EDA016 Programmeringsteknik för D

Läsvecka 2: Kodstruktur

Björn Regnell

Datavetenskap, LTH

Lp1-2, HT 2015

2 Kodstruktur

- Algoritmer
- Loop-strukturer
- Varför behövs kodstruktur?
- Filstruktur
- Objekt

Vad är en algoritm?

En algoritm är en sekvens av instruktioner som beskriver hur man löser ett problem

Exempel:
matrecept

Vad är en algoritm?

En algoritm är en sekvens av instruktioner som beskriver hur man löser ett problem

Exempel:
- matrecept
- uppdatera highscore i ett spel
- ...

Algoritm-exempel: Highscore

Problem: Uppdatera high-score i ett spel

Varför?

Algoritm-exempel: Highscore

Problem: Uppdatera high-score i ett spel

Varför? Så att de som spelar uppmuntras att spela mer :)

Algoritm:

Algoritm-exempel: Highscore

Problem: Uppdatera high-score i ett spel

Varför? Så att de som spelar uppmuntras att spela mer :)

Algoritm:

1. *points* $\leftarrow$ poängen efter senaste spelet
2. *highscore* $\leftarrow$ bästa resultatet innan senaste spelet
3. **`om`** *points* är större än *highscore*
   Skriv ”Försök igen!”
   `annars`
   Skriv ”Grattis!”

Algoritm-exempel: Highscore

Problem: Uppdatera high-score i ett spel

Varför? Så att de som spelar uppmuntras att spela mer :)

Algoritm:

1. *points* $\leftarrow$ poängen efter senaste spelet
2. *highscore* $\leftarrow$ bästa resultatet innan senaste spelet
3. **om** *points* är större än *highscore*
   Skriv ”Försök igen!”
   annars
   Skriv ”Grattis!”

Hittar du buggen?

Algoritm-exempel: Highscore

```
1  import java.util.Scanner;
2
3  public class HighScore {
4      public static void main(String[] args){
5          Scanner scan = new Scanner(System.in);
6          System.out.println("Hur många poäng fick du?");
7          int points =  scan.nextInt();
8          System.out.println("Vad var higscore före senaste spelet?");
9          int highscore = scan.nextInt();
10         if (points > highscore) {
11             System.out.println("GRATTIS!");
12         } else {
13             System.out.println("Försök igen!");
14         }
15     }
16 }
```

Det finns en bugg i denna implementation. Vilken?
Fanns buggen redan i algoritmdesignen?

Abstraktion – varför?

1. Dela upp problem i delproblem
2. Skapa ”byggblock” av kod som kan återanvändas
3. Dölja komplexiteten i lösningar
4. Abstraktion är själva **essensen i all programmering**

```
public static void main(String[] args){
 askUser();
 updateHighscore();
}
```

Kolla hela programmet här:
https://github.com/bjornregnell/lth-eda016-2015
i filen:

`lectures/examples/terminal/highscore/HighScoreAbstraction.java`

Vår första algoritmkluring: SWAP

Problem: läs in och byt plats på två tal i minnet

Vår första algoritmkluring: SWAP

Problem: läs in och byt plats på två tal i minnet

Algoritm:

1. skapa en Scanner
2. läs in x
3. läs in y
4. Skriv ut x och y
5. byt plats på värdena mellan x och y
6. Skriv ut x och y

Varför kan det vara bra att kunna byta plats på olika värden?

Steg 5 är egentligen en **abstraktion** av själva problemet SWAP, som inte är så lätt som det verkar och behöver delas upp i flera steg för att det ska vara rakt fram att översätta till exekverbar kod i t.ex. Java.

Vår första algoritmkluring: SWAP

```
1  import java.util.Scanner;
2
3  public class SwapQuest {
4      public static void main(String[] args){
5          //Steg 1: skapa en Scanner
6          Scanner scan = new Scanner(System.in);
7
8          int x = scan.nextInt();  //Steg 2: läs in x
9          int y = scan.nextInt();  //Steg 3: läs in y
10
11         //Steg 4: Skriv ut x och y
12         System.out.println("x: " + x + "  y: " + y);
13
14         //Steg 5: byt plats på värdena mellan x och y  HUR???
15         // ... skriv SWAP-satser här ...
16         //Steg 6: Skriv ut x och y
17         System.out.println("x: " + x + "  y: " + y);
18     }
19 }
```

Vår första algoritmkluring: SWAP

```
1  import java.util.Scanner;
2
3  public class SwapSolution {
4      public static void main(String[] args){
5          Scanner scan = new Scanner(System.in);
6
7          int x = scan.nextInt();
8          int y = scan.nextInt();
9
10         System.out.println("x: " + x + "  y: " + y);
11
12         int temp = x;
13         x = y;
14         y = temp;
15
16         System.out.println("x: " + x + "  y: " + y);
17     }
18 }
```

Övning: Rita hur minnet ser ut efter respektive raderna 7, 8, 12, 13, 14

Loop-strukturer

Mitt första program: en oändlig loop

```
10 print "hej"
20 goto 10
```

Mitt första program: en oändlig loop

```
10 print "hej"
20 goto 10
```


```
hej
hej
hej
hej
hej
hej
hej
hej
hej
hej
hej
hej
hej
<Ctrl+C>
```

Repetition med **while**-sats

```
1  public class InfiniteLoop {
2
3      public static void main(String[] args){
4
5          while (true) {
6              System.out.println("Hej!");
7          }
8
9      }
10 }
```

Repetition med `while`-sats

```
1  public class InfiniteLoop {
2
3      public static void main(String[] args){
4
5          while (true) {
6              System.out.println("Hej!");
7          }
8
9      }
10 }
```

- En av de saker en dator är *extra* bra på är att göra samma sak om och om igen utan att tröttna!
  Och det är ju människor *extra* dåliga på :)
- Med klockfrekvens i storleksordningen 10^9 Hz är det ganska många instruktioner som kan göras per sekund...

Oändlig **`while`**-loop med räknare

```
1  public class InfiniteLoopWithCounter {
2
3      public static void main(String[] args){
4
5          int i = 0;
6          while (true) {
7              System.out.println("Hej " + i);
8              i = i + 1;
9          }
10
11     }
12 }
```

Ändlig **while**-loop med räknare

```
1  public class FiniteWhileLoopWithCounter {
2
3      public static void main(String[] args){
4
5          int i = 0;
6          while (i < 5000) {
7              System.out.println("Hej " + i);
8              i = i + 1;
9          }
10
11     }
12 }
```

`for`-loop med räknare

```
1  public class ForLoopWithCounter {
2
3      public static void main(String[] args){
4
5          for (int i = 0; i < 5000; i =  i + 1){
6              System.out.println("Hej " + i);
7              i = i + 1;
8          }
9
10     }
11 }
```

Denna sats är ekvivalent med **`while`**-satsen på föregående bild.[1]

[1] Förutom att variabeln `i` finns efter **`while`**-satsen men *inte* efter **`for`**-satsen

Ändlig **while**-loop med timer

```
1  public class LoopWithTimer {
2
3      public static void main(String[] args){
4
5          long startTime = System.currentTimeMillis();
6          int i = 0;
7          int max = 5000;
8          while (i < max) {
9              System.out.println("Hej " + i);
10             i = i + 1;
11         }
12         long stopTime = System.currentTimeMillis();
13         long duration = stopTime-startTime;
14         System.out.println(
15             "Det tog " + duration +
16             " ms att räkna till " + max);
17     }
18 }
```

Övning: Skriv om till **for**-loop och kolla om den är lika snabb som **while**

Algoritm: MIN/MAX

Problem: hitta största talet

Algoritm: MIN/MAX

Problem: hitta största talet

Algoritm:

1. *scan* ← en Scanner som läser det användaren skriver
2. *maxSoFar* ← ett heltal som är *mindre* än alla andra heltal
3. **sålänge** det finns fler heltal att läsa:
   x ← läs in ett heltal med hjälp av *scan*
   om *x* är större än *maxSoFar*
   maxSoFar ← *x*
4. skriv ut *maxSoFar*

Övning 1: Kör algoritmen med papper och penna med indata:
`0 41 1 45 2 3 4`

Övning 2: skriv om så att algoritmen istället hittar *minsta* talet.

Övning: Implementera algoritmen MIN/MAX i Java

Några ledtrådar:

1. Man kan få det minsta heltalet med `Integer.MIN_VALUE` (negativt värde)
2. Man kan få det största heltalet med `Integer.MAX_VALUE`
3. Dokumentation av klassen Scanner finns här: `https://docs.oracle.com/javase/8/docs/api/`
4. Man kan kolla om det finns mer att läsa med `scan.hasNextInt()`
5. Man läser nästa heltal med `scan.nextInt()`

Googlingstävling 1: Vem hittar först största Double-värdet i Java?

Googlingstävling 2: Vem hittar först minsta Double-värdet i Java?

Varför kodstruktur?

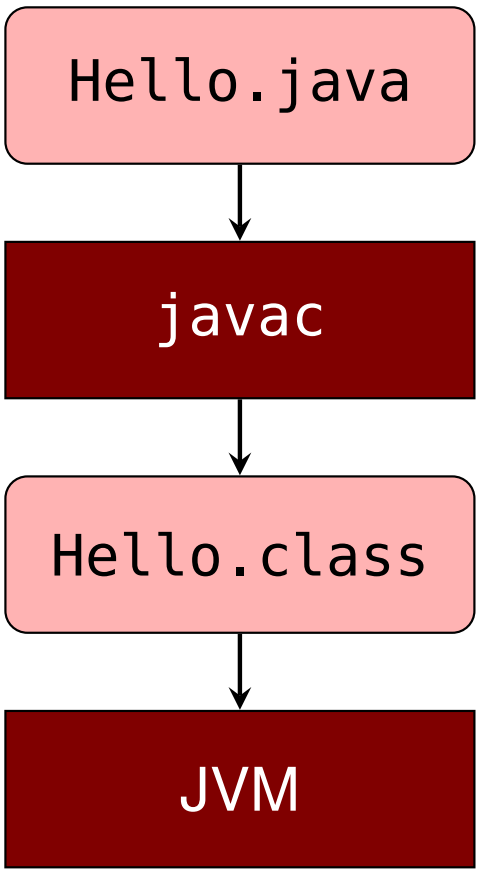
- Stora programdelar behöver delas upp annars blir det mycket svårt att förstå programmet.
- Vi behöver kunna välja namn på saker i koden *lokalt*, utan att det krockar med samma namn i andra delar av koden.
- Abstraktioner hjälper till att hantera och kapsla in komplexa delar så att de blir enklare att använda om och om igen. Exempel på **abstraktionsmekanismer** i Java:
  - Paket används för att organisera koddelar som samverkar i en hierarkisk katalogstruktur
  - Klasser används för att skapa ”byggblock” med kod, s.k. objekt, som innehåller delar som hör ihop
  - Metoder är programdelar finns i klasser och används för att lösa specifika uppgifter. Exempel: `updateHighScore()`

Exempel på olika sorters metoder:

- **Procedurer** är metoder som *gör* något men som inte returnerar något värde. Avsaknad av värde anges i Java med nyckelordet `void`
- **Funktioner** är metoder som beräknar och *returnerar* ett specifikt värde av en viss typ (gärna ngt som övriga programdelar har nytta av...)
- En funktion *utan* **sidoeffekter** ger alltid samma resultat varje gång den anropas med samma parametrar och den *ändrar inte* något som märks ”utanför” funktionen.
- En funktion *med* sidoeffekter returnerar ett värde men kan också göra något som påverkar **tillståndet** hos programdelar utanför funktionen och ger *inte* garanterat samma resultat varje gång den anropas med samma parametrar.

Filstruktur

Källkodsfiler och klassfiler



Källkodsfil

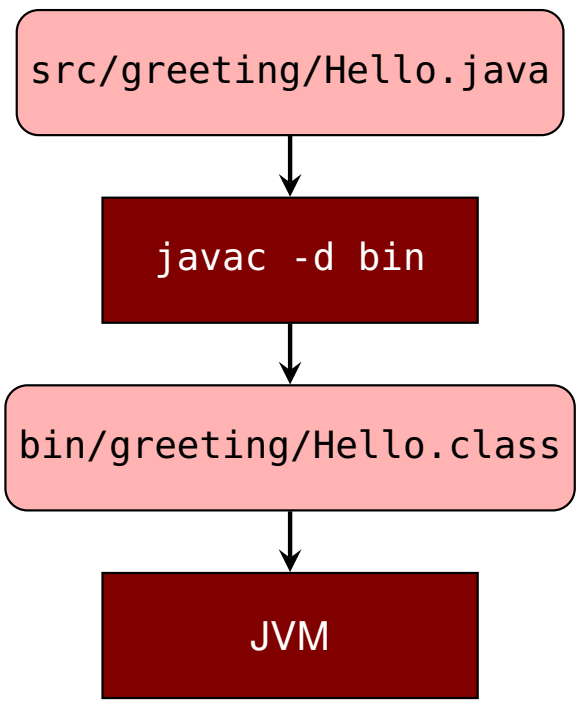
Kompilator

Fil med byte-kod

Java Virtual Machine
Översätter till maskinkod
som passar din specifika CPU
medan programmet kör

Paket

Katalogstrukturen för källkoden måste i Java motsvara paketstrukturen.
Byte-koden placeras av kompilatorn i katalogstruktur enligt paketstrukturen.




```
package greeting;
public class Hello { ...
```

Paketens bytekod hamnar i katalog med samma namn som paketnamnet

Dokumentation

Objekt

Objekt och referensvariabler

Ni har redan på övning 1 och labb 1 skapat ett objekt och deklarerat en referensvariabel som refererar till objektet:

```
Scanner scan = new Scanner(System.in);
```