Objektorienterad Programmering (TDDC77)

Föreläsning XI: åsidosättning, gränssnitt, uppräkning, hierarkier

Ahmed Rezine

IDA, Linköpings Universitet

Hösttermin 2017

Outline

Klassen calculator

Calculator
memory: double result: double
Calculator() Calculator(mem: double) add(a: double): double add(a: double, b: double): double store():void load():void

```
class Calculator{
  private double result;

  public Calculator(){
  result = 0;
  }

  public Calculator(double mem){
  result = mem;
  }

  public double add(double a){
    result += a;
    return result;
  }

  public double add(double a, double b){
    result = a + b;
    return result;
  }

  public void reset(){
    result = 0;
  }
}
```


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

Överlagring

- En metod överlagrar en annan om den har samma namn (identifierare) men olika signatur
- Returtyp spelar ingen roll för överlagring vilket innebär att två metoder med olika returtyp men samma signatur inte är tillåtet (kompilatorn kan inte skilja på dem).

Outline

Arv relationen (inheritance)

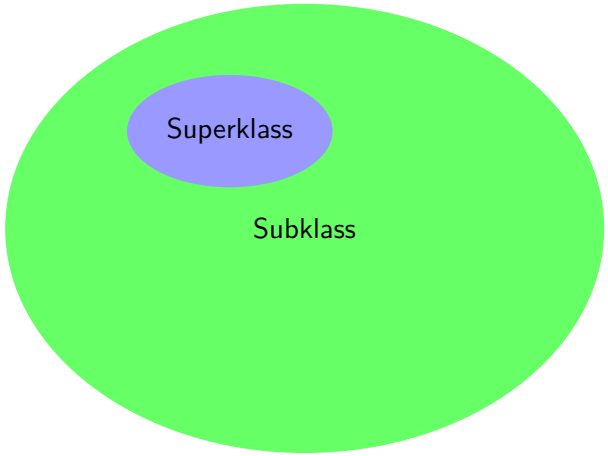
```
class Animal {
    public String name="";

    public Animal(String _name){ name = _name; }

    public String speak() { return  "hello  ...."; }
}
```

```
class Bird extends Animal {
    public Bird(String birdName){ super(birdName); }

    public String iam(){ return "I am " + name + " the bird!"; }
}
```

Arv relationen (inheritance)

```
class AnimalDriver{

    public static void main(String[] args){

     Animal djur = new Animal("Smith");
     Duck anka = new Duck("Donald");
     Bird fågel = new Bird("Pipi");

     System.out.println("Djur:");
     System.out.println(djur.speak());

     System.out.println("\nAnka:");
     System.out.println(anka.speak());
     System.out.println(anka.iam());

     System.out.println("\nFågel:");
     System.out.println(fågel.speak());
     System.out.println(fågel.iam());

    }
}
```

Arv (inheritance)



Protected eller getters och setters

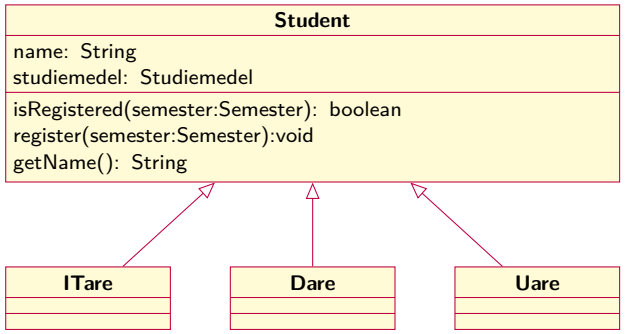
- Vill man komma åt instansvariabler i basklassen så fungerar inte det om de är deklarerade som private.
- Lösning: deklarera som protected : som private men synlig i subklasser
- Bättre lösning: använd getters och setters

Att prata om arv

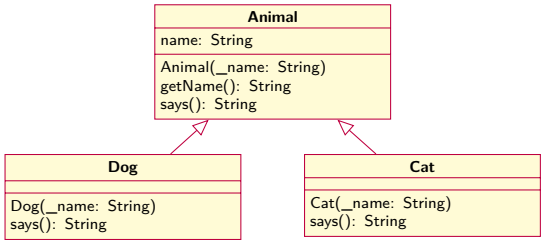
- Subklasser ärver tillstånd och tjänster från superklasser
- Ett sätt att dela på gemensam kod
- En subklass utöker eller specialiserar sin superklass
- En subklass är en subtyp till superklassens datatyp
- En superklass kan ha flera subklasser

Arv (inheritance)

- Antag att klass B ärver från klass A. Detta illustreras genom att en icke ifylld “pil” ritas från B till A



Polymorfism via Arv (inheritance)




```
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());
```


Outline

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

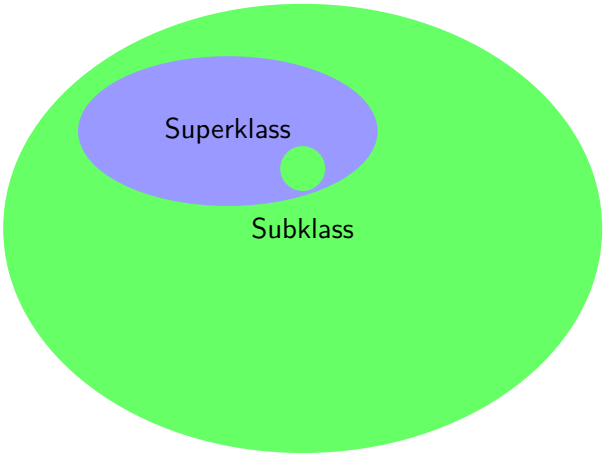
Åsidosättning

- 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

```
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;
    }
}
```

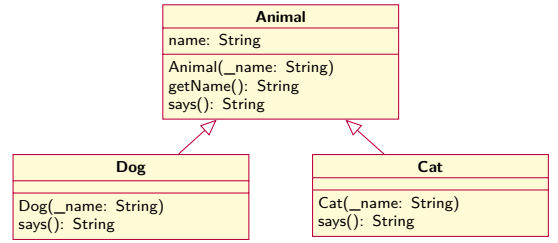

Åsidosättning



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




```
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

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());
   }
}
```


Outline

Multiple arv?

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

Gränsnitt (Interface)

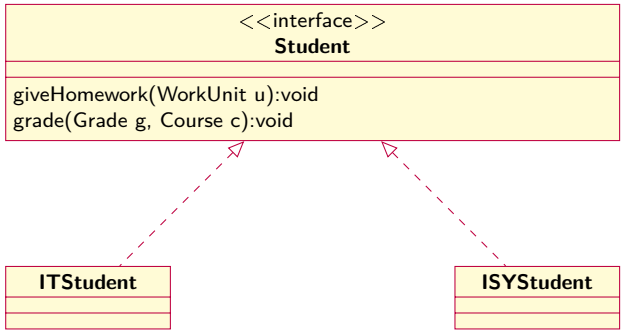
- Interface är lite som helt abstrakta klasser
- De definierar vilka metoder som ska finnas, men får inte innehålla definitioner av dessa
- De kan också definiera konstanter
- Man kan inte instansiera ett interface
- En klass implementerar ett interface genom att definiera metoder för alla abstrakta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

    }
  }
```