EDA016 Programmeringsteknik för D

Läsvecka 2: Kodstruktur

Björn Regnell

Datavetenskap, LTH

Lp1-2, HT 2015

2 Kodstruktur

- Att göra denna vecka
- Algoritmer
- Loop-strukturer
- Abstraktionsmekanismer
- Filstruktur
- Klasser och objekt

Att göra i Vecka 2: Fatta kodstruktur

1. Läs följande kapitel i kursboken:
   2.1-2.6, 4, 5.4, 7.2, 7.5-7.6, 7.8-7.9; Begrepp:
   algoritm, pseudokod, abstraktion, oändlig loop,while-sats,
   for-sats, paket, import, referensvariabel, objekt,
   referenstilldelning, referenslikhet
2. Gör övning 2: Paket, kodfiler, och dokumentation
3. OBS! Ingen lab denna vecka
4. Träffas i samarbetsgrupper och hjälp varandra att förstå
5. Gör klart samarbetskontrakt och visa för handledare på resurstid
6. **Koda på resurstiderna** och få hjälp och tips!
   Varför var de så få kom kom till resurstiderna vecka 1?

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?

- Dela upp problem i delproblem
- Skapa ”byggblock” av kod som kan återanvändas
- Dölja komplexiteten i lösningar
- 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 på ABC80

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

Mitt första program: en oändlig loop på ABC80

```
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 }
```

Ändlig 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 föregående **`while`**-sats.[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 med abstraktion?

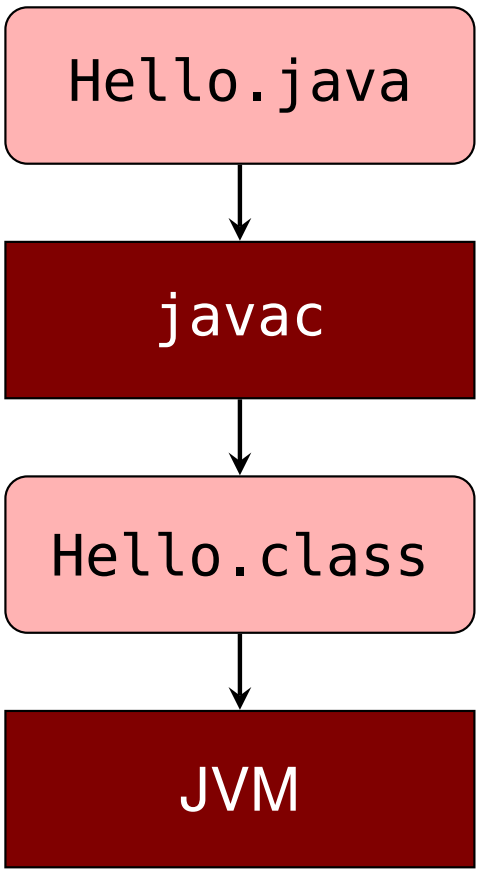
- Stora programdelar behöver delas upp annars blir det mycket svårt att förstå programmet.
- Vi behöver kunna välja namn på saker i koden *lokalt*, utan att det krockar med samma namn i andra delar av koden.
- Abstraktioner hjälper till att hantera och kapsla in komplexa delar så att de blir enklare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



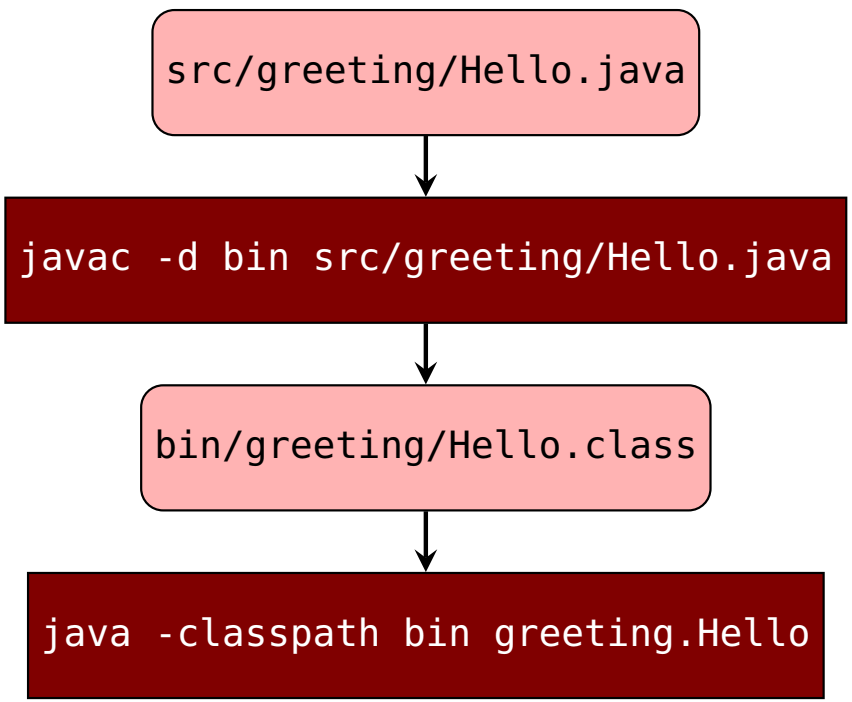
Källkodsfil

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

Import

Med hjälp av punktnotation kommer man åt klasser i ett paket.
`SimpleWindow` ligger i paketet `window` som ligger i paktetet `pt` ...

```
SimpleWindow w = new se.lth.cs.pt.window.SimpleWindow(200, 200, "");
```

En **`import`**-sats i början av kodfilen:

```
import se.lth.cs.pt.window.SimpleWindow;
```

...gör så att kompilatorn ”ser” klassnamnet, och man slipper skriva paketsökvägen före klassnamnet:

```
SimpleWindow w = new SimpleWindow(200, 200, "");
```

Man säger att namnet `SimpleWindow` hamnar ***in scope***.

Dokumentation

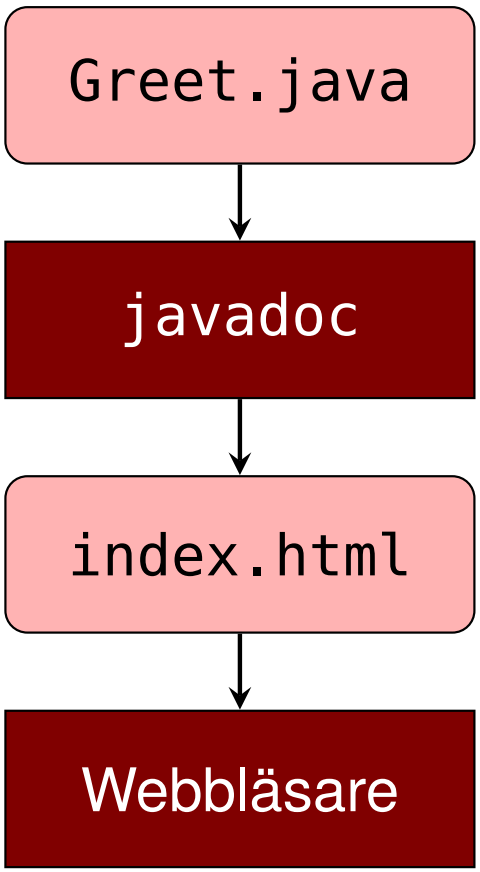
För att kod ska bli begriplig för människor är det bra att dokumentera vad den gör. Det finns tre sorters kommentarer man kan skriva direkt i Java-koden, som kompilatorn struntar fullständigt i när den gör byte-kod:

```
// Enradskommentarer börjar med dubbla snedstreck
//       men de gäller bara till radslut


/* Flerradskommentarer börjar med snedstreck-asterisk
   och slutar med asterisk-snedstreck.
*/


/** Dokumentationskommentarer placeras före en klass
*   eller en metod och berättar vad som görs
*   och vad eventuella parametrar används till.
*   Börjar med snedstreck-asterisk-asterisk.
*   Varje ny kommentarsrad börjar med asterisk.
*   Avslutas med asterisk-stjärna.
*/
```

Exempel på dokumentationskommentarer

Exempel på dokumnetationskommentar som ges i **övning 2** i kompendiet finns här:
https://github.com/bjornregnell/lth-eda016-2015/blob/master/lectures/examples/terminal/greet
i filen Greet.java

Generera dokumentation för webbläsare



Källkodsfil med dokumentationskommentarer

...och en massa andra filer...

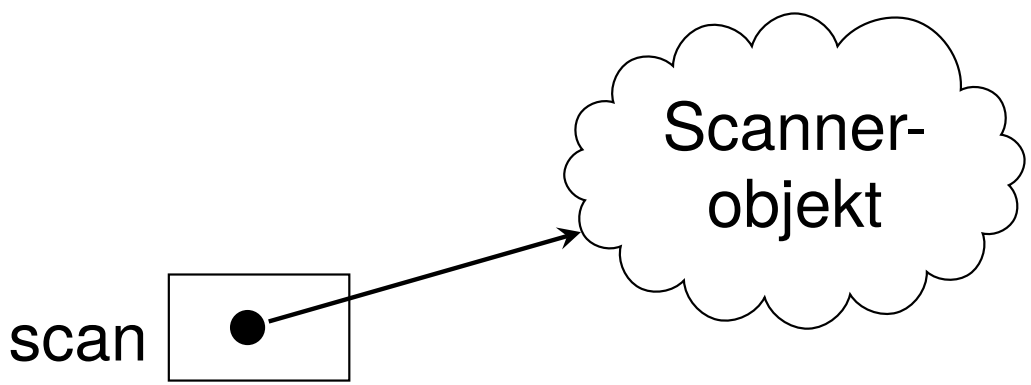
Klasser och objekt

Objekt och referensvariabler

Ni har redan på övning 1 och lab 1 skapat ett objekt av typen `Scanner` och deklarerat en **referensvariabel** `scan` som refererar till objektet.

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

Nyckelordet **`new`** skapar plats någonstans i minnet för objektet:



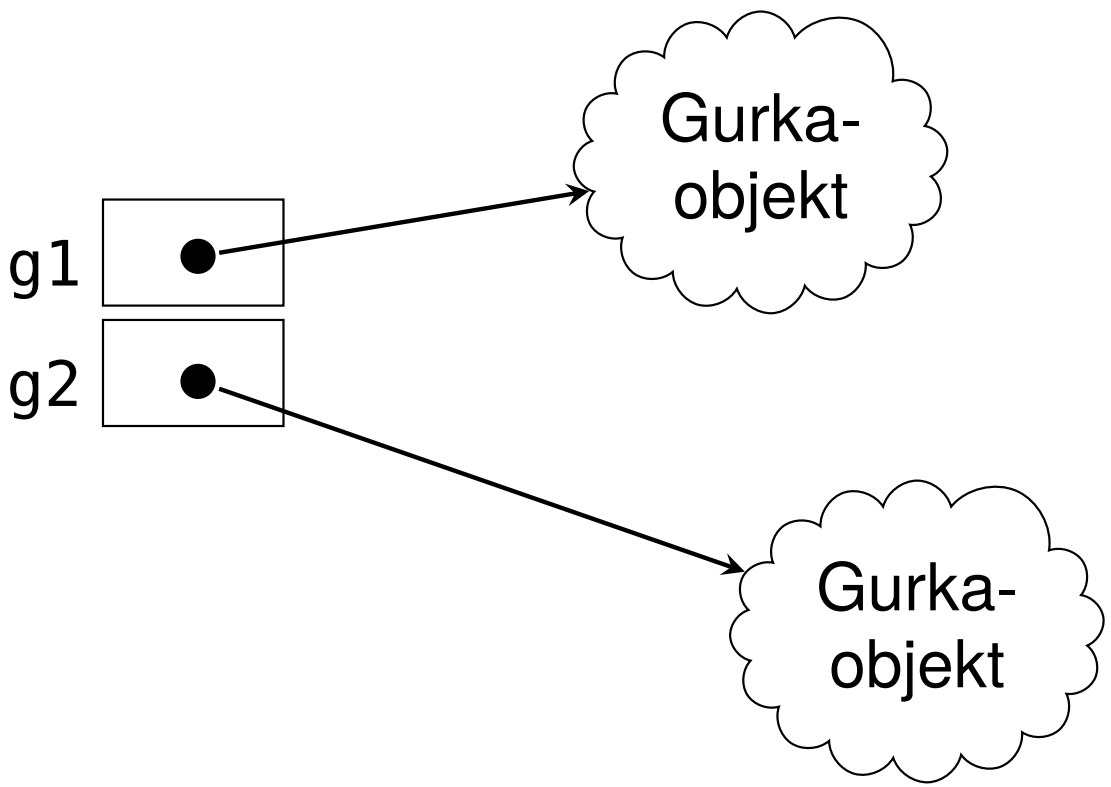
Objekt och referensvariabler

```
1  class Gurka {
2      public int vikt = 100; //gram
3  }
4
5  public class ReferenceVariables {
6      public static void main(String[] args){
7          Gurka g1 = new Gurka();
8          Gurka g2 = new Gurka();
9          g2.vikt = 200;
10         System.out.println("Gurka 1 väger: " + g1.vikt + "g");
11         System.out.println("Gurka 2 väger: " + g2.vikt + "g");
12         g1.vikt = 200;
13         System.out.println("Gurka 1 väger nu: " + g1.vikt + "g");
14         if (g1 == g2) {
15           System.out.println("samma");
16         } else {
17           System.out.println("olika");
18         }
19     }
20 }
```

Objekt och referensvariabler

```
7         Gurka g1 = new Gurka();
8         Gurka g2 = new Gurka();
```

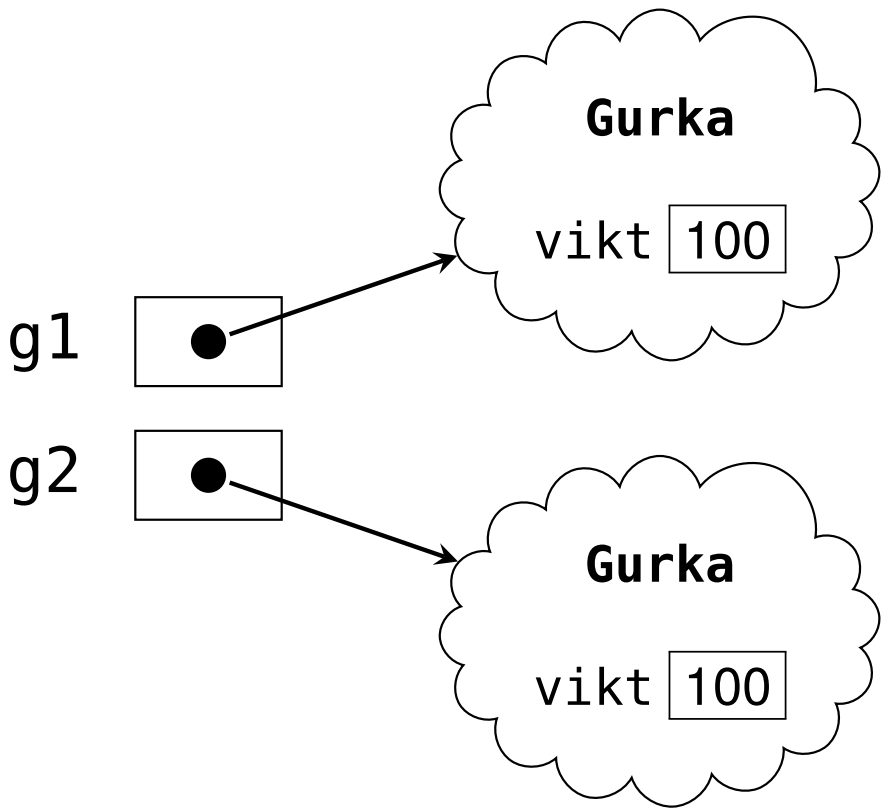
Efter rad 8 ser det ut såhär i minnet:



Objekt och referensvariabler

```
7         Gurka g1 = new Gurka();
8         Gurka g2 = new Gurka();
```

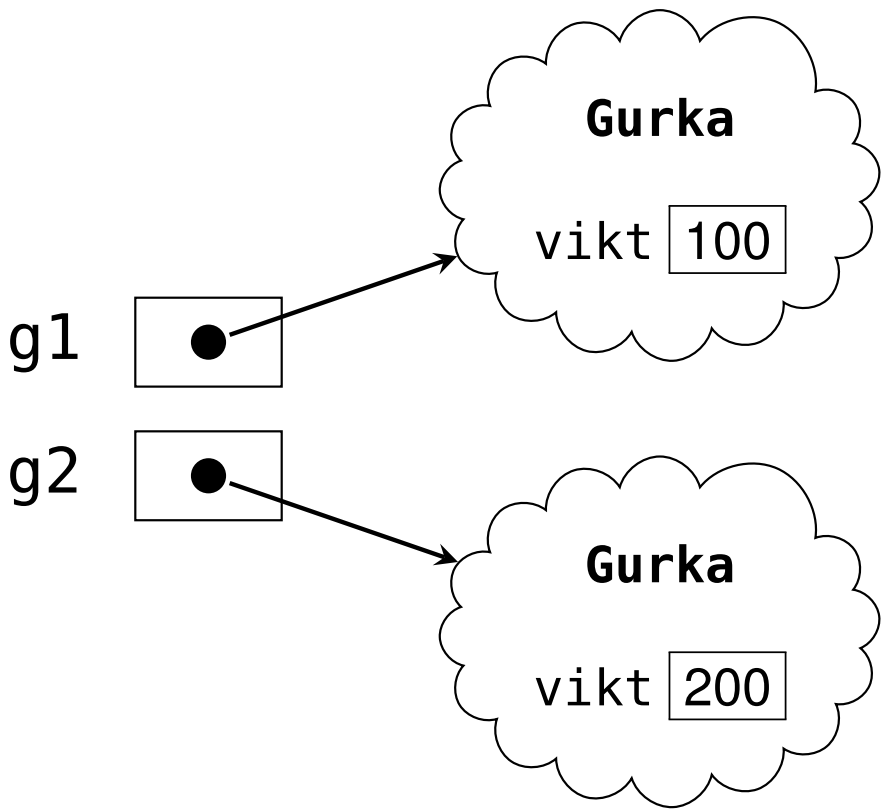
En mer detaljerad bild av minnet efter rad 8:



Punktnotation för att komma åt klassmedlemmar

```
9         g2.vikt = 200;
```

Efter rad 9 ser det ut såhär i minnet:



Deklarera och anropa metoder

```
1  class Gurka {
2      public int vikt = 100; //en variabel "överst" i en klass kallas attribut (eller fält)
3
4      public void halvera(){ //denna metod är en procedur
5          vikt = vikt / 2;
6      }
7
8      public double kilo(){  //denna metod är en funktion utan sidoeffekt
9          return vikt / 1000.0;
10     }
11
12     public void visa(){    //denna metod är en procedur
13         System.out.println("Gurkan väger " + kilo() + "kg");
14     }
15 }
16
17 public class MethodsExamples {
18     public static void main(String[] args){
19         Gurka g = new Gurka();
20         g.visa();
21         g.vikt = 256;
22         g.visa();
23         g.halvera();
24         g.visa();
25     }
26 }
```

Förhindra att variabler ändras med `final`

Attributet `latinsktNamn` nedan är en **konstant**.
Kompilatorn hjälper oss att kolla så att vi inte råkar ändra på det vi har deklarerat som **`final`**.

```
1  class Gurka {
2      public int vikt = 100; //gram
3
4      public final String latinsktNamn = "Cucumis sativus";   //*1
5
6      public String toString() {
7          return "Denna gurka (" + latinsktNamn + ") väger " + vikt + "g";
8      }
9  }
10
11 public class Constant {
12     public static void main(String[] args){
13         Gurka g = new Gurka();
14         g.vikt = 200;
15         g.latinsktNamn = "Tomat";  // ERROR: ger kompileringsfel! Vilket?
16         System.out.println(g.toString());  // *2
17     }
18 }
19
20 // *1: final brukar även deklareras static; det behövs ju bara en enda
21 // *2: .toString behövs ej
```