EDA016 Programmeringsteknik för D

Läsvecka 11: Polymorfism

Björn Regnell

Datavetenskap, LTH

Lp1-2, HT 2015

11 Polymorfism

- Att göra denna vecka
- Repetition: arv
- Polymorfism
- Definitiva metoder och klasser

Att göra i Vecka 11: Förstå arv och polymorfism.

1. Läs följande kapitel i kursboken: 9
   Begrepp: polymorfism, klassificering, polymorfa variabler, dynamisk bindning, överskugga (override), virtuell metod, definitiva metoder och klasser
2. Gör övning 10: arv
3. Träffas i samarbetsgrupper och hjälp varandra
4. Gör Lab 9: grupplabb TurtleRace

Repetition: Vad är arv? Motivering och terminologi

- Med hjälp av **arv** mellan klasser kan man göra så att en klass **ärver** (”får med sig”) innehållet i en *annan* klass.
- Varför vill man det?
  1. Dela upp ansvar mellan klasser och bryta ut gemensamma delar så att man slipper duplicerad kod.
  2. Skapa en klassificering av objekt utifrån relationen **X är en Y**.
     Exempel 1: En gurka är en grönsak. En tomat är en grönsak.
     Exempel 2: En cykel är ett fordon. En bil är ett fordon.
- Nyckelordet `extends` används för att ange arv i Java.
  Exempel: `class TalkingRobot extends Robot`
- Klassen som ärver (utökar) kallas **subklass**
- Klassen som blir utökad kallas **superklass** (även *basklass*)
- Läs mer om arv (eng. *inheritance*) här:
  https://sv.wikipedia.org/wiki/Arv_%28programmering%29

Skydd i samband med arv

Använd **`protected`** för synlighet bara i subklasser:

```
public class A {
    private int x;
    protected int y;
    public int z;
}
```

```
public class B extends A {
    // här är de ärvda attributen y och z tillgängliga,
    // x är inte tillgängligt
}
```

- Läs om skyddsregler i ankboken 9.2 och officiella java tutorial.

Konstruktorer och arv

Konstruktorn i subklassen måste **först** anropa superklassens konstruktor med **`super`**:

```
public class A {
    private int a;

    public A(int a){
        this.a = a;
    }
}
```

```
public class B extends A {
    private int b;

    public B(int a, int b){
        super(a);
        this.b = b;
    }
}
```

Abstrakt klass

En abstrakt klass får **inte** instansieras.
Vid försök blir det **kompileringsfel**:

```
public abstract class A {
    private int a;

    public A(int a){
        this.a = a;
    }
}
```

```
    A a = new A(42);  // compile error: Cannot instantiate type A
```

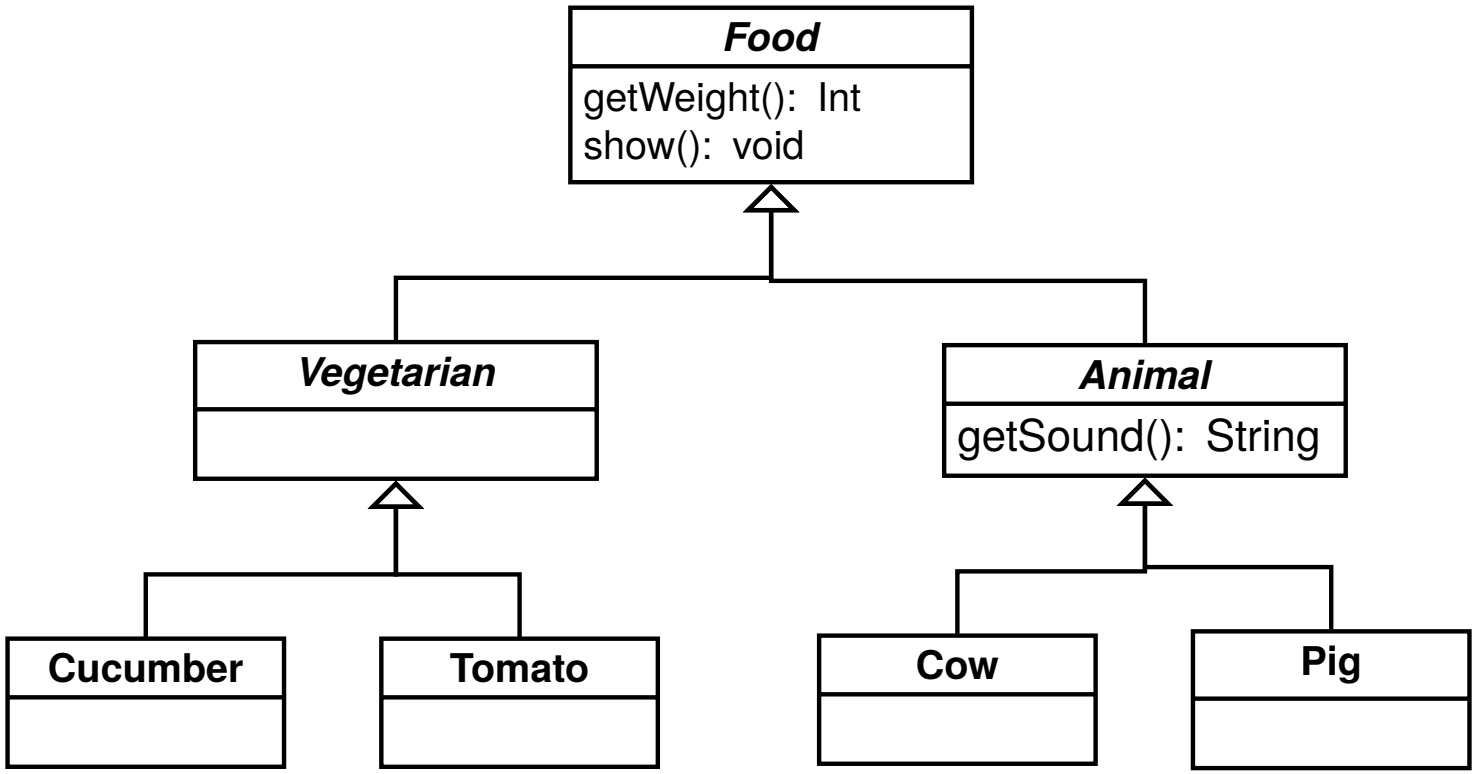
- Ska konstruktorer i abstrakta klasser vara **`public`** eller **`protected`**?
  Läs mer på SO: abstract-class-constructor-access-modifier

Polymorfism

Polymorfism betyder **många olika skepnader**.

- Det finns flera olika slags polymorfism, bland andra:
  - **Subtypning**: Variabler av en supertyp kan innehålla värden av olika subtyp. I Java används arv för att åstadkomma detta, t.ex. genom att en referensvariabel av typen `Shape` kan referera till *olika* slags grafiska objekt, så som `Polygon` och `Circle`
  - **Parametrisk polymorfism**: En metod eller klass kan göra generisk och implementeras oberoende av vilken typ som hanteras. I java, t.ex.: `ArrayList<E>`
- Läs mer här
  - svenska wikipedia
  - engelska wikipedia
  - java tutorial

Exempel på polymorfism: Klassificering av mat



Metoden `show()` förekommer **i många skepnader**, beroende på vilken konkret subklass som instansieras. Vid *körtid* avgörs vilken som anropas. Detta kallas **dynamisk bindning** och metoden `show()` kallas **virtuell**.

Den abstrakta klassen Food

```
1  package week11.polymorphism;
2
3  public abstract class Food {
4      private int weight;
5
6      public Food(int weight) {
7          this.weight = weight;
8      }
9
10     public int getWeight() {
11         return weight;
12     }
13
14     public void show() {
15         System.out.println("I am abstract Food!");
16     }
17
18     // public abstract void show();
19
20 }
```

lecture-examples/src/week11/polymorphism/Food.java

Den abstrakta klassen `Animal`

```
1  package week11.polymorphism;
2
3  public abstract class Animal extends Food {
4      private String sound;
5
6      public Animal(int weight, String sound) {
7          super(weight);
8          this.sound = sound;
9      }
10
11     public String getSound() {
12         return sound;
13     }
14
15     @Override
16     public void show() {
17         super.show();
18         System.out.println("I am abstract Animal!");
19     }
20 }
```

lecture-examples/src/week11/polymorphism/Animal.java

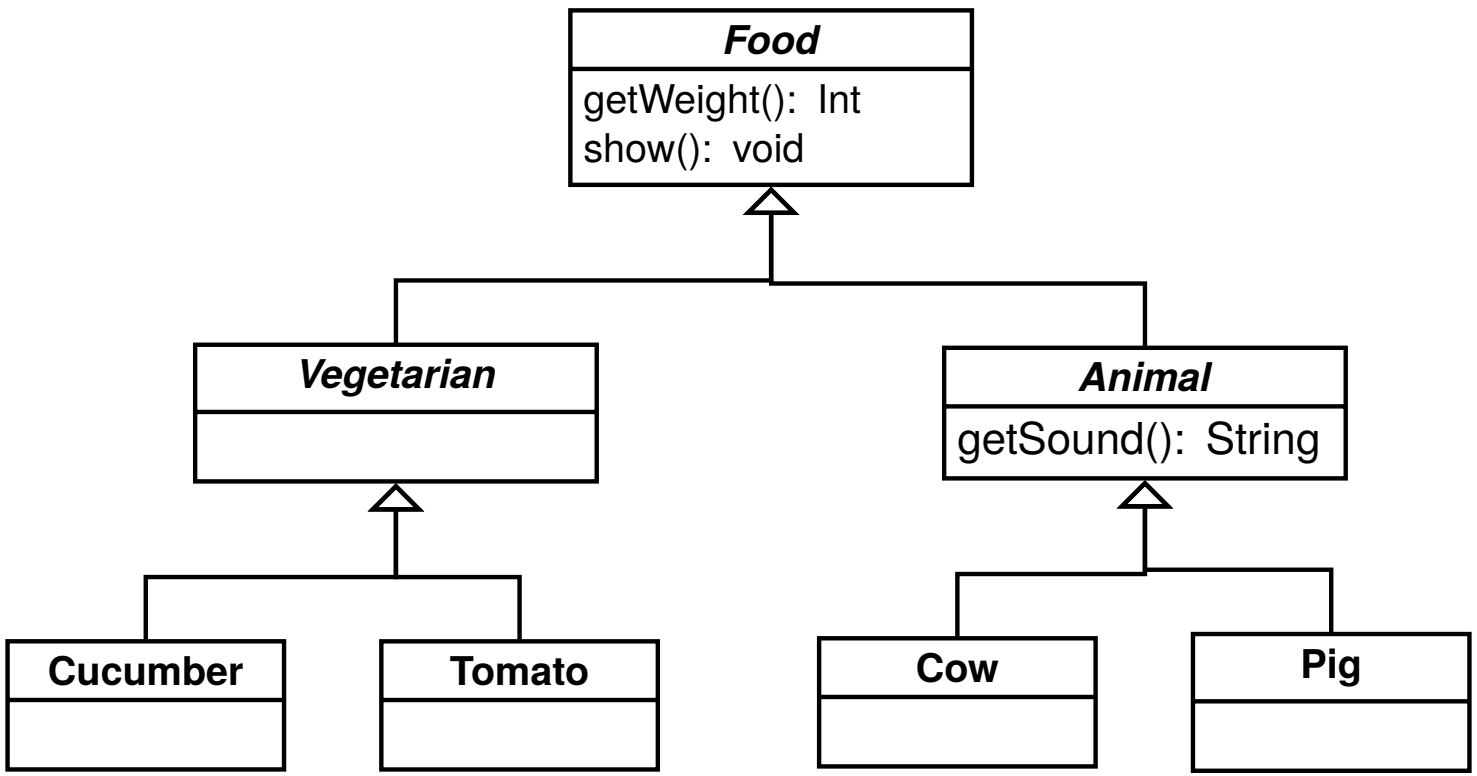
Den konkreta klassen Cow

```
1  package week11.polymorphism;
2
3  public class Cow extends Animal {
4
5      public Cow(int weight) {
6          super(weight, "Muuuu!");
7      }
8
9      @Override
10     public void show() {
11         super.show();
12         System.out.println("I am a concrete Cow!");
13     }
14
15
16 }
```

lecture-examples/src/week11/polymorphism/Cow.java

Övning på polymorfism

Övning: Med papper och penna, implementera klasserna `Vegetarian`, `Cucumber` och `Pig`. Diskutera gärna parvis hur det kommer att bli när man skapar olika slags matobjekt och anropar metoden `show()`.



Polymorfism med referensvariabler, listor och vektorer

lecture-examples/src/week11/polymorfism:

```
1  // Food f = new Food(42);   // compile error
2  Food t1 = new Tomato(42);
3  Tomato t2 = new Tomato(42);
4  // t2 = new Cucumber(42);   // compile error
5  ArrayList<Food> foodList = new ArrayList<Food>();
6  foodList.add(t1);
7  foodList.add(t2);
8  foodList.add(new Pig(84));
9  foodList.add(new Cow(168));
10 foodList.add(new Cucumber(21));
11 for (Food f: foodList){
12     f.show();
13     int weight = f.getWeight();
14     // String sound = f.getSound(); // compile error
15     System.out.println("Weight: " + weight);
16 }
17 Animal[] animalArray =
18     {new Pig(100), new Cow(500), new Pig(100)};
19 for (Animal a: animalArray){
20     String sound = a.getSound();
21     System.out.println(sound);
22 }
```

Övning: Rita minnet efter raderna 5, 10, 18

```
I am abstract Food!
I am abstract Vegetarian!
I am a concrete Tomato!
Weight: 42
I am abstract Food!
I am abstract Vegetarian!
I am a concrete Tomato!
Weight: 42
I am abstract Food!
I am abstract Animal!
I am a concrete Pig!
Weight: 84
I am abstract Food!
I am abstract Animal!
I am a concrete Cow!
Weight: 168
I am abstract Food!
I am abstract Vegetarian!
I am a concrete Cucumber!
Weight: 21
Nöff Nöff!
Muuuu!
Nöff Nöff!
```

Definitiva metoder med `final`

```
public class ChessAlgorithm {
    public static final int WHITE = 1;
    public static final int BLACK = 2;
    //...
    final int getFirstPlayer() {
        return WHITE;
    }
    //...
}
```

På grund av **`final`** före metoden får ingen subklass överskugga `getFirstPlayer()`

Se java tutorial om ”Final classes and methods”

Definitiva klasser med `final class`

```
public final class MyInteger {
    private final int value;

    public MyInteger(int value) {
        this.value = value;
    }

    public int intValue() {
        return value;
    }
}
```

På grund av **`final`** före **`class`** får ingen göra **`extends`** på denna klass och då kan vi vara helt säkra på att ingen subklass ändrar beteendet på `MyInteger`.

Regler för grupplabbar

- Diskutera i din samarbetsgrupp hur ni vill dela upp ansvaret och arbetet för olika delar av ko den. Det är lämpligt att varje klass har en huvudansvarig. Flera kan hjälpas åt med samma klass, t.e x. genom att implementera olika metoder.
- När ni redovisar er lösning ska ni också kunna redogöra för hur ni delat upp ansvar och arbete mellan er. Var och en redovisar sina delar.
- Grupplaborationer görs i huvudsak som **hemuppgift**. Salstiden används primärt för redovisning.

Diskutera gärna med handledare på resurstid om ni behöver hjälp med hur ni ska dela upp arbetet.

Morgondagens föreläsning: Grumligtlådan

Inför nästa vecka: Algoritmer

- Repetera algoritmer: min/max, linjärsökning, registrering
- Nästa vecka:
  binärsökning, sortering, algoritmisk komplexitet