Et Udviklingseksempel

Styklister

Michael R. Hansen
Anne Haxthausen
mrh@imm.dtu.dk

Informatics and Mathematical Modelling
Technical University of Denmark

Oversigt

- Eksempel: Styklister
  - Uformel beskrivelse
  - Analyse
  - Design
  - Implementering (i Java)
  - Test
- Observationer
  - Klassediagrammer: bindeled mellem beskrivelse og program
  - Et lille, men ikke-trivielt, eksempel

Uformel beskrivelse: Styklister

I produktionsomgivelser bliver komplekse produkter samlet ud fra komponenter, som selv kan være simple dele eller komplekse produkter. Et lagerstyringssystem skal

- indeholde et katalog for simple dele,
- holde styr på hvordan produkterne stykkes sammen,
- kunne beregne prisen for et produkt, og
- meget andet.

Eksempel:

- Cykel
  - Stel
  - Hjul (2 styk)
    - Dæk
    - Slange
    - ...
  - ...

Analyse: væsentlige begreber

• komponent	component
• simpel del	part
• komplekst produkt	assembly
• katalogindgang	catalogue entry
• katalog	udelades her
• prisberegning	cost

En *begrebsliste*, bestående af begreber med tilhørende definitioner, er et vigtigt del-resultat af en analyse.

Kandidater kan fås via en markering af navneord og verber i teksten

- find de væsentligste begreber og gør dem så enkle som muligt — kompleksitet af begreber vokser med detaljeringsgraden.

Design

Skal give en oversigt over de væsentligste systemkomponenter, herunder deres

- tilstand,
- grænseflade, og
- indbyrdes sammenhænge.

To formål

- vise hvordan problemstillingens væsentlige begreber repræsenteres og behandles
- give et overblik over programmet abstraktion

UML diagrammer benyttes ofte til at beskrive design af objekt-orienterede programmer, f.eks. Java programmer.

Simpelt Klassediagram

Klassediagram for katalogindgange:

CatalogueEntry
-name: String -number: long -cost: double
+getName(): String +getNumber(): long +getCost(): double

Navn

Attributter

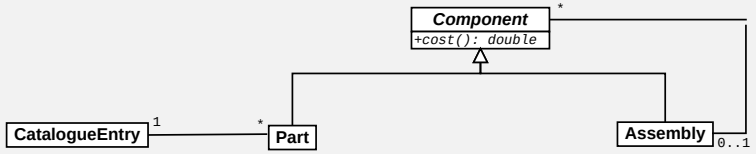
Operationer

Bemærkninger:

- '-'et ved attributterne angiver at de er *private*
- '+'et ved operationer angiver at de er *public*
- for et objekt, der er en instans af klassen, angiver
  - attributternes værdi objektets *tilstand*, og
  - operationerne objektets *grænseflade*

Klassers indbyrdes sammenhænge

- en simpel del er en komponent **is-a**, generalization
- et komplekst produkt er en komponent **is-a**, generalization
- en simpel del er tilknyttet en katalogindgang association
- et komplekst produkt er tilknyttet en liste af komponenter association



Component og *cost* er *abstrakte* størrelser (klasse og operation).

Såvel `Part` som `Assembly` er komponenter. En abstrakt klasse specificerer fælles egenskaber ved subklasserne, her en `cost` operation. En abstrakt klasse kan ikke instantieres.

Implementerings skelet

Implementeringens struktur kan direkte afledes af diagrammet

```
abstract class Component {
   public abstract double cost();
}

class Part extends Component {
   private CatalogueEntry entry;
   ...
}

class Assembly extends Component {
   private  Collection<Component> components = new ArrayList<Component>();
   ...
}
```

Diagrammet giver derfor et overblik over koden.

Implementering af Part

Prisen for en del findes i katalogindgangen for delen

```
class Part extends Component {
   private CatalogueEntry entry;

   public Part(CatalogueEntry ent)
   {
      entry = ent;
   }

   public double cost()
   {
      return entry.getCost();
   }
}
```


Implementering af Assembly

Prisen for et komplekst produkt findes vha. komponenternes pris:

```
class Assembly extends Component {

   private Collection<Component> components = new ArrayList<Component>();

   public void add(Component c)
   {  components.add(c); }

   public double cost()
   {
      double total = 0.0;
      for (Component c : components)
         total += c.cost();
      return total;
   }

}
```

- *Polymorfi* via den abstrakte klasse `Component`.
  Den rigtige `cost` operation vælges under kørslen.

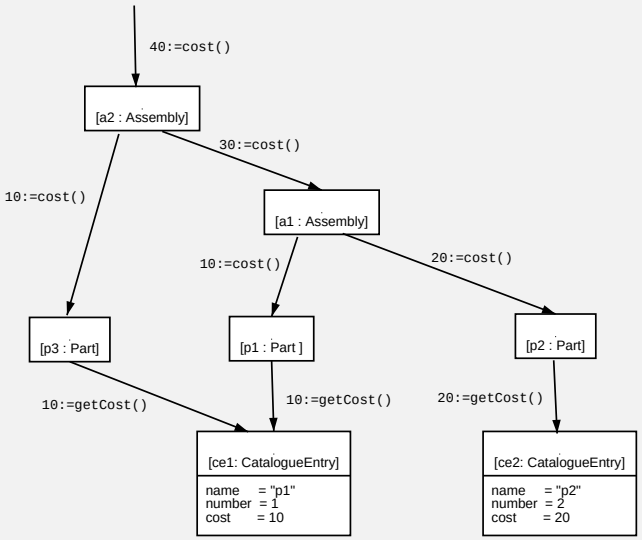
Eksempel på kørsel

```
CatalogueEntry ce1 = new CatalogueEntry("p1", 1, 10);
CatalogueEntry ce2 = new CatalogueEntry("p2", 2, 20);

Part p1 = new Part(ce1);
Part p2 = new Part(ce2);
Part p3 = new Part(ce1);

Assembly a1 = new Assembly();
Assembly a2 = new Assembly();

a1.add(p1);
a1.add(p2);

a2.add(p3);
a2.add(a1);

System.out.println(a1.cost());
System.out.println(a2.cost());
```

Hvad udskrives?

Objekt Model

En samling objekter, hver med en selvstændig tilstand.
Beregninger foregår ved at objekterne udveksler meddelelser.



Eksempel på JUnit test case

```
public class TestComputeCosts extends TestCase {
   CatalogueEntry stel, daek, slange;

   public void setUp() {
      stel = new CatalogueEntry("Stel",1,700);
      daek = new CatalogueEntry("Dk",2,100);
      slange = new CatalogueEntry("Slange",3,50);
   }

   public void testCostsPart() {
      Component stelPart = new Part(stel);
      assertEquals(700.0,stelPart.cost());
   }

   public void testCostsAssembly() {
      Component daekPart1 = new Part(daek);
      Component slange1 = new Part(slange);
      Assembly hjul1 = new Assembly();
      hjul1.add(daekPart1); hjul1.add(slange1);
      assertEquals(150.0,hjul1.cost());
   }
}
```


Opsummering

- Uformel beskrivelse
- Analyse
- Design
- Implementering i Java
- Test