Objektorienterad Programmering (TDDC77)

Föreläsning XIII: Kollektioner (kont.) [hashing, iteratorer], Undantag

Ahmed Rezine

IDA, Linköpings Universitet

Höstermin 2016

Outline

Collection

Iterable

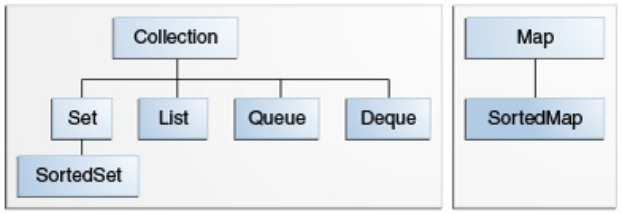
Hashing

Undantag

Collection gränsnittet

- 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: en mängd där inget element är duplicerat.

List: en sekvens. Elementerna kan vara duplicerat

Queue: typisk FIFO (fast kan involvera prioritet)med metoder för att lägga till och ta bort element.

Deque: Som en Queue, fast både LIFO och FIFO.

Map: associera (max) ett värde (value) per nyckel (key).

Set (Mängd)

```java
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> set = new HashSet<String>();
      for (int i=0; i<args.length; i++)
            if (!set.add(args[i]))
                System.out.println("Duplicate detected: " + args[i]);

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

Set (Mängd)

```java
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 FindDuplicatesIterator {
    public static void main(String[] args) {
      Set<String> set = new HashSet<String>();
      for (int i=0; i<args.length; i++)
            if (!set.add(args[i]))
                System.out.println("Duplicate detected: " + args[i]);

        //System.out.println(set.size() + " distinct words: " + set);
     Iterator<String> it = set.iterator();
     while(it.hasNext()){
         String e = it.next();
         System.out.print(" " + e);
     }
     System.out.println();
      }

    for(String e: set)
     System.out.print(" " + e);
    System.out.println();
}
```

List (sekvens)

- kan använda index som i arrayer (get, set, remove, ...)
- kan söka efter ett element (indexOf, lastIndexOf, ...)
- kan "gå igenom elementerna i sekvens" med list iteratorer
- implementationer exempelvis ArrayList och LinkedList

```java
import java.util.*;

/* The program Directions takes a sequence of Directions
 * in {right, left} and perform some simple operations on them
 * example: java Directions left left left right left right
 */
class DirectionsIterators {
    public static void main(String[] args) {
     List<Integer> list = new LinkedList<Integer>();
        Integer i = new Integer(5);
     list.add(i);
     System.out.println(list);
 }
}
```

List (sekvens)

```java
import java.util.*;

/* The program Directions takes a sequence of Directions
 * in {right, left} and perform some simple operations
 * on them
 * example: java Directions left left left right left right
 */
class Directions {
    public static void main(String[] args) {
     List<String> list = new LinkedList<String>();

     for (int i=0; i<args.length; i++)
         list.add(args[i]);

        //System.out.println(set.size() + " distinct words: " + set);
     for(int i=0; i<list.size(); i++){
         System.out.println("att step " + i + " turn " + list.get(i));
        }
     System.out.println();

     System.out.println("You should turn right at step " + list.indexOf("right"));
     list.remove(list.lastIndexOf("left"));
     System.out.println("truncated list: " + list);
    }
}
```

Boxed primitives

- Collections och generics fungerar bara för objekt (inte primitiva typer som int, double osv.)
- Java innehåller därför klasser som motsvarar dessa: Byte, Double, Float, Integer, Long, och Short
- De är i princip bara klasser som kapslar in sina primitiva motsvarigheter

Map

```
import java.util.*;
// Counts the frequency of each argument
//java Freq if it is to be it is up to me to delegate
public class Freq {
    public static void main(String[] args) {
        Map<String, Integer> m = new HashMap<String, Integer>();

        // Initialize frequency table from command line
        for (int i=0; i<args.length; i++) {
            Integer freq = m.get(args[i]);
         if(freq==null)
          m.put(args[i], 1);
         else
          m.put(args[i],freq+1);
          //m.put(args[i],10);
        }

     //System.out.println(m);

     Set<String> keys = m.keySet();
     Iterator<String> it = keys.iterator();
     while(it.hasNext()){
         String key = it.next();
         Integer freq = m.get(key);
         System.out.println(key + " ----> " + freq);
     }

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

        //System.out.println(m);
    }
}
```

Map

```
import java.util.*;
//Illustrate keySet, values and entrySet for a Map
//java Freq if it is to be it is up to me to delegate
class MoreFreq {
    public static void main(String[] args) {
     Map<String, Integer> m = new HashMap<String, Integer>();
     for (String a : args) {
         Integer freq = m.get(a);
         m.put(a, (freq == null) ? 1 : freq + 1);
     }

     System.out.println("Keys: ");
     for(String key: m.keySet())
         System.out.print(" " + key);

     System.out.println("\nValues: ");
     for(Integer val: m.values())
         System.out.print(" " + val);

     System.out.println("\nEntrySet: ");
     for(Map.Entry<String, Integer> entry: m.entrySet())
         System.out.print(" (" + entry.getKey()
                    + "->" + entry.getValue() + ")");
     System.out.println();
    }
}
```

Att gå genom en Array med **for**

```
//Programet tar ett antal argument och summera de
class SumFor{

    public static void main(String[] args) {
     int sum=0;
     for(yint i=0; i<args.length; i++){
         sum = sum + Integer.parseInt(args[i]);
     }
     System.out.println("Summan är: " + sum);
    }
}
```

Att gå genom en Array med **for-each**

```
//Programet tar ett antal argument och summera de
class Sum{

    public static void main(String[] args) {
     int sum=0;
     //''syntactic sugar''
     for(String arg: args){
         sum = sum + Integer.parseInt(arg);
     }
     System.out.println("Summan är  :" + sum);
    }
}
```

Outline

Collection

Iterable

Hashing

Undantag

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
}
```

Att gå genom en Kollektion med en iterator

```
import java.util.*;

// Räkna upp antalet olika jämna nummer
// med Iterator
class CountEvensIteration {

    public static void main(String[] args) {
     Set<Integer> s = new HashSet<Integer>();
     for(String a: args)
         s.add(Integer.parseInt(a));

     int resultatet=0;
     for(Iterator<Integer> it = s.iterator(); it.hasNext(); ){
         if(it.next() % 2 == 0){
          resultatet++;
         }
     }
     System.out.println("Antalet olika jämna nummer är:"
                  + resultatet);
    }
}
```

Iterator: Enklare med **for-each**

```
import java.util.*;

// Räkna upp antalet olika jämna nummer
// med Iterator
class CountEvensIterationForEach {

    public static void main(String[] args) {
     Set<Integer> s = new HashSet<Integer>();
     for(String a: args)
         s.add(Integer.parseInt(a));

     int resultatet=0;
     for(Integer i : s){
         if(i % 2 == 0){
          resultatet++;
         }
     }
     System.out.println("Antalet olika jämna nummer är:"
                  + resultatet);
    }
}
```

Använd iterator om du vill ta bort några element

```
import java.util.*;
// Filtrera bort de jämna nummer
// från mängden av olika input värden
class FilterEvens {

    public static void main(String[] args) {
     Set<Integer> s = new HashSet<Integer>();
     for(String a: args)
         s.add(Integer.parseInt(a));

     for(Iterator<Integer> it = s.iterator(); it.hasNext(); ){
         if(it.next()%2 == 0){
          it.remove();
         }
     }
     System.out.println("Filtrerat mängd:" + s); }
}
//java FilterEvens 0 1 1 2 3 5 8
```

Klassen Collections

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

Klassen Collections

```
import java.util.*;
// Illustrate usage of Collection static methods
//UsingCollections 0 1 1 2 3 5 8
public class UsingCollections {

    public static void main(String[] args) {
     List<Integer> s = new ArrayList<Integer>();
     for(String a: args)
         s.add(Integer.parseInt(a));

     System.out.println("listan: " + s);
     Collections.reverse(s);
     System.out.println("reversed: " + s);
     Collections.shuffle(s);
     System.out.println("shuffled: " + s);
    }
}
```

Outline

Collection

Iterable

Hashing

Undantag

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 det
- Idéen är att man beräknar för varje objekt ett så kallad hash (heltal)

Hashing

- 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()**

Hashing

```
import java.util.*;

//Basket päron melon apelsin melon
public class Basket {

    public static void main(String[] args) {
     Set<Fruit> set = new HashSet<Fruit>();
     for(String in: args)
         set.add(new Fruit(in));

     System.out.println("Mängden av olika input är: " + set);
    }
}
```

Hashing

```
class Fruit {
    private String name;

    public Fruit(String name){this.setName(name);}

    public String getName() {return name;}

    public void setName(String name) {this.name = name;}

    public String toString(){return name;}

    public int hashCode(){return name.hashCode();}

    public boolean equals(Object other){
     if(other==null || !(other instanceof Fruit))
         return false;
     return name.equals(((Fruit)other).name);
    }
}
```

Outline

Collection

Iterable

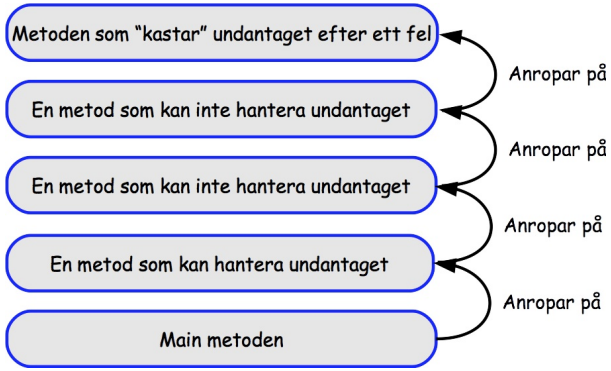
Hashing

Undantag

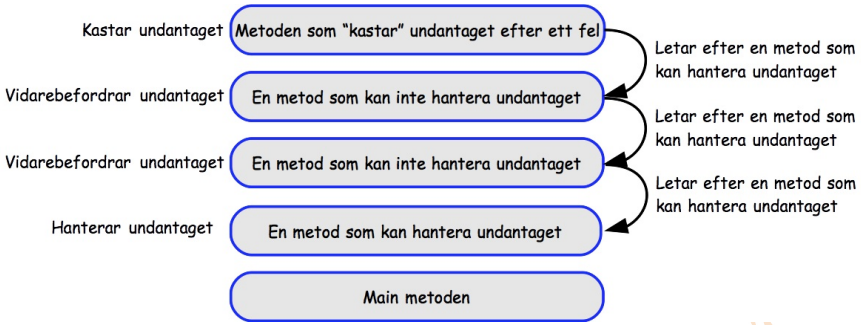
Undantag - Exceptions

- Fel som uppstår under körning kallas undantag (exceptions)
- Metoder kan kasta undantag (throw)
- Metoder kan fånga undantag (catch)

Undantag - Exceptions



Undantag - Exceptions

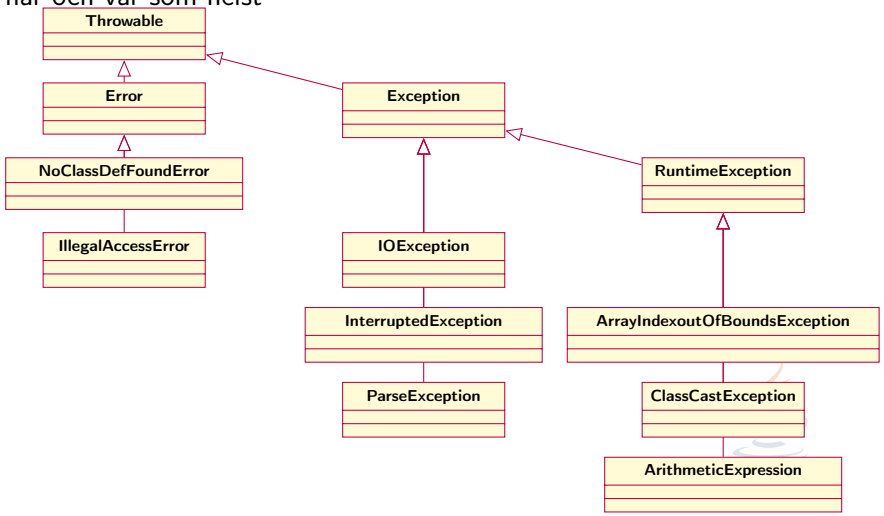


Kasta undantag

```
class DateParsing{

    private static int getMonth(String input) throws DateFormatException {
     if(input.length()!=10)
         throw new DateFormatException("Incorrect date format, should be dd/mm/yyyy");
     return Integer.parseInt(input.substring(3,5));
    }

    private static String printMonth(String date) throws DateFormatException{
     int month = getMonth(date);
     switch(month){
     case 1: return "January";
     case 2: return "February";
     case 3: return "March";
     default: return "December";
     }
    }

    public static void main(String[] args){
     try{
         System.out.println("The month in: " + args[0] + " is " + printMonth(args[0]));
     } catch(DateFormatException e){
         System.out.println(e.getMessage());
     }
    }
}
```

Typer av undantag

Vissa undantag måste man förvarna för, medan andra kan uppstå när och var som helst



Typer av undantag

- **Throwable** - Allt som kan kastas och fångas
- **Errors** - Jättealvarliga fel i själva java. Normalt sett bör dessa inte hanteras av programmeraren.
- **Exceptions** (utom **RuntimeException**) - Förutsedda undantag, dessa är del av normal funktion och måste hanteras
- **RuntimeExceptions** - Undantag som kan inträffa under normal körning, men normal sett beror på att programmeraren gjort fel. Behöver inte fångas

Typer av undantag

- Checked exception:
  - e.g. **Exception, ParseException, IOException,** ...
  - Måste hanteras (dvs. antigen med **try-catch** eller genom att skriva **throws** i metodens deklaration)
  - När undantaget förväntas uppstå
  - När det går att rädda systemet
- Unchecked exception:
  - e.g. **RuntimeException, ArrayIndexOutOfBoundsException,** ...
  - Måste inte hanteras ((dvs. behöver inte ha en **try-catch** eller skriva **throws** i metodens deklaration))
  - När undantaget inte förväntas uppstå
  - Men ifall det ske, "Something went horribly wrong!"

Egna undantag

```
class DateFormatException extends Exception{
    public DateFormatException (String message){
     super(message);
    }
}
```

Kasta vidare undantag

- En kedja av metoder kan ha throws i sin signatur
- Lämnar över ansvaret att fånga högre upp i anroskedjan

```
class Undantag {
  public static void main(String[] args) {
    System.out.println("Prog börjar");
    foo1();
    System.out.println("Prog slutar");
  }
  public static void foo1(){
    System.out.println("foo1 börjar");
    try{
      foo2();
    }
    catch (ArithmeticException e){
      System.out.println("foo1 catch1");
    }
    catch(NumberFormatException e){
      System.out.println("foo1 catch2");
    }
    finally{
      System.out.println("foo1 finally");
    }
    System.out.println("foo1 slutar");
  }
  public static void foo2(){
    System.out.println("foo2 börjar");
    try{
      System.out.println(Integer.parseInt("a"));
    }
    catch(ArithmeticException e){
      System.out.println("foo2 catch1");
    }
    //catch(NumberFormatException e){
    //  System.out.println("foo2 catch2");
    //}
    finally{
      System.out.println("foo2 finally");
    }
    System.out.println("foo2 slutar");
  }
}
```

Stack trace

```
Exception in thread "main" java.lang.NumberFormatException: For input st
     at sun.misc.FloatingDecimal.readJavaFormatString(FloatingDecimal.ja
     at java.lang.Double.parseDouble(Double.java:510)
     at lincalc.LinCalcJohn.calc(LinCalcJohn.java:139)
     at lincalc.LinCalc.calc(LinCalc.java:35)
     at lincalc.LinCalc.evaluate(LinCalc.java:45)
     at lincalc.LinCalc.main(LinCalc.java:28)
```