Föreläsning 5

Mer om klasser och objekt

Klassvariabler och klassmetoder

En klass kan innehålla klassvariabler och klassmetoder.

Det som skiljer en klassvariabel från en instansvariabel och en klassmetod från en instansmetod är att:

- klassvariabler och klassmetoder är *gemensamma för alla objekt som tillhör en klass*
- instansvariabler och instansmetoder finns i *unika uppsättningar i varje objekt som tillhör en klass.*

Detta betyder att:

- metoder som påverkar någon instansvariabel eller behöver tillgång till någon instansvariabel för att kunna fullgöra sin uppgift, måste implementeras som instansmetoder
- metoder *inte* påverkar eller behöver tillgång till någon instansvariabel bör implementeras som klassmetoder.

Klassvariabler och klassmetoder

Exempel:

Antag att vi har en klass `Person` för att avbilda personer. Antag vidare att vi i ett program skapat instanserna `kalle` och `sture` av klassen `Person`

```
Person kalle = new Person("Kalle");
Person sture = new Person("Sture");
```

I klassen `Person` finns vidare metoderna `getVikt()` och `getVikt(Person p)` som t.ex kan anropas på följande sätt:

```
double vikt1 = kalle.getVikt();
double vikt2 = kalle.getVikt(sture);
```

som levererar svaren på frågorna:

Kalle, vad väger du?

respektive

Kalle, hur mycket väger Sture?

Metoden `getVikt()` måste vara en instansmetod, medan metoden `getVikt(Person p)` bör vara en klassmetod. **Varför?**

Klassvariabler och klassmetoder

Modifieraren **static** används för att ange att en variabel är en klassvariabel respektive att en metod är en klassmetod:

```
private static double rate;
public static final double G = 6.6743e-11;

public static double getVikt(Person p) {
    . . .
}//getVikt

public static void main(String[] arg) {
    . . .
}//main
```

De enda publika klassattribut som bör finnas är konstanter.

Klassvariabler och klassmetoder

Access till publika klassattribut fås genom att ange klassens namn, följt av en punkt ‘.’, följt av attributets namn

```
Math.PI
Interger.MAX_VALUE
```

Access till publika klassmetoder fås genom att ange klassens namn, följt av en punkt ‘.’, följt av metodens namn och argumentlista.

```
String indata = JOptionPane.showInputDialog("Ge talet:");
double tal1 = Double.parseDouble(indata);
double tal2 = Math.pow(tal1, 3);
System.out.println("Det beräknade talet blev: " + tal2);
```

Har man skapat objekt av en klass kan klassvariabler och klassmetoder även refereras via dessa objekt.

Exempel: Klassvariabler och klassmetoder

Skapa en klass `Account` för att avbilda bankkonton.

Ett konto skall innehålla uppgifter om

- vem som äger kontot
- kontonummer
- saldo
- räntesats

På ett konto skall det vara möjligt att

- göra transaktioner (insättningar eller uttag)
- ta reda på räntesatsen
- ta reda på saldo
- ta reda på kontots ägare
- ta reda på kontots kontonummer
- ändra räntesats
- lägga till årsräntan

Alla konton skall ha samma räntesats.

Alla konton skall ha unika kontonummer, vilket är ett löpnummer i den ordning konton skapas.

Implementation av klassen `Account`

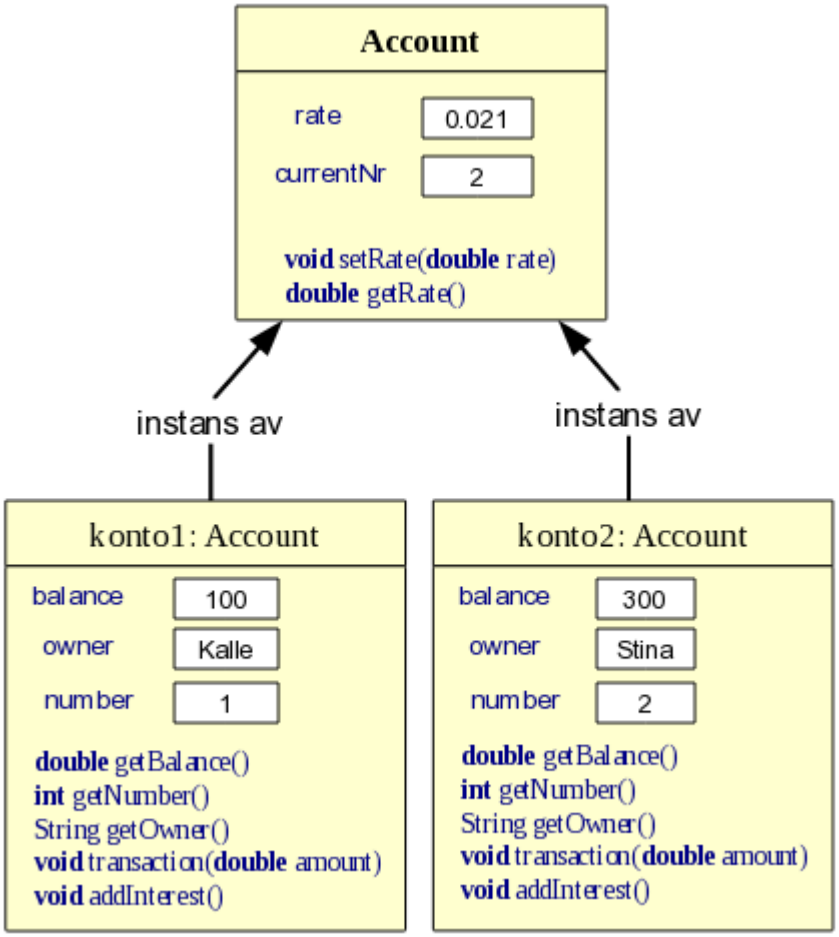
```
public class Account {
   private static double rate;            //klassvariabel
   private static int currentNr = 0;      //klassvariabel
   private double balance;                //instansvariabel
   private String owner;                  //instansvariabel
   private int number;                    //instansvariabel
   public Account(String owner) {
      this.owner = owner;
      currentNr = currentNr + 1;
      number = currentNr;
   } //konstruktor
   public static void setRate(double newRate) {
      rate = newRate;
   } //setRate
   public static double getRate() {
      return rate;
   } //getRate
   public double getBalance() {
      return balance;
   } //getBalance
   public int getNumber() {
      return number;
   } //getNumber
```

```
   public String getOwner() {
      return owner;
   } //getOwner
   public void transaction(double amount) {
      if (amount < 0 && balance + amount < 0)
         System.out.println("Not money enought!!");
      else
         balance = balance + amount;
   } //transaction
   public void addInterest(double interest) {
      balance = balance + balance*interest;
   }//addInterest
} //Account
```

Klassvariabler och klassmetoder

Antag att följande satser har gjorts:

```
Account konto1 = new Account("Kalle");
Account konto2 = new Account("Stina");
Account.setRate(0.024);
konto1.transaction(100);
konto1.transaction(300);
konto2.setRate(0.021);
```



Vidstående bild illustrerar vilka objekt som finns samt vilka värden respektive instansvariabel och klassvariabel har.

En klassmetod får inte referera till någon instansvariabler! Varför?

Problemexempel

Uppgift

Utveckla en klass för att avbilda tärningar.

Frågor att ta ställning till:

Vad skall vi kalla klassen?

- vi döper klassen till `Dice`

Vilket beteende har en tärning (= vad vill man kunna göra med en tärning)?

- slå tärningen
- läsa av värdet på tärningen

Vilka tillstånd kan en tärning ha?

- värdet, dvs vilken siffra har ovansidan på tärningen

Specifikation av klassen `Dice`

För att handha tärningens värde behövs en instansvariabel, som vi kan kalla `dots`. Eftersom värdet är ett heltal skall `dots` vara av typen **int**:

```
private int dots;
```

För att kasta tärningen behövs en instansmetod som slumpmässigt ändrar tärningens värde, låt oss kalla denna metod för `roll`:

public void roll()	kastar tärningen

För att läsa av tärningens värde behövs en instansmetod som returnerar värdet av instansvariabeln `dots`:

public int getDots()	returnerar tärningens värde

Vi väljer att låta klassen ha två konstruktorer. Den första konstruktorn sätter ett slumpmässigt startvärde (mellan 1 och 6) på tärningen. Den andra konstruktorn tillåter användaren att initiera tärningens startvärde. Om detta värde inte är mellan 1 och 6, sätts ett slumpmässigt startvärde.

public Dice()	skapar en tärning med en slumpmässig sida upp
public Dice(**int** dots)	skapar en tärning med sidan `dots` upp

Metoden `roll` använder ett objekt av klassen `Random` för att erhålla ett slumpmässigt värde. Detta objekt görs till en klassvariabel.

```
private static Random diceRandom = new Random();
```

Implementering av klassen `Dice`

```
import java.util.*;
public class Dice {
   private int dots;
   private static Random diceRandom = new Random();
   public Dice() {
      roll();
   }// konstruktor
   public Dice(int dots) {
      if (dots > 6 || dots < 1)
         roll();
      else
         this.dots = dots;
   } //konstruktor
   public int getDots() {
      return dots;
   } //getDots
   public void roll() {
      dots = diceRandom.nextInt(6) + 1;
   } //roll
} //Dice
```

Varför har `diceRandom` implementerats som en klassvariabel?

Varför finns det ingen metod `setDots` för att sätta värdet på instansvariabeln `dots`?

Problemexempel

Använd klassen `Dice` för att skriv ett program som genom simulering beräknar sannolikheten för att erhålla den sammanlagda summan 7 när man kastar två tärningar.

Analys:

Diskussion: För att beräkna sannolikheten görs ett tillräckligt stort antal kast med tärningarna. Vi räknar hur många gånger de båda tärningarnas sammanlagda värde blir 7. Detta antal dividerat med totala antalet tärningskast ger sannolikheten för att erhålla värdet 7.

Utdata: Sannolikheten att erhålla värdet 7.

Design:

Algoritm:

1. Sätt *nrOfSeven* till 0.
2. Upprepa *nrOfTurns* gånger
   - 2.1. Kasta tärning *t1*.
   - 2.2. Kasta tärning *t2*.
   - 2.2. Läs av tärningarnas värden och lagra den sammanlagda summan i *sum*.
   - 2.3. Om *sum* har värdet 7 så

     öka *nrOfSeven* med 1.
3. Skriv ut värdet av *nrOfSeven / nfOfTurns*

Datarepresentation:

Tärningarna *t1* och *t2* är av klassen *Dice*.

nrOfSeven, *nrOfTurns* och *sum* är av typen **int**.

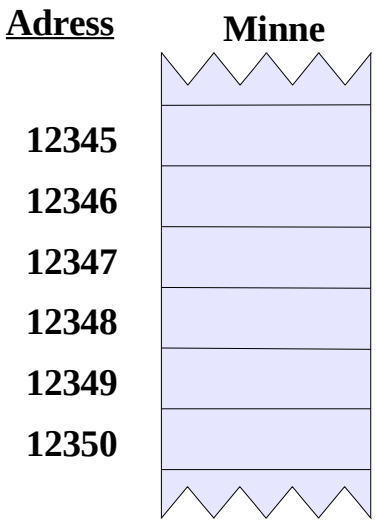
Implementation:

```
// Programmet beräknar genom simulering sannolikheten för att erhålla
// värdet 7 vid kast med två tärningar
public class TestTwoDice {
   public static void main (String[] args) {
      Dice t1 = new Dice();
      Dice t2 = new Dice();
      final int nrOfTurns = 100000;
      int sum;
      int nrOfSeven = 0;
      int i = 0;
      while (i < nrOfTurns) {
         t1.roll();
         t2.roll();
         sum = t1.getDots() + t2.getDots();
         if (sum == 7)
            nrOfSeven = nrOfSeven + 1;
         i = i + 1;
      }
      System.out.println("Sannolikheten att få värdet 7 är " + ((double) nrOfSeven) / nrOfTurns);
   } // main
} //TestTwoDice
```

Enkla variabler kontra referensvariabel

Datorns minne är uppdelat i ett antal dataord, där varje dataord är associerat med en specifik adress, enligt bilden bredvid.

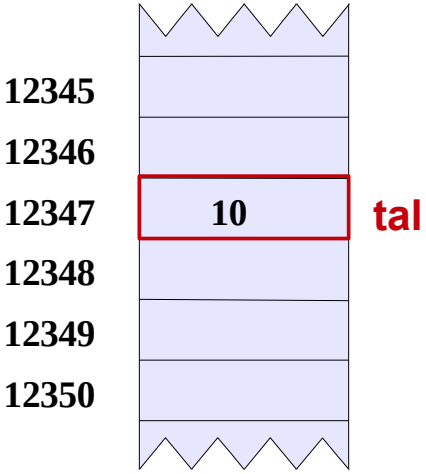
En variabel är ett *namngivet* minnesutrymme i datorns primärminne i vilket information lagras i form av bits.



Resultatet av deklarationssatsen

```
int tal = 10;
```

som innebär att en heltalsvariabel `tal` deklareras och initieras till värdet 10, kan illustreras med vidstående bild:



Enkla variabler kontra referensvariabel

En *referensvariabel* är en variabel i vilken en adress till datorns minne lagras. En referensvariabel *refererar* således till en annan plats i datorns minne (därav namnet referensvariabel).

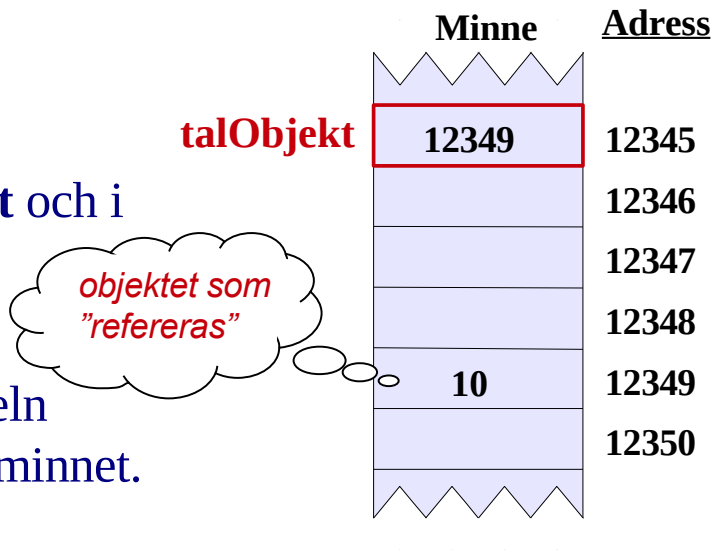
Referensvariabler används när objekt skapas dynamiskt med operatorn **new**.

Som exempel kan vi skapa ett objekt av klassen `Integer` med hjälp av deklarationen

```
Integer talObjekt = new Integer(10);
```

Ett objekt av klassen `Integer` har endast en instansvariabel, typen på instansvariabeln är **int** och i vårt exempel blir dess värde 10.

Det objekt som vi skapat kommer att lagras någonstans i datorns minne och referensvariabeln `talObjekt` tilldelas adressen till denna plats i minnet. Det hela kan illustreras med vidstående figur.



Omslagsklassen `Integer`

Constructor Summary

	Integer(int value) Constructs a newly allocated Integer object that represents the primitive int argument.
	Integer(String s) Constructs a newly allocated Integer object that represents the value represented by the string.

Method Summary

byte	**byteValue**() Returns the value of this Integer as a byte.
int	**compareTo**(Integer anotherInteger) Compares two Integers numerically.
int	**compareTo**(Object o) Compares this Integer to another Object.
static Integer	**decode**(String nm) Decodes a String into an Integer.
double	**doubleValue**() Returns the value of this Integer as a double.
boolean	**equals**(Object obj) Compares this object to the specified object.
float	**floatValue**() Returns the value of this Integer as a float.
static Integer	**getInteger**(String nm) Determines the integer value of the system property with the specified name.
static Integer	**getInteger**(String nm, int val) Determines the integer value of the system property with the specified name.
static Integer	**getInteger**(String nm, Integer val) Returns the integer value of the system property with the specified name.
int	**hashCode**() Returns a hashcode for this Integer.

Omslagsklassen `Integer`

Internet Lookup New&Cool Netcaster

int	**intValue**() Returns the value of this Integer as an int.
long	**longValue**() Returns the value of this Integer as a long.
static int	**parseInt**(String s) Parses the string argument as a signed decimal integer.
static int	**parseInt**(String s, int radix) Parses the string argument as a signed integer in the radix specified by the second argument.
short	**shortValue**() Returns the value of this Integer as a short.
static String	**toBinaryString**(int i) Creates a string representation of the integer argument as an unsigned integer in base 2.
static String	**toHexString**(int i) Creates a string representation of the integer argument as an unsigned integer in base 16.
static String	**toOctalString**(int i) Creates a string representation of the integer argument as an unsigned integer in base 8.
String	**toString**() Returns a String object representing this Integer's value.
static String	**toString**(int i) Returns a new String object representing the specified integer.
static String	**toString**(int i, int radix) Creates a string representation of the first argument in the radix specified by the second argument.
static Integer	**valueOf**(String s) Returns a new Integer object initialized to the value of the specified String.
static Integer	**valueOf**(String s, int radix) Returns a new Integer object initialized to the value of the specified String.

Methods inherited from class java.lang.Object

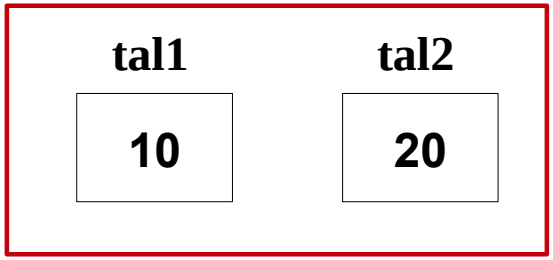
clone, finalize, getClass, notify, notifyAll, wait, wait, wait

Skillnaden mellan en variabel och en referensvariabel

Om vi gör nedanstående deklarationer

```
int tal1 = 10;
int tal2 = 20;
```

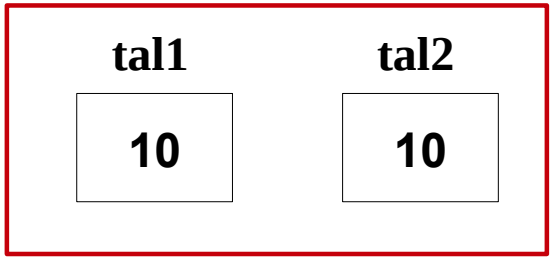
kan detta illustreras med följande bild



Görs sedan satsen

```
tal2 = tal1;
```

får vi bilden

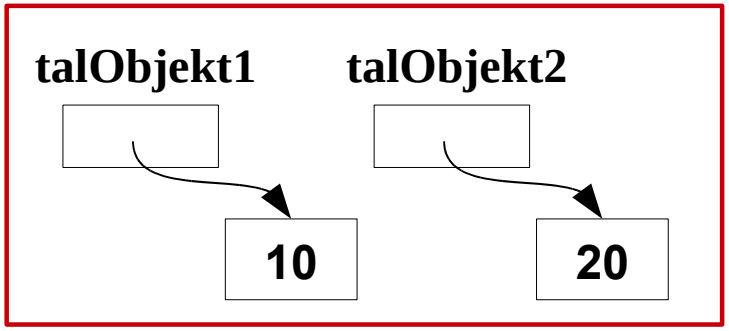


Skillnaden mellan en variabel och en referensvariabel

Om vi gör nedanstående deklarationer

```
Integer talObjekt1 = new Integer(10);
Integer talObjekt2 = new Integer(20);
```

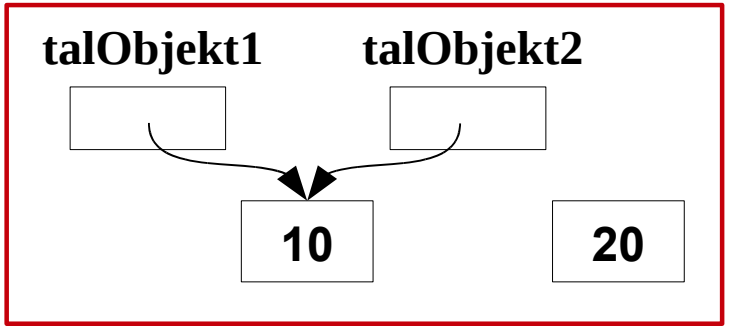
Kan detta illustreras med följande bild:



Görs sedan satsen

```
talObjekt2 = talObjekt1;
```

får vi bilden:



Ett litet testprogram

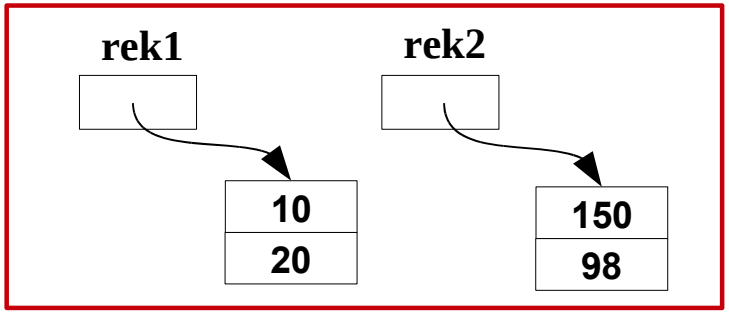
```
import javax.swing.*;
public class TestRectangle {
    public static void main (String[] args) {
        Rectangle rek1 = new Rectangle(10, 20);
        Rectangle rek2 = new Rectangle(150, 98);
        rek2 = rek1;
        rek1.setWidth(50);
        rek2.setHeight(200);
        JOptionPane.showMessageDialog(null, "Rek1:\n" + rek1 +
                                                "\nRek2: \n" + rek2);

    }//main
}//TestRectangle
```

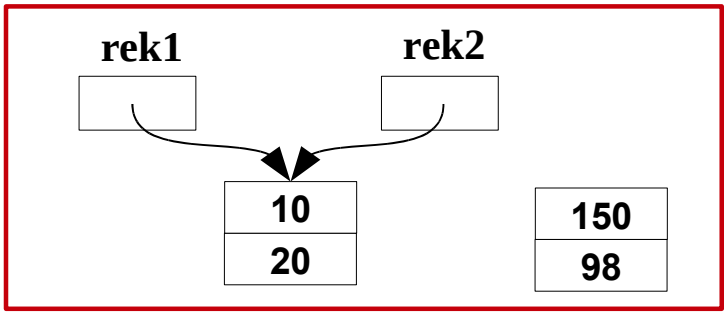
Vad blir utskriften?

Förklaring

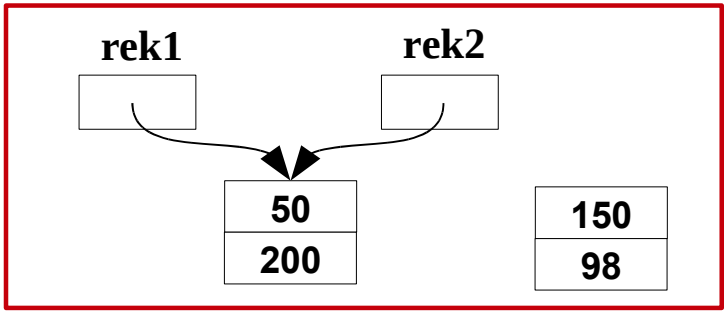
```
Rectangle rek1 = new Rectalgle(10, 20);
Rectangle rek2 = new Rectangle(150, 98);
```




```
rek2 = rek1;
```




```
rek1.setWidth(50);
rek2.setHeight(200);
```

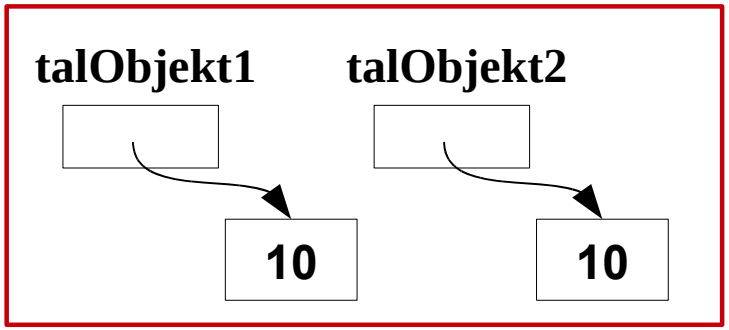


När är två objekt lika?

Om vi gör följande deklarationer

```
Integer talObjekt1 = new Integer(10);
Integer talObjekt2 = new Integer(10);
```

kan detta illustreras med följande bild:



Vad kommer jämförelsen

```
(talObjekt1 == talObjekt2)
```

att få för värde?

Resultatet blir **false**, eftersom det är *referenserna* till objekten som jämförs!!

Operatorn '==' jämför på *identitet*, inte på likhet!

När är två objekt lika?

För att jämföra likhet på själva objekten tillhandahåller klassen `Integer` instansmetoden `equals`. Anropet

```
talObjekt1.equals(talObjekt2)
```

kommer att returnera värdet **true**.

Vill man kunna jämföra två objekt av en viss klass måste klassen tillhandahålla instansmetoden `equals`.

Uppgift:

Utöka klassen `Rectangle` med instansmetoden `equals` som jämför om två objekt är lika! Hur skall likhet definieras?

Implementation av `equals` i `Rectangle`

```
public class Rectangle {
   private double width;
   private double height;
   . . .
   public boolean equals(Rectangle other) {
      if (this.width == other.width && this.height == other.height)
         return true;
      else
         return false;
   }//equals
}//Rectangle
```

Autoboxing och auto-unboxing

Typomvandling sker automatiskt mellan primära datatyper och motsvarande omslagsklass. Detta kallas för *autoboxing* respektive *auto-unboxing*.

Istället för att skriva

```
Integer talObjekt = new Integer(10);
. . .
int tal = talObjekt.toValue();
```

kan man skriva

```
Integer talObjekt = 10;        //autoboxing
. . .
int tal = talObjekt;           //auto-unboxing
```