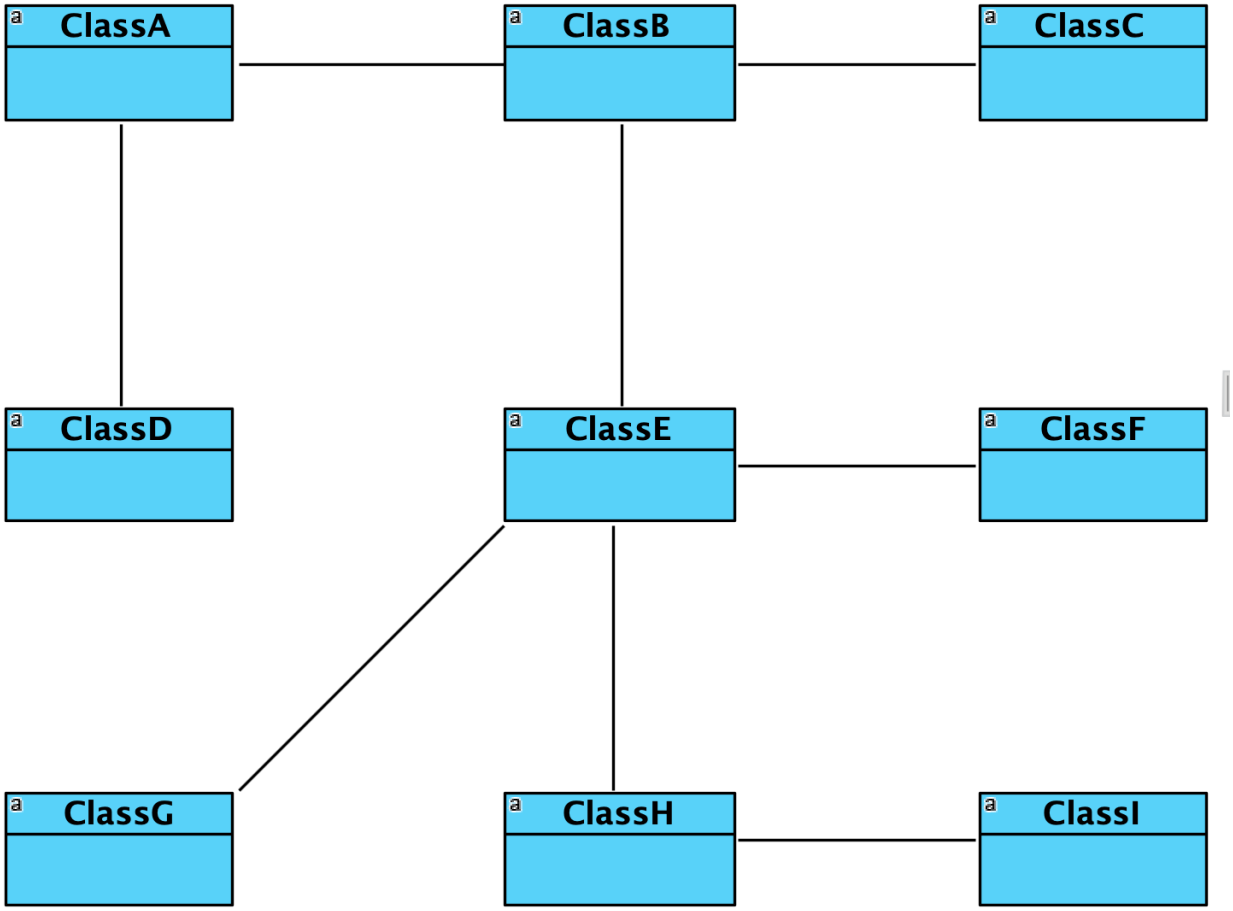
Lav Kobling (Low coupling)

- En klasse skal være så uafhængig som muligt
- En klasse skal kun associeres med de få klasser, der er nødvendige for, at den kan opfylde sit ansvarsområde
- Dermed lettere at forstå og vedligeholde

Høj Kobling (Bad ☹)



Lav Kobling (Good 😊)



Law of Demeter
(Principle of Least Knowledge)

- Tale kun med dine nærmeste naboer (og ikke dine naboers naboer)
- Minimere afhængigheder mellem moduler (klasser)
    - opnå lav kobling

Eksempel i Adventure spil

- "Talking to strangers" (violating Law of Demeter)
- Dvs. kalde metode på et objekt, som man har fået fra et andet kald.

Adventure klasse:

```
public boolean takeItemDemo(String itemName) {
  Room currentRoom = player.getCurrentRoom();
  List<Item> items = currentRoom.getItemsDemo();
  for(Item item : items) {
    if(item.getName().equalsIgnoreCase(itemName)) {
      return true;
    }
  }
  return false;
}
```

Objekt orienteret principper i Adventure

- Single Responsibility
- Creator
- Controller
- Information Expert
- Law of Demeter
- Lav kobling

Objekt orienteret principper i Adventure

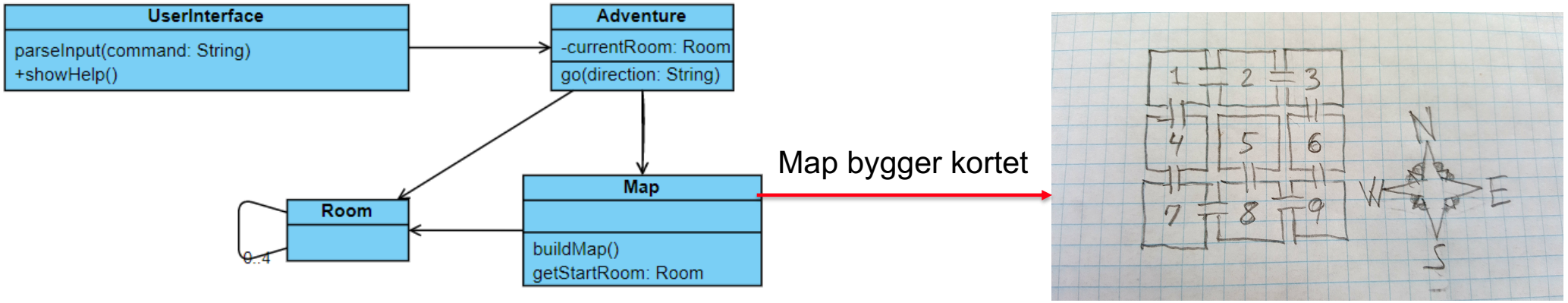
- Hvordan har I anvendt følgende? Diskuter i gruppen?
  - Single Responsibility
  - Creator
  - Controller
  - Information Expert
  - Law of Demeter
  - Lav kobling

Creator

- Hvem har ansvaret for at skabe et nyt objekt?
- Normalt vil ”container” klasse få tildelt ansvaret for at oprette ”indeholdte” (contained) objekter.

- Hvem skal lave kort over alle rum, forbindelser ml. rummene og lægge items i rum?
- Det er specialopgave at konfigurere kortet ved opstart af programmet.
- Vi skal passe på med at overbebyrde Adventure med for mange forskellige typer opgaver (Single Responsibility princippet)
- I flg. problemdomænet, vil en ny klasse ”Map” være et oplagt bud

Creator


UserInterface
parseInput(command: String)
+showHelp()
Adventure
-currentRoom: Room
go(direction: String)
Room
0..4
Map
buildMap()
getStartRoom: Room
Map bygger kortet
1
2
3
4
5
6
7
8
9
N
W
E
S

Controller

- Hvem styrer og koordinerer programflow?
- Adventure er oplagt kandidat
- (Vi kan udelukke UserInterface, fordi den allerede har ansvar for brugerdialog – også Map, som står for konfiguration af kortet)
- Adventure bliver ”single point of entry” for UserInterface, og kommer til at styre spillet.
- UserInterface har kun brug for Scanner og Adventure som attributter:

Controller

```
public class UserInterface {

  private Scanner scanner = new Scanner(System.in);
  private Adventure adventure;
```

Controller

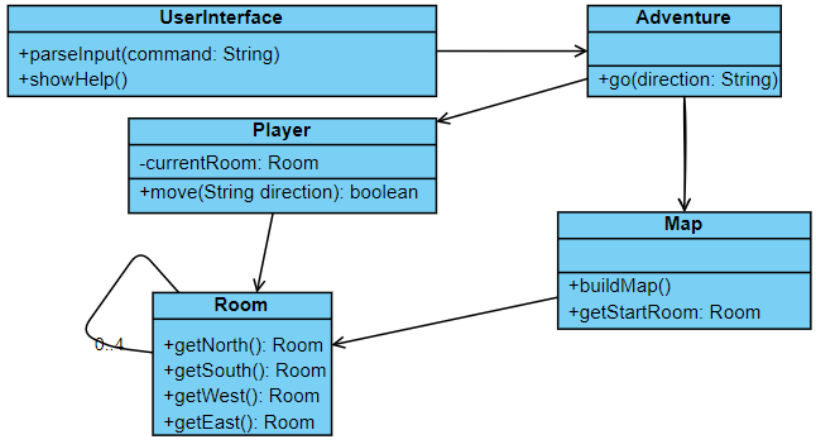
- I Adventure delopgave 2 skal man kunne meget forskelligt, f.eks. få vist hvad spilleren bærer rundt på, spiller skal kunne tage ting i rum og efterlade ting i et rum.

- Det bliver for meget for Adventure klassen både at skulle koordinere spillet og udføre spil-kommandoerne (inventory take, drop).

- Derfor må delopgaver (dvs. udførelsen af selve spil-kommandoerne) uddelegeres til relevante objekter.

Controller

- Adventure bliver controller og står for
  - adskillelsen ml. brugergrænsefladen og spillet (domæneobjekterne).
  - styring og koordinering af domæneobjekter
- Ny Player klasse overtager kendskab til aktuelle Room og står:
  - Flyt mellem rum
  - Lede i rum efter items
  - Tage item op i rum
  - Efterlade item i rum
  - osv.



Information Expert

- Adventure beder Player om at flytte til et andet room

Player.java

```
103 @   public boolean move(String direction) {
104
105         Room desiredRoom = switch (direction) {
106             case "north", "n" -> currentRoom.getNorthRoom();
107             case "south", "s" -> currentRoom.getSouthRoom();
108             case "east", "e" -> currentRoom.getEastRoom();
109             case "west", "w" -> currentRoom.getWestRoom();
110             default -> null;
111         };
112
113         if(desiredRoom != null) {
114             currentRoom = desiredRoom;
115             return true;
116         }
117         else {
118             return false;
119         }
120     }
```

Information Expert

- Adventure kunne godt bede Player om at returnere currentRoom og selv foretage flytning men...

- ...det er i modstrid med Informationsekspert princippet (og law of Demeter), som altid uddelegerer opgaven til den som er tættest på data.

Inden Adventure del 3

- Refactor!