JAVAUTVECKLING – LEKTION 11

Mahmud Al Hakim
mahmud.al.hakim@nackademin.se
www.alhakim.se

AGENDA

Mer om klasser och objekt

Statiska/instans-metoder

Överlagrade metoder

Inkapsling

Konstruktorer

UPPGIFT 13.6

Deklarera en klass **Kort** som kan användas för att beskriva kort i en vanlig kortlek.

Använd heltalen 0,1,2,3 för att presentera de fyra kortfärgerna, klöver, ruter, hjärter resp. spader. Använd konstanter så att man slipper komma ihåg vilket nummer som betyder vilken färg.

Skriv sedan ett program som skapar två kort.

Initiera korten så att de innehåller spader dam och klöver 7.

UPPGIFT 13.6 – KLASSEN KORT

```
public class Kort {
  // klassvariabler
  public static final int KLÖVER  = 0;
  public static final int RUTER   = 1;
  public static final int HJÄRTER = 2;
  public static final int SPADER  = 3;
  // instansvariabler
  int färg;
  int valör;
}
```

UPPGIFT 13.6 –KORT , TESTPROGRAM

```
public class KortTest {
  public static void main(String[] arg) {
    Kort k1 = new Kort();
    Kort k2 = new Kort();
    k1.färg = Kort.SPADER;
    k1.valör = 12;
    k2.färg = Kort.KLÖVER;
    k2.valör = 7;
  }
}
```

INSTANSMETODER

modifieare resultattyp namn (typ1 p1, typ2 p2,…)

{

  lokala deklarationer och satser

}

p1, p2, … är parametrar.

Modifierare kan vara public, private eller protected.

OBS! ordet static får inte vara med.

ANROP AV INSTANSMETODER

Objekt.metodnamn(a1, a2, … an)

Objekt är referens till ett objekt.

a1, a2, … an är argument. Deras typer bör överrensstämma med motsvarande parametrar.

ÖVNING 15.1

Deklarera en klass som beskriver rektanglar.

Låt klassen innehålla instansvariabler som beskriver en rektangels startpunkt (dess övre vänstra hörn) samt dess höjd och bredd.

Klassen skall också innehålla instansmetoder som sätter höjden, sätter bredden, beräknar arean och beräknar omkretsen.

ÖVNING 15.1
KLASSEN REKTANGEL – INSTANSVARIABLER

```
public class Rektangel {
  // instansvariabler
  double x, y;     // startpunktens koordinater
  double b, h;     // bredd och höjd
}
```

ÖVNING 15.1
KLASSEN REKTANGEL – INSTANSMETODEN SÄTTBREDD

```
public void sättBredd(double br) {
    if (br >= 0)
      b = br;
    else
      throw new IllegalArgumentException("Negativ bredd");
  }
```

ÖVNING 15.1
KLASSEN REKTANGEL – INSTANSMETODEN SÄTTHÖJD

```
public void sättHöjd(double hö) {
    if (hö >= 0)
      h = hö;
    else
      throw new IllegalArgumentException("Negativ höjd");
  }
```

ÖVNING 15.1 – KLASSEN REKTANGEL
INSTANSMETODERNA AREA OCH OMKR

```
public double area() { // beräknar arean
    return b * h;
}

public double omkr() { // beräknar omkretsen
    return 2 * b + 2 * h;
}
```

ÖVNING 15.2

```
Rektangel r = new Rektangel();
while (true) {
  String s = JOptionPane.showInputDialog("Ange rektangelns bredd och höjd");
  if (s == null) break;
  Scanner sc = new Scanner(s);
  r.sättBredd(sc.nextDouble());
  r.sättHöjd(sc.nextDouble());
  JOptionPane.showMessageDialog(null, String.format
        ("Rektangeln har arean %.3f " + "och omkretsen %.3f\n",
         r.area(), r.omkr()));
}
```

INKAPSLING (*ENCAPSULATION*)

Inkapsling innebär att ett objekts inre uppbyggnad ska vara ointressant för den som använder objektet.

Detta är en grundprincip i objektorienterad programmering.

Allt en programmerare behöver veta är vad objektet kan göra och hur man ber objektet att göra det.

KLASSEN PERSON

```
public class Person {
  private String förnamn;     // privat instansvariabel
  private String efternamn;   // privat instansvariabel
  // instansmetoder
  public void setNamn(String förnamn, String efternamn) {
    this.förnamn = förnamn;
    this.efternamn = efternamn;
  }
  public String getNamn() {
    return förnamn + " " + efternamn;
  }
}
```

ÖVERLAGRADE METODER

Två eller flera metoder i en klass får ha samma namn.

Antalet parametrar eller parametrarnas typer måste då vara olika.

Det räcker inte att returtyperna är olika.

Vid anrop väljer kompilatorn den version där parametrarna stämmer med argumenten i anropet.

PERSON – TESTPROGRAM

```
Person p1 = new Person();

p1.setNamn("Mahmud", "Al Hakim");

System.out.println(p1.getNamn() );
```

OBS!
Två argument skickas till metoden setNamn

KLASSEN PERSON – METODEN SETNAMN (V2)

```
public void setNamn(String namn) {
   int k = namn.indexOf(',');
    if (k < 0)
      throw new IllegalArgumentException("Fel i namn");
    efternamn = namn.substring(0, k).trim();
    förnamn = namn.substring(k+1).trim();
  }
```

PERSON – TESTPROGRAM

```
Person p1 = new Person();

p1.setNamn("Mahmud, Al Hakim");

System.out.println(p1.getNamn() );
```

OBS!
Ett argument skickas till metoden setNamn

KLASSEN PERSON – METODEN SETNAMN (V3)

```
public void sätt(String s, char avgränsare) {
    int k = s.indexOf(avgränsare);
    if (k < 0)
      throw new IllegalArgumentException("Fel i namn");
    efternamn = s.substring(0, k).trim();
    förnamn = s.substring(k+1).trim();
  }
```

PERSON – TESTPROGRAM

```
Person p1 = new Person();

p1.setNamn("Mahmud;Al Hakim” , ';');

System.out.println(p1.getNamn() );
```

OBS!
Två argument skickas till metoden setNamn

Vad händer om vi skickar ";" istället för ';'

METODEN TOSTRING

Metoden toString() ger en ”textversion” av det aktuella objektet.

```
@Override
public String toString() {
     return this.förnamn + " " + this.efternamn;
}
```

KONSTRUKTORER

Har samma namn som klassen.

Får inte ha returtyp.

Får finnas flera, men då måste de ha olika antal parametrar eller olika typer av parametrar.

Om man inte definierar några egna konstruktorer, får man automatiskt en parameterlös konstruktor, en s.k. defaultkonstruktor.

Om man definierar egna konstruktorer, måste man själv definiera en parameterlös konstruktor om man vill ha en sådan.

KLASSEN PERSON

```
public class Person {
  private String förnamn;     // privat instansvariabel
  private String efternamn;   // privat instansvariabel
  public Person() { }         // Defaultkonstruktor
  // Konstruktor med två parametrar
  public  Person(String förnamn, String efternamn){
        setNamn(förnamn,efternamn);
  }
  ...
}
```