Praktische Softwaretechnologie

Lecture 7.

Károly Bósa
(Karoly.Bosa@jku.at)

Research Institute for Symbolic Computation
(RISC)

java.lang.Math

Karoly.Bosa@jku.at

Mathematical standard Functions

(see JavaDoc)

java.Math

Karoly.Bosa@jku.at

The package java.Math contains the following classes:

- BigInteger
- BigDecimal

For arithmetic on demand accuracy:

BigInteger is an integer number with demanded length

BigDecimal is a scalable number $m*10^n$ (where m and n are integer)

Example for BigInteger


```
import java.math.BigInteger;

public class BigIntegerExample
{
  public static void main(String[] args)
  {
    BigInteger bigInteger1 = new BigInteger("98765432112345678 9");
    BigInteger bigInteger2 = new BigInteger("11233445566");
    BigInteger bigIntResult = bigInteger1.multiply(bigInteger2);
    System.out.println("Result is ==>"+bigIntResult);
  }
}
```

Output:

```
c:\___Class\tmp>java BigIntegerExample
Result is ==>1109476105436503580498464757 4
```

The I/O Classes

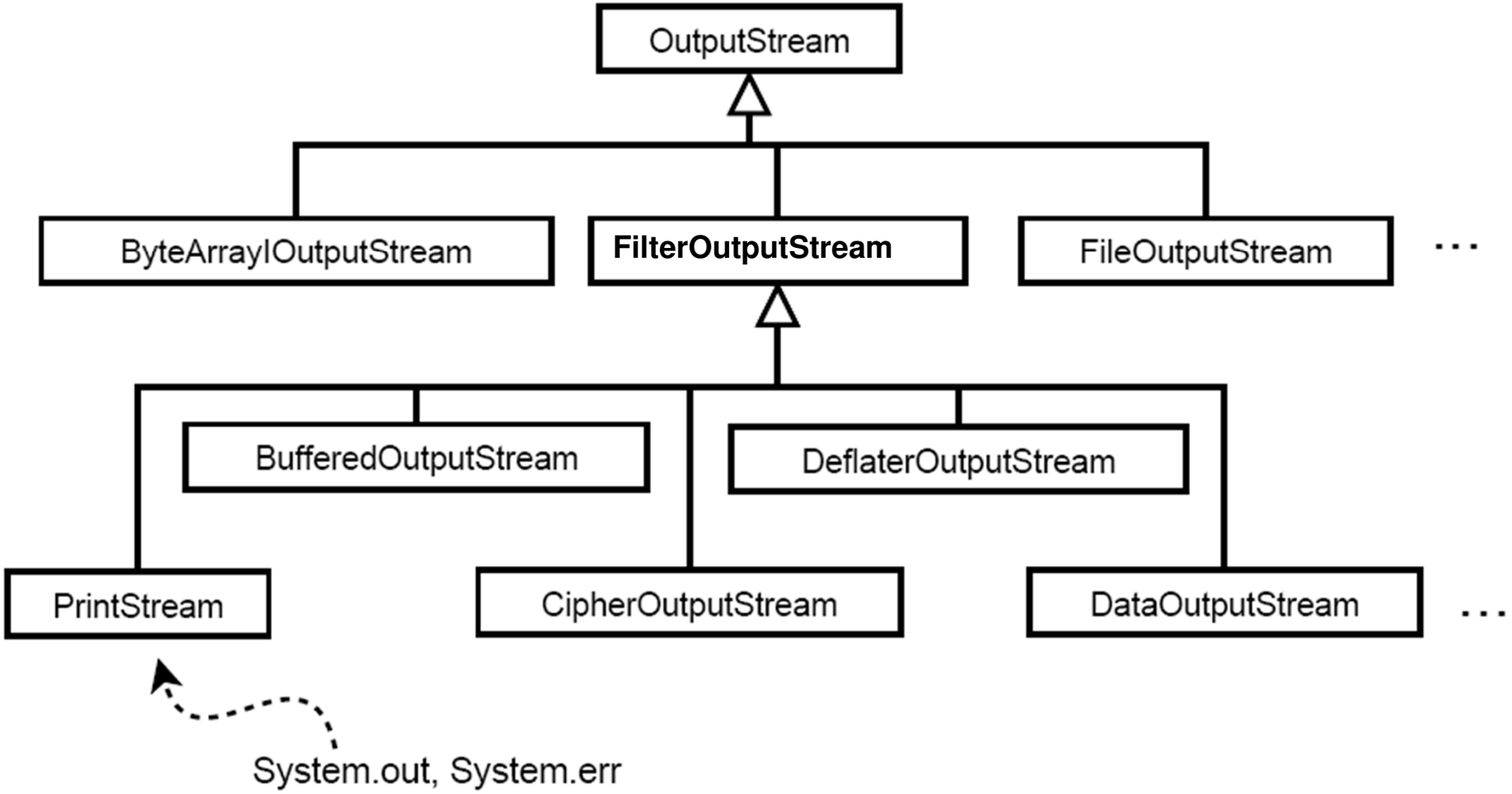
There are two families of the I/O channels

- InputStream and OutputStream for byte sequences
- Reader and Writer for Text = Character sequences

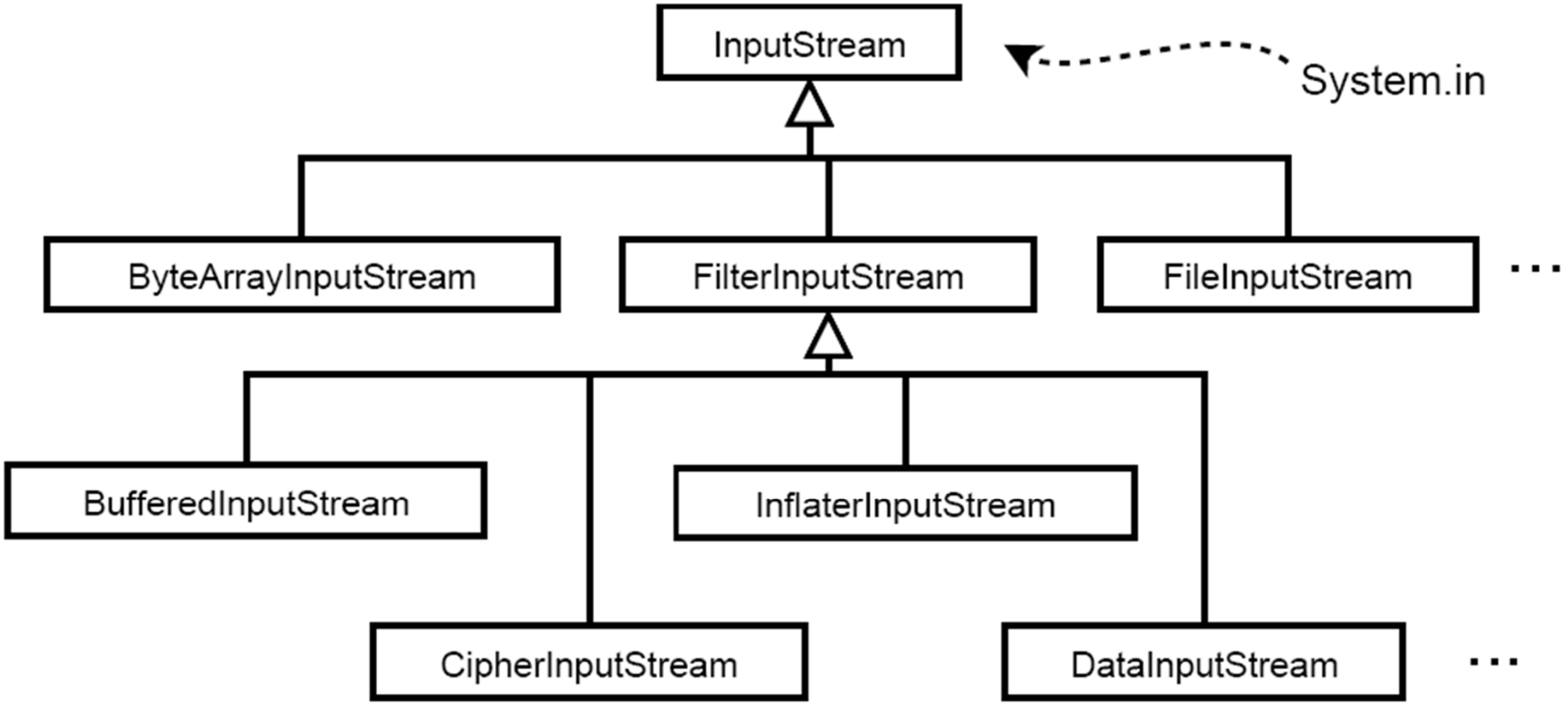
- Representation:
  sequence of 16-bit char ⇔sequence of bytes

OutputStream Classes

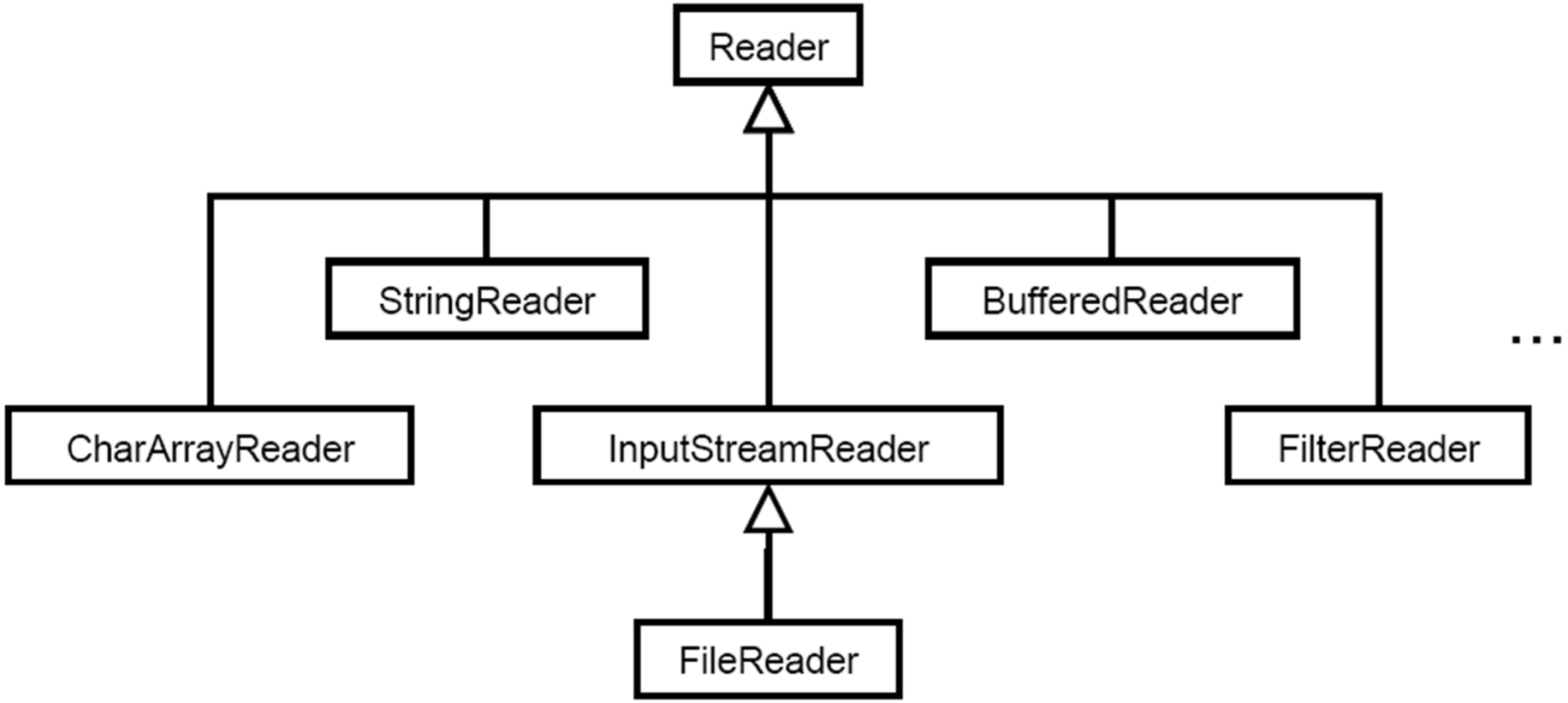
Karoly.Bosa@jku.at



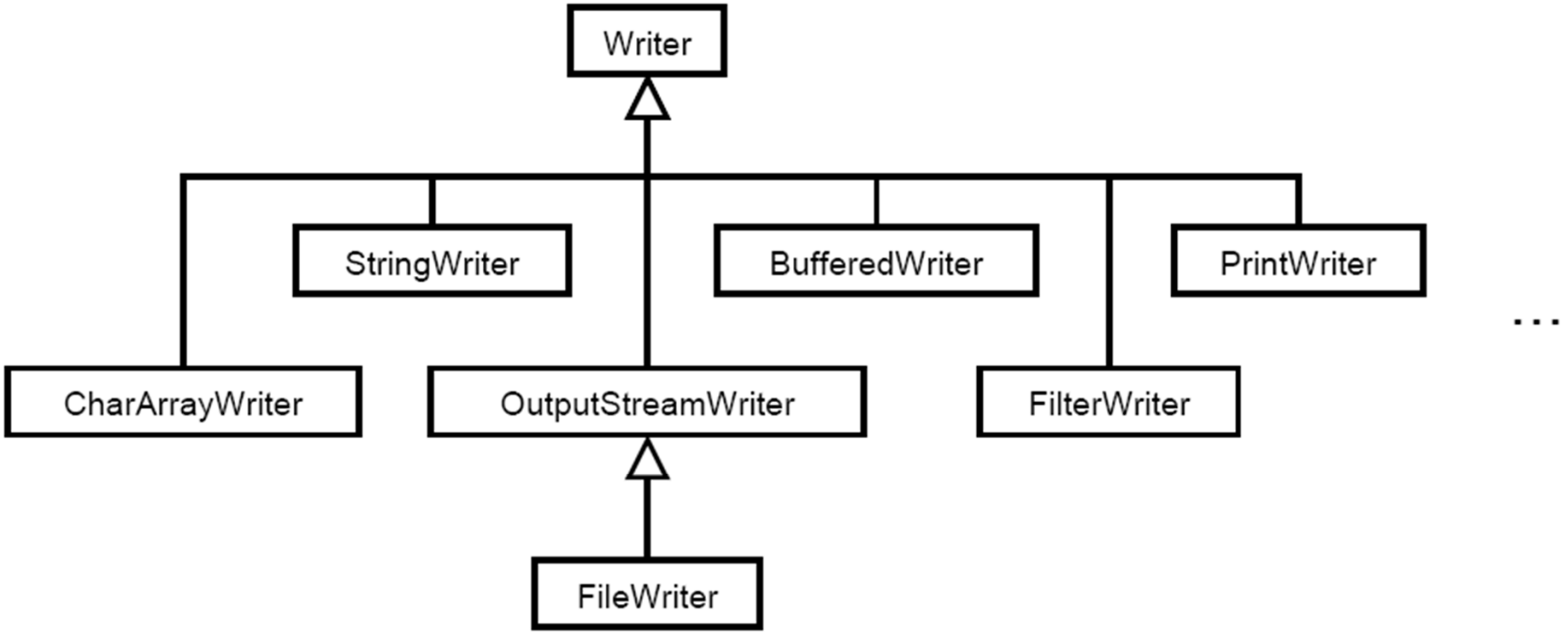
InputStream Classes



Reader Classes



Writer Classes



Copying a File Byte by Byte


```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
public class CopyBytes {
  public static void main(String[] args) throws IOException {
     FileInputStream in = null;
     FileOutputStream out = null;
     try {
       in = new FileInputStream(“source.exe");
       out = new FileOutputStream(“destination.exe"); //we can specify a path, too
       int c;
       while ((c = in.read()) != -1) { out.write(c); }
     } finally {
      if (in != null) { in.close(); }
      if (out != null) { out.close(); }
     }
   }
}
```

Copying Lines from a Text File from Another

Karoly.Bosa@jku.at

```
import java.io.FileReader;
import java.io.FileWriter;
import java.io.BufferedReader;
import java.io.PrintWriter;
import java.io.IOException;
public class CopyLines {
  public static void main(String[] args) throws IOException {
      BufferedReader inputStream = null;
      PrintWriter outputStream = null;
      try {
         inputStream = new BufferedReader(new  FileReader("source.txt"));
         outputStream = new PrintWriter(new FileWriter("destination.txt"));
         String l;
         while ((l = inputStream.readLine()) != null) { outputStream.println(l); }
      } finally {
         if (inputStream != null) { inputStream.close(); }
         if (outputStream != null) { outputStream.close(); }
      }
  }
}
```

Output: Make it Easy

- Writing text onto the stdout:

```
System.out.println(...);
```

- Writing text into a file:

```
PrintWriter pw =  new PrintWriter(new FileWriter("blabla.txt"));
pw.println(...); pw.close();
```

- Writing binary data into a file:

```
FileOutputStream fos = new    FileOutputStream ("data.bin");
DataOutputStream dos = new DataOutputStream(fos);
dos.writeInt(23);
dos.writeDouble(12.3);
dos.close();
```

Input: Make it Easy

Karoly.Bosa@jku.at

- Reading text from stdin line by line

```
InputStreamReader isr = new InputStreamReader(System.in);
BufferedReader br = new BufferedReader(isr);
String line = br.readLine();
```

- Reading text from a file line by line

```
FileReader fr = new FileReader("blabla.txt");
BufferedReader br = new BufferedReader(fr);
String line = br.readLine();
if (line == null) {...end of file...}
br.close();
```

- Reading binary data from a file

```
FileInputStream fis = new FileInputString("data.bin");
DataInputStream dis = new DataInputStream(fis);
int i = dis.readInt();
double d = dis.readDouble();
dis.close();
```

Recommended to Read

Reading and completing the course material from the online Java Tutorial:

http://download.oracle.com/javase/tutorial/java/index.html

- Essential Java Classes: Basic I/O

Exercise 9

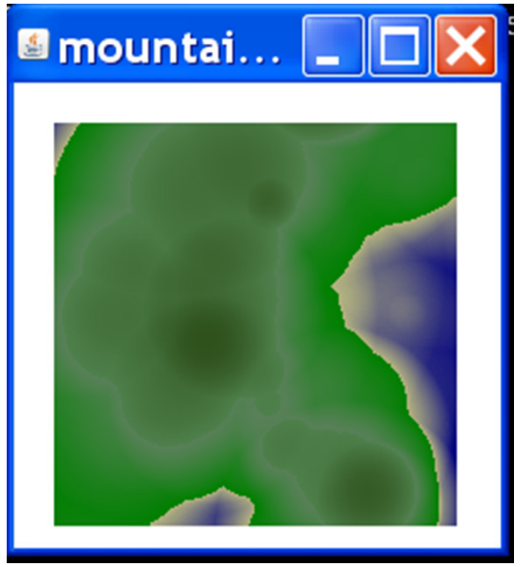
Deadline: 30.04.2014

Karoly.Bosa@jku.at

Download map.zip from the course web page. This zip file consists of the following two files :

- **Map.java**
- **mountains.map** (It describes a surface map with the size the 201x201. It specifies an altitude value (byte) value for each point of the map.

Your task is to implement the method ***readMapFromFile()*** of the class ***Map*** which should read the given altitude (byte) values from file **mountains.map** into the predefined array **map[]** (Attention! It is a vector, not a matrix)**.**



If you do the right job you should see the following window on the screen after you run the class ***Map***.

Generic Types

Interface Stack so far:

```
interface Stack {
    void push(String s);
    String pop();
    boolean isEmpty();
}
```

➔ It works only with String Objects

Generic Types

Interface Stack for any Object:

```
interface Stack {
    void push(Object s);
    Object pop();
    boolean isEmpty();
}
```

Generic Types

Interface Stack for any Object:

```
interface Stack {
    void push(Object s);
    Object pop();
    boolean isEmpty();
}
```

Application with Strings:

```
Stack stringStack = new ArrayStack();
stringStack.push("Hallo");
...
String s = (String)(stringStack.pop());
```

➔ Complicated and error prone

Generic Types

Generic Stack Interface

```
interface Stack<E> {
    void push(E s);
    E pop();
    boolean isEmpty();
}
```

Generic Types

Generic Stack Interface

```
interface Stack<E> {
    void push(E s);
    E pop();
    boolean isEmpty();
}
```

Application with Strings:

```
Stack<String> stringStack = new ArrayStack<String>();
stringStack.push("Hallo");
...
String s = stringStack.pop();
```

Generic Types

Generic Stack Interface

```
interface Stack<E> {
    void push(E s);
    E pop();
    boolean isEmpty();
}
```

Application with ints:

```
Stack<Integer> intStack = new ArrayStack<Integer>();
intStack.push(23);          // auto-boxing
...
int i = intStack.pop();     // auto-unboxing
```

Generic Types

Generic Stack Interface

```
interface Stack<E> {
    void push(E s);
    E pop();
    boolean isEmpty();
}
```

Application with ints:

```
Stack<Integer> intStack = new ArrayStack<Integer>();
intStack.push(23);          // auto-boxing
...
int i = intStack.pop();     // auto-unboxing
```

➔Reading and Completing in the Java Tutorial

Implementation of Generic Types

The type-parameter is known only by the compiler

```
interface Stack<E> {
   push(E o);
   E pop();
}
```

```
interface Stack {
   push(Object o);
   Object pop();
}
```

Hence, restrictions:

- We cannot apply E or E[] in static methods
- o instanceof E or o instanceof Stack<String> is not allowed
- Cast E and E[] without runtime check
  (but there is an "Unchecked" warning from the compiler)

Example for “Unchecked” Cast


```
public class ArrayStack<E> {
    E[] elements;
    int top = 0;

    ArrayStack(int initialSize) {
        elements = new E[initialSize];
    }
}
```

Compiler error: “generic array creation”

Example for “Unchecked” Cast


```
public class ArrayStack<E> {
   E[] elements;
   int top = 0;

   ArrayStack(int initialSize) {
      elements = new Object[initialSize];
   }
}
```

Compiler error: “incompatible types”

```
found : java.lang.Object[]

required: E[]
```

Example for “Unchecked” Cast

Karoly.Bosa@jku.at

```
public class ArrayStack<E> {
    E[] elements;
    int top = 0;

    ArrayStack(int initialSize) {
        elements = (E[]) new Object[initialSize];
    }
}
```

Compiler warning:

```
Note: stack/ArrayStack.java uses unchecked or unsafe operations.

Note: Recompile with -Xlint:unchecked for details.
```

Example for “Unchecked” Cast


```
public class ArrayStack<E> {
   E[] elements;
   int top = 0;

   ArrayStack(int initialSize) {
      elements = (E[]) new Object[initialSize];
   }
}
```

Compiler warning with –Xlint:unchecked:

```
stack/ArrayStack.java:6: warning: [unchecked] unchecked cast
found   : java.lang.Object[]
required: E[]
```

Comparable

In order to be able to sort the objects:

Implement the interface java.lang.Comparable<T> with the method int compareTo(T o) such that:

- < 0 then this < o
- = 0 then this = o
- > 0 then this > o

The best: It is 0 if and only if this.equals(o).

➔ It determines the “natural” order of the objects

Example for Comparable

Sorting the points on the basis of x and y coordinates

```
public class Point implements Comparable<Point> {
   private int x;
   private int y;

   public int compareTo(Point other) {
      if( this.x == other.x ) {
         return this.y - other.y;
      } else {
         return this.x - other.x;
      }
   }
}

   Point[] points = new Point[20];
   java.util.Arrays.sort(points);
```

General Quick Sorting

Karoly.Bosa@jku.at

Problem: Writing a program that is able to sort the element of an array whose type is not specified.

Requirement: The type of the elements must be a class which implements the interface java.lang.Comparable<T>.

General Quick Sorting


```
public class GeneralQuickSort {

 public static void sort(Comparable[] a) {
   quick(a, 0, a.length-1);
 }

 private static void quick(Comparable[] a, int lower, int upper) {
 …
 }
}
```

General Quick Sorting


```
private static void quick(Comparable[] a, int lower, int upper) {
   Comparable middleValue = a[(lower+upper)/2];
   int i = lower;
   int j = upper;
   Comparable tmp;

   do {
    while (a[i].compareTo(middleValue) < 0) { i++;}
    while (a[j].compareTo(middleValue) > 0) {j--;}

    if (i < j) {tmp = a[i]; a[i] = a[j]; a[j] = tmp;}

    if (i<=j) {
      i++;
      j--;
    }
   } while (i<=j);

   if (lower<j) quick(a, lower, j);
   if (i<upper) quick(a, i, upper);
 }
```

General Quick Sorting – Testing

The original program was able to sort an array of integer numbers.

The *GeneralQuickSort* can be applied on the following classes:

Authenticator.RequestorType, BigDecimal, BigInteger, Boolean, Byte, ByteBuffer, Calendar, Character, CharBuffer, Charset, CollationKey, CompositeName, CompoundName, Date, Date, Double, DoubleBuffer, ElementType, Enum, File, Float, FloatBuffer, Formatter.BigDecimalLayoutForm, FormSubmitEvent.MethodType, GregorianCalendar, IntBuffer, Integer, JTable.PrintMode, KeyRep.Type, LdapName, Long, LongBuffer, MappedByteBuffer, MemoryType, ObjectStreamField, Proxy.Type, Rdn, RetentionPolicy, RoundingMode, Short, ShortBuffer, SSLEngineResult.HandshakeStatus, SSLEngineResult.Status, String, Thread.State, Time, Timestamp, TimeUnit, URI, UUID

General Quick Sorting – Testing w. Integer


```
public class TestQuick {
  public static void main(String[] args) {
     Integer[] arrayOfNumbers = {47, 74, 21, 99, 51, 15, 65, 36, 83, 59};
     GeneralQuickSort.sort(arrayOfNumbers);

     for (int i = 0; i < arrayOfNumbers.length; i++) {
      System.out. print(arrayOfNumbers[i]+" ");
    }
    System.out.println();
  }
}
```

General Quick Sorting – Testing w. String


```
public class TestQuick2 {
  public static void main(String[] args) {
    String[] arrayOfString =  {"Richard", "Zak", "Tom", "George", "Tracy",
                               "Ernie", "Tim", "Ashley", "Jack", "Alan", "Charles"};
    GeneralQuickSort.sort(arrayOfString);

    for (int i = 0; i < arrayOfString.length; i++) {
     System.out.println(arrayOfString[i]);
    }
  }
}
```

Exercise 10

Deadline: 30.04.2014

Karoly.Bosa@jku.at

Recall the stack framework of Exercise 6:

- Modify the interface *Stack* (introduce a generic type parameter), such that a stack which implements this interface will be able to store any kind of data.

```
public interface Stack<E> {
    void push(E o);
    E pop();
    ...
}
```

- Modify the original abstract class *AbstractStack* and class *BoundedStack* such that they implement the new *Stack* interface.

- Test the new stack implementation with different data types (int, double, String,…).