

## Java I/O

### Reference:

[java.sun.com/docs/books/tutorial/essential/io/](http://java.sun.com/docs/books/tutorial/essential/io/)

Java Programming: I/O

1

---

---

---

---

---

---

---

---

## Java Support for I/O

- Traditional Java IO (package **java.io**)
- "New" stuff (package **java.nio**)
  
- We will focus on java.io (for now).
  
- Very different than C or C++ I/O!
  - printf/scanf, cin, cout

Java Programming: I/O

2

---

---

---

---

---

---

---

---

## Java IO Classes

- Java I/O is based on a class hierarchy.
- Base classes are used to describe the basic functionality required.
- Derived classes provided the functionality for specific kinds of I/O environments.
  - Files, Pipes, Networks, IPC, etc.
- The Java I/O package is a nice example of OOP programming!

Java Programming: I/O

3

---

---

---

---

---

---

---

---

## Some Abstract Base Classes

- **InputStream**
  - for reading a stream of bytes
- **OutputStream**
  - for writing a stream of bytes
- **Reader**
  - reading a stream of characters
- **Writer**
  - writing a stream of characters.

Java Programming: I/O

4

---

---

---

---

---

---

---

---

## Class **InputStream**

- All *byte* stream readers are based on the class **InputStream** (they extend **InputStream**)
- Some of the methods:
  - `int read()` throws `IOException`
  - `int read(byte[] b)` throws `IOException`
  - `int available()` throws `IOException`
  - `void close()` throws `IOException`

Java Programming: I/O

5

---

---

---

---

---

---

---

---

## Class **OutputStream**

- All *byte* stream writers are based on the class **OutputStream** (they extend **OutputStream**)
- Some of the methods:
  - `void write(int b)` throws `IOException`
  - `void write(byte[] b)` throws `IOException`
  - `void flush()` throws `IOException`
  - `void close()` throws `IOException`

Java Programming: I/O

6

---

---

---

---

---

---

---

---

### throws `IOException`

- Everything here can throw an `IOException`, so I'll stop mentioning it.
- If you use any of these methods, you must catch `IOException` (or `Exception`).

---

---

---

---

---

---

---

---

### Class `Reader`

- All *character* stream readers are based on the class `Reader` (they extend `Reader`)
- Some of the methods:
  - `int read()`
  - `int read(char[] cbuf)`
  - `boolean ready()`
  - `void close()`

---

---

---

---

---

---

---

---

### Class `Writer`

- All *character* stream writers are based on the class `Writer` (they extend `Writer`)
- Some of the methods:
  - `void write(int c)`
  - `void write(char[] cbuf)`
  - `void write(String str)`
  - `void flush()`
  - `void close()`

---

---

---

---

---

---

---

---

## Abstract Classes

- You can't create an `InputStream` object!
  - or `OutputStream` or `Reader` or `Writer`
- You create some object of a derived class
  - a specific kind of `InputStream` object, perhaps a `FileInputStream`
  - but your code can be written to work with any kind of `InputStream`

Java Programming: I/O

10

---

---

---

---

---

---

---

---

## `System.out`

- `System.out` is actually a kind of `OutputStream`:
  - it's a `PrintStream` object, but you don't need to know that to use it as an `OutputStream`

```
OutputStream stdout = System.out;
stdout.write(104); // ASCII 'h'
stdout.flush();
```

Java Programming: I/O

11

---

---

---

---

---

---

---

---

Sample Code: [SysOut.java](#)

## IOExceptions!

```
public static void main(String[] args) {
    OutputStream stdout = System.out;

    try {
        stdout.write(104); // 'h'
        stdout.write(105); // 'i'
        stdout.write(10); // '\n'
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Java Programming: I/O

12

---

---

---

---

---

---

---

---

## Another way...

```
public static void main(String[] args)
    throws IOException {

    OutputStream stdout = System.out;
    stdout.write(104); // 'h'
    stdout.write(105); // 'i'
    stdout.write(10); // '\n'
}
```

Java Programming: IO

13

---

---

---

---

---

---

---

---

Sample Code: [SysIn.java](#)

### System.in

- System.in is a type of InputStream

```
byte[] b = new byte[10];
InputStream stdin = System.in;
stdin.read(b);
```

```
System.out.write(104); // ASCII 'h'
System.out.flush();
```

Java Programming: IO

14

---

---

---

---

---

---

---

---

## try it with available()

```
InputStream stdin = System.in;
byte[] b = new byte[stdin.available()];
int len = stdin.read(b);
for (int i=0;i<len;i++)
    System.out.write(b[i]);
System.out.flush();
```

Sample Code: [SysInAvailable.java](#)

Java Programming: IO

15

---

---

---

---

---

---

---

---

## Character Streams

- Remember that Java supports unicode!
  - support for non-ASCII characters.
- Java also supports ASCII (obviously).
- BUT – since characters are not necessarily a single byte – we need more than byte streams.

Java Programming: I/O

16

---

---

---

---

---

---

---

---

## InputStreamReader

- An `InputStreamReader` is a bridge from byte streams to character streams.
  - When you create one you can specify the encoding (or use the system default).
  - You need to have an `InputStream` to feed the `InputStreamReader`.
- Each call to `InputStreamReader.read()` returns one char, but may read multiple bytes from the `InputStream`.

Java Programming: I/O

17

---

---

---

---

---

---

---

---

## InputStreamReader attached to stdin

```
InputStreamReader isr = new
    InputStreamReader( System.in );

char c;
c = (char) isr.read();
System.out.write( c);
```

Sample Code: [InputReader.java](#)

Java Programming: I/O

18

---

---

---

---

---

---

---

---

## BufferedReader

- A type of `Reader` that does internal buffering.
  - more efficient.
- Provides everything from `Reader`, plus:

`String readLine()`

- reads up to `'\n'`, `'\r'` (or both).
- `String` returned does not include the line termination char(s).

Java Programming: I/O

19

---

---

---

---

---

---

---

---

## Attaching a `BufferedReader` to `stdin`

```
InputStreamReader isr =  
    new InputStreamReader(System.in);
```

```
BufferedReader bf =  
    new BufferedReader(isr);
```

```
String foo = bf.readLine();
```

Java Programming: I/O

20

---

---

---

---

---

---

---

---

## Character Stream Output

- `OutputStreamWriter`: a bridge (converts from characters to byte stream).
- `BufferedWriter`: efficient `Writer`.

```
BufferedWriter bw =  
    new BufferedWriter(  
        new OutputStreamWriter(System.out) );
```

**Java promotes strong fingers!**

Java Programming: I/O

21

---

---

---

---

---

---

---

---

### **PrintWriter and PrintStream**

- **PrintWriter** is a writer that provides a bunch of overloaded `print()` methods.
  - can print just about anything!
- A **PrintStream** also has many `print()` methods.
- **System.out** is a **PrintStream**

Java Programming: I/O

22

---

---

---

---

---

---

---

---

### **Enough already!**

- We've looked at lots of classes, but so far all we can do is use them to mess with `stdin`, `stdout` (which can already do all this stuff).
- Next time we explore File I/O.
- Later: Network I/O.

Java Programming: I/O

23

---

---

---

---

---

---

---

---