 capacité 1

תרגול 8 – קלט/פלט
רובי בים ומתי שמרת
A Typical Program

- Most applications need to process some input and produce some output based on that input.
- The Java IO package (java.io) is to make that possible in Java.
Streams

- A program that needs to read data from a source needs an **input stream** or **reader**

- A program that needs to write data to a destination needs an **output stream** or **writer**
Sources and Destinations

- Typical sources and destinations are:
  - Files
  - Pipes (inter-process communication)
  - Network connections
  - In-memory buffers (e.g. arrays)
  - Console (system.in, System.out, System.err)

- The Java IO package provides classes to handle all types of sources and destinations.
Using Streams

- create a stream
- while more information
- read/write information
- close the stream

- This is the general flow no matter what the source / destination is
- All streams are automatically opened when created

Depends on source / destination

Does not depend on specific source / destination
Types of Streams

- Character based streams
  - Used for working with textual data

- Byte based streams
  - Used for other types of data (binary)

Class Name suffix:

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
<th>Byte</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Output</td>
<td>InputStream</td>
<td>Reader</td>
</tr>
<tr>
<td>Output</td>
<td></td>
<td>OutputStream</td>
<td>Writer</td>
</tr>
</tbody>
</table>
For now, think of Reader/Writer as an interface defining the fundamental read/write functionality

<table>
<thead>
<tr>
<th>Source / Destination</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic</td>
<td>Reader</td>
<td>Writer</td>
</tr>
<tr>
<td>Arrays</td>
<td>CharArrayReader</td>
<td>CharArrayWriter</td>
</tr>
<tr>
<td>Files</td>
<td>FileReader</td>
<td>FileWriter</td>
</tr>
<tr>
<td>Buffering</td>
<td>BufferedReader</td>
<td>BufferedWriter</td>
</tr>
<tr>
<td>String</td>
<td>StringReader</td>
<td>StringWriter</td>
</tr>
</tbody>
</table>
Java IO Class Overview (Byte based)

<table>
<thead>
<tr>
<th>Source / Destination</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic</td>
<td>InputStream</td>
<td>OutputStream</td>
</tr>
<tr>
<td>Arrays</td>
<td>ByteArrayInputStream</td>
<td>ByteArrayOutputStream</td>
</tr>
<tr>
<td>Files</td>
<td>FileInputStream</td>
<td>FileOutputStream</td>
</tr>
<tr>
<td>Data</td>
<td>DataInputStream</td>
<td>DataOutputStream</td>
</tr>
<tr>
<td>Buffering</td>
<td>BufferedInputStream</td>
<td>BufferedOutputStream</td>
</tr>
<tr>
<td>Objects</td>
<td>ObjectInputStream</td>
<td>ObjectOutputStream</td>
</tr>
</tbody>
</table>

**InputStreamReader** and **OutputStreamWriter** are bridge classes converting byte based stream to a character based one.
**IO Errors**

- When using IO operations we may encounter problems
  - Attempt to open a file that doesn’t exist
  - Corrupted data
  - ...

- An exception is thrown to indicate an error
  - We will learn a great deal more about exceptions later on

- For now, we just need to know how to handle exceptions
Handling Exceptions

1. Catch the exception and handle it
   - try-catch block
2. Ignore the exception, but indicate that you are throwing one now
   - add a throws clause at the end of your method signature

For now we will choose option 2
Console I/O

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

- System.in - InputStream
- System.out - PrintStream
- System.err - PrintStream
public static void copy() throws IOException {
    Reader in = new InputStreamReader(System.in);
    Writer out = new OutputStreamWriter(System.out);
    int c;
    while ((c = in.read()) != -1) {
        out.write(c);
        out.flush();
    }
    in.close();
    out.close();
}
Copy input to output

Don’t handle the exception, but indicate that we might be throwing one as well (the one being thrown from the method we use)

```java
public static void copy() throws IOException {
    Reader in = new InputStreamReader(System.in);
    Writer out = new OutputStreamWriter(System.out);

    int c;
    while ((c = in.read()) != -1) {
        out.write(c);
        out.flush();
    }
    in.close();
    out.close();
}
```

Don't handle the exception, but indicate that we might be throwing one as well (the one being thrown from the method we use)
public static void copy() throws IOException {
    Reader in = new InputStreamReader(System.in);
    Writer out = new OutputStreamWriter(System.out);

    int c;
    Convert the input stream attached to the keyboards into a Reader

    while ((c = in.read()) != -1) {
        out.write(c);
        out.flush();
    }

    in.close();
    out.close();
}
public static void copy() throws IOException {
    Reader in = new InputStreamReader(System.in);
    Writer out = new OutputStreamWriter(System.out);

    int c;
    Convert the PrintStream (byte based) attached to the console into a Writer
    
    }
    in.close();
    out.close();
}
public static void copy() throws IOException {
    Reader in = new InputStreamReader(System.in);
    Writer out = new OutputStreamWriter(System.out);

    int c;
    while ((c = in.read()) != -1) {
        out.write(c);
        out.flush();
    }
    in.close();
    out.close();
}
Java Files

- To read from a file use FileInputStream or FileReader
- To write to a file use FileOutputStream or FileWriter
- To access information about a file (length, exist?, directory?) use the File class
Copy one file to another

The path to the file we’re reading from e.g. C:\Software1\example.txt

```java
public static void copy(String src, String dst) throws IOException {
    Reader in = new FileReader(src);
    Writer out = new FileWriter(dst);

    int c;
    while ((c = in.read()) != -1) {
        out.write(c);
        out.flush();
    }
    in.close();
    out.close();
}
```

The path to the file we’re writing to

Create a FileReader

Create a FileWriter

Exactly as before
The File Class

- Represents pathname (file or directory)
- Retrieve meta data about a file
  -.isFile / isDirectory
  - length
  - exists
  - …
- Performs basic file-system operations:
  - removes a file: delete()
  - creates a new directory: mkdir()
  - checks if the file is writable: canWrite()
  - …
public class DirectoryListing {
    public static void main(String[] args) throws IOException {
        File file = new File(args[0]);
        System.out.println("Path = " + file.getCanonicalPath());
        if (file.isDirectory()) {
            for (File f : file.listFiles()) {
                System.out.printf("%c\t%10s\t%dn", f.isDirectory() ? 'd' : 'f', f.getName(), f.length());
            }
        }
    }
}
Parsing

- Breaking text into a series of tokens
- The **Scanner** class is a simple text scanner which can parse primitive types and strings using regular expressions
- The source can be a stream or a string
The Scanner Class

- Breaks its input into tokens using a delimiter pattern (default: whitespace)
- The resulting tokens may then be converted into values

```java
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 the user will type-in the correct input?
Example - Scanner

```java
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();
```
Reverse Polish Notation

Reverse Polish notation (RPN) is a mathematical notation wherein every operator follows all of its operands.

- $3 \ 4 + = 7$
- $3 \ 2 * = 6$
- $5 + ((1 + 2) * 4) - 3$ is $5 \ 1 \ 2 + 4 * + 3 -$ in RPN
Evaluating RPN Expressions

While there are input tokens left
- Read the next token from input
- If the token is a value
  - Push it onto the stack
- Otherwise, the token is an operator
  - Pop the top values from the stack
  - Evaluate the operator, with the values
  - Push the returned results, back onto the stack

If there is only one value in the stack
- That value is the result of the calculation
Simple RPN Calculator

```java
public class RPNCalculator {

    public double evaluate(String expr) {
        return evaluate(new Scanner(expr));
    }

    private double evaluate(Scanner s) {
        ...
    }

    ...
}
```

See supplement code
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/