2I1049 Föreläsning 5

Objektorienterad programmering i Java

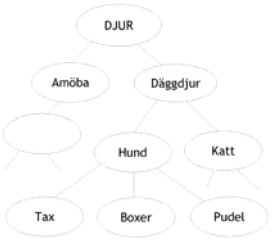
KTH-MI Peter Mozelius

Objektorientering

- Världar uppbyggda av objekt
- Inte helt olikt vår egen värld
- Ett sätt att modularisera våra system
- Objekten delas in i klasser
- Klasserna kan ärva varandra
- En klass utökar/specialiserar en annan
- Ett antal klasser bildar en taxonomi

Objektorientering

En liten taxonomi



Klasserna ordnas i en hierarki som motsvarar deras inbördes ordning

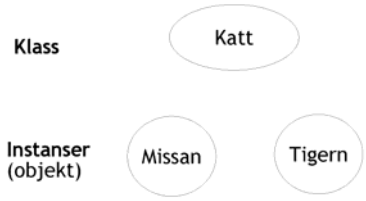
Objektorientering

- En klass innehåller en specifikation som definierar beteenden och egenskaper hos de klassinstanser (objekt) som sedan skapas (instansieras) av klassen
- Av en klass Tax kan man sedan skapa en mängd olika taxar. I Java görs detta enligt:

```
Tax t1 = new Tax();
```

Objektorientering

Klasser och klassinstanser



Objektorientering i Java

- En enkel klass med en konstruktor:

```
public class Katt {
  private String namn;
  public Katt(String s){
    namn = s;
  }//konstruktor
}//klassen Katt
```

Objektorientering i Java

- I en annan klass kan vi sedan instansiera ett valfritt antal kattobjekt:

```
public class KattProgram {
   public static void main(String[] arg){
     Katt k1 = new Katt(”Jameson”);
     Katt k2 = new Katt(”Schrödinger”);
   }//main
}//KattProgram
```

Är det något som saknas i klassen Katt ?

Inkapsling

- Att klasserna är enheter som innehåller både attribut och de operationer som bearbetar klassens attribut
- I Java innebär detta att klassens variabler ska hanteras av de metoder som finns definierade i samma klass
- Ett designmönster (design pattern) i UML:

High cohesion - Low coupling

Datagömning

- Att utöka idén om inkapsling och aktivt hindra andra klassers metoder från att komma åt klassens data
- Ger bättre säkerhet
- Minskar risken för namnkonflikter
- För att systemutvecklaren ska kunna välja olika grader av datagömning så finns det i Java olika **modifierare**

Javas modifierare

- För klasser, variabler och metoder finns det reserverade ord för åtkomstkontroll i Java
- De fyra modifierare som finns är:
  private
  protected
  public
  och om inte annat anges
  package/paketåtkomst

Datagömning i Java

- Data deklareras så *snålt* som möjligt :

```
private int x;
```

- Och åtkomsten sköts sedan med hjälp av åtkomstmetoder enligt:

```
public int getX(){
   return x;
}
```

Datagömning i Java

- På samma sätt bygger man även metoder som kan ändra värdet på instansvariabler:

```
public void setX(int i){
   x = i;
}
```

- **Accessmetoder**
- Inspektorer – Mutatorer

Rast 15 min!

Överlagring

- Metoder och konstruktorer i en klass med samma namn MEN olika parameterlistor
- Exempel med konstruktorer:

```
public Hund() {}
public Hund(String namn) {}
public Hund(String namn, String ras) {}
public Hund(String namn, String ras,
                    boolean biterFolk) {}
```

Polymorfism

- **polymorfism** = mångformighet
- I en klasshierarki kan det i de olika klasserna finnas metoder med samma namn och samma argument/returtyp MEN med olika metodkroppar
- Objektorienterade språk har inbyggda mekanismer för **dynamisk bindning**
- Rätt metod körs automatiskt när programmet exekveras

Polymorfism

I en abstrakt basklass:

```
public abstract class Figur
```

Finns det en abstrakt metod:

```
public abstract double visaArea();
```

I den ärvande klassen Rektangel:

```
public class Rektangel extends Figur
```

har metoden formats så att den passar för att räkna ut arean hos just en rektangel

Polymorfism

I klassen Rektangel:

```
public double visaArea() {
   return bredd * höjd;
}//visaArea i klassen Rektangel
```

Medan den i klassen Cirkel **överskuggas** enligt:

```
public double visaArea() {
   return PI * radie * radie;
}//visaArea i klassen Cirkel
```

Överskuggning

- När en eller flera av superklassens instans-metoder omdefinieras i en eller flera subklasser
- Vid exekveringen så är det objektets klass som avgör vilken metod som anropas
- Olika objekt i en klasshierarki kan på detta sätt behandlas enhetligt utan att varje enskilt objekt måste klassbestämmas
- Att rätt metod automatiskt anropas under programkörningen kallas **dynamisk bindning**

Dynamisk bindning i Java

- Om en instansmetod *metod* anropas via referensen *ref* enligt:

  `ref.metod();`
- Så undersöker javainterpretatorn vilken klass objektet har som ref refererar till
- Om objektets klass har en passande metod() så körs denna
- Om inte, så letar javatolken vidare uppåt bland superklasserna tills en metod återfinns

Några reserverade ord

Följande reserverade ord är bra att känna till:

- **this** syftar på den egna klassen
- **super** syftar på basklassen
- **abstract** förhindrar instansiering
- **final** förhindrar vidare arv
- **static** markerar klasstillhörighet (inte instanstillhörighet)

Klassvariabler

- De flesta attribut är instansvariabler
- Alla objekt får då egna variabler
- Det finns dock undantagsfall då objekten i en klass behöver ha gemensamma klassvariabler
- I Java markeras detta med ordet **static** och en klassvariabel kan deklareras enligt:

```
private static int klassVariabel;
```

Klassvariabler

- En vanlig användning är när klassen behöver en räknare:

```
public class Katt {
  private static int kattNummer;
  public Katt(){
    ++kattNummer;
  }
}
```

Klassen Object

- Alla klasser i Java har en gemensam grundläggande basklass: `java.lang.Object`
- De metoder som finns i klassen Object ärvs därför av samtliga klasser och kan överskuggas
- Två metoder från klassen Object som ofta överskuggas är:

```
public String toString()
public boolean equals(Object obj)
```

toString()

- Bör överskuggas så att lämplig information ges vid utskrift av klassens instanser
- I en klass för att representera punkter:

```
public class Punkt {
  private int x;
  private int y;
```

så kan toString() överskuggas enligt:

toString()

```
public String toString() {
  return ”[” + x + ”,” + y + ”]”;
}
```

En utskrift av:

```
Punkt punkt = new Punkt(3, 4);
System.out.println(punkt);
```

ger utskriften: `[3,4]`

Klasspecifikationer

- Specifikationer av Javas färdiga klasser:
  http://java.sun.com/j2se/1.5.0/docs/api/index.html
- **API** = Application Programmable Interface
- Två andra länkar med information om Java:
  http://www.javasoft.com
  http://www.javaworld.com

Hemsida

- Laborationsdelens hemsida:
  http://www.dsv.su.se/~mio

Tack för idag!!

För er som vill så visar jag gärna:

ArgoUML: http://argouml.tigris.org/