Objektorienterad Programmering (TDDC77)

Föreläsning XI: åsidosättning, gränssnitt, uppräkning, hierarkier

Ahmed Rezine

IDA, Linköpings Universitet

Hösttermin 2016

Outline

Åsidosättning

Abstrakta klasser

Gränssnitt

Uppräkningar (enum)

Hierarkier

Signatur

```
public static int parseInt(String s)  throws NumberFormatException {...}
```

- En konkret metod deklareras med hjälp av sex komponenter
  - Accessmodifierare såsom public och protected
  - Returtyp eller void
  - Metodens namn (identifierare) som ska börja med liten bokstav
  - Parameterlista med noll eller fler datatyper och identifierare
  - Eventuella "checked exceptions" som metoden kastar
  - Metodens kropp inom måsvingar
- Metodens signatur betäms av metodens namns samt datatyper och ordning på parameterlistan

Åsidosättning

- En metod åsidosätter en annan om den har samma signatur, samma eller mer synlig accessmodifierare, och samma returtyp. Med samma returtyp menar man här ett av följande:
  - void
  - exakt samma returtyp (vid primitivtyp eller referenstyp)
  - en subtyp till returtypen (bara vid referenstyp)

Åsidosättning

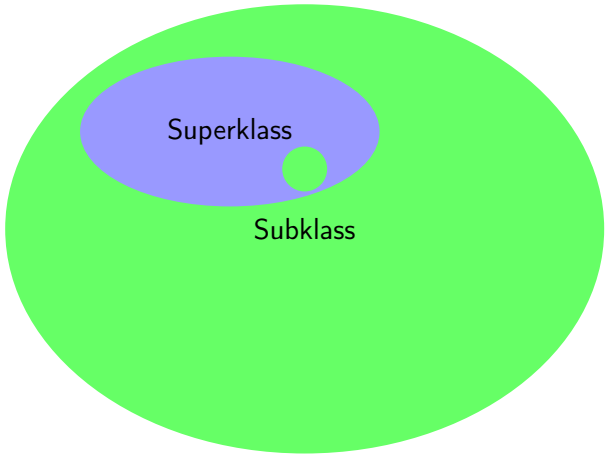
```java
class Student{
    private int points;
    private Studiemedel studiemedel;

    public int getPoints(){
     return points;
    }
}

class ITare extends Student {
    private int pointsDatavetenskap;

    public int getPoints(){
     return super.getPoints() + pointsDatavetenskap;
    }
}
```

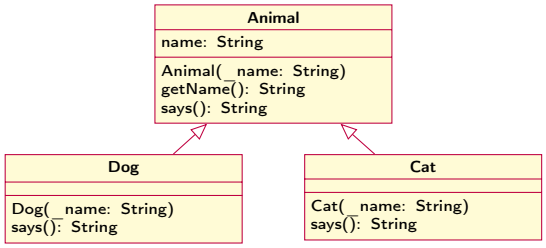
Åsidosättning



Åsidosättning

- Synliga metoder åsidosättningsbara i subklasser
- **this** och **super** skapas automatiskt i alla instansmetoder
- **this** refererar till det aktuella objektet
- **super** refererar till superklassdelen av det aktuella objektet

Arv




```java
Animal hunden = new Dog("bobby");
Animal katten = new Cat("kitty");

System.out.println(hunden.getName() + " says " + hunden.says());
System.out.println(katten.getName() + " says " + katten.says());
```

Arv från Object

- Alla objekt ärver automatiskt från **java.lang.Object**
- Några viktiga metoder som ärvs:
  - **equals()**
  - **toString()**
  - **hashCode()** - senare
  - **clone()** - senare

Outline

Åsidosättning

Abstrakta klasser

Gränssnitt

Uppräkningar (enum)

Hierarkier

Abstrakta klasser

- Abstrakta klasser kan inte instansieras

```
abstract class Person{
   private String name;

   public Person(String name){
    this.name = name;
   }

   public String getName(){
    return name;
   }
}
```


Abstrakta metoder

- Abstrakta metoder saknar "kropp" och <u>måste</u> åsidosättas i alla subklasser som inte är abstrakta
- Bara tillåtet i abstrakta klasser

```
abstract class Person{
   private String name = "";

   public Person(String name){
    this.name = name;
   }

   public String getName(){
    return name;
   }

   public abstract void greet();
}
```

Åsidosätta abstrakta metoder

```
class Man extends Person{
   publlic greet(){
    System.out.println("Mr." + getName());
   }
}
```


Outline

Åsidosättning

Abstrakta klasser

Gränssnitt

Uppräkningar (enum)

Hierarkier

Multiple arv?

- I Java får man endast ärva från en klass
- Arv från flera klasser kallas multiplelt arv, och är inte tillåtet

Gränsnitt (Interface)

- Interface är lite som helt abstrakta klasser
- De definierar vilka metoder som ska finnas, men får inte innehålla definitioner av dessa
- De kan också definiera konstanter
- Man kan inte instansiera ett interface
- En klass implementerar ett interface genom att definiera metoder för alla abstrakta metoder som finns i interfacet.
- En klass kan implementera flera interface.

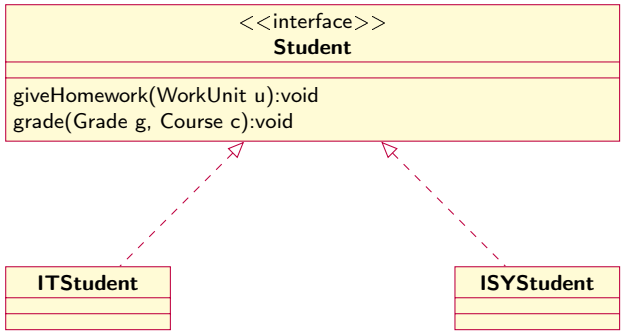
Interface

```
public interface Student{
    giveHomework(WorkUnit u);
    grade(Grade g, Course c);
}

public class ITStudent extends Person implements Student{
    ...
    // Måste implementera metoderna ovan
    ...
}
```

UML Klassdiagram för Gränsnitt

- Man brukar rita interface som klasser, men märka upp dem med **«interface»** ovanför klassnamnet
- Implementering av interface ritas som arc, men med streckad linje



Polymorfism via Interface

```
interface Talker {
  String speak();
}

class Cat implements Talker {
  private String name;

  public Cat (String name){
    this.name = name;
  }

  public String speak() {
    return name + " says Miao!";
  }
}
```

```
class Driver {

  public static void main(String[] args) {

    Talker[] assembly = new Talker[2];
    assembly[0] = new Cat("kitty");
    assembly[1] = new Dog("bobby");

    for(int i=0; i<assembly.length; i++)
      System.out.println(assembly[i].speak());

  }
}
```

Outline

Åsidosättning

Abstrakta klasser

Gränssnitt

Uppräkningar (enum)

Hierarkier

En enkel uppräkningar (enum)

```
public enum Color {red, blue, green}

String skrivUt(Color c){
  if(c == Color.red)
     return "röd";
  if(c == Color.green)
     return "grön";
  return "blå";
}
```

Uppräkningar

```
enum Seasons{
    winter ("vinter"), spring ("vår"), summer ("sommar"), autumn ("höst");

    private String name;

    Seasons(String name){
     this.name = name;
    }

    public String getName() {
     return name;
    }
}
```

```
class SeasonsDriver {

    public static void main(String[] args) {

     Seasons[] values = Seasons.values();

     for(int i=0; i < values.length; i++){
         if(values[i]==Seasons.autumn)
          System.out.println(values[i] + " : " + values[i].getName());
     }
    }
}
```

Outline

Åsidosättning

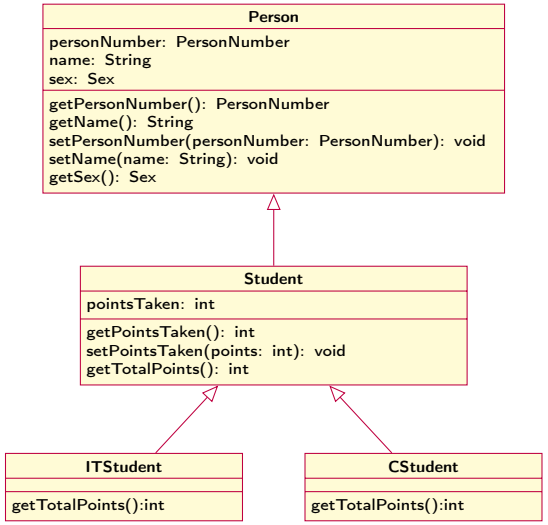
Abstrakta klasser

Gränssnitt

Uppräkningar (enum)

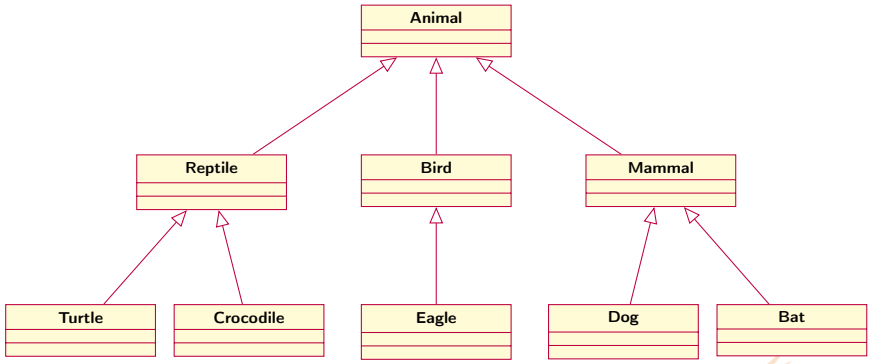
Hierarkier

Klass Hierarky

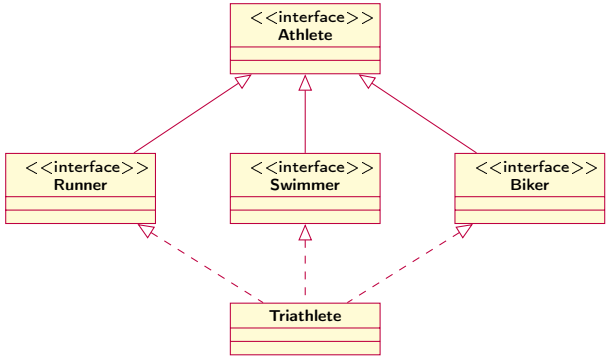


- Identifiera alla arv
- Identifiera alla inkapsling
- Identifiera alla åsidosatta metoder
- Identifiera alla överlagrade metoder
- Skriv kod för att skapa de olika klasserna
- Person kapslar in personnummer, detta kan ritas annorlunda i UML, hur?

Klass Hierarki



Interface Hierarkier




```
class Triathlete implements Runner, Swimmer, Biker {
  ...
}
```