Program input and output

- Computer programs are only useful if they interact in some way with the rest of the world - this interaction is called input / output (I / O).

- In Java, input / output is most commonly used, which includes files and computer networks via a streams mechanism - streams are objects that support I / O commands

- Standard output (**System. out)** and standard input (**System. in)** are examples of flows

- Working with streams and files in Java requires knowledge of the mechanism for handling errors using exceptions

Working with flows

- When working with input / output, we distinguish two basic categories of data:
  - machine-formatted data - composed of bytes and
  - text that people can read - made up of characters
- Thus, in Java there are two basic types of flows:
  - **Byte streams** i
  - **Character streams**
- Flow classes are part of the package java.io which must be introduced at the beginning of the program
- Streams are necessary in Java programs for working with files and communicating over a network

Standard data flows in Java

- Java includes three standard data streams:

1. Standard input - typically used to load with
   keyboard overhang System.in

2. Standard output - typically used for screen printing
   over System.out

3. Standard error - also typically connects to the screen,
   is used to print errors via System.err

Working with flows

- An object that writes data to **byte stream** belongs to one of the subclasses of the abstract class OutputStream, while the one that reads from the byte stream belongs to one of the subclasses of the abstract class InputStream.

- An object that writes data to **character stream** belongs to one of the subclasses of the abstract class Writer, while the one who reads from the flow of signs belongs to one of the subclasses of the abstract class
  Reader.
- Byte streams are useful in machine communication, as well as for efficient storage of very large amounts of data, e.g. in large databases, but they can also handle ASCII characters (since they are 1 B in size), but not UNICODE characters that require character flow!

Example 4.3 - I / O stream byte

- A program that loads bytes from an input file and writes them to an output file - we use it for binary data and ASCII

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class Copy Bytes {
        public static void main (String [] args ) throws IOException {
                FileInputStream entrance = null ;
                FileOutputStream output = null ;
                try {
                        entrance = new FileInputStream ( "ulazBajt.txt" );
                        output = new FileOutputStream ( "outputByte.txt" );
                        int c ;
                        while (( c = entrance .read ())! = -1) {
                                output .write ( c );
                        }
                }
                . . .
```

Example 4.3 - I / O stream byte

- A program that loads bytes from an input file and writes them to an output file - we use it for binary data and ASCII

```
. . .
catch ( IOException e ) {System. out . println ( e .getMessage ());

}
finally {
                    if ( entrance ! = null ) {
                                entrance .close ();
                    }
                    if ( output ! = null ) {
                                output .close ();
                    }
          }
}
}
```

Example 4.4 - I / O flow character

- Program that loads bytes from the input file and writes them to the output file - use for UNICODE characters

```
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class Copy Bytes {
        public static void main (String [] args ) throws IOException {
                FileReader entrance = null ;
                FileWriter output = null ;
                try {
                        entrance = new FileReader ( "ulazKarakter.txt" );
                        output = new FileWriter ( "outputKarakter.txt" );
                        int c ;
                        while (( c = entrance .read ())! = -1) {
                                output .write ( c );
                        }
                }
                . . .
```

Example 4.4 - I / O flow character

- A program that loads bytes from an input file and writes them to an output file - binary for UNICODE characters

```
. . .
catch ( IOException e ) {System. out . println ( e .getMessage ());

}
finally {
                    if ( entrance ! = null ) {
                                entrance .close ();
                    }
                    if ( output ! = null ) {
                                output .close ();
                    }
          }
}
}
```

Buffered data streams

- Buffered data streams are especially important when working with large files

- The buffered input stream reads data from a portion of memory known as a buffer; the native input API is called only when the buffer is empty. Similarly, a buffered output stream writes data to the buffer and the native output API is called only when the buffer is full
- In order to use buffered I / O in the previous example, it is necessary to call the appropriate constructors:

```
input = new BufferedReader ( new FileReader ( "Unos.txt" )); output = new BufferedWriter ( new FileWriter ( "Print.txt" ));
```

- BufferedInputStream i BufferedOutputStream create buffered byte streams, doc BufferedReader i
  BufferedWriter create buffered character streams

Class Scanner

- Class Scanner works as a wrapper around the input data source.
  The source can be Reader, InputStream, String or File

- Scanner works with tokens (the shortest meaningful through characters) and delimiters

- Example of using delimiters:

```
String entrance = "10 tea 20 coffee 30 fruit juice" ; Scanner s = new Scanner ( entrance
) .useDelimiter "\\ s" ); System. out . println ( s .nextInt ());

System. out . println ( s .next ());
System. out . println ( s .nextInt ());
System. out . println ( s .next ());
s .close ();
```

Output:

10

tea

20

coffee

Example 4.5 - class Scanner

- Example:

```
import java.util.Scanner;

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

                Scanner sc = new Scanner (System. in );
                System. out . println ( "Enter your JMBG:" ); String jmbg = sc .next ();
                System. out . println ( "Enter your name:" ); String name = sc .next ();
                System. out . println ( "Enter your salary:" );


                double salary = sc .nextDouble (); System. out . println ( "JMBG:" + jmbg + "Name:" + name
                +
                                                        "Salary:" + salary );
                sc .close ();
        }
}
```

Serialization of objects during U / I

- Object serialization is the process of representing objects as data sequences of primitive types that can become elements of byte or character flows. Upon entry, the serialized data should be loaded and a copy of the original object reconstructed based on them.

- There are ready-made classes in Java for this purpose ObjectInputStream i ObjectOutputStream
- Methods for I / O work with objects are readObject (), u ObjectInputStream, i writeObject (Object obj) u ObjectOutputStream. These methods can generate an IOException

- ObjectInputStream i ObjectOutputStream work only with class objects that implement the interface Serializable

Task for class work

- Make a package employed and implement it:

1. An abstract class Worker whose protected data are: employee's name, employee's last name, JMBG, current account number and education coefficient, and public: method for loading employee data from a text file, method for entering name, surname, current account number and employee's salary (for a given value of labor costs) in one line of a text file and an abstract method for calculating the worker's salary.

2. class ActiveWorker derived from an abstract class Worker, which as a private data contains a variable coefficient (accordingly, the loading method should be predefined).
3. class RadnikNaBolovanju also derived from the class Worker.

   In class Main create a payroll of employees of one company based on the contents of the input file *spisak.txt.*

Task for class work

NOTE: The salary of an active worker should be calculated according to the form:

*salary = (expertSecurity coefficient + variableCoefficient) * labor price*

and the salary of sick workers according to the form:

*0.8 * expertSecurity Coefficient * Labor Price*

The input file contains first the price of labor, then the number of workers in the company, and then the data on all workers. Data on one worker starts with a line in which the symbol + or - is written. The symbol + indicates that the data on the active worker follow, and - that the data on the worker who is currently on sick leave follow. Print the payroll to the output file *plate.txt.*

Note: class envelopes for primitive types from can be used for conversion java.lang such as Integer:

Example for input: n = Integer.parseInt (bafUlaz.readLine ());

Example output: dat.write (new Double (plata) .toString ());

Exercise task

- Make a package mathematics and implement it:

1. Interface Function which contains methods for loading function parameters from a text file, calculating the function value at a given point, examining whether the function has real zeros, and finding the function zeros.

2. Class LinearFunction (to represent shape functions
   y = ax + b) which implements the interface Function.

3. Class SquareFunction (to represent shape functions
   $y = ax^2 + bx + c)$ which also implements the interface Function.

In class Main define two variables of interface type
Function, assign one object type LinearFunction,
another type SquareFunction. Load function parameters from files *function1.txt* i *funkcija2*
respectively, and print their zeros (if any) in the output files *nule1.txt* i

nule2.txt, respectively.

JAVAPLATFORM