תוכנה 1

תרגול 9: קלט/פלט מבואר

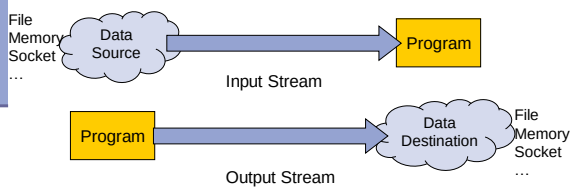
אסף זריצקי ומתי שמרת

I/O Streams

- An ***I/O Stream*** represents an input source or an output destination
  - disk files, network, memory etc.
- Simple model: a sequence of data
- All kind of data, from primitive values to complex objects

I/O Streams

- Streams are one-way streets
  - ***Input*** streams for reading
  - ***Output*** streams for writing



Streams Usage Pattern

- Usage Flow:

```
create a stream
while more data
    Read/Write data
close the stream
```

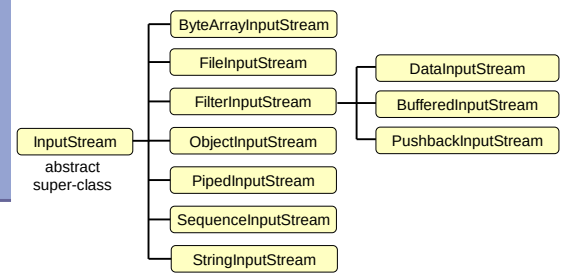
- All streams are automatically opened when created

Streams

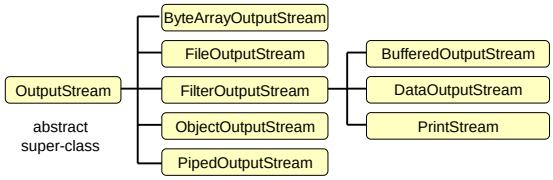
- There are two categories of streams:
  - ***Byte streams*** for reading/writing binary data
  - ***Character streams*** for reading/writing text
- Suffix Convention:

direction \ category	Byte	Character
Input	InputStream	Reader
Output	OutputStream	Writer

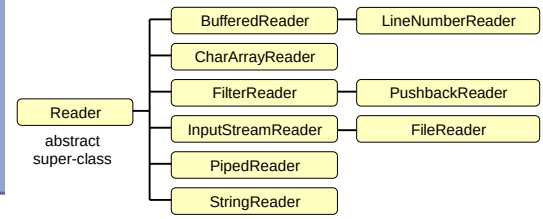
InputStreams Hierarchy



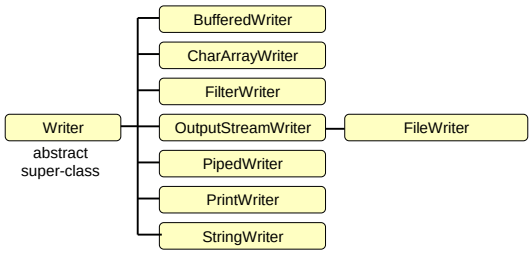
OutputStreams Hierarchy



Readers Hierarchy



Writers Hierarchy



InputStream Example

- Reading a single byte from the standard input stream:

```
try {
    int value = System.in.read();
    if (value != -1) {
        ...
    }
    ...
} catch (IOException e) {
    ...
}
```

an int with a byte information

is thrown in case of an error

returns -1 if a normal end of stream has been reached

Character Stream Example

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

    try {
        in = new FileReader(src);
        out = new FileWriter(dst);

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

Create streams

Copy input to output

Close streams

Almost

```
public static void copy(String src, String dst)
    throws IOException {
        ...
    } finally {
        closeIgnoringException(in);
        closeIgnoringException(out)
    }
}

private static void closeIgnoringException(Closeable c) {
    if (c != null) {
        try {
            c.close();
        } catch (IOException e) {
            // Deliberately left empty; There is nothing we
            // can do if close fails
        }
    }
}
```

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();
```

13

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) {...}
```

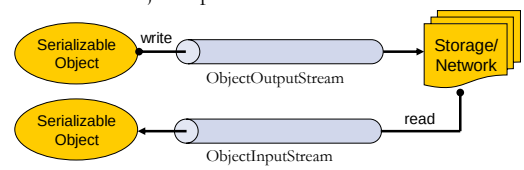
Object Serialization

- 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

15

The Default Mechanism

- The default mechanism includes:
  - The Serializable interface
  - The ObjectOutputStream
  - The ObjectInputStream



16

The Serializable Interface

- Objects to be serialized must implement the java.io.Serializable interface
- An empty interface
- Some types are Serializable:
  - Primitives, Strings, GUI components etc.
- Subtypes of Serializable types are also Serializable

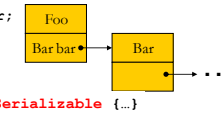
17

Recursive Serialization

- Can we serialize a Foo object?

```
public class Foo implements Serializable {
    private transient Bar bar;
    ...
}

public class Bar implements Serializable {…}
```



- No, since Bar is not Serializable
- Solutions:
  1. Implement Bar as Serializable
  2. Mark the bar field of Foo as transient
  3. Customize the serialization process

18

HashMap Serialization

```
Map<Integer, String> map = new HashMap<…>();
...
ObjectOutputStream out = null;
try {
    out = new ObjectOutputStream(
                  new FileOutputStream("map.s"));
    out.writeObject(map);
} catch (IOException e) {
    ...
} finally  {
    ...
}
```

HashMap is Serializable, so are all the other **concrete** collection types we've seen

19

Reading Objects

```
ObjectInputStream in = null;
try {
    in = new ObjectInputStream(
              new FileInputStream("map.s"));
    Map<Integer, String> map =
        (Map<Integer, String>) in.readObject();
    System.out.println(map);
} catch (Exception e) {
    ...
} finally {
    ...
}
```

20

Reminder – The File Class

- Abstract representation of files and directories
- Meta information, no data
  - Use streams for data
- Immutable

21

Java 7 – nio2

- Path equivalent of File
  - manipulate pathes
  - locate file / directories
- Better support for directory listing
- Support for symbolic links
- File change notification
- Define your own file system (advanced)

22