JAVA Cheat Sheet 6

I/O

"A **stream** is a sequence if data."
"A program uses an **input stream** to read data from a source."
"A program uses an **output stream** to write data to a destination." *(docs.oracle.com)*

Byte Streams

In Java all stream types are built on **ByteStreams**.

```
FileInputStream in = new FileInputStream(fileName);
FileOutputStream out = new FileOutputStream(fileName);
int c;
while((c = in.read()) != -1){
   out.write(c);
}
```

Character Streams

FileReader and **FileWriter** use *FileInputStream* and *FileOutputStream* internally.

```
FileReader in = new FileReader(fileName);
FileWriter out = new FileWriter(fileName);
int c;
while((c = in.read()) != -1){
   out.write(c);
}
```

Use ints to store last 16 bits instead of last 8 bits.

Line-Oriented I/O

BufferedReader.readLine() uses line terminators to split the lines.

```
BufferedReader in = new BufferedReader(new
FileReader(fileName));
PrintWriter out = new PrintWriter(new
FileWriter(fileName));
String line;
while((line = in.readLine()) != null){
   out.println(line);
}
```

Scanner

```
Scanner s = new Scanner(¡Stream¿);
while(s.hasNext()){
   doSomething...
}
```

Using different delimiter with: *s.useDelimiter(Regex);*

I/O From Command Line

InputStreamReader in = new InputStreamReader(System.in);
Or use **Console**:
Console c = System.console();
Returns *null* if not available.

Data Streams

Streams to read and write primitive data types.

```
DataInputStream in = new DataInputStream(new
FileInputStream(fileName));
DataOutputStream out = new DataOutputStream(new
BufferedOutputStream(new FileOutputStream(fileName)));
in.readDouble(); / out.writeDouble(someDouble);
in.readInt(); / out.writeInt(someInt);
in.readUTF(); / out.writeUTF(someString);
```

Object Streams

Java objects can be written to files if they implement the **Serializable** marker interface.
Every reference inside this object will also be written to the file.
Classes to use are **ObjectInputStream** and **ObjectOutputStream**.

File I/O With NIO

New since Java7.
Class **Path** to represent a path.
Path p = Paths.get("/tmp/foo");
For releasing resources after use, use **try-with-resources**:
try(BufferedWriter writer = Files.newBufferedWriter(file, charset));
For file handling use class **Files**.
Files.write(Path, byte[], OpenOption...);
OpenOptions:
{WRITE, APPEND, TRUNCATE_EXISTING, CREATE_NEW, CREATE, DELETE_ON_CLOSE, SPARSE, SYNC, DSYNC}