Objektorienterad Programmering (TDDC77)

Föreläsning XII: Interfaces, Enums, Generics, Collections

Links for: interfaces, enums, generics, collections

Ahmed Rezine

IDA, Linköpings Universitet

Hösttermin 2017

Outline

Abstrakta klasser

Gränssnitt

Uppräkningar (enum)

Hierarkier

Generics

Kollektioner

Iterable

Outline

Abstrakta metoder

- Metoder som krävs hos alla subklasser men som är meningslösa att implementera hos superklassen.

```
class Animal {
    protected String name="anonymous";

    public Animal(){}

    public void setName(String str){ name = str; }

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

```
class Duck extends Animal {
    public String speak(){ return "couac! I am " + name; }
}
```

Abstrakta klasser

- Abstrakta klasser kan inte instansieras

```
abstract class Animal {
    protected String name="anonymous";

    public Animal(){}

    public void setName(String str){ name = str; }

    public abstract String speak();
}
```

```
class Duck extends Animal {
    public String speak(){ return "couac! I am " + name; }
}
```

Abstrakta metoder

- Abstrakta metoder saknar “kropp” och måste åsidosättas i alla subklasser som inte är abstrakta
- Bara tillåtet i abstrakta klasser

```
class Duck extends Animal {
    public String speak(){ return "couac! I am " + name; }
}
```

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)

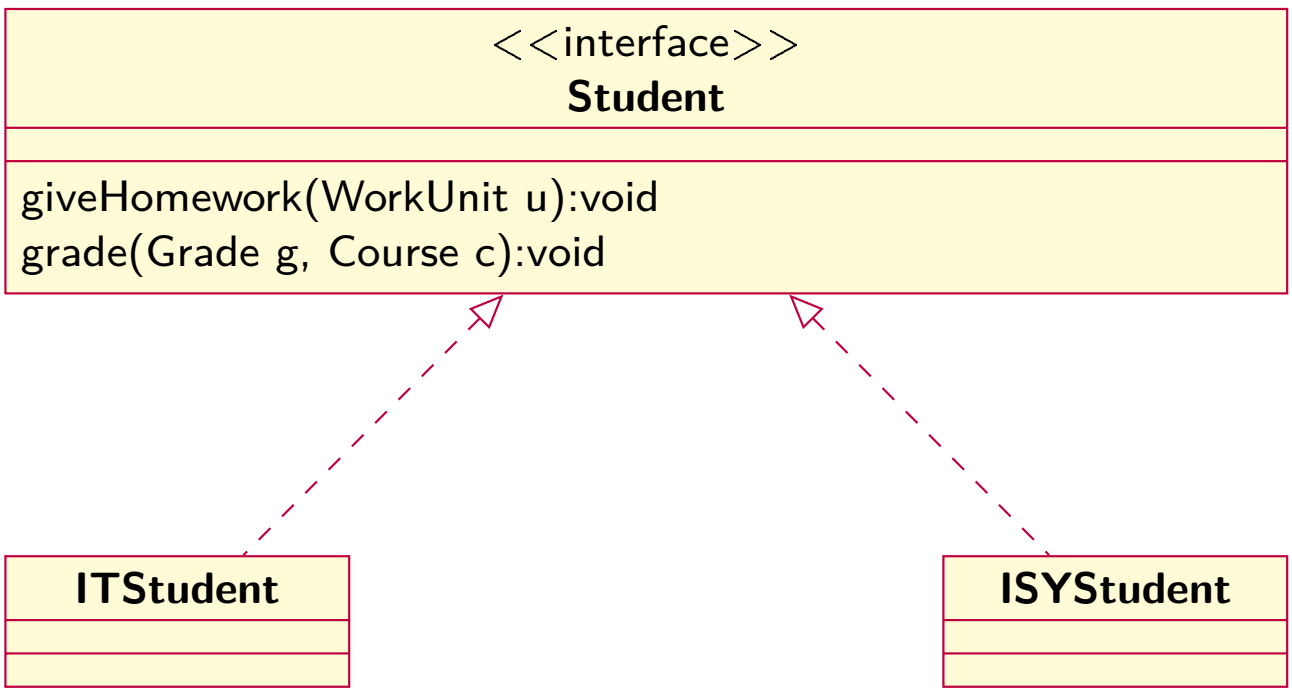
- Interface är lite som helt abstrakta klasser
- De definierar vilka metoder som ska finnas, men får inte innehålla definitioner av dessa
- De kan också definiera konstanter
- Man kan inte instansiera ett interface
- En klass implementerar ett interface genom att definiera metoder för alla abstrakta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

    }
  }
```

Outline

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";
}
```

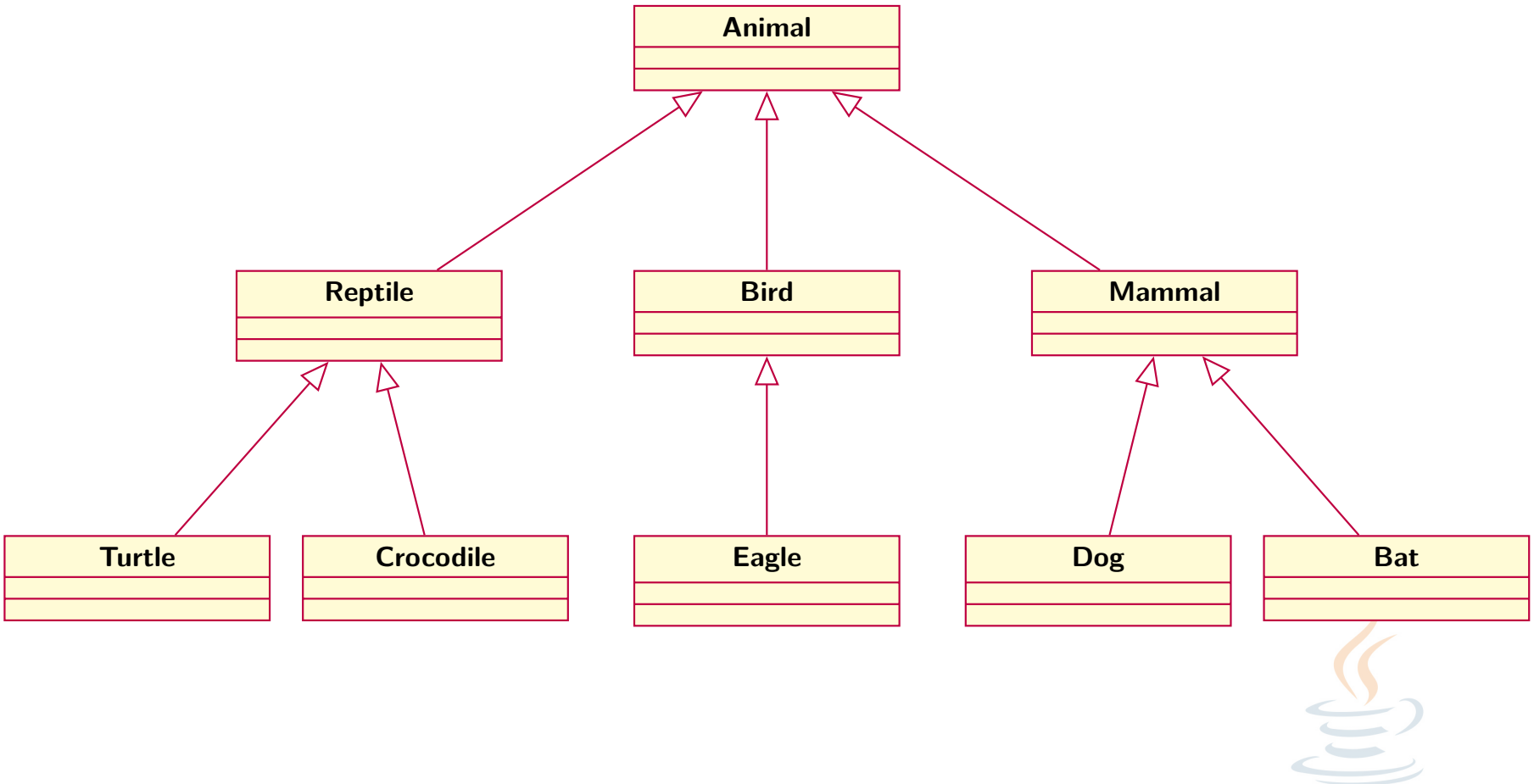
Uppräkningar

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

    private String name;

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

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

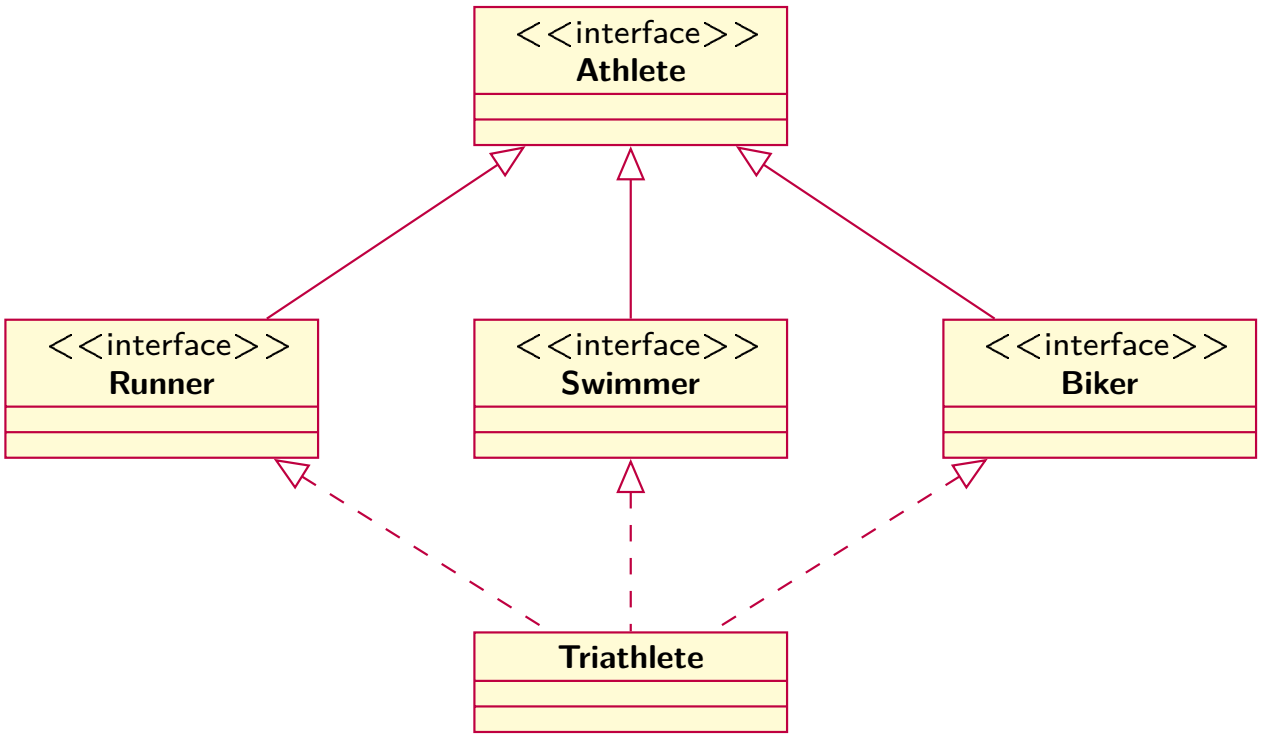
```
import java.util.Scanner;

class SeasonsDriver {

    public static void main(String[] args) {

     Seasons[] values = {Seasons.winter, Seasons.spring, Seasons.summer, Seasons.autumn}

     System.out.println("Choose among: ");
     for(int i=0; i < values.length; i++)
         System.out.println(i + ". " + values[i]);

     Scanner scan = new Scanner(System.in);
     int i = scan.nextInt();
     System.out.println("You chose: " + values[i]);
    }
}
```

Outline

Klass Hierarki



Interface Hierarkier




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

Outline

Generics

En generic typ är en klass eller ett gränsnitt som är parameteriserad med en eller flera typer.

```
class MyPair<T1, T2> {
   private T1 first;
   private T2 second;

   MyPair(T1 t1, T2 t2){
     first = t1;
     second = t2;
   }

   public String toString(){
     return "(" + first  + ", " + second + ")";
   }
 }
```

```
class MyPairDriver {
    public static void main(String[] args) {

     MyPair<String,Integer> p1 = new MyPair<String,Integer>("Monday",1);
     System.out.println(p1);

     MyPair<Integer,Integer> p2 = new MyPair<Integer,Integer>(3,7);
     System.out.println(p2);
    }
}
```

Outline

Att lagra värden

- En variabel: Data av en sort
- Problem: om det blir många variabler
- En array: En rad av data av samma sort
- Problem: Måste veta storlek i förväg, även svårt att bearbeta data i arrayen (söka, sortera osv)

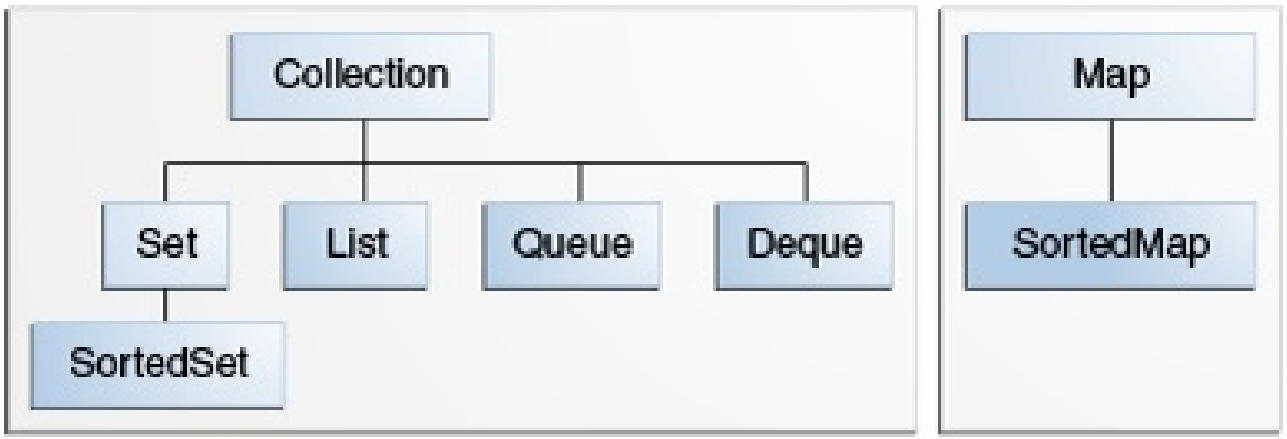
Kollektion (Collection, ibland Container)

- är ett objekt som grupperar flera element i en enda enhet.
- används för att lagra, hämta, manipulera, och kommunicera aggregerade data.
- bildar en naturlig grupp, såsom en kort hand (en samling av kort), en post mapp (en samling av brev) eller en telefonkatalog (en kartläggning av namn till telefonnummer).

Lösningen heter **Collection**

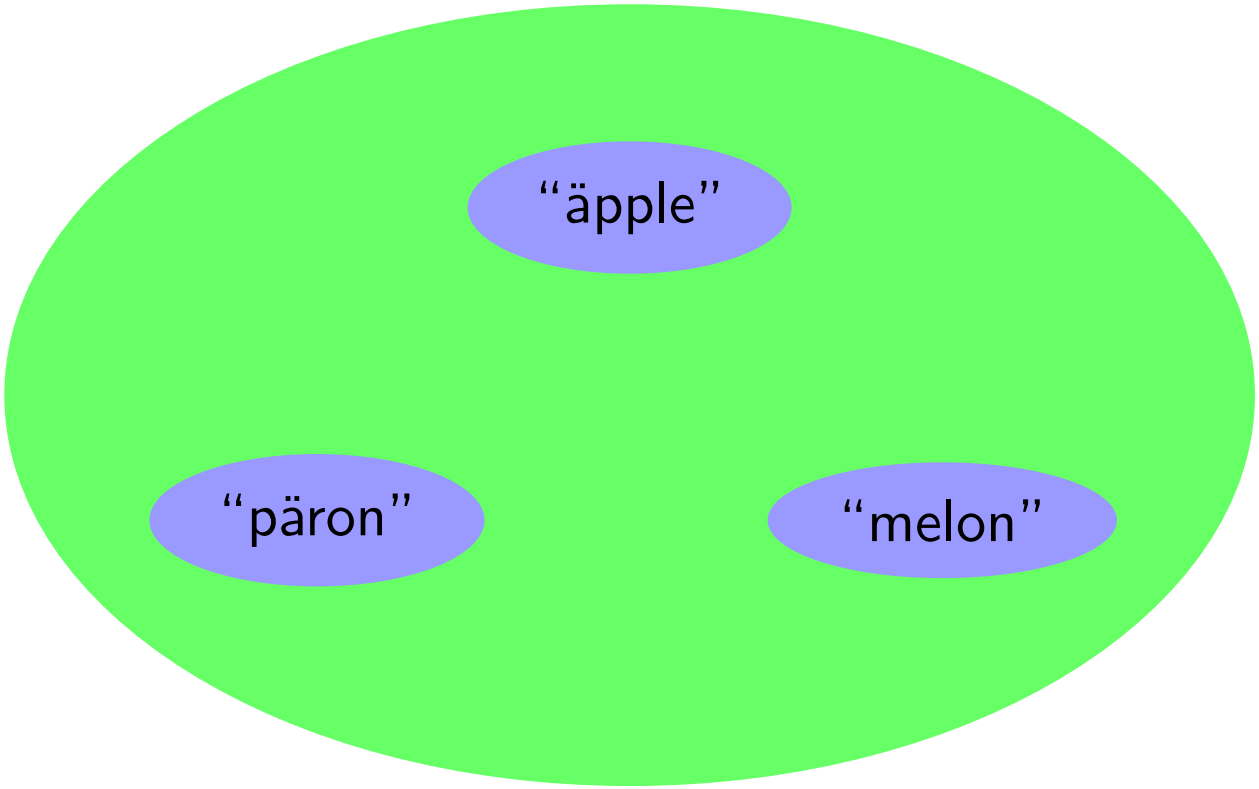
- Samling av klasser för effektiv och avancerad lagring av data
- Finns i paketet java.util
- Är uppdelade i Interface och implementationer
- Sök efter “Collection” i API:et !
- `http://docs.oracle.com/javase/7/docs/api/`
- Finns även bra tutorials, ex `http://docs.oracle.com/javase/tutorial/collections`

Lösningen heter **Collection**



Set - Mängd

- Oordnad mäng av data utan kopior
- Interfacet heter Set
- Implementationer exempelvis HashSet

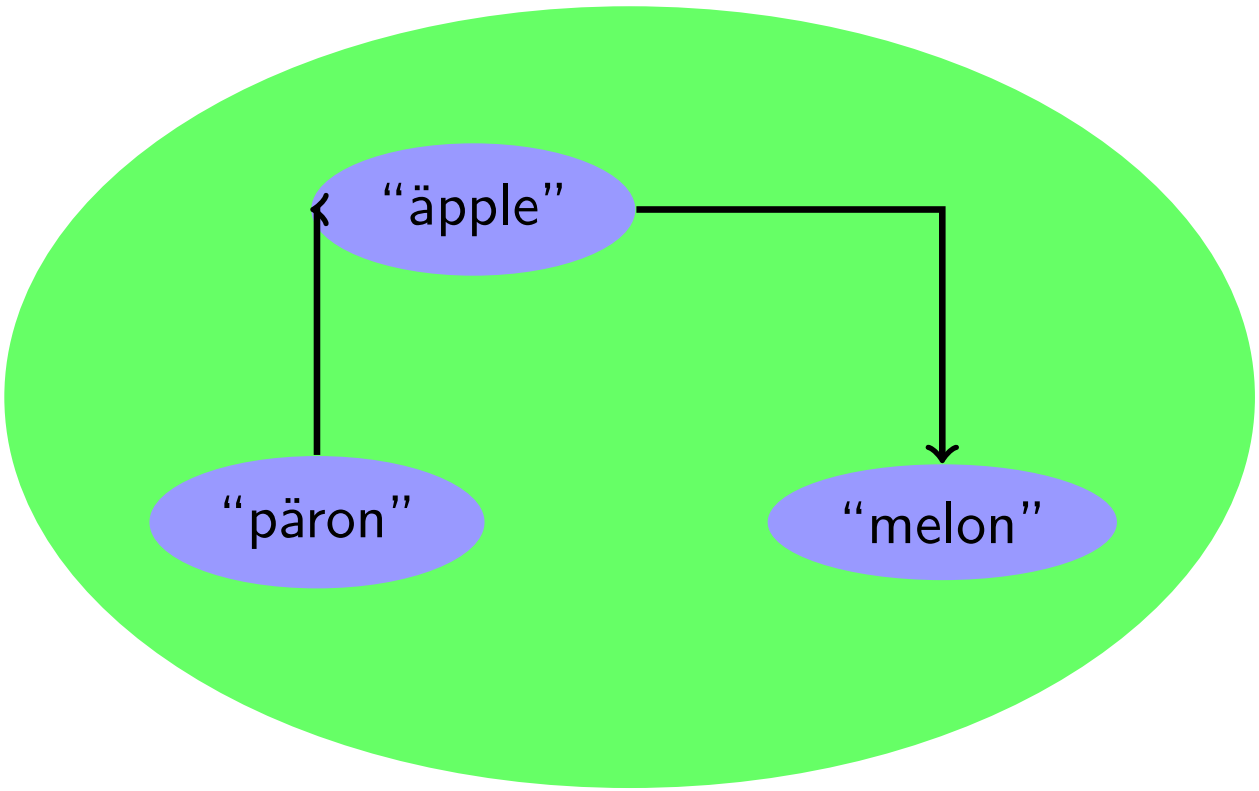


Set

```
import java.util.*;

/* The program FindDuplicates prints messages for each duplicated
 * argument it is passed.
 * example: java FindDuplicates i came i saw i left
 */
class FindDuplicates {
    public static void main(String[] args) {
     Set<String> minSet = new HashSet<String>();
     for (int i=0; i<args.length; i++)
            if (minSet.add(args[i]) == false)
                System.out.println("Duplicate detected: " + args[i]);

       System.out.println(minSet.size() + " distinct words: " + minSet);

      }
}

//      for(String str: args)
```

List -Lista

- Sekvens av data
- Interfacet heter List
- Implementationer exempelvis ArrayList och LinkedList

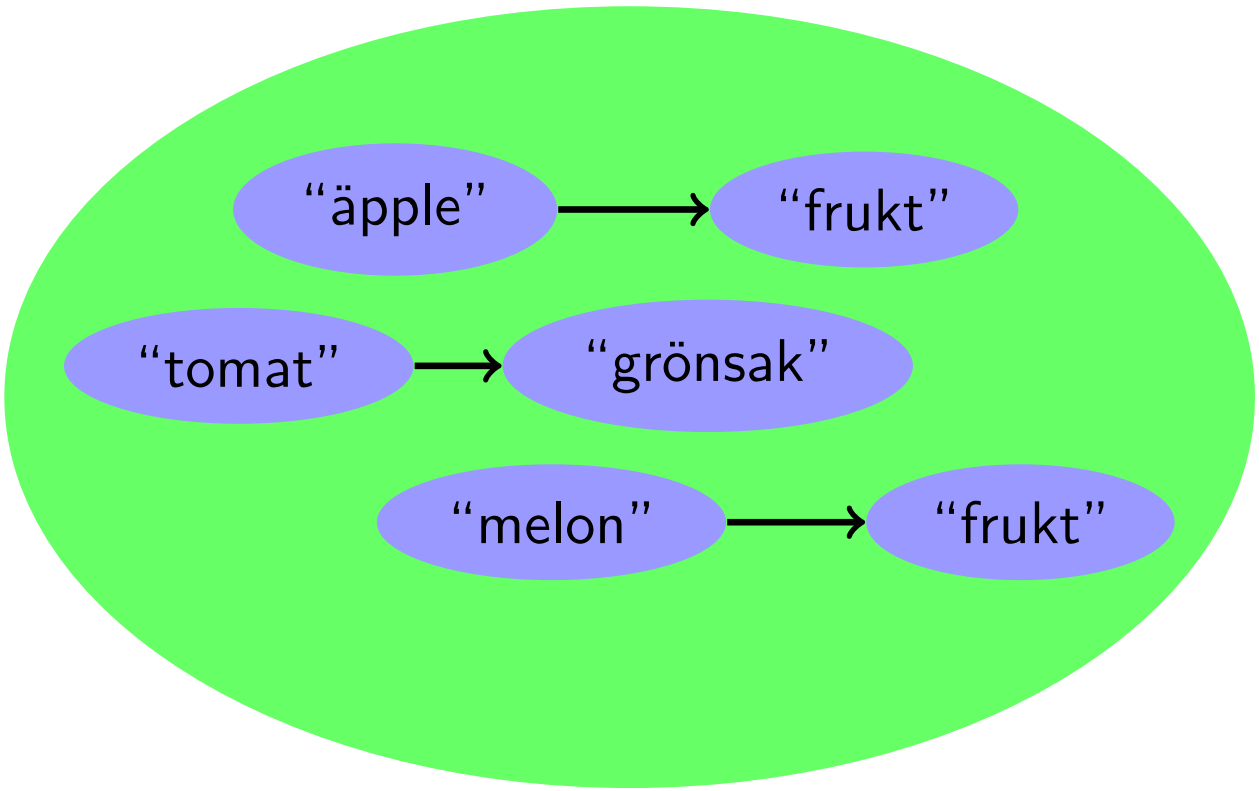


List

```
import java.util.*;
class Panier {
  public static void main(String[] args) {
    List<String> panier = new ArrayList<String>();

    panier.add("melon");
    panier.add("banan");
    panier.add("hallon");
    panier.add("hallon");

    System.out.println("panier in order:");
    // for(Iterator<String> it = panier.iterator();
    //     it.hasNext();
    //     ){
    //     String frukt = it.next();
    //     System.out.println("contains: " +  frukt);
    // }

       for(String str: panier)
     System.out.println(str);

    //System.out.println("panier contains:");
    //for(ListIterator<String> it = panier.listIterator(panier.size()); it.hasPrevious()
    //String frukt = it.previous();
    //     System.out.println("contains: " +  frukt);
    //}

  }
}
```

Map

- Kopplade par av data
- Interfacet heter Map
- Implementationer exempelvis HashMap



Map

```
import java.util.*;

public class Freq {
    public static void main(String[] args) {
        Map<String, Integer> m = new HashMap<String, Integer>();

        // Initialize frequency table from command line
        for (String a : args) {
            Integer freq = m.get(a);
            m.put(a, (freq == null) ? 1 : freq + 1);
        }

        System.out.println(m.size() + " distinct words:");
        System.out.println(m);
    }
}

//java Freq if it is to be it is up to me to delegate
```

Hashing

- Alla klasserna ovan som innehåller Hash i namnet använder något som heter hashing för att öka hastigheten på vissa operationer, exempelvis när man ska göra snabba sökningar i en lista
- Det ingår inte i kursen att förstå exakt hur det fungerar, men ni måste förstå tillräckligt för att kunna använda hjälpligt

Hashing

- Hashing går ut på att man för varje objekt beräknar ett så kallad hash (heltal)
- Heltalet används sedan för att dela upp mängden i flera mindre mängder, vill man veta vilken del man ska leta i behöver man bara hasha värdet man letar efter
- Hashfunktionen måste man alltid returnera samma värde för samma objekt, och för alla andra objekt där **equals()** returnerar sant
- För er ordlistlaboration så räcker det bra att returnera värdet för textsrängens **haschCode()**

Klassen Collections

- Klassen Collections har statiska metoder för att manipulera listor, mängder m.m.
  - **frequency()**
  - **min()**, **max()**
  - **reverse()**
  - **sort()**, **shuffle()**

```
Collections.reverse(pl);
```

Outline

Iterable

- Iterable är ett interface som uppfylls av alla klasser som man kan gå igenom element för element
- Iterable innehåller bara en metod: **iterator()** som returnerar ett speciellt Iterator-objekt som man kan använda för att gå igenom mängden

Iterator

- Iterator är ett interface med metoderna
  - **hasNext()**
  - **next()**
  - **remove()** //valfritt
- Man får inte ändra på den underliggande samlingen medan iteratorn används, då blir det fel

Iterator

- Använda en iterator

```
Person person;
for(Iterator<Person> iterator = pl.iterator();
    iterator.hasNext();
    person = iterator.next()){
    // Gör något
}
```

Iterator

- Enklare med for-each

```
for (Person person : pl) {
    // Gör något
}
```