Objektorienterad Programmering DAT043

Föreläsning 2
15/1 -18
Moa Johansson

Dagens föreläsning

- Loopar
- Arrayer
- Objekt och metoder
- I/O
- Paket och importer

for-loop

Före ingång i loopen

Om sant, kör loopen ett varv

Efter varje varv i loopen

```
for(init; villkor; uppdatering){
   s1;
   s2;
   ...
}
```

```
for(int i = 1; i <= 10; i++){
   System.out.println(i);
}
```


För upprepade satser som ska köras gång på gång. Satserna inne i loopen körs om och om igen. För varje “varv” kollas om villkoret är sant, isåfall exekveras satserna, och därefter uppdateringen. Om villkoret är falskt är loopen färdig och programmet fortsätter med det som kommer efter loopen.

Kan utelämna init,villkoret, och uppdateringen men då måste man använda kommandot break för att få loopen att stanna och inte repetera för evigt.

while-loop

```
while(villkor){
   s1;
   s2;
   ...
}
```

```
int x = 1;
while(x <= 10){
 System.out.println(x);
 x++;
}
```

```
while (gissning != slumptal) {
      System.out.println("Fel. Gissa igen.");
      gissning = Integer.parseInt(System.console().readLine());
    }
System.out.println("Rätt!");
```


Samma som for-loop utan initiering och uppdateringsuttryck. Man kan ofta byta en loop för en annan, men ibland kan en typ kännas mer naturlig.
While-loopar kan kännas mer naturliga om t.ex. I/O dialog ska upprepas till användaren skriver ett avslutningskommando (eller som här, gissar rätt).

for-loopar kan vara mer naturliga när utför någon form av beräkning, och har en variabel som explicit räknar upp/ner antalet itertioner (en s.k. loop-counter). Vilken variabel som fungerar som loop-counter görs tydligare i en for-loop, där den deklarares i headern.

Notera metoden Integer.parseInt: Detta är en speciell metod som “gör om” inputen (vilket har lästs in som en sträng) till ett värde av typ int, motsvarande den siffra som finns i sträng-form. Motsvarande finns även för andra primitiva typer.

break

```
while (true) {
     int gissning = Integer.parseInt(System.console().readLine());
     if (gissning == slumptal)
         break;
     System.out.println("Fel. Gissa igen.");
    }
System.out.println("Rätt!");
```

Går ofta att använda antingen while-loop eller for-loop. Vilket är snyggast?

```
for (;;) {
     int gissning = Integer.parseInt(System.console().readLine());
     if (gissning == slumptal)
         break;
     System.out.println("Fel. Gissa igen.");
    }
System.out.println("Rätt!");
```


Vanligtvis skulle vi ju inte vilja att en loop aldrig stannar (while (true) eller for(;;)), men om man har ett break så går det bra. Break gör att programmet hoppar ut ur loopen och fortsätter exekvera den kod som kommer direkt efter.
Demo: ExempelLoop.java

Nästlade loopar

Loopar kan vara nästlade inuti en annan loop:

```
public static void gångertabell(){
    for (int i  = 1; i <= 9; i++) {
      for (int j = 1; j <= 9; j++) {
        System.out.format("%3d", i * j);
      }
      System.out.println("");
    }
  }
```


Ibland är det praktiskt att ha en loop inuti en annan loop.

Demo: ExempelForLoop.java

do-while loop

do-while loopar exekveras minst en gång, till skillnad från while-loopar.

```
do {
  s1;
  s2;
  ...
} while (e);
```

```
do {
  System.out.println(“Do-while”);
   x--;
} while (x > 0);
```


Som while-loop, men exekverar loopen en gång innan den beräknar villkoret första gången. Alltså skiljer den sig från en vanlig while loop i det att en do-while exekveras minst en gång, medan en while-loop kanske inte exekveras alls, om villkoret inte håller.

Arrayer

```
int[] xs;                          // Deklaration av array variabel xs.
xs = new int[10];                  // Initiering av ny array med längd 10.
boolean[] bs = new boolean[5]; // Går även att göra i ett steg.
```

```
int len = xs.length;      // Ta reda på en arrays längd.
int först = xs[0];        // OBS! Indexering börjar på 0...
int sist = xs[len-1];     // Sista element har alltså index len-1.
```

- Längden på en array kan inte ändras.
- Istället: Ersätt en array med en annan om det behövs.

```
boolean[] bs = new boolean[5];
bs = new boolean[10];       // Vi “byter” ut bs mot en ny array.
```


Liknar listor i Haskell, men innehållet lagras efter varandra i minnet. Man kan komma åt ett visst element i en array direkt genom att ange dess index.

Liksom i listor så består arrayer av element med samma typ. När vi deklarerar en ny array får alla element typens förvalda värde (se förra föreläsningen). För int är detta 0 och för boolean false.

Vanligt misstag: Index för array börjar på 0 och slutar alltså på length-1. Försöker vi med något annat får vi ett felmeddelande när vi kör programmet (ArrayIndexOutOfBoundsException).

Längden på en array kan inte ändras. Men vi kan ersätta den med en längre array om det behövs. Notera dock att vi i så fall explicit måste kopiera värdena på elementen i den gamla arrayen.

Arrayer: initiera element

```
int[] xs = {2,4,6,8}; // Går att explicit initiera med givna värden.
```

```
int [] xs = new int[100];
for (int i = 0; i < xs.length; i++){ // OBS! index börjar på 0.
     xs[i] = 2*(i+1);        // Initiera till jämna tal, större än 0.
   }
```

- Vanligt att använda en loop för att initiera längre arrays.
- Viktig att komma ihåg att indexeringen börjar på 0!

Vad innehåller xs efter den första raden? Den har 100 element, men vad är dessa?

Hur fyller vi arrayer med något annat än de förvalda värdena? Vi kan antingen göra det direkt vid initieringen, men det kan var opraktiskt för längre arrayer. Då är det oftast lämpligt att använda en loop som initierar varje element i tur och ordning. Vi måste dock komma ihåg att första index är 0 och att loopen ska sluta ett steg "innan" indexet för längden.

Demo: ExempelArray.java. Visa även vad som händer om man itererar för långt, till i <= x.length

Flerdimensionella arrayer

```
int[][] x = {{1, 7}, {7, 6}, {0, 3}};
int[][] y = {{8, 3}, {2, 5}, {1, 7}};
int[][] z = new int[x.length][x[0].length];
```

```
// Addition av matriser.
for (int i = 0; i < x.length; i++) {
      for (int j = 0; j < x[0].length; j++) {
        z[i][j] = x[i][j] + y[i][j];
      }
    }
```


Här har arrayerna x och y samma dimensioner, och x[i].length är samma för alla i. Detta behöver dock inte alltid vara fallet, så här får man se upp! Vi skulle kunna sätta t.ex. x[1] = new int[10]. Rätt sätt att se på flerdimensionella arrayer är alltså som arrayer av arrayer (inte matriser, även om de kan användas så).

Statiska Metoder

- Statiska metoder: Hör till en klass

```
public static returtyp metodnamn(argtyp1 argnm1, ...){
   s1;
   s2;
   ...
   return ... ;
}
```

- Exempel:

```
public static int pow(int x, int y) {
  int res = 1;
  for (; y > 0; y--) res *= x;
  return res;
}
```

Anta att metoden pow finns i klassen Exempel

```
Exempel.pow(2, 2);      //Anropar metoden pow.
```


Lite repetition:
Hittills har vi bara tittat på s.k. statiska metoder (keyword static). Vi kommer senare att även se andra typer av metoder. Vi har även bara tittat på publika metoder (keyword public), vilket betyder att metoden kan anropas även från andra klasser, dvs, metoden är synlig även utanför klassen där den definierats.
Listan med argument och deras typer kan vara tom. Då skriver man bara tom parentes.

uttrycket return... anger vad metoden returnerar. Typen anges i headern.

Statiska metoder hör till den klass där den definierats. Ska man anropa en sån metod (förutsatt att den är public) skriver man Klassnamn.metodnamn(argument). Om man anropar en statisk metod i samma klass som den är definierad i räcker det att skriva metodnamn(argument), man kan hoppa över klassnamnet.

Statiska metoder forts.

```
public static int abs(int x) {
  if (x < 0) {
    return -x;
  } else {
    return x;
  }
```


En metod kan ha flera return-uttryck och de behöver inte nödvändigtvis vara det sista. T.ex. kan man returnera olika saker beroende på branch i ett if-else.

Metoder utan returvärde

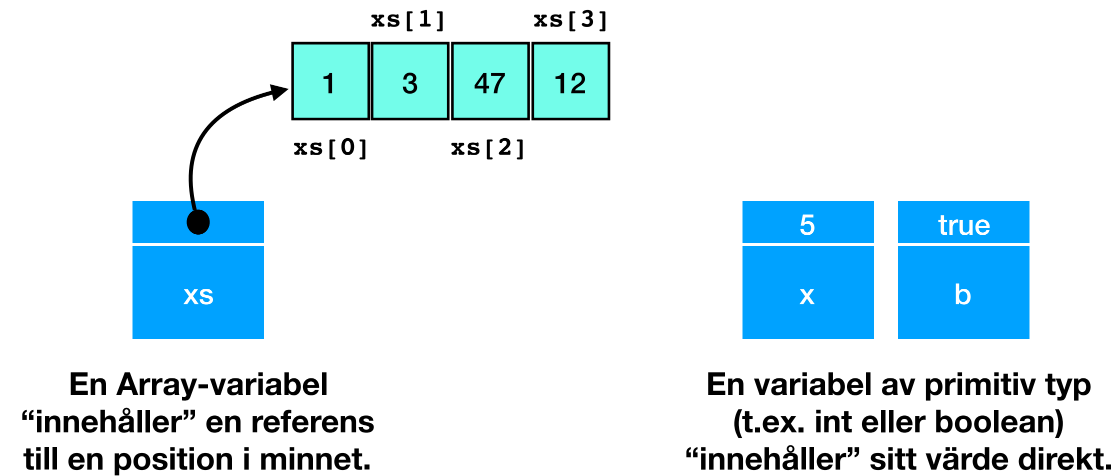
void: returnerar
inget värde

```
public static void printNumber(int x) {
  System.out.println(x);
  return; // antingen tomt return, eller inget alls.
}
```


Alla metoder returnerar inget värde. De kanske bara exempelvis skriver ut något till stadard out, eller utför någon form av uppdatering. Vi har sett sådana, t.ex. main. Dessa metoder har den speciella "returtypen" void.

För dessa metoder kan man antingen ha en tom return:
return;
eller helt utesluta return.

Arrayer som argument




```
int[] xs = {1,2,3,4,5,6};
print(xs); // Skriver ut 1,2,3...
reverse(xs); // reverse har typ void
print(xs); // Skriver ut 6,5,4...
```

```
int x = 5;
System.out.println(x); // Skriver 5
negera(x);
System.out.println(x); // Skriver 5
```


Man kan skicka arrayer som argument. Detta skiljer sig från primitiva typer (såsom int) genom att det inte är värdet (d.v.s. alla elementens värde) som skickas till metoden utan en referens till en position i minnet.
Det är en fördel för stora arrayer på det sättet att det går snabbare. Det har också den effekten att ändringar som metoden gör i arrayen finns kvar när exekveringen av den anropande metoden fortsätter. Här gäller det att ha tungan rätt i mun. Imperativ programmering kryllar av sidoeffekter av det här slaget, men det kan leda till svårhittade fel.

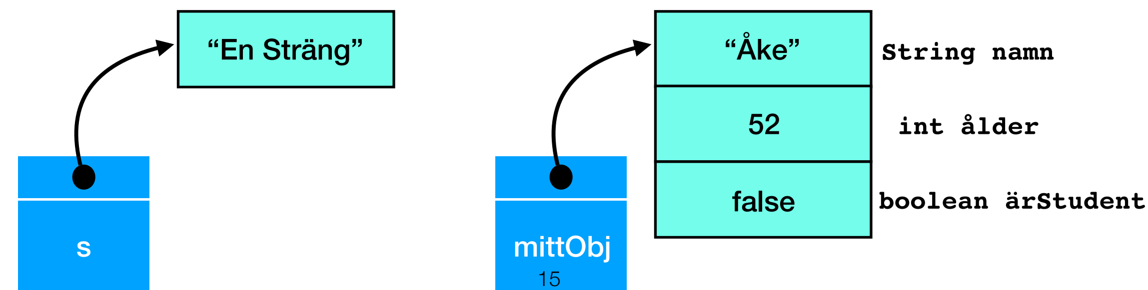
DEMO: ExempelArray2.java

I exemplet till vänster, notera att vi för arrayen får nya värden *trots att vi inte skrivit xs = reverse(xs)* och reverse har returtyp void! Referensen är den samma, men det är det den pekar på som har ändrats. En metod med returtyp void kan alltså ha sidoeffekter som uppdaterar något i minnet, trots att den inte explicit returnerar något.

I det högra exemplet måste vi explicit skriva: x = negera x. negera måste ha returtypen int. Gör vi den till void kan vi varken returnera ett värde, eller uppdatera argumentet då det är av primitiv typ.

Kort om: Objekt

- Utöver primitiva värden och typer (e.g. int, boolean...) och arrayer finns även objekt, vars typ definieras i en klass.
- Objekt innehåller data (samling av värden). Jmfr. med tuple i Haskell.
- Likt arrayer "innehåller" ett objekt en referens till data i minnet. Kan alltså modifieras av metoder.
- Initieras med **`new`**



Här har variabeln s typ "String", Javas inbyggda typ av strängar. Vi säger att "s är ett objekt av typ String".

Variabeln mittObj innehåller en tuple som består av en sträng, en int och en boolean. Varje värde har ett namn (här namn, ålder, ärStudent).
Vi kommer att återkomma till hur man definierar sina egna objekttyper i kommande föreläsningar.

Liksom för arrayer anropar man new när man vill skapa ett nytt objekt i ett initialt tillstånd. Man använder punkt för att komma åt ett metod (eller ett värde) i ett objekt.
obj.metod(...)

Exempel: För att komma åt värdet "namn" i mittObj skriver vi:
String namnet = mittObj.namn;
Jämför med hur vi kommer åt längden på en Array: xs.length. "length är egentligen ett värde i array-objektet, och inte en metod! Vi kan se detta eftersom det inte skrivs någon parentes efter length.

Kort om: Instansmetoder

- Typer för objekt definieras i tillhörande klass, med samma namn.
- Här definieras även metoder som t.ex. *uppdaterar och använder* data i objektet, sk. ***instansmetoder*** (kallas även ***objektmetoder)***.
- Dessa metoder anropas *på specifika objekt*. Till skillnad från statiska metoder som vi sett hittills.

```
obj.metod(...)
```

```
String s = “Min sträng”;
s.length();
```


Jämför med notationen för att anropa en statisk metod: Klassnamn.metodnam.
För instansmetoder är det istället variabelnamn.metodnamn (variabeln måste referera till ett objekt).
Man kan tänka sig att objektet på vilket vi anropar metoden är som ett extra argument till metoden.

Vi kommer att återkomma till skillnaden mellan statiska metoder och instansmetoder i kommande föreläsningar. För tillfället kan vi nöja oss med att tänka oss att statiska metoder utför någon form av beräkning till exempel (se tidigare exemplet med `pow`, `Math.floor` och liknande metoder), medan instansmetoder använder information och/eller uppdaterar information som lagrats i ett objekt.

Strängar

```
// Deklaration av ny strängvariabel, till en tom sträng.
String s = new String();
String s2 = “Hej”;     // Deklaration och initiering.
```

```
String mellanrum = “ “;
// + slår ihop strängar.
System.out.println(s2 + mellanrum + “Moa!”);
```

```
String str = "åtta sju";
str.length()         // --> 8
str.charAt(3)        // --> 'a'
str.substring(2, 6)  // --> "ta s"
```

```
String str = "åtta sju";
str == "åtta sju”;        // returnerar false!
str.equals("åtta sju")    // returnerar true.
```


Strängar är objekt i Java. Klassen där de är definierade heter String (kolla gärna upp den i Javas API).
Här ger vi några exempel på vanliga metoder. Notera att de är instansmetoder, vi anropar dem alltid på ett objekt av typ String. Varför är de instansmetoder? För att t.ex. hitta en substring behöver vi “gå in” i String-objektet och inspektera innehållet. Det behövde vi inte när vi t.ex. beräknade `pow` i en statisk metod, här kunde vi direkt använda argumenten.

DEMO: StringExample.java (notera skillnaden mellan metoder vi sett förut som var statiska, nu kallas de på objekt)

Notera att length ska ha parentes, till skillnad från arrayer. Vad betyder detta? length för strängar är en metod som beräknar strängens längd, medan length för Arrayer är ett värde som sparats “innuti” Array-objektet...
Precis som för Arrayer så börjar index i en String på 0 (vi kan tänka på en sträng som en array av chars).

Notera att vi ska använda metoden `equals` om vi vill jämföra strängar. (`==` kommer att jämföra referenserna, dvs om de två objekten pekar till samma position i minnet!).

Strängar, forts

```
String str = "12,25,3,0,128";
String[] ss = str.split(",");
```

```
String s = "1a25b32345c0d128 ";
String[] ss = s.split("\\d+"); //Reg-exp: Ta bort alla siffror ur s.
// Skriver ut a b c d på varsin rad
for (int i = 0; i < ss.length; i++) {
  System.out.println(ss[i]);
}
```

```
byte[] bs = {97,98,99,100,101,102};
String str = new String(bs);
System.out.println(str);     // Skriver ut abcdef!
```


Split är en användbar metod för att göra om en sträng till en array. Argumentet talar om för oss att vi ska dela strängen vi ",". Man kan även använda s.k. regular expressions för att splitta strängar. Det är väldigt användbart om vi t.ex. vill ta bort alla siffror eller whitespaces i en sträng (se Oracles dokumention för mer info om olika reg-exps).

Vi kan även skapa strängar på lite mer obskyra sätt, t.ex. genom en array av bytes. Vi får en sträng som omvandlar bytesekvensen till tecken enligt plattformens förvalda teckenuppsättning. På min Mac betyder tydligen sekvensen i exemplet a b c d e f.

Programargument

- Äntligen kan vi förstå vad argumentet `args` till main metoden i Java är: En array med en strängrepresentation av argument som givits programmet.
- Argumenten finns i varsin sträng. Första argumentet är `args[0]`, andra `args[1]` o.s.v.
- Antalet argument är `args.length`.

```
public static void main(String [] args){
    int inputArg = Integer.parseInt(args[0]);
}
```


Argumenten läses in in stängform. Kom ihåg att det finns behändiga metoder som t.ex. parseInt, parseDouble osv som man kan använda för att konvertera argumenten till andra typer om det behövs. Kolla upp dessa i Javas online API! De primitiva typerna har där motsvarande klasser för metoder som dessa.

(Demo om tid finns: modifiera t.ex. StringExample till att läsa in ett argument).

För laboration 1 behöver man kunna ange ett argument när programmet körs. Det är lätt när man kompilerar och kör program i terminalfönstret/kommandotolken. I Eclipse finns “Run Configurations…” under menyn “Run”. Där kan man ställa in programargument under fliken “Arguments”. Motsvarande funktion finns i övriga IDEer.

Läsa in filer

- Javas bibliotek innehåller naturligtvis många olika sätt att läsa in filer.

```
String filnamn = "minFil.txt";
byte[] bs = java.nio.file.Files.readAllBytes(java.nio.file.Paths.get(filnamn));
```

- Tänk på att filnamn och sökväg antingen ska vara absoluta eller beror på var de ligger i förhållande till var dit Javaprogram exekveras.

Innehållet i filen med namnet "minFil.txt" läses in till en array av bytes. java.nio.file.Paths.get omvandlar en sträng till en abstrakt representation av en sökväg i filsystemet, ett objekt av klassen Path.

java.nio.file.Files.readAllBytes läser sedan in innehållet i filen med denna sökväg.
Om något går snett när detta ska utföras så kastas ett undantag (exception). Vi kommer behandla undantag senare och även gå igenom läsning och skrivning till filer och IO i allmänhet mer noggrant.

I labb 1 är argumentet en fil som ska hittas. Antingen kan man ange den absoluta sökvägen till filen. Annars kan man chansa på att programmet när det körs har den katalogen som filen finns i som sin arbetskatalog (aktuella katalog). Om så inte är fallet går även detta att ställa in på samma flik som programargument.

Paket och import

- Javas bibliotek är organiserade i paket.
- `java.nio.file.Files.readAllBytes` är en metod som finns i klassen Files, som ligger i paketet file, vilken i sin tur finns i paketet nio som finns i toppnivåpaketet java.
- För att slippa långa namn kan man importera klasser, eller hela paket.

```
import java.nio.file.Files;
...
byte[] bs = Files.readAllBytes(java.nio.file.Paths.get(filnamn));
```

```
import java.nio.file.*;
...
byte[] bs = Files.readAllBytes(Paths.get(filnamn));
```


En klass behöver inte tillhöra något paket.
Testprogrammet i labb 1 försöker anropa metoden sort i klassen Lab1, d.v.s. Lab1.sort. Inget paket anges så därför behöver klassen Lab1 befinna sig på rotnivå i pakethierarkin. IDEer kan automatiskt placera klasser i paket. Ha detta i åtanke om du stöter på problem när du ska köra testprogrammet.

För att slippa skriva ut vägen i paketstrukturen till metoder om och om igen i koden kan man i början av filen tala om att ett vissa pakets namespace ska vara tillgängligt.

Javadoc

```
// Jag är en kommentar i Java
/* Kommentarer kan gå över
   flera
   rader */
```

- Javadoc är ett verktyg som låter dig automatiskt skapa html-dokumentation för din kod om du skriver kommentarer till klasser och metoder på ett speciellt format.

```
/**
 * Returns x^y.
 * <p>
 * This method compute the power function for integers.
 *
 * @param  x  the base value
 * @param  y  the exponent
 * @return    x raised by y
 */
public static int pow(int x, int y) {
  ...
```


Javas online API (som ni bör titta på!) är genererat med Javadoc.