Files

Input/Output

Files

- In Java all files(including directories) are objects
- Files(including directories) are considered part of the I/O API
- The File class is java.io.File
- The constructor creates a File only by its path
  - Path is the way to the file
  - Relative path – the path from the root of the project to the accessed file
  - Absolute path – path from the root of the file system tree to the accessed file

Files

File constructors

Using absolute path to file

```
File file1 = new File("C:\\Softacad\test.txt");
File file2 = new File("..","\\test.txt");
File file3 = new File("C:\\Softacad");
System.out.println(file1==file2);
boolean file1Exists = file1.exists();
boolean file1IsFile = file1.isFile();
boolean file3IsDir = file3.isDirectory();
File files[]= file3.listFiles();
file1.delete();
```

Using relative path to file

Checks the file for existance

Prints false, although the file is one and the same

Checks file type- Either file or directory

Lists all files in the directory

Deletes the file

Exercise

- Create a class FileDemo with a main method
  - Check for folder *iotest* in directory D:\javaTest and if it does not exist, create it firstly
  - Create a new file *test.txt* in *iotest*
  - Via Windows Explorer, create 3 files in the iotest dir
  - List all of the files from dir iotest and write their names in the console
  - Delete files, with names, starting with letter „t“

Introduction

- Input/Output – basic and main mechanism for reading and writing data from and to external resources

  for example reading from files, writing to files

- System.out.println() is already known approach for writing data

  therefore System.out.println() is part of I/O

- I/O functionality in java is kept in the package java.io

Basic Input and Output

Input and Output is a flow of data i.e of bytes and because of this flow I/O objects are called streams

Example: Imagine live streaming of TV program or Radio streaming

Logically *Input* is used for reading (i.e the data comes in) and *Output* for writing (i.e the data goes out)

Basic Input and Output

The streaming means and derives from the following features

- The data in the stream is ordered – we can't swap the position of the bytes
- The data flows sequentially – i.e byte X can't be read before all bytes from 1 to X-1 have been read. Data can't be accessed randomly.
- Reading and Writing always starts from the beginning
- Reading can proceed till end of file

Basic Input and Output

- Reading and writing occurs only in one direction – forwards. Streams can't go backwards – i.e once a byte has been read it can't be read again by the same stream
- When reading or writing to file a connection to the file is open so after reading or writing the stream MUST be closed
- Each connection should be closed every time the work with the file ends
- End of file is presented through byte value of -1

Streams in Java

- Byte streams
- Character streams
- Buffered streams
- Object streams
- Scanner and PrintStream

Streams in Java

- On top of all streams are
  - InputStream for reading
  - OutputStream for writing
- InputStream and OutputStream are abstract and can't be instantiated

Streams in Java

- InputStream reads data through the method read()
- read(byte[] array) – writes the read data into the byte array
- OutputStream writes data through the method write()
- If the read byte is -1 the End Of File is reached
- OutputStream writes the data into the file
- IOException and FileNotFoundException should be caught and handled

File Streams

- FileStreams are basic byte streams
- Using the methods of InputStream and OutputStream:
  - Read the bytes from test.txt
  - Write some new data into it

FileInput/OutputStream

```
FileInputStream input = new FileInputStream(file);
int b=0;
while(b!=-1){
    b= input.read();
}
input.close();
```

While the read byte is different from EOF

Reading byte and assigning it to b

Closing the stream

```
FileOutputStream output = new FileOutputStream(file);
output.write(24);
output.write('c');
output.close();
```

Writing bytes

Closing the stream

Exercise

Create a program that compare two .jpg files

Character Streams

- The character streams are Reader and Writer
- Reader and Writer are specified only for characters
- Reader and Writer are abstact and refer to charachter files as InputStream and OutputStream do for byte streams
- FileWriter and FileReader can be used to write/read text to/from files.

Scanner and PrintStream

- Scanner is a utility class which uses regular expressions for easier parsing a character source
- Scanner is not a stream but can work as such

  or it can use a stream to file as well
- PrintStream can be used for writing into a file

Scanner and PrintStream

Defining Scanner through a file

```
Scanner sc = new Scanner(file);
while(sc.hasNextLine()) {
    System.out.println(sc.nextLine());
}
```

```
InputStream stream = new FileInputStream(file);
Scanner sc = new Scanner(stream);
while(sc.hasNextLine()) {
    System.out.println(sc.nextLine());
}
```

Defining Scanner through a stream

Serialization and Serializable

- It's possible to read bytes from file, write bytes from file, read and write strings. What about an object?
- Serialization is the feature for presenting an object as a sequence of bytes.
- Deserialization is the feature of presenting bytes to objects
- Serializable in java is an interface which marks the class for able for serialization
- Serializable doesn't have any methods

Serializable and object streams

- Writing an object as a sequence of bytes into a file is enabled through the ObjectOutputStream
- Reading objects as a sequence of bytes from a file is enabled through ObjectInputSteam
- Non-serializable objects cannot be written – java.io.NotSerializableException is thrown
- If a field should not be serialized it should be marked as transient

Object Streams

```
OutputStream os = new FileOutputStream(file);
ObjectOutputStream oos = new ObjectOutputStream(os);
oos.writeObject(bmw);
oos.close();
os.close();

InputStream is = new FileInputStream(file);
ObjectInputStream ois = new ObjectInputStream(is);
Car isBMW = (Car) ois.readObject();
is.close();
ois.close();
```

Serializing object

Deserializing object

```
public class Car implements Serializable{
    String model;
    String color;
    transient double price;
    transient Person owner;
}
```

These fields will be serialized

These fields won't be serialized

Serializable and object streams

- Exercise: Create a class which implements Serializable. Create instances of it and write them to a file. Try reading them through ObjectInputStream and FileInputStream