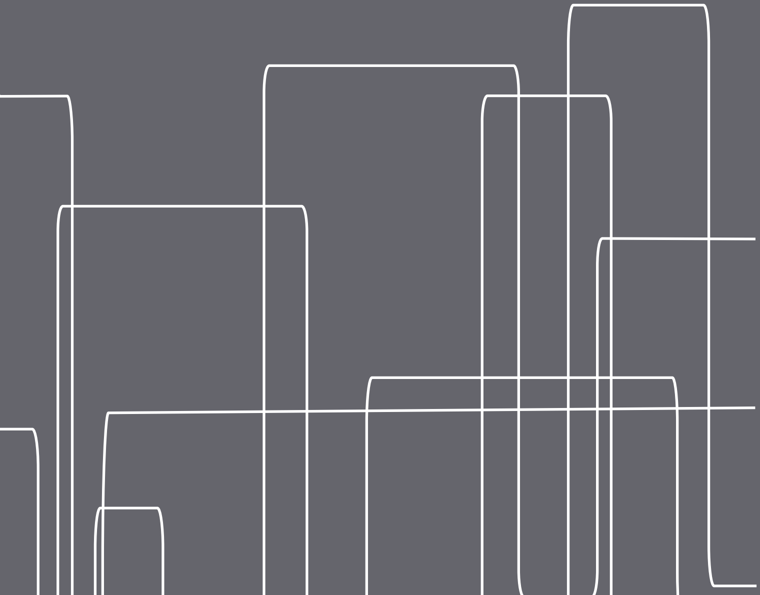
Objektorienterad Programkonstruktion

Föreläsning 9

30 nov 2016



Collections

- Ett samlingsnamn på objekt som innehåller en samling av andra objekt
- Det finns många olika sorters Collections, t.ex listor, träd, mängder, mm
- I Java finns gränssnittet `Collection`, som definierar ett antal metoder som är gemensamma, bl.a
  - `add(Element e)` - garanterar att elementet `e` finns (valfri)
  - `addAll(Collection c)` – garanterar att alla element i `c` finns (valfri)
  - `contains(Object o)` - returnerar `true` om o finns
  - `containsAll(Collection c)` - returnerar `true` om alla element i c finns
  - `clear()` - tömmer samlingen
  - `size()` - returnerar antalet element
  - `toArray()` - returnerar en `Array` med alla element i
  - `iterator()` - returnerar en `Iterator`

Collections

- Med Collection-gränsnittet behöver man inte veta hur den interna representationen ser ut, man kan ändå stoppa in och plocka ut element
- Sett åt andra hållet, om man själv skriver en klass som använder sig av gränsnittet, så kan man använda många inbyggda funktioner i Javabiblioteken, som tex **for each**

```
MyCollection collection = new MyCollection();
...
for (Object o : collection){
    System.out.println(o);
}
```

Exempel på Collections i Java

- **`HashSet`** - Tillåter ett exemplar av varje element, använder hashtabeller för att ge snabba operationer. Har ingen specifik ordning.
- **`ArrayList`** - Tillåter flera exemplar av varje element, använder array:er internt för att lagra element. Har en väldefinierad ordning.
- **`LinkedList`** - Tillåter flera exemplar av varje element, använder en dubbellänkad lista för intern representation. Har en väldefinierad ordning.
- **`TreeSet`** - Tillåter bara ett exemplar av varje element, använder ett sorterat träd för sin interna representation, kräver att de lagrade elementen är `Comparable`, och har en väldefinierad ordning

Map

- En samling av nyckel-element-par
- Kan liknas vid en funktion: om man anger nyckeln så får man tillbaka det tillhörande elementet
- Samma element kan förekomma flera gånger, men en viss nyckel får bara finnas en gång
- När man skapar en Map, eller när man lägger till fler element måste man alltid förse varje element med en tillhörande nyckel
- När man vill plocka ut ett element anger man nyckeln och får det tillhörande elementet
- Exempel:
  - Elementet är ett objekt som innehåller studieresultat,nyckeln är ett personnummer
  - Elementet är en förklarande text, nyckeln är ett ord (ordlista)

Map i Java

- `Map` är ett gränsnitt
- Implementerande klasser är t.ex: `HashMap`, `Hashtable`, `TreeMap`
- Det definierar bl.a följande metoder
  - `put(key, value)`
  - `get(key)`
  - `remove(key)`
  - `values()`
- `values()` returnerar en Collection med samma innehåll, som man alltså kan ta sig igenom med en Iterator:

```
for(Object o: myMap.values()){
    ....
}
```

Iterator (GoF)

- *Behaviorial Pattern*
- **Iteratorer** är objekt som tillhandahåller ett sätt att stega sig igenom element i en **Collection.**
- Ett sätt att möjliggöra algoritmer som verkar på datastrukturer utan att man vet hur strukturen ser ut.
- Tillåter multipla åtkomster av samma underliggande struktur, med separata minnen av hur långt man har kommit

Iterator i Java

- Gränssnittet Iterator definierar tre metoder:
  - hasNext() - returnerar true om det finns element kvar
  - next() - returnerar nästa element
  - remove() - plockar bort senast returnerat element (frivillig)
- Om en klass tillhandahåller en metod `Iterator()` som returnerar ett iteratorobjekt över klassen, kan den implementera gränssnittet `Iterable`, och användas i en for-each-loop
- `Collection` är ett exempel som ärver från `Iterable`
- `Scanner` är ett exempel på en `Iterator` över ett `String`-objekt

Composite (GoF)

- *Structural Pattern*
- Låter en behandla grupper av objekt på samma sätt som ett enskilt objekt
- Passar när man har en trädstruktur och vill göra exakt samma sak med en nod och alla dess barn
- Definiera en abstrakt **Component**-klass som rymmer alla de metoder man vill kunna anropa, en struktur för att rymma barn-noder, samt metoder för att lägga till barn och komma åt dessa
- Nu kan man definiera en **Leaf**-klass och en **Composite-**klass som ärver från **Component**
- Ett program som arbetar med dessa strukturer behandlar alla delar som om de vore **Component**-objekt, och ett anrop av någon metod utförs för samtliga **Leaf**-objekt nedanför i hierarkin

Composite (GoF)

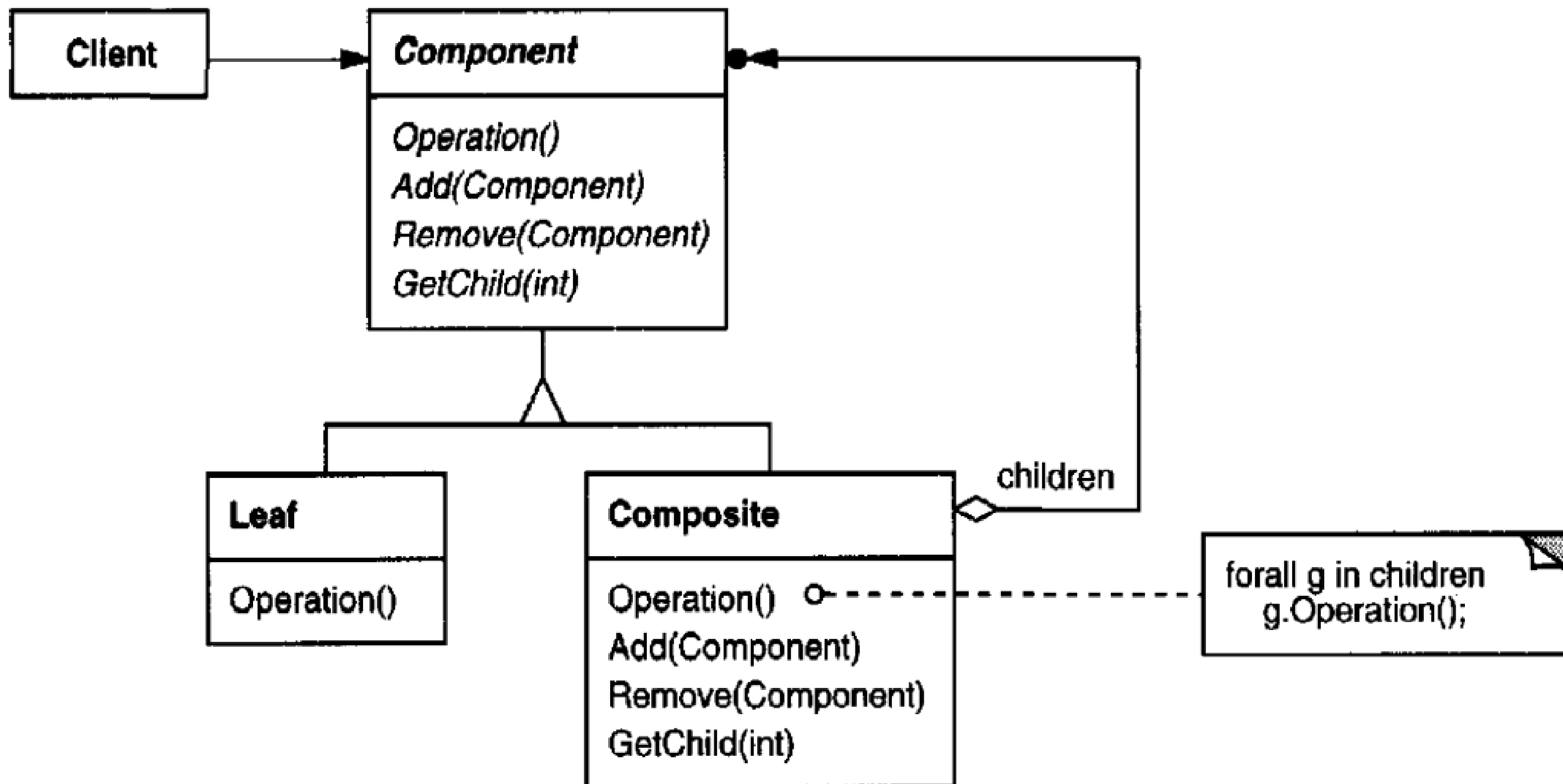
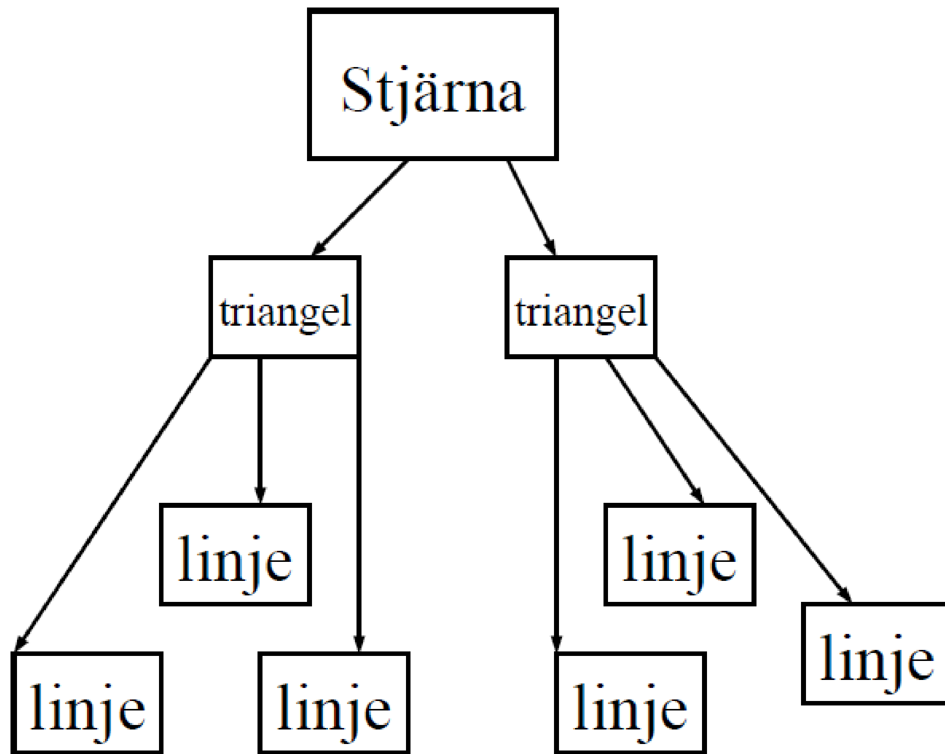


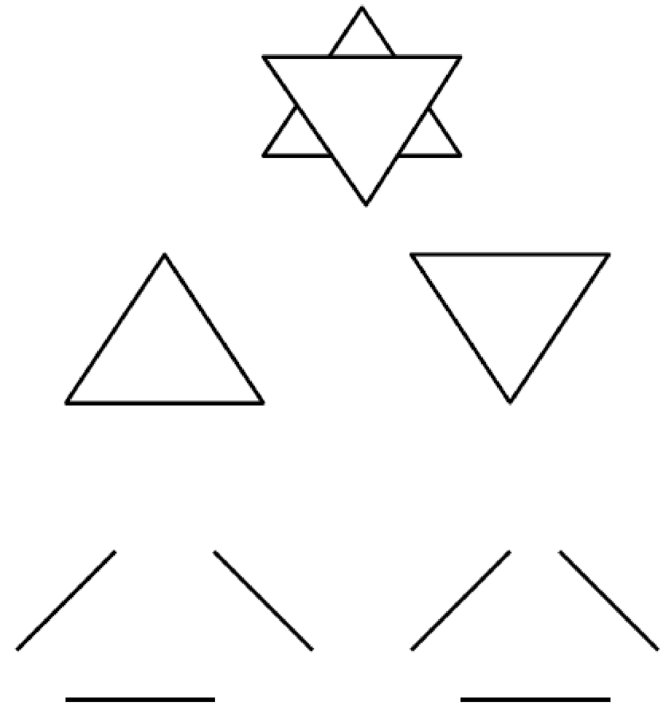
bild fr. "Design Patterns" av GoF

Composite-exempel

- I grafik kan varje bild bestå av linjer
- Dessa linjer blir till olika polygoner
- Polygonerna blir till avancerade bilder
- Man vill kunna flytta en bild, då anropas flytta-metoden I alla polygoner den består av, som i sin tur anropar flytta-metoden i sina beståndsdelar linjerna.
- På samma sätt vill man kunna byta färg på en enskild linje, en enskild polygon, en grupp av polygoner eller hela föremålet

Composite-exempel





Composite-exempel

- Man kan representera filsystem som **Composite**
- Katalogerna är Composite-objekt, och filerna blir Leaf-objekt
- Det blir då enkelt att genomföra samma operation på alla underkataloger och filer, tex
  - Byta ägare eller åtkomst
  - Kopiera
  - Beräkna storlek