

# Föreläsning 1–2: Inledande exempel

## 1 Variabler och typer

Variabler använder vi för att hålla reda på den information som ett program bearbetar. Variabler är av olika sorter, och i Java håller man mycket strikt koll på variabelernas innehåll. Varje variabel deklarerar som tillhörig en viss typ.

```
int mittHeltal = 3;  
String mittOrd = "tomaterna";
```

Dessa satser deklarerar de två variablerna `mittHeltal` och `mittOrd` som varande av typerna `int` och `String`. Typen `int` avser heltal (eng. "integer") med vissa storleksbegränsningar och `String` avser strängar, alltså sekvenser av tecken. Dessa satser tilldelar även variablerna konkreta värden, och visar hur heltal och strängar skrivs.

Andra satsen kunde även skrivas som två, en för deklarationen och en för tilldelningen.

```
String mittOrd;           // deklaration  
mittOrd = "tomaterna";   // tilldelning
```

Det vanliga är dock att man deklarerar och gör en första tilldelning (initierar variabeln) på samma rad.

Notera även att vi kan skriva kommentarer med hjälp av `//`. Allt som kommer efter `//` på en rad räknas som kommentar och tolkas ej som programkod.

## 2 Objekt och metoder, några första påpekanden

En viktig typ av värden på variabler är s.k. objekt. Dessa är givetvis av central betydelse i objekt-orienterad programmering, som Java exemplifierar. Typen `String` är en objektstyp (dock inte `int`, som "bara" är en "enkel typ").

En annan viktig sak i objekt-orienterad programmering är metoder. Dessa representerar saker man kan göra och man kan definiera egna metoder eller använda befintliga (som finns färdiga i Java). Ett viktigt sätt att använda metoder är att utföra dem på objekt. Objekt och metoder passar ihop, och detta är en grundprincip i objekt-orienterad programmering.

Två metoder som man kan utföra på `String`-objekt är `substring` och `length`. `substring` är en fördefinierad metod som utförs på ett `String`-objekt och utifrån två heltal ger oss en delsträng, nämligen den delsträng som börjar på positionen som anges av första talet och avslutas vid positionen som anges av andra talet (tecknet där inkluderas ej). Det är viktigt att bestämma att returtypen för metoden just är `String`. De två heltalen skickas till metoden som s.k. argument. Vi får på så sätt ett metodanrop genom att t.ex. skriva `mittOrd.substring(mittHeltal, 6)`, där variablerna införts som ovan. Detta metodanrop returnerar "ate" (som ligger på positionerna 3–5 i `mittOrd = "tomaterna"`).

Sammanfattningsvis tillhör metoden klassen `String` och dess returtyp är (också) `String` och den tar två argument, båda av typen `int`. I dokumentationen ställs det upp så här, (se <http://java.sun.com/javase/6/docs/api/java/lang/String.html>):

```
public String substring(int beginIndex,
                       int endIndex)
```

Metoden `length` är enklare. Den ger ett `int`-värde från strängen men tar inget argument.

```
public int length()
```

Det kan se ut så här när vi använder denna metod.

```
int mittAndraHeltal = mittOrd.length();
```

### 3 Exempel1.java – skicka argument till program och villkor

Detta är filen `Exempel1.java`, som är ett Javaprogram:

---

```
public class Exempel1 {

    public static void main (String[] kommRadsArg) {
        if (kommRadsArg.length > 0) {
            System.out.println("Första argumentet " +
                               "från kommandoraden: " +
                               kommRadsArg[0]);
        }
        else {
            System.out.println("Inget argument gavs på kommandoraden");
        }
    }
}
```

---

- `public class Exempel1:Exempel1` är en allmänt synlig (`public`) klass. Den måste definieras i filen `Exempel1.java`.
- `public static void main...`: Här definieras metoden `main`. Den är allmänt synlig (`public`). Den är statisk, vilket enkelt uttryck innebär att den inte utförs på någon objektsinstans. Den returnerar inget värde (`void`).
- `String` namn på standardklassen av strängar, alltså teckensekvenser.
- `String[]` typen för att fält av `String`-objekt. Ett fält är en linjär sekvens av saker, numrerade 0, 1, 2, etc.
- `main (String[] kommRadsArg)` betyder i denna kontext att metoden `main` tar ett fält (se mer nedan) av strängar (`String`) som argument. Argument är de data vi ger till en metod och som metoden kan göra något med. Detta argument (argumentvariabel) får här namnet `kommRadsArg`.
- `(kommRadsArg.length > 0)`: Här testas vi om fältets längd är större än 0. Detta är ett uttryck av typen `boolean`, som kan anta ett sanningsvärde. (Dessa heter `true` och `false` i Java.)
- `+`: är en operator som (här) sätter ihop strängar (till längre strängar). I andra kontexter kan `+`, som man kan förvänta sig, stå för addition.

- `System.out` är systemets standard output (som är en outputström).
- `println` är en metod som utförs på en outputström. Den skriver en sträng och avslutar raden. Strängen ges som argument till metoden. Här är argumentet:  
`"Första argumentet " + "från kommandoraden: " + kommRadsArg[0]`  
 Detta blir konkret denna sträng i exempelanropet nedan.  
`"Första argumentet från kommandoraden: Detta"`  
 Notera att `"` omsluter strängar i Java-notation.
- `if Test Block1 else Block2`: En villkorssats: gör `Block1` om `Test` blir `true`, annars `Block2`.

### 3.1 Kompilera ett program

Så här kompilerar vi vårt program i Linux, med kommandot `javac` (ompilator). Detta kan ge oss varningar och felmeddelanden, men i detta fall är kompilatorn helt nöjd. Den har därmed skapat en fil som heter `Exempell.class` med sådan bytekod som Javainterpretatorn kör.

---

```
mindator> javac Exempell.java
```

---

### 3.2 Köra ett program

Vi kan bara köra ett program efter kompilering, och det vi kör är bytekodsfilen (i detta fall `Exempell.class`). Vi kan t.ex. anropa den så här (extensionen `.class` skrivs inte ut):

---

```
mindator> java Exempell Detta är mina argument
Första argumentet från kommandoraden: Detta
```

---

Som du ser, så är det första fallet i villkorssatsen som aktiveras. Varför?

Jo, när man kör en klass med `java Exempell` så är det klassens `main`-metod som anropas. Metoden `me` det namnet har alltså en speciell ställning. Linux och Javainterpretatorn ser till att knyta de eventuella ord som kommer efter `java Exempell` till det fält av `String`-objekt (fältet är då av typen `String[]`) som är `main`-metodens argument.

Detta argument får namnet `kommRadsArg` inne i metodens kod. Alltså utgår metoden från ett läge där:

```
kommRadsArg[0] är "Detta"
kommRadsArg[1] är "är"
kommRadsArg[2] är "mina"
kommRadsArg[3] är "argument"
```

I Java skriver man `kommRadsArg[0]` för att komma åt första (nollte) värdet i detta fält.

Här är `kommRadsArg.length` lika med 4. Testet (`kommRadsArg.length > 0`) blir `true`. Som vi istället skriver så här, så är det `else`-fallet i villkorssatsen som aktiveras.

---

```
mindator> java Exempell
Inget argument gavs på kommandoraden
```

---

Varför är det `else`-fallet som aktiveras? Jo, eftersom det inte kommer något mer än programnamnet på kommandoraden så finns det inget argument. Fältet `kommRadsArg` blir tomt, vilket är samma sak som att `kommRadsArg.length` är lika med 0.

Testet (`kommRadsArg.length > 0`) ger sålunda `false` och det är `else`-blocket som exekveras.

## 4 Exempel2.java – mer om fält och om iteration

---

```
public class Exempel2 {

    public static void main (String[] kommRadsArg) {

        String[] arrayOne = new String[3];
        arrayOne[0] = "Hej";
        arrayOne[1] = "på";
        arrayOne[2] = "dig!";

        String[] arrayTwo = {"Hej", "på", "dig!"};

        for (String str: arrayOne) {
            System.out.println(str);
        };

    }
}
```

- 
- `String[] arrayOne = new String[3]`: För det första skall `arrayOne` som värde ha ett fält av `String` (alltså `String[]`). För det andra skapar vi ett nytt sådant fält med tre platser med `new String[3]`.
  - `arrayOne[0] = "Hej"` tilldelar plats 0 i detta fält värdet "Hej".
  - `String[] arrayTwo = {"Hej", "på", "dig!"}` skapar fältet och tilldelar dess platser värden i ett "svep". Detta är bara ett mer kompakt sätt att säga samma sak som på fyra rader ovan.
  - Uttrycket '`for (Typ Variabel: Samling) Block`' är en s.k. förenklad `for`-sats. Den utförs genom att objekten i *Samling* går igenom ett och ett och instruktionerna i *Block* utförs. Objekten i *Samling* skall vara av typen *Typ* och de knyts i varje vända till variabeln *Variabel*.

Sådan exekvering där ett och samma sak upprepas flera gånger kallas *iteration*.

I det aktuella exemplet får variabeln `str` värdena "Hej" och sedan "på" och slutligen "dig!".

Det ser alltså ut så här om vi kompilerar och kör programmet.

Notera att programmet aldrig tittar på `kommRadsArg`!

---

```
mindator> javac Exempel2.java
mindator> java Exempel2
Hej
på
dig!
```

---

## 5 Exempel3.java – ta isär strängar och iteration på lite nytt sätt

---

```
public class Exempel3 {
    public static void main (String[] kommRadsArg) {
        if (kommRadsArg.length > 0) {
            System.out.print(kommRadsArg[0].substring(0,1));
            // skriver ut första tecknet, position 0
            for (int i=1; i < kommRadsArg[0].length(); i++){
                System.out.print("-" + kommRadsArg[0].substring(i,i+1));
                // skriver ut återstående detcken ett efter ett med "-" före
            };
            System.out.println("");
            // avslutar raden
        }
    }
}
```

---

- `print` är en metod som utförs på en outputström. Den är som `println` (den skriver en sträng) men den avslutar inte raden.
- `length` (som beskrivits ovan) är en metod som utförs på ett `String`-objekt och ger dess längd. Metoden tar inga argument. Detta indikeras med `()`.

`kommRadsArg[0].length()` är alltså längden på första (nollte) kommandoradsargumentet (från Linux).

- Uttrycket '`for (Initiering; Villkor; Ändring) Block`' en s.k. (vanlig) `for`-sats. *Initiering* utförs först. Sedan utförs *Block* och *Ändring* så länge *Villkor* är sant.

`for (int i=0; i < kommRadsArg[0].length(); i++)` innebär att variabeln `i` som skall vara av typen `int` (heltal, eng. "integer") först får värdet 0. Variabeln `i` skall vara mindre än `kommRadsArg[0].length()` för att *Block* skall upprepas. I varje vända räknas `i` upp ett snäpp. Kommandot för det är `i++`. Om `kommRadsArg[0].length()` är 4, så får `i` alltså värdena 0, 1, 2 och 3. Sedan när `i` blir 4 så blir villkoret `i < kommRadsArg[0].length()` falskt (får värdet `false`), varför exekveringen av `for`-satsen avbryts.

- `substring` är en metod som utförs på ett `String`-objekt och utifrån två heltalsargument ger oss en delsträng, nämligen den som börjar på positionen som anges av första argumentet och avslutas vid positionen som anges av andra argumentet (tecknet där inkluderas ej).

`kommRadsArg[0].substring(i, i+1)` tar alltså ut delsträngen ur `kommRadsArg[0]` (alltså, första, nollte, kommandoradsargumentet) av längden ett som innehåller tecknet vid position `i`. Även i `String`-objekt räknar man positioner från 0 och uppåt.

Summan av kardemumman är alltså att programmet tar första ordet bland argumenten och skriver ut tecknen i det med bindestreck emellan:

---

```
mindator> javac Exempel3.java
mindator> java Exempel3 hej du
h-e-j
```

---

## 6 Exempel4.java – bryta ut en procedur

En viktig sak i Exempel3.java är att gå från en sträng, t.ex. "hej", till utskriften med bindestreck emellan tecknen, t.ex. h-e-j. Vi kan bryta ut den delen av det hela som en metod, här kallad `printExpanded`. Det kan bli så här.

---

```
public class Exempel4 {

    private static void printExpanded(String str) {
        System.out.print(str.substring(0,1));
        // skriver ut första tecknet, position 0
        for (int i=1; i < str.length(); i++){
            System.out.print("-" + str.substring(i,i+1));
            // skriver ut återstående tecken ett efter ett med "-" före
        };
        System.out.println("");
    }

    public static void main (String[] kommRadsArg) {
        if (kommRadsArg.length > 0) {
            printExpanded(kommRadsArg[0]);
        };
    }
}
```

---

Detta program arbetar precis som Exempel3.java. Metoden `printExpanded` är void, d.v.s. returnerar inget värde, och det behövs inte, för den verkställer ju utskriften ändå. Metoden är även `private`, alltså ej tillgänglig utanför denna klass `Exempel4`.

## 7 Exempel5.java – med metod som returnerar värde

Vi kan göra ungefär som i Exempel4.java, fast ha en metod som returnerar ett värde. Det vore användbart att kunna gå från en sträng, t.ex. "hej", till motsvarande sträng med bindestreck emellan tecknen, t.ex. "h-e-j". (I Exempel4.java sätts aldrig den strängen samman, utan tecknen hamnar direkt i output.) I detta exempelprogram gör vi på detta sätt. Vi skriver den givna strängens bitar i metoden `stringExpanded` till `String`-variabeln `expanded` en efter en med bindestreck emellan och gör sedan `return expanded` varvid dess värde returneras av metoden `stringExpanded` (som ju tar ett `String`-argument och har `String` som returtyp).

---

```
public class Exempel5 {

    private static String stringExpanded(String str) {
        String expanded = str.substring(0,1);
        // expanded blir första tecknet, position 0
        for (int i=1; i < str.length(); i++){
            expanded = expanded + "-" + str.substring(i,i+1);
            // skriver ut återstående tecken ett efter ett med "-" före
        };
        return expanded;
    }

    public static void main (String[] kommRadsArg) {
        if (kommRadsArg.length > 0) {
            System.out.println(stringExpanded(kommRadsArg[0]));
        };
    }
}
```

---