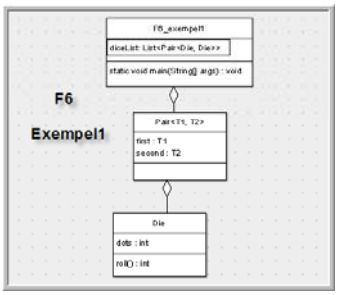
ITK:P2 F6

Sortering av generiska containerklasser

DSV Peter Mozelius

Generiska klasser



- En generisk klass lagras i en generisk container-klass

En tärningsklass

```
public class Die {
  private int dots;

  public Die() {
    dots = roll();
  }

  public int roll(){
    return (int)(Math.random() * 6) + 1;
  }

  public int getDots(){
    return dots;
  }
}
```

En generisk parklass

- En lite modifierad variant av den klass som finns i kursbokens kap 17:

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

  public Pair() {}

  public Pair(T1 x1, T2 x2) {
   first = x1;
   second = x2;
 }
```

4

En generisk parklass

```
public void setFirst(T1 x) {
   first = x;
 }

 public void setSecond(T2 x) {
   second = x;
 }

 public T1 getFirst() {
   return first;
 }

 public T2 getSecond() {
   return second;
 }
```

5

En generisk containerklass

```
import java.util.*;

public class F6_exempel1 {

  List<Pair<Die, Die>> diceList;

  public F6_exempel1() {
    diceList =
       new LinkedList<Pair<Die, Die>>();
    initList();
    showList();
  }
```

6

En generisk containerklass

```
public void initList(){
  diceList.add(
    new Pair<Die, Die>(new Die(), new Die()));

  diceList.add(
   new Pair<Die, Die>(new Die(), new Die()));

  diceList.add(
   new Pair<Die, Die>(new Die(), new Die()));

  diceList.add(
   new Pair<Die, Die>(new Die(), new Die()));
}
```

7

En generisk containerklass

```
public void showList(){

  Iterator<Pair<Die, Die>> i =
    diceList.iterator();

  while(i.hasNext()){
    Pair<Die, Die> pair = i.next();
    int first = pair.getFirst().getDots();
    int second = pair.getSecond().getDots();
    System.out.println(first + second);
  }

}
```

8

Jokertecken / wildcards

- Ett typuttryck Typ<T>
- Kan kombineras med:

?	? = vilken typ som helst
? extends X	? = X eller subtyp till X
? super X	? = X eller supertyp till X

9

Java Collections Framework

- Ett ramverk med 3 delar:
  - Interface
  - Implementationer
  - Algoritmer
- Polymorfa algoritmer:
  - Samma metod kan användas för olika implementationer av ett interface

10

11

Klassen Collections

- **java.util.Collections**
  - En samling statiska metoder
  - Metoder som jobbar med datasamlingar
  - Metoder för sökning
    - `min(Collection<? extends T> coll)`
    - `max(Collection<? extends T> coll)`
    - `shuffle(List<?> list)`
    - `swap(List<?> list, int i, int j)`
    - `sort(List<T> list)`
    - `sort(List<T> list, Comparator<? super T> c)`

12

Att sortera datasamlingar

- En sorterbar lista med naturligt sorterbara element kan sorteras med hjälp av:

```
java.util.Collections.sort(List<T> list)
```

Vi tittar nu på ett enkelt kodexempel:

```
import java.util.*;
public class F6_exempel2{
```

13

Att sortera datasamlingar

```
Map<String,Integer> hMap2;
List<Integer> lista;

  public F6_exempel2(){
    hMap2 =
       new HashMap<String,Integer>();
    initiera();
    skrivUt(sortera());
  }
```

14

Att sortera datasamlingar

```
  public void initiera(){
    hMap2.put("Mikael",
                new Integer(4));
    hMap2.put("Cecilia",
                new Integer(21));
    hMap2.put("Peter",
                new Integer(9));
    hMap2.put("Johan",
                new Integer(7));

  }
```

15

Att sortera datasamlingar

```
public List<Integer> sortera(){
  lista = new ArrayList<Integer>();
  Iterator<Integer> i1 =
       hMap2.values().iterator();

  while (i1.hasNext())
    lista.add(i1.next());

  Collections.sort(lista);
  return lista;
}
```

16

Att sortera datasamlingar

```
public void skrivUt(List<Integer> lista){
  Iterator<Integer> i2 =
           lista.iterator();

  while(i2.hasNext())
     System.out.println(i2.next());
}

public static void main(String[] args){
   new F6_exempel2();
}
}
```

17

Att sortera datasamlingar

- En sorterbar lista som **inte har** naturligt sorterbara element kan sorteras med hjälp av:

```
sort(List<T> list, Comparator<? super T> c))
```

Ett komplett körbart kodexempel med en klass som implementerar *interfacet Comparator* finns för avhämtning på kurshemsidan

18

Interfacet Comparator

- Innehåller 2 abstrakta metoder
  - [booleean equals(T1, T2)]
  - **int compare(T1, T2)**

Metoden compare() returnerar:
Ett negativt heltal om T1 **<** T2
0 om T1 **==** T2
Ett positivt heltal om T1 **>** T2

19

Att jämföra våra tärningar

```
public class DiceComparator implements
            Comparator<Pair<Die, Die>> {

 public int compare(Pair<Die, Die> p1,
                    Pair<Die, Die> p2) {
   return (p1.getFirst().getDots() +
           p1.getSecond().getDots())
           -
          (p2.getFirst().getDots() +
           p2.getSecond().getDots()) ;
   }
}
```

20

Att sortera strängar

- Att sortera strängar är lättare
- Gör som i kursbokens kap 10 med:
  - **java.util.Comparator**
  - **java.text.Collator**

- **Collator.PRIMARY:**
  - ”Java” = ”java”
  - ”Entre” = ”entré”

21

Att sortera svenska tecken

- I teckentabellerna är det fel ordning (för det svenska alfabetet) när det gäller tecknen **Å**, **Ä** och **Ö**
- Avänd klassen **`java.util.Locale`**
- Läs mera i kursbokens kap 9.2

22

Testfrågor

- Detta var allt för denna gång
- Kontrollera att du har hängt med genom testet på kurshemsidan
- Dax nu också att köra igång med Uppgift B!

Hej då!

23