ex1classes

Joachim von Hacht

Instansmetoder

```
class Player {
   int points;  // Instance variable

    // Declare instance method
   void incPoints(){
      points++;
   }
}

// Later elsewhere, create object
Player p = new Player();
p.incPoints();  // Call instance metod
```

2

Klasser kan innehålla metoder!

Instansmetoder är metoder som är deklarerade i en klass.

- De kan anropas på alla objekt skapade utifrån klassen
- Det måste finnas ett objekt för att kunna anropa metoderna.
  - Punktoperatorn används.
- I vilken ordning instansmetoder deklareras spelar ingen roll.
- Alla deklarationer ligger på samma nivå i klassdeklarationen (ej nästlade)

Mer om Instansvariabler

```
public class Person {
   private final String name;
   private int age;
   private double income;

   Person(String name, int age, double income) {
       this.name = name;
       this.age = age;
       this.income = income;
   }

   int getAge() {
       return age;
   }

   void setAge(int age) {
       this.age = age;
   }

      // Objects has the data to be able to answer questions!
   boolean isRetired(int retireAge) {
       return age >= retireAge;
   }
  ....
```

Synlighets område

3

Instansvariabler deklareras i klassen men utanför all metoder (vi skriver dem ofta överst i klassen)

- Synlighetsområdet för instansvariabler är hela klassen
  - I alla metoder!!
- Lokala variabler kan ha samma namn som en instansvariabel men isf döljs instansvariabeln av den lokala variabeln
  - Kan komma åt med this (som vi sett).
- Ett problem med instansvariabler är om något blir fel, vilken metod orsakade felet??
  - Undvik instansvariabler om det går, föredra lokala variabler.
  - Oftast behövs dock instansvariabler. Så fort ett objekt måste ihåg något mellan metodanrop måste vi använda instansvariabler.
- Alla instansvariabler kommer att initieras med förvalda värden, t.ex. 0 för int.

Varför Metoder i Klasser?

```
// Not so good...
boolean isWinner(Player p, int winPoints) {
   // ... must extract data from player, then use it.
   return p.points >= winPoints;
}


// Better put method in object (class)
class Player {
   String name;
   int points;

    boolean isWinner(int winPoints) {
       return points >= winPoints;  // Use data in class
   }
}
```

4

Varför instansmetoder?

- Vi vill samla instansvariabler och metoder som använder dessa på samma ställe.
- Eftersom variablerna (datan) finns i en klass lägger vi metoderna i klassen.
- Vi behöver inte plocka ut data ur objektet, metoderna kan använda datan som finns i objektet.
    - Färre parametrar behövs för metoderna.
- Objekten hanterar själva sin data. Vi vet var saker sker.

Kedjade Metodanrop

```
class Counter {
   int count;

   // Method in class but not chainable
   void inc() {
       count++;
   }

   // Chainable method return this
   Counter incC() {
       count++;
       return this;   // Return this!
   }
}

Counter c = new Counter();
c.incC().incC().incC();  // Chained calls
```


Programmeringsteknik som kan ge smidigare kod.

- Låt instansmetoder returnera this.

Generiska Klasser

```
// A generic class/type
class Box<T> {       // T is a type variable
  T toRemember;      // T is any reference type
}

// Must say what T should be (here Integer)
Box<Integer> m1 = new Box<>();
// T is of type String
Box<String> m2 = new Box<>();

m1.toRemember = 4;            // Ok (boxing)
//m1.toRemember = "Hej";      // Bad type

//m2.toRemember = 56;          // Bad type
m2.toRemember = "Abraxas";
```

I Java kan man skapa **generiska klasser** d.v.s. klasser som använder en typvariabel som typ för någon instansvariabel.

- Typevariabeln kallas ofta T eller E etc. (en stor bokstav)
- Anges inom vinkelparenteser (som generiska metoder)

Då man skall deklarera en variabel för en generisk typ måste man ange någon existerande typ för typvariabeln

- Anges inom vinkelparenteser vid deklarationen.
- Måste vara en referenstyp

Mer om detta i senare kurser, här bara som en bakgrund.