תוכנה 1

תרגול מס' 4

שימוש במחלקות קיימות:

קלט/פלט (IO)

OOP and IO

- IO – Input/Output
- What is IO good for?
- In OOP services are united under Objects
- IO is also handled via predefined classes
- These objects are defined in the java.io package

The java.io package

- The java.io package provides:
  - Classes for reading input
  - Classes for writing output
  - Classes for manipulating files
  - Classes for serializing objects

Online Resources

- JAVA API Specification:

  http://java.sun.com/j2se/1.6.0/docs/api/index.html

- The Java Tutorial (Sun)

  http://java.sun.com/docs/books/tutorial/essential/io/

Handling IO Problems

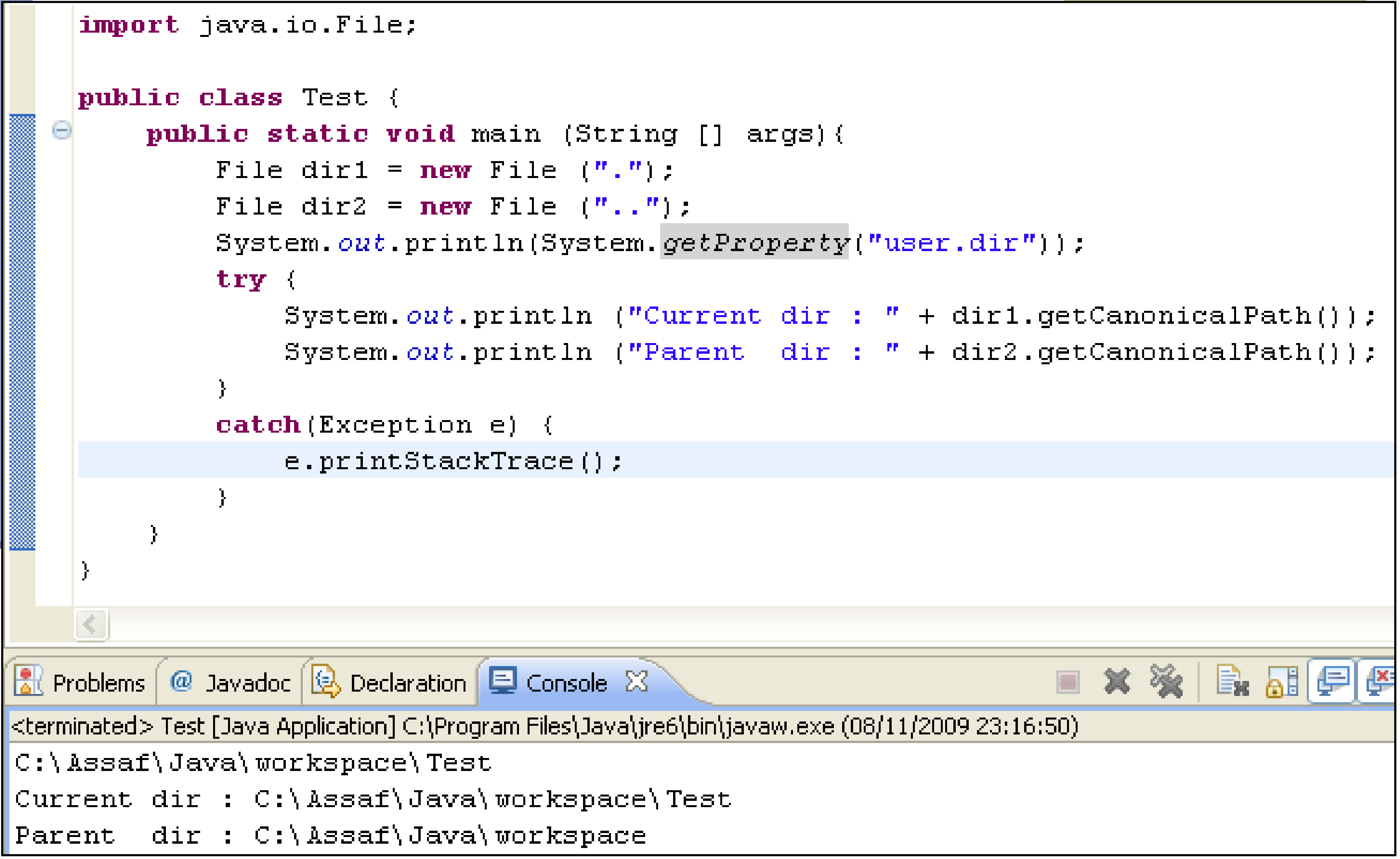
- The result of an IO operation might be problematic
- Thus IO operations are defined as “throwing” exceptions
- We shall learn about it later this course
- Meanwhile, we just have to know how to handle it
- Try-catch block

The File Class

- Represents a file or directory pathname
- Performs basic file-system operations:
  - removes a file: `delete()`
  - creates a new directory: `mkdir()`
  - checks if the file is writable: `canWrite()`
- No method to create a new file
- No direct access to file data
- Use file streams for reading and writing (later)

The File Class

Example – get the current directory



```
import java.io.File;

public class Test {
    public static void main (String [] args){
        File dir1 = new File (".");
        File dir2 = new File ("..");
        System.out.println(System.getProperty("user.dir"));
        try {
            System.out.println ("Current dir : " + dir1.getCanonicalPath());
            System.out.println ("Parent  dir : " + dir2.getCanonicalPath());
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
<terminated> Test [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (08/11/2009 23:16:50)
C:\Assaf\Java\workspace\Test
Current dir : C:\Assaf\Java\workspace\Test
Parent  dir : C:\Assaf\Java\workspace
```

The File Class
Constructors

- Using a full pathname:

```
File f = new File("/doc/foo.txt");
File dir = new File("/doc/tmp");
```

- Using a pathname relative to the current directory of the Java interpreter:

```
File f = new File(“foo.txt”);
```

Note: `System.getProperty(“user.dir”)` returns the current directory of the interpreter

File Tests and Utilities

- File information:
  - `String getName()`
  - `String getPath()`
  - `String getAbsolutePath()`
  - `String getParent()`
  - `long lastModified()`
  - `long length()`

- File modification:
  - `boolean renameTo(File newName)`
  - `boolean delete()`

File Tests and Utilities

- Directory utilities:
  - `boolean mkdir()`
  - `String[] list()`

- File tests:
  - `boolean exists()`
  - `boolean canWrite()`
  - `boolean canRead()`
  - `boolean isFile()`
  - `boolean isDirectory()`
  - `boolean isAbsolute()`

The File Class
Pathnames

- Pathnames are system-dependent
  - `"/doc/foo.txt"` (UNIX format)
  - `"D:\doc\foo.txt"` (Windows format)

- On Windows platform Java excepts path names either with `'/'` or `'\'`

- The system file separator is defined in:
  - `File.separator`
  - `File.separatorChar`

The File Class
Directory Listing

- Printing all files and directories under the current directory:

```
File file = new File(".");

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

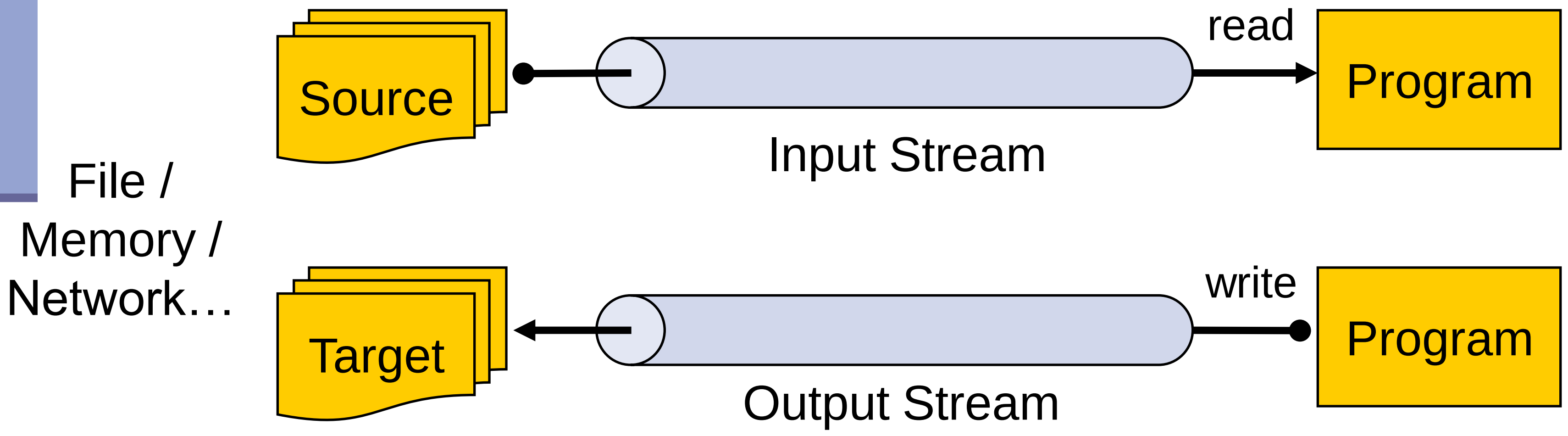
The File Class
Directory Listing (cont.)

- Print all files and directories under a given directory with ".txt" suffix

```
File file = new File("C:/Assaf/Junk/");
String[] files = file.list();
for (int i=0 ; i<files.length ; i++) {
     if (files[i].endsWith(".txt"))
          System.out.println(files[i]);
}
```

Streams

- A ***stream*** is a sequential flow of data
- Streams are one-way streets.
  - ***Input streams*** are for reading
  - ***Output streams*** are for writing



Streams

- Usage Flow:

```
open a stream
while more information
    Read/write information
close the stream
```

- All streams are automatically opened when created.

Streams

- There are two types of streams:
  - ***Byte streams*** for reading/writing raw bytes
  - ***Character streams*** for reading/writing text

- Class Name Suffix Convention:

	Byte	Character
Input	InputStream	Reader
Output	OutputStream	Writer

Terminal I/O

- The System class provides references to the standard input, output and error streams:

```
InputStream stdin = System.in;
PrintStream stdout = System.out;
PrintStream stderr = System.err;
```

The Scanner Class

- Breaks its input into tokens using a delimiter pattern (default: whitespace)
- http://www.j2ee.me/javase/6/docs/api/java/util/Scanner.html
- The resulting tokens may then be converted into values

```
Scanner s = new Scanner(System.in);
int anInt = s.nextInt();
float aFloat = s.nextFloat();
String aString = s.next();
String aLine = s.nextLine();
```

How can we be sure that user will type-in the correct input?

Example - Scanner
Set delimiters

```
String input = "1 fish 2 fish red fish blue fish";
Scanner s =
   new Scanner(input).useDelimiter(" *fish *");
while (s.hasNext())
      System.out.println(s.next());
s.close();
```

Example - Scanner
Input from the user

```
Scanner s = new Scanner(System.in);
System.out.println("enter line:");
while (s.hasNext())
      System.out.println(s.next());
```

Example
The whole loop

- Input from the user:
  - Directory
  - File suffix
- Output: all file that match the given suffix file-type at the given directory

Example
The whole loop

```
Scanner s = new Scanner(System.in);
System.out.println("Please enter directory and file-suffix:");
String dir = s.next();
String suffix = s.next();

File file = new File(dir);

String[] files = file.list();
for (String filename : files) {
     if (filename.endsWith(suffix))
          System.out.println(filename);
}
```

Object Serialization

- A mechanism that enable objects to be:
  - saved and restored from byte streams
  - persistent (outlive the current process)

- Useful for:
  - persistent storage
  - sending an object to a remote computer