TDDC77
Objektorienterad Programmering

Föreläsning 8

Sahand Sadjadee
IDA, Linköpings Universitet

Outline

- **Åsidosättning (overriding)**
- **Nyckelordet super(17.4)**
- **Klassen *Object***
- **Abstrakta klasser(17.6)**
- **Gränssnitt(17.7)**

Åsidosättning

LINKÖPINGS UNIVERSITET

Åsidosättning

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

- En konkret metod deklareras med hjälp av fyra komponenter
    - Returtyp eller void
    - Metodens namn (identifierare) som ska börja med liten bokstav
    - Parameterlista med noll eller fler datatyper och identifierare
    - Metodens kropp inom måsvingar
- Metodens signatur bestäms av metodens namn samt datatyper och ordning på parameterlistan

Åsidosättning

- En metod åsidosätter en annan ärvd metod om den har samma signatur.
- Det är inte tillåtet att åsidosätta en metod med en mindre synlig accessmodifierare eller mindre returtyp.

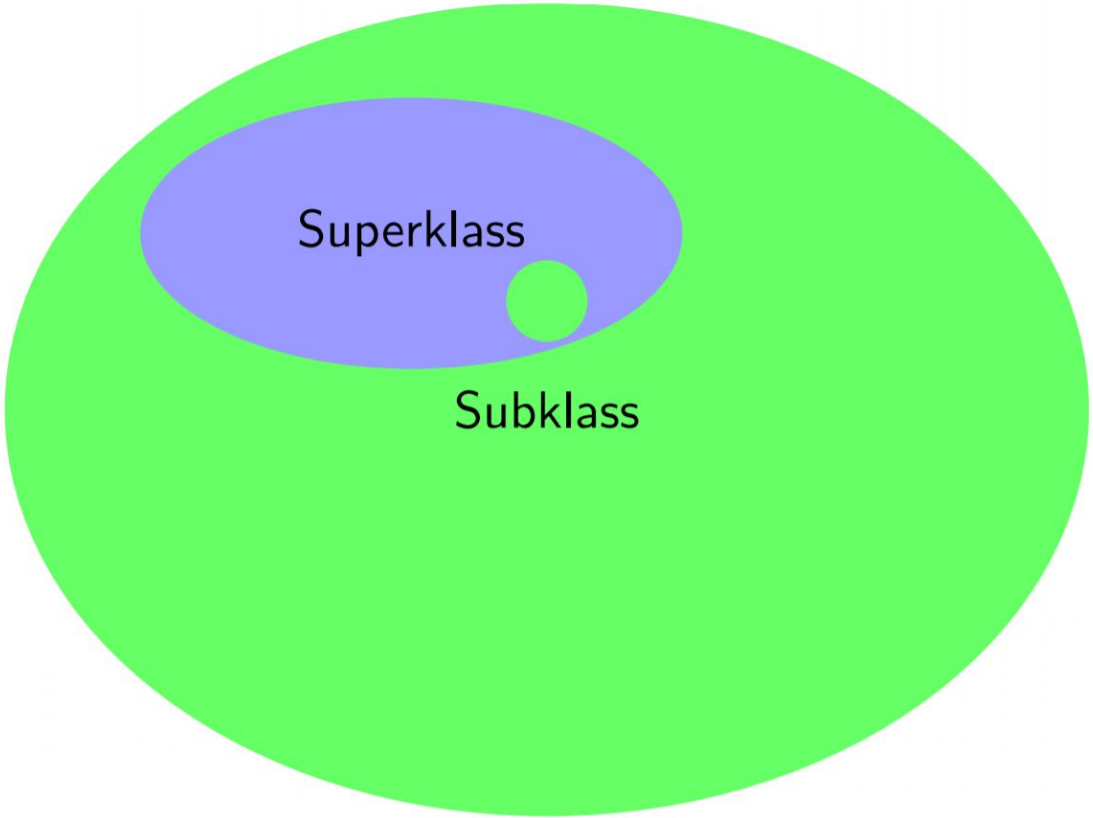
Åsidosättning

```
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

```
class ITare extends Student{

   protected int getPoints(){} //fel

   public short getPoints(){}  //fel

   protected short getPoints() //fel

   protected short getPoints(int term) //rätt?

}
```

Åsidosättning



Åsidosättning

- Synliga metoder är 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.

super

`super` kan användas för att referera till alla ärvda metoder/fält i nuvarande objektet. I sånt fall kan man använda `this` också för att ärvda metoder/fält också tillhör till nuvarande objektet.

```
class ITare extends Student{

   public int getPoints(){

       super.getAllPoints(); /*eller*/ this.getAllPoints();

}
```

super

`super` kan användas i åsidosättande metoden för att referera till ärvda metoden som åsidosätts.

```
class ITare extends Student{

   public int getPoints(){

       super.getPoins(); //varför behöver man göra ett sånt anrop?

       this.getPoints(); //det här anropet fungerar inte lika bra!

                         // Varför?

   }

}
```

`super`

Hur kan man behöva anropa/använda metoden som man åsidosätter i åsidosättande metoden?

super

`super` kan användas för att anropa konstruktorer i superklassen. Regeln är att anropet själv ska ligga i första raden i någon konstruktor i subklassen.

```
class ITare extends Student{

    Public ITare(int term){

        super(); //det anropar default konstruktorn i super
    klassen.

    }

}
```

`super`

Hur kan man behöva anropa/använda en konstruktor i superklassen?

Arv från klassen `Object`

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

https://docs.oracle.com/en/java/javase/11/

toString()

Returnerar en presentation av objektets innehåll i text.

```
class ITare extends Student{

    public String toString(){

        return “I have passed ” + this.getPoints() + “ points”;

    }

}
```

equals()

Används för att jämföra två objekt.

```
class ITare extends Student{

     public boolean equals(Object o) {
         if (o == this)
             return true;
         if (!(o instanceof ITare))
             return false;
         else if (this.getPoints() == ((ITare) o).getPoints())
             return true;
          else
               Return false;
     }
}
```

equals() och equalsIgnoreCase()

Används för att jämföra två strängar:

```
class Main{
     Public static void main(String[] args){
          String s1 = “Hej”;
          String s2 = “Hej”;
          if (s1 == s2) // false
          if (s1.equals(s2)) //true
          String s3 = “Hej”;
          String s4 = “hej”;
          if (s3 == s4) // false
          if (s3.equals(s4)) //false
          if (s3.equalsIgnoreCase(s4)) //true

     }

}
```

Abstrakta klasser

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 måste 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());
    }
}
```

Gränssnitt

Multipelt arv

- I Java får man endast ärva från en klass
- Arv från flera klasser kallas multipelt arv som inte är tillåtet i Java.

Gränsnitt (Interface)

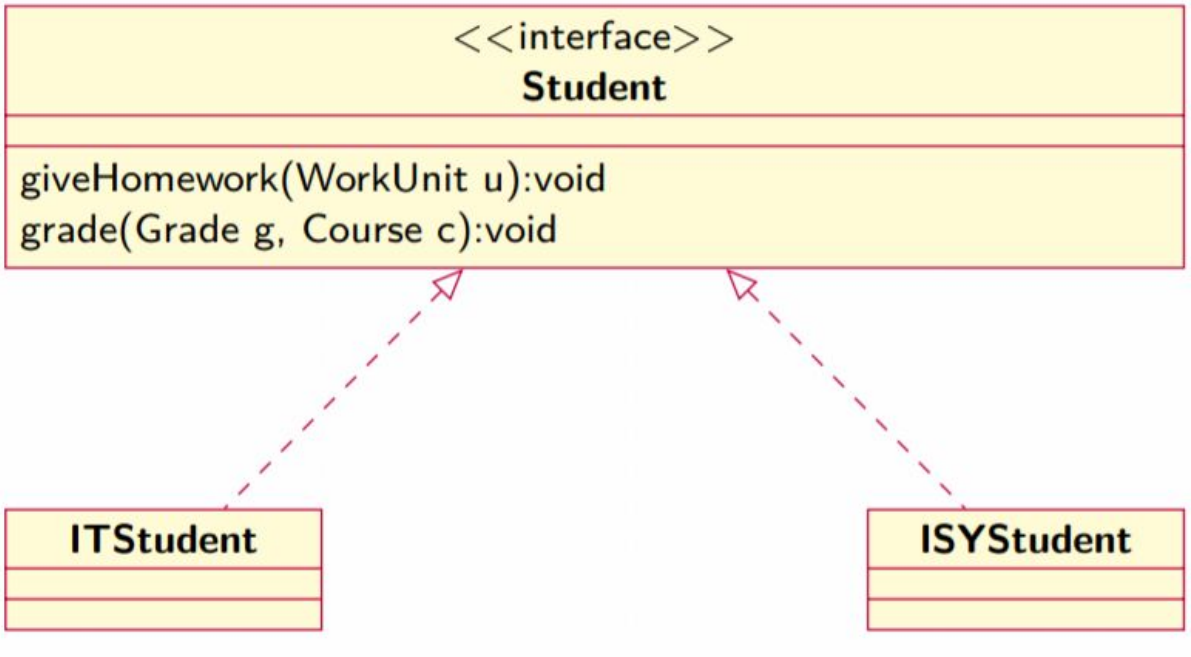
- Interface är lite som helt abstrakta klasser.
- De definierar vilka metoder som ska finnas, men får inte innehålla definitioner av dessa.
- De kan också definiera konstanter.
- Man kan inte instansiera ett interface.
- En klass implementerar ett interface genom att definiera metoder för alla abstrakta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

Gränsnitt (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 pil, men med streckad linje.



Thanks for listening!

LINKÖPINGS UNIVERSITET