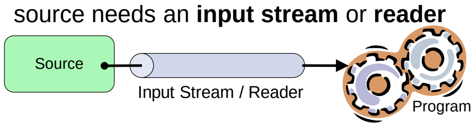
תוכנה 1

תרגול 8 – קלט/פלט

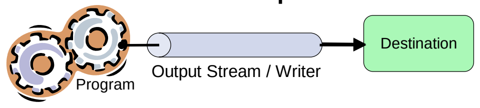
הדס צור ואסף זריצקי

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

Depends on source / destination

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

Does not depend on specific source / destination

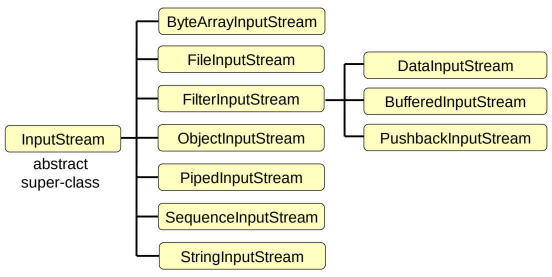
- This is the general flow no matter what the source / destination is
- All streams are automatically opened when created

Streams

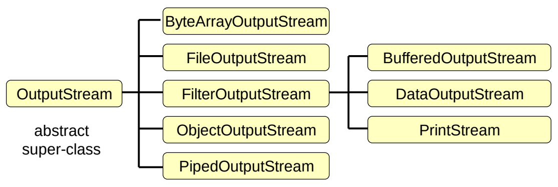
- There are two categories of streams:
  - ***Byte streams*** for reading/writing binary data
  - ***Character streams*** for reading/writing text
- <u>Suffix Convention</u>:

category / direction	Byte	Character
Input	InputStream	Reader
Output	OutputStream	Writer

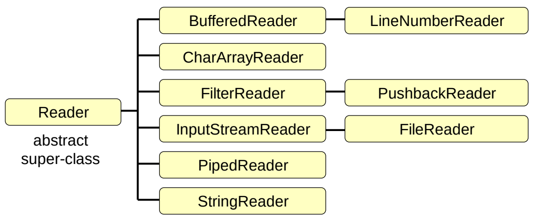
InputStream Class Hierarchy



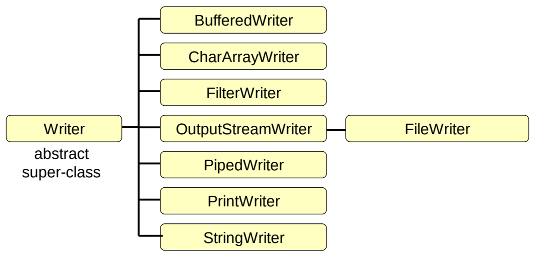
OutputStream Class Hierarchy



Reader Class Hierarchy



Writer Class Hierarchy



Handling Exceptions

- Handle exception
  - using a try-catch block
- Propagate the exception to the caller
  - Add throws declaration
- finally block is always executed at the end of the try block

Console I/O

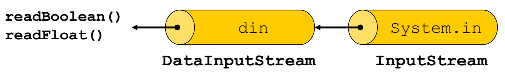
- The System class provides references to the standard input, output and error streams:

```
System.in    - InputStream
System.out   - PrintStream
System.err   - PrintStream
```

Stream Wrappers

- Some streams wrap others streams and add new features.
- A wrapper stream accepts another stream in its constructor:

```
DataInputStream din =
    new DataInputStream(System.in);
double d = din.readDouble();
```



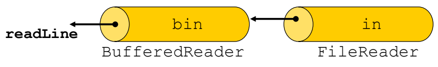
Stream Wrappers Example

- Reading a line of text from a file:

```
try {
    FileReader in =
        new FileReader("FileReaderDemo.java");

    BufferedReader bin = new BufferedReader(in);

    String text = bin.readLine();
    ...
} catch (IOException e) {...}
```



Copy input to output

```
public static void copy() throws IOException {
                                   amReader(System.in);
                                   eamWriter(System.out);

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

Read a single character from the source

-1 indicates the end of the input

Write a single character to the destination

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)

```
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();
}
```

might throw an exception

Copy input to output

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

    int c;

    }
    in.close();
    out.close();
}
```

Convert the input stream attached to the keyboards into a Reader

Copy input to output

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

    int c;

    }
    in.close();
    out.close();
}
```

Convert the PrintStream (byte based) attached to the console into a Writer

Copy input to output

```
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();
}
```

Close the streams once we're done

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`

The path to the file we're writing to

```
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();
}
```

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()
  - …

Directory Listing

```
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%d\n",
                             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

```
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?

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/