Kommentarer till föreläsningsunderlaget och dess exempel, F1

Rubriker

Om kursen

På kurssidan hittar du bl.a.:

- Föreläsningsunderlag
- Övningar
- Kommentarmaterial
- Programmeringsuppgifter
- Planeringsdokument
- Aktuella länkar

Veckans material handlar dels om att komma igång med Java och är dels repetition av viktiga delar från den grundläggande kursen. Laborationen består av två delar (fyra dokument):

- Eclipse.pdf, NetBeans.pdf.
  För att komma igång med Java. Välj Eclipse eller NetBeans. Dokumenten används på den grundläggande kursen i Java och är därför väldigt grundläggande.
- DA147AL1aVT14.pdf och DA147AL1bVT14.pdf
  Repetition av klasser, metoder, variabler mm.Grundläggande strukturer som selektioner (if-else, switch) och iterationer (for, while, do) får du repetera på egen hand.
  Eventuellt har du redan stött på laborationerna på den grundläggande kursen.

Klassens struktur

En del av det som står under denna rubrik får sin förklaring senare denna föreläsning eller i kommande föreläsningar. Du kommer eventuellt att gå tillbaka till dessa kommentarer senare.

```
public class Klassnamn extends Superklass implements … {
```

Denna rad talar om att:

- klassen är **public** - Normal synlighet för en klass. Den innebär att klassen är tillgänglig även från klasser i andra paket. Om du placerar flera klasser i samma .java-fil så får endast en vara public nämligen den med samma namn som filen. Övriga filer deklareras med paket-synlighet (ingen synlighet angiven).
- klassen heter **Klassnamn**. Naturligtvis med stor bokstav.
- klassen **ärver** klassen Superklass. Om ingen klass ärvs så utgår **extends**.Jag återkommer till detta i föreläsning 3.
- klasser kan **implementera** interface. ... står för ett eller flera interface. Om inget interface implementeras så utgår **implements** . interface repeteras kort i föreläsning 2..

```
// instansvariabler
private int balance;
```

Instansvariabler skapas då objekt av klassen skapas. Det innebär att varje objekt av denna typ har en variabel **balance**. Om man skapar många objekt av samma typ så kommer samtliga att ha sin egen variabel **balance**.

```
// klassvariabler
public static double interestRate;
```

Klassvariabler skapas då klassen läses in av den virtuella maskinen. Klassvariableln **interestRate** kommer att skapas en gång och kommer att vara gemensam för alla objekt av typen Klassnamn. Klassvariabler deklareras med modifieraren **static**. Om klassvariabeln dessutom är **public** så är den alltid tillgänglig från andra klasser. Man kommer åt variabeln med hjälp av klassnamnet:

```
Klassnamn.interestRate = 4.2;
```

```
// klassmetoder
public static long total(int balance, double interestRate, int year) {
  :
}
```

Klassmetoder är på samma sätt som klassvariabler tillgängliga från andra klasser om de är **public**. Man anropar dem med hjälp av klassnamnet:

```
int long result = Klassnamn.total(1000,3.2,10);
```

En klassmetod får inte referera till en instansvariabel eller instansmetod i samma klass. Däremot kan man anropa andra klassmetoder. En klassmetod får alltid all information som behövs via parametrar.

```
// instansmetoder (metoder)
public int getBalance() {
  return balance;   // instansvariabel får användas i metoden
}
```

Instansmetoder, eller kortare **metoder**, finns i en uppsättning för varje klass. Om en metod är **public** kan den anropas från andra klasser. Den stora skillnaden mot en klassmetod är att instansmetoden anropas med hjälp av ett objekt av klassen. Det anropande objektet har en uppsättning av instansvariabler och dessa får användas i metoden. Metoden får därför även anropa andra instansmetoder i klassen.

```
Klassnamn cls = new Klassnamn();     // objekt av typen Klassnamn skapas
int sum = cls.getBalance();          // metoden getBalance anropas med hjälp av
objektet som klass refererar till
```

Instansmetoden kan givetvis anropa klassmetoder.

```
    // inre klasser
    class InreKlass {
      :
    }
```

Man kan skriva en klass inuti en annan klass. Jag återkommer till detta i föreläsning 4.

```
} // Slut på klassen
```

Enkla variabler

I java finns följande enkla datatyper:

Datatyp	Bitar	Värde
boolean	8	true / false
char	16	Ett tecken (Unicode)
byte	8	-128 - 127
short	16	-32768 - 32767
int	32	-2147483648 - 2147483647
long	64	ung ± 9*10^18
float	32	± 3,40292347*10^38 (Noggrannhet: 9 siffror)
double	64	± 10^308 (Noggrannhet: 18 siffror)

Dessutom finns det **referensvariabler.** Dessa refererar till någon typ av objekt.

Instansvariabel / Attribut

När man skriver en klass så upptäcker man ofta att det som klassen beskriver har en eller flera **egenskaper**. T.ex. har en elev på gymnasieskolan kanske ett namn, ett program och en årskurs.

Student("Eva Månsson", "Samhällsprogrammet", 2)
Student("Bo Ek", "Handelsprogrammet",3)
Student("Evert Tall", "Teknikprogrammet",1)

Man använder variabler som placeras direkt i klassen för att lagra dessa egenskaper, t.ex.

```
public class Student {
    private String name;
    private String program;
    private int grade;
    // här följer eventuella metoder i klassen
}
```

Dessa variabler kallas för **instansvariabler** eller **attribut**. Jag placerar alltid instansvariablerna först i klassen, det är enkelt att hitta dem då. I java är det även vanligt att instansvariablerna placeras sist i klassen.

Instansvariablerna ska nästa alltid **private**-deklareras. Detta innebär att man inte kommer åt variablerna från andra klasser, t.ex.

```
Student student = new Student();
student.name = "Ing-Britt Svensson";  // Kompilatorn säger till att detta inte är tillåtet.
```

Om en användare av klassen ska kunna ändra en instansvariabels värde lägger man till en set-metod för instansvariabeln:

```
public class Student {
    private String name;
    private String program;
    private int grade;

    public void setName(String nameOfTheStudent) {
        name = nameOfTheStudent;
    }
}
```

```
Student student = new Student();
student.setName("Ing-Britt Svensson"); // går bra
```

Vid anropet `elev.setNamn("Ing-Britt Svensson");` "överförs" värdet `"Ing-Britt Svensson"` till parametern `elevensNamn` (Detta är egentligen inte sant men vi kan se det så under denna föreläsning). Sedan överförs värdet som `elevensNamn` håller till instansvariabeln `namn`.

Om användaren ska kunna avläsa värdet i en instansvariabel lägger man till en get-metod för instansvariabeln:

```
public class Student {
    private String name;
    private String program;
    private int grade;

    public void setName(String nameOfStudent) {
        name = nameOfStudent;
    }

    public String getName() {
        return name;
    }
}
```

```
Student student = new Student();
student.setName("Ing-Britt"); // går bra

String n = student.getName();
System.out.println(n); // ger utskriften: "Ing-Britt"

// System.out.println(student.getName()); // går också bra
```

Vid anropet student.getName() returnerar metoden innehållet i attributet name.

Flera objekt av samma typ i ett program

Det går naturligtvis bra att ha flera objekt av typen Student i samma program.

```
Student stud1 = new Student();
Student stud2 = new Student(), stud3 = new Student();
stud1.setName("Ing-Britt Svensson");
stud2.setName("Ingvar Hansson");

System.out.println(stud2.getNamn());
// Vad händer om man skriver ut stud3.getNamn() efter ovanstående rader? Prova!
```

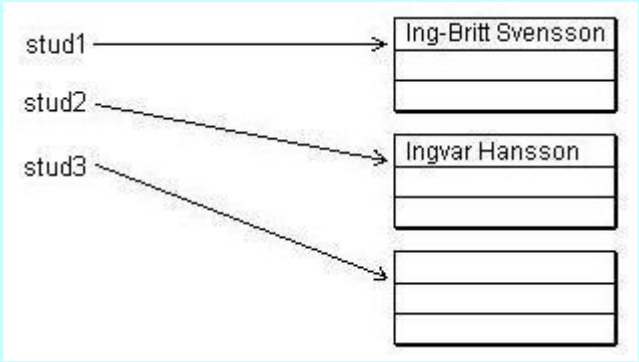
När ovanstående program körs så sker följande

```
Student stud1 = new Student ();
```
Utrymme för elev-data reserveras i datorns minne, dvs utrymme för två String och en int. stud1 håller reda på detta objekt.

```
Student stud2 = new Student(),stud3 =
new Student();
```
Utrymme för 2 uppsättningar elev-data reserveras i datorns minne. stud2 håller reda på en uppsättning och stud3 håller reda på en.

```
stud1.setName("Ing-Britt Svensson");
```
Värdet "Ing-Britt Svensson" sätts i attributet namn i det objekt som stud1 håller reda på.

```
stud2.setName("Ingvar Hansson");
```
Värdet "Ingvar Hansson" sätts i attributet namn i det objekt som stud2 håller reda på.



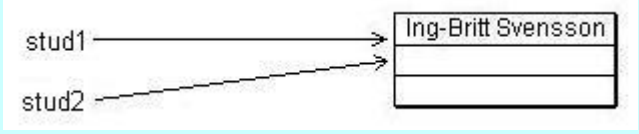
Referensvariabel

I exemplet ovan kallas variablerna stud1, stud2 och stud3 för **referensvariabler**. En referensvariabel refererar till ett objekt. men inte vilket objekt som helst utan endast den typen av objekt som anges vid variabeldeklarationen.

```
Student stud1;  // stud1 kan endast referera till ett Student-objekt. stud1 = new
String();  // fungerar inte
String str;  // str kan endast referera till ett String-objekt
```

Det går utmärkt att tilldela en referensvaraibel värdet av en annan referensvaraibel. Resultatet blir att referensvariablerna håller reda på samma objekt.

```
Student stud1 = new Student();
Student stud2;
stud1.setName("Ing-Britt Svensson");
stud2 = stud1;  // går utmärkt
```



Instansmetod

En klass kan innehålla metoder. Dessa kallas normalt för **instansmetoder**, eller kortare **metoder** . Det finnsytterligare en typ av metoder som kan vara placerade i en klass, nämligen

klassmetoder (har static i deklarationen). Vi återkommer till denna typ av metoder senare på kursen.

Följande egenskaper har en **instansmetod**:

- Den kan använda instansvariabler i klassen
- Den kan anropa andra instansmetoder i klassen

Instansmetodens **synlighet** avgör hur den kan användas. Om den deklareras som **public** så kan den anropas från metoder i andra klasser. Detta har vi gjort många gånger på kursen.

```
public class Input {
  public void example() {
    String name; // String   lagrar ett antal
tecken

    name = JOptionPane.showInputDialog("Ange
ditt namn");
    System.out.println("Hej " + name + "!");
    System.out.println("Jag tycker " + name +"
...namn!");
  }
}
```

```
public class StartInput {
  public static void
main(String[] args) {
    Input prog = new Input();
    prog.example();
  }
}
```

Om den däremot deklareras som **private** kan den endast anropas från instansmetoder i den egna klassen. Samma sak gäller för **instansvariablerna**. Om de deklareras som **public** så kan de användas från metoder i andra klasser medan **private**-deklarerade instansvariabler endast kan användas inom den egna klassen.

Konstanta objekt

Konstanta objekt är av speciellt intresse i Java. Detta beror på att vid metodanrop där objekt används som argument så kopieras objekt-referensen och metoden arbetar därmed direkt mot det objekt som utgör argument vid anropet.

Klassen Point1 ger inte konstanta objekt. Klassen har ju bl.a. set-metoder för att ändra x- respektive y-koordinat. Programmet till höger är i två versioner.

- Version 1 visar hur en illa skriven metod ändrar värdet i ett objekt av typen Point1.
- Version 2 visar hur man kan skydda sig mot dåligt skrivna metoder. Man skickar en kopia av objektet som argument till metoden om man vill vara säker på att objektets instansvariabler inte ändras i metoden.
  `ChangePoint1.info(new Point1(p));`

```
public class Point1 {
    private int x;
    private int y;

    public Point1() {}
```

```
Version 1 - osäker version

public class BadCode {

    public static void info(Point1 point) {
        Random rand = new Random();
        System.out.println("Detta är ett
```

```
    public Point1(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public Point1(Point1 p) {
        x = p.getX();
        y = p.getY();
    }

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }

    public void setX(int x) {
        this.x = x;
    }

    public void setY(int y) {
        this.y = y;
    }

    public String toString() {
        return "("+x+";"+y+")";
    }
}
```

```
Point1-objekt");
            point.setX(rand.nextInt(100)); // Hoppsan
        }

        public static void main(String[] args)  {
            Point1 p = new Point1(2,4);
            System.out.println(p);
            ChangePoint1.info(p);
            System.out.println(p);
        }
    }
```

Exempel på körresltat

```
(2;4)
Detta är ett Point1-objekt
(85;4)
```

Version 2 - säker version

```
public class SafeBadCode {

    public static void info(Point1 point) {
        Random rand = new Random();
        System.out.println("Detta är ett
Point1-objekt");
        point.setX(rand.nextInt(100));
    }

    public static void main(String[] args)  {
        Point1 p = new Point1(2,4);
        System.out.println(p);
        ChangePunkt.info(new Point1(p)); // Kopia som
argument
        System.out.println(p);
    }
}
```

Exempel på körresltat

```
(2;4)
Detta är ett Point1-objekt
(2;4)
```

Klassen Point2 innehåller inga metoder (t.ex. set-metoder) för att ändra attributens värde. Därmed går det inte ens att skriva den dåliga metoden info. Kompilatorn kommer inte ens att godkänna koden - det finns ju ingen setX-metod i klassen.

```
public class Point2 {
    private int x;
    private int y;

    public Point2(int x, int y) {
        this.x = x;
        this.y = y;
```

```
    }

    public Point2(Point2 p) {
        x = p.getX();
        y = p.getY();
    }

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }

    public String toString() {
        return "("+x+";"+y+")";
    }
}
```