

# EDA016 Programmeringsteknik för D

## Läsvecka 3: Systemutveckling

Björn Regnell

Datavetenskap, LTH

Lp1-2, HT 2015

## 2 Systemutveckling

- Att göra denna vecka
- Klasser och objekt
- Metoder och parametrar
- Synlighet
- Konstruktorer
- Oföränderlighet (immutability)
- Specifikation versus implementation
- Integrerad utvecklingsmiljö

# Att göra i Vecka 3: Förstå hur systemutveckling går till med klasser och objekt i en integrerad utvecklingsmiljö

- 1 Läs följande kapitel i kursboken:  
2, 6.3, 7.1, 7.2, 7.3, 5.1, 5.2, 5.3  
Begrepp: klass, objekt, specifikation, referensvariabel, instans, IDE, arbetsområde, avlusare, brytpunkt,
- 2 Gör övning 3: beräkningar, klasser och objekt
- 3 Träffas i samarbetsgrupper och hjälp varandra förstå
- 4 Gör Lab 2: Eclipse

# Klasser och objekt

# Klasser och objekt

Några viktiga begrepp:

- En **klass** samlar variabler och kod som hör ihop.  
`class Klassnamn { /*klassmedlemmar*/ }`
- En klass är en mall som kan användas för att skapa **objekt**.  
`Klassnamn referensvariabelnamn = new Klassnamn();`
- Varje gång man gör **new** skapas en nytt objekt.  
Objektet kallas även en **instans** av klassen.
- Objektens variabler kallas **instansvariabler** och finns i en ny upplaga för varje instans och kan ha olika värden.
- Om man deklarerar en variabel **static** kallas den för **klassvariabel** och den finns i en enda upplaga som är gemensam för alla objekt.

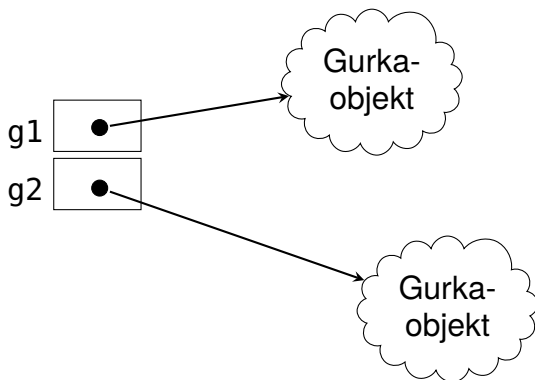
# Objekt och referensvariabler

```
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 { // g1 och g2 refererar till OLIKA objekt!
17             System.out.println("olika");
18         }
19     }
20 }
```

# Objekt och referensvariabler

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

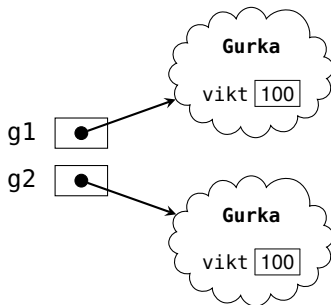
Efter rad 8 ser det ut såhär i minnet:



# Objekt och referensvariabler

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

En mer detaljerad bild av minnet efter rad 8:

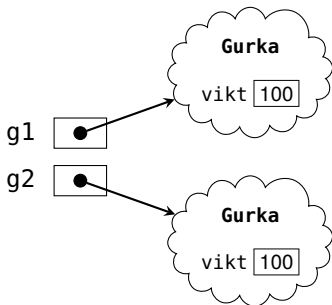




# Objekt och referensvariabler

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

En mer detaljerad bild av minnet efter rad 8:



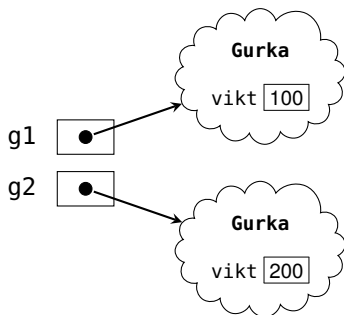
Referensvariablerna `g1` och `g2` pekar på *olika* objekt, sålunda är uttrycket `g1 == g2` **false**, även om objektens *innehåll* är lika och `g1.vikt == g2.vikt` är **true**.

# Punktnotation för att komma åt klassmedlemmar

9

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

Efter rad 9 ser det ut såhär i minnet:



# Metoder och parametrar

# Deklarera och anropa metoder

```
1  class Gurka {
2      public int vikt = 100; //en variabel i en klass kallas attribut (eller fält/field)
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 }
```

# Parametrar

```
1  class Gurka {  
2      public int vikt = 100; //gram  
3  
4      public void kapa(int gramAttKapa){ //denna metod är en procedur med parameter  
5          vikt = vikt - gramAttKapa;  
6      }  
7  
8      public void visa(){  
9          System.out.println("Gurkan väger " + vikt + "g");  
10     }  
11 }  
12  
13 public class MethodWithParameter {  
14     public static void main(String[] args){  
15         Gurka g = new Gurka();  
16         g.visa();  
17         g.kapa(75);  
18         g.visa();  
19     }  
20 }
```

# Synlighet

# Varför styra synlighet?

I storskalig systemutveckling utvecklas en stor kodbas som många människor ska läsa och bidra till...

- Då är det praktiskt med **lokala** namn som inte "krockar"; det vore *mycket opraktiskt* om man hela tiden måste hitta på globalt unika namn
- Om man kan **kapsla in** variabler så att de inte går att komma åt från andra delar av koden, förhindrar man att någon "utifrån" av misstag ändrar på en variabel och därmed sänks risken för rälige buggar

# Public, private och protected

Nyckelord för att styra **synlighet** i Java:

- Med **private** förhindrar man att namn syns "utanför" klassen
- Med **public** gör namn tillgängliga "utåt", för alla andra klasser och paket i kodbasen
- Med **protected** begränsas synligheten till egna paketet och egna subklasser
- Om man inget skriver, syns namnen i det egna paketet men inte i egna subklasser



# Gör attribut privata och slipp oönskade förändringar!

```
1  class Cucumber {
2      private int weight = 100;  // a private field
3
4      public int getWeight() {
5          return weight;
6      }
7
8      public void setWeight(int newWeight){
9          if (newWeight > 0) { // prohibit negative weight
10             weight = newWeight;
11          } else {
12             weight = 0;
13          }
14      }
15
16      public void show(){
17          System.out.println("Cucumber(" + weight + ")");
18      }
19  }
20
21  public class PrivateAttribute {
22      public static void main(String[] args){
23          Cucumber c = new Cucumber();
24          //c.weight = -42; // COMPILE TIME ERROR (What error msg?)
25          c.setWeight(-42);
26          c.show();
27      }
28  }
```

# Konstruktörer

# Använd konstruktor för att ge attribut startvärden

- När ett objekt skapas anropas en **konstruktor**
- En deklaration av en konstruktor liknar en metoddeklaration, men namnet ska vara samma som klassen och ingen returtyp får anges

```
public Klassnamn(parametrar) { ... }
```

```
public class Gurka {  
    private int vikt;  
  
    public Gurka(){ // en konstruktor utan parameter  
        vikt = 100;  
    }  
  
    public Gurka(int startVikt){ // en konstruktor med parameter  
        vikt = startVikt;  
    }  
}
```

# Exempel: implementation av klass

I filen Cucumber.java

```
1 public class Cucumber {
2     private int weight;
3
4     public Cucumber(int startWeight){ // a constructor
5         if (startWeight > 0) {
6             weight = startWeight;
7         } else {
8             startWeight = 0; // This is a bug! Why? Will the compiler help us?
9         }
10    }
11
12    public int getWeight() {
13        return weight;
14    }
15
16    public void eat(int bite){
17        if (bite <= weight) {
18            weight = weight - bite;
19            System.out.println("Eating " + bite + " grams");
20        } else {
21            weight = 0;
22            System.out.println("Ate the last bit!");
23        }
24    }
25
26    public void show(){
27        System.out.println("Cucumber(" + weight + ")");
28    }
29 }
```

# Exempel: test av klass-implementation

Kör main-metoden klassen CucumberTest

```
1  public class CucumberTest {  
2      public static void main(String[] args){  
3          Cucumber c = new Cucumber(42);  
4          c.show();  
5          c.eat(40);  
6          c.show();  
7          c.eat(40);  
8          c.show();  
9      }  
10 }
```

- Vad skriver programmet ut?
- Kommer detta testprogram hitta buggen i klassen Cucumber?
- Lägg till kod som testar den felaktiga koden i konstruktorn i Cucumber
- Hur vet vi vad som är "rätt" beteende?

# Exempel: test av klass-implementation

Kör main-metoden klassen CucumberTest

```
1  public class CucumberTest {  
2      public static void main(String[] args){  
3          Cucumber c = new Cucumber(42);  
4          c.show();  
5          c.eat(40);  
6          c.show();  
7          c.eat(40);  
8          c.show();  
9      }  
10 }
```

- Vad skriver programmet ut?
- Kommer detta testprogram hitta buggen i klassen Cucumber?
- Lägg till kod som testar den felaktiga koden i konstruktorn i Cucumber
- Hur vet vi vad som är "rätt" beteende? Kolla specifikationen (om sådan finns)

## Exempel: En specifikation av klassen Cucumber

Så här vill författaren av denna specifikation att vår gurk-klass ska fungera:

### Cucumber

```
/** Skapar en gurka som väger startWeight gram.  
    Om startWeight är negativt blir gurkans vikt 0. */  
Cucumber(int startWeight);  
  
/** Returnerar gurkans vikt i gram */  
int getWeight();  
  
/** Minskar gurkans vikt med bite gram */  
void eat(int bite);  
  
/** Skriver ut gurkans vikt */  
void show();
```

Observera att implementationsdetaljer inte visas, t.ex. namnet på det privata attributet.

# Komma runt överskuggning med **this**

```
public class Gurka {  
    private int vikt;  
  
    public Gurka(int vikt){//parameterns namn krockar med attributet  
        this.vikt = vikt;  
    }  
}
```

Nyckelordet **this** ger oss en referens till instansen.  
Med punktnotation kan vi komma åt attributet vars namn  
överskuggas av parameternamnet.



# Lokala namn, överskuggning och **this**

Vad händer här? OBS! Tre *olika* variabler, men med samma namn...

```
1  class Gurka {
2      public int vikt = 100;
3
4      public void visaLokalVariabel(){
5          int vikt = 0; //ny lokal variabel med ett namn som överskuggar attributets namn
6          System.out.println("Lokal variabel vikt: " + vikt);
7          System.out.println("Attributet vikt: " + this.vikt);
8      }
9
10     public void visaVikt(){
11         System.out.println("Attributet vikt: " + vikt);
12     }
13 }
14
15 public class LocalVar {
16     public static void main(String[] args){
17         int vikt = 42; //lokal variabel i metoden main
18         Gurka g = new Gurka();
19         g.visaLokalVariabel();
20         g.visaVikt();
21         System.out.println("Lokal vikt i main: " + vikt);
22     }
23 }
```

# Aktiveringspost

# Oföränderlighet (immutability)

# 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 visa() {
7          System.out.println("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          g.visa();
17      }
18  }
19
20  // *1: final deklarereras gärna även static om det bara behövs en enda
```

# Oföränderligt objekt

```
1  class Gurka { // exempel på oföränderligt objekt (eng. immutable objekt)
2      private int vikt;
3
4      void Gurka(int vikt) {
5          this.vikt = vikt; //endast här tilldelas attributet ett värde
6      }
7
8      public Gurka halva(){ // förändrar inte denna instans, skapa ny istället
9          return new Gurka(vikt/2);
10     }
11
12     public void visa() {
13         System.out.println("Denna gurk-instans väger för alltid " + vikt + " gram");
14     }
15 }
16
17 public class ImmutableObject {
18     public static void main(String[] args){
19         Gurka g1 = new Gurka(42);
20         Gurka g2 = g1; // g1 och g2 refererar till samma objekt
21         g1 = g1.halva(); // förändringen av g1 påverkar inte g2
22         g1.visa();
23         g2.visa();
24     }
25 }
```

# Specifikation versus implementation

# Krav – Design – Implementation – Test

# Integrerad utvecklingsmiljö



# Eclipse