Ursäkta svengelskan. Vi kör svenska instruktioner men design-delarna är på engelska.

Larman, s 250, särskilt det stora diagrammet/den stora rutan med SuperClassFoo

Sista sidan, insidan av pärmen, Särskilt Class Diagram:
ClassX
Composition
Association
staticAttribute
+publicAttribute
-privateAttribute

Man kan skriva klassen Room, så långt som vi kom i måndags, så här:

```
+-------------------------------------------------------------------+
|                                Room                               |
+-------------------------------------------------------------------+
| -north : Room [0..1]  // rummen kan vara null                     |
| -south : Room [0..1]                                              |
| -west  : Room [0..1]                                              |
| -east  : Room [0..1]                                              |
| -things : Thing [*]   // något slags lista av Thing               |
| -description : String // Vi skulle kunna göra UML av String också...|
|                       // men vi skiter i det - vi kollar Javas API  |
|                       // i stället                                 |
+-------------------------------------------------------------------+
|      // beteende - metoder - här - vad kan vi be ett rum göra?    |
+-------------------------------------------------------------------+
```

Klassen Thing blev “nästan färdig”

```
+---------------------------+
|         Thing             |
+---------------------------+
| -name : String            |
+---------------------------+
| +Thing(name : String)     | // constructor
| +thing() : String         | // returnerar namet
+---------------------------+
```

```
// Skriv ingen kod, detta är bara ett exempel på hur ovanstående
// skulle kunna se ut i Java!
public class Thing {
  private String name;
```

```
  public Thing(String name) {
    this.name = name;
  }
  public String name() {
    return name;
  }
}
```

```
+---------------------------+
|          Player           |
+---------------------------+
| -currentRoom : Room       | // Player har referens till aktuellt rum
| -inventory : Thing[*]     | // Player har noll eller flera saker
+---------------------------+
| //beteende/metoder        | // Vad kan vi be Player göra?
+---------------------------+
```

```
A Room has references to at most four other Room:s.

The references are called north, south, east, west. If any is null, it signifies “no other room in that
direction”, otherwise (not null) they signify the Room in that direction.

This allows us to build the whole “Cave” from the database.

Each Room object will have references to the Room:s in the directions north, south, east, west (if it
has such a connecting room).
```

För att förklara hur ett Room kan ha referenser till andra Room, så tog vi ett exempel från bankvärlden. Banken i Kvibille, Dragans Bank och Pantbank, skulle kunna ha ett system för sina kunder. En klass i systemet skulle kunna vara Customer för att systemet skall kunna skapa Customer-objekt som representerar bankens kunder.

En Customer kan beskrivas som ett objekt av typen Customer som har följande instansvariabler:

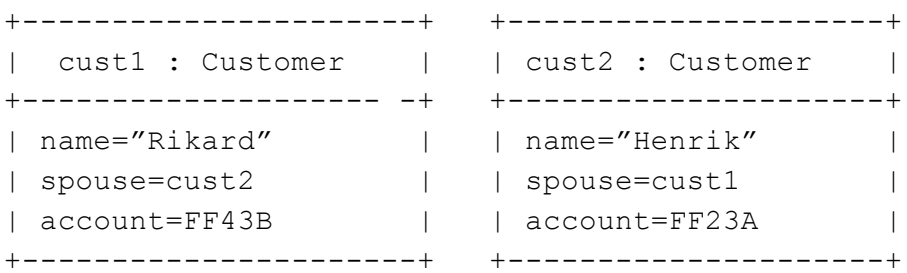
name : String
account : BankAccount
spouse : Customer

Wait, what? Kan ett Customer-objekt har en referens till ett annat Customer-objekt? Ja, varför inte? Om vi kan ha en instansvariabel av typen String för “name”, så kan vi ha en instansvariabel spouse av typen Customer.

Variabler av referenstyp (String, List, Object, HashMap, File osv) lagrar ju blott en referens (typ en adress) till ett objekt, eller specialvärdet null (betyder “refererar till inget objekt alls). Så variabeln name i ett Customer-objekt refererar till ett objekt av typen String. Adressen till objektet lagras i variabeln. Själva objektet finns i det så kallade heap-minnet i JVM när programmet körs.

Tänk er följande två instanser (färdiga objekt som skapats av applikationen när den körs):

cust1, cust2



På adress FF43B ligger ett namnlöst objekt av typen Accound och samma sak på adressen FF23A.

Rikards spouse är Henrik och vice versa.

Ovan är, by the way, UML för objekt! Inte klasser. Två namngivna objekt cust1 och cust2. Vi låtsas att konto-objekten för cust1 och cust2 har de angivna adresserna (man ser ju aldrig adresserna i Java så vi får låtsas). Poängen är att ett Customer-objekt kan ha en referens till ett annat Customer-objekt precis på samma sätt som det kan ha en referens till ett Konto-objekt eller ett String-objekt.