

# Input & Output

Tim Kovacs  
kovacs@cs.bris.ac.uk

Frederik Vercauteren  
frederik@cs.bris.ac.uk

- End of line in files, `\r`, `\n` or `\r\n`
- Character set encodings: ASCII, UTF, Unicode, ...
- Binary representation of numbers: big-endian vs. little-endian
- Filename and path conventions: `/` vs. `\`

- Java package `java.io` defines more than 60 classes
- `InputStream` and `OutputStream`: reading and writing **bytes**
- `Reader` and `Writer`: reading and writing **characters**
  - Convert between Unicode and system's representation

## File Class

- Functionality to work with **file system** on user's machine
- A **File** object represents a **file or directory**
  - Name, parent directory, size, permissions
  - No methods for reading the contents of file
- Constructor takes the full file name
  - No path name supplied, then Java uses current directory

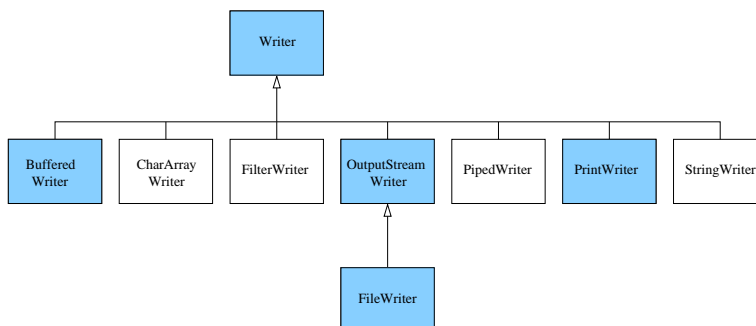
```
File test = new File("test.txt");
```
- No exception thrown if the file does not exist
  - **No file is created / opened**

## Readers & Writers

- Convert **external representations** of characters to **Unicode**
- Base abstract classes: `Reader` and `Writer`
  - Use `int` as representation for Unicode

```
abstract int read() // returns -1 at the end of the file
int read(char[] cbuf)
abstract void write(int b)
void write(char[] cbuf)
abstract void close()
```
- `InputStreamReader`: turns bytes into Unicode characters
- `OutputStreamWriter`: turns Unicode into particular encoding
- Convenience classes to work with **files**
  - `FileReader` and `FileWriter`

## Writer Classes



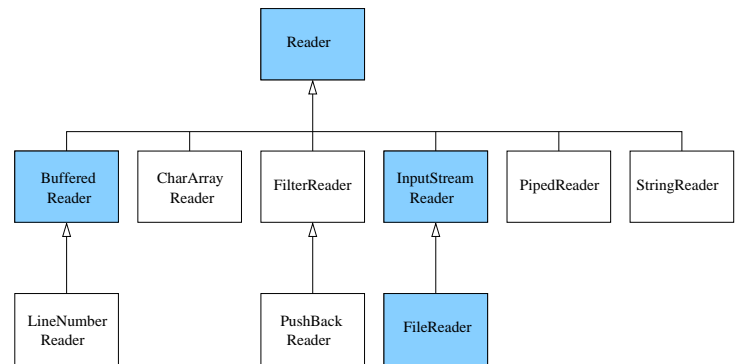
## File Class

- `File.separator`: default name-separator character

- Method summary of `File` class:

```
boolean canRead()
boolean canWrite()
boolean createNewFile()
static File createTempFile(String prefix, String suffix)
boolean delete()
void deleteOnExit()
boolean exists()
String getName()
String getParent()
String getPath()
boolean isDirectory()
boolean isFile()
long length()
File[] listFiles()
boolean mkdir()
```

## Reader Classes



## InputStreamReader OutputStreamWriter

- Methods allow reading/writing **char** or **char[]**

```
InputStreamReader(InputStream in)
InputStreamReader(InputStream in, String charsetName)
void close()
int read()
int read(char[] cbuf, int offset, int length)
```

```
OutputStreamWriter(OutputStream out)
OutputStreamWriter(OutputStream out, String charsetName)
void close()
void flush()
void write(char[] cbuf, int off, int len)
void write(int c)
void write(String str, int off, int len)
```

- Reading keystrokes from **console**

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

```

FileReader(File file)
FileReader(String fileName)

FileReader in = new FileReader("input.txt");
// Equivalent to
InputStreamReader in = new InputStreamReader(new
    FileInputStream("input.txt"));

```

- **FileWriter**: adds no extra methods to **OutputStreamWriter**

```

FileWriter(File file)
FileWriter(File file, boolean append)
FileWriter(String fileName)
FileWriter(String fileName, boolean append)

FileWriter out = new FileWriter("output.txt");
// Equivalent to
OutputStreamWriter out = new OutputStreamWriter(new
    FileOutputStream("output.txt"));

```

```

try
{
    // Try to open the file
    FileReader inputFile = new FileReader(filename);
    // Process the file's contents
    ...
    // Close the file now that you're done with it.
    inputFile.close();
}
catch(FileNotFoundException e)
{
    System.out.println("Unable to open " + filename);
}
catch(IOException e)
{
    // The file could not be read or closed
    System.out.println("Unable to close " + filename);
}
}

```

## Copying a Text File

```

void copyFile(FileReader inputFile, FileWriter outputFile)
    throws IOException
{
    // Create buffer of 1024 characters
    final int bufferSize = 1024;
    char[] buffer = new char[bufferSize];

    // Read the first chunk of characters.
    int numberRead = inputFile.read(buffer);
    while(numberRead > 0)
    {
        // Write out what was read.
        outputFile.write(buffer,0,numberRead);
        numberRead = inputFile.read(buffer);
    }

    outputFile.flush();
}

```

## Copying a Text File

```

void copyFile(BufferedReader reader, BufferedWriter writer)
    throws IOException
{
    // Read the first line.
    String line = reader.readLine();

    // null returned on EOF.
    while(line != null)
    {
        // Write the whole line.
        writer.write(line);

        // Add the newline character.
        writer.newLine();

        // Read the next line.
        line = reader.readLine();
    }
}

```

## StringTokenizer Class

- Splits strings into separate **String tokens**
  - Delimiter characters are user settable (**whitespace by default**)
  - Will also return delimiters if required

```

StringTokenizer(String str)
StringTokenizer(String str, String delim)
boolean hasMoreTokens()
String nextToken()

```

- Splitting String into words

```

String line = in.readLine();
while(line != null)
{
    StringTokenizer tokenizer = new StringTokenizer(line,";:.\\" \t");
    while(tokenizer.hasMoreTokens())
        String word = tokenizer.nextToken();
    line = in.readLine();
}

```

## Buffered Reader & Writer Classes

- Input-output is generally a **very slow process**
  - Disk and network access are often involved
- Large read/write takes little more time than a small read/write
- Buffered classes bundle **multiple read/write** operations into **fewer**
  - Read more than is required, delay writing until more is ready

```

BufferedReader(Reader in)
BufferedReader(Reader in, int sz)
String readLine() // reads line of text as String

BufferedWriter(Writer out)
BufferedWriter(Writer out, int sz)
void flush()
void newLine()

```

## PrintWriter

- Can write both **Strings** and **numbers in text format**

```

PrintWriter(OutputStream out)
PrintWriter(Writer out)
void println()
void print || println(boolean b)
void print || println(char c)
void print || println(char[] s)
void print || println(double d)
void print || println(float f)
void print || println(int i)
void print || println(long l)
void print || println(Object obj)
void print || println(String s)

```

- **Default class** to use for **writing text** to a file (**no analog for input**)

```

PrintWriter out = new PrintWriter(
    new BufferedWriter(
        new FileWriter("output.txt")));

```

## Input & Output Streams

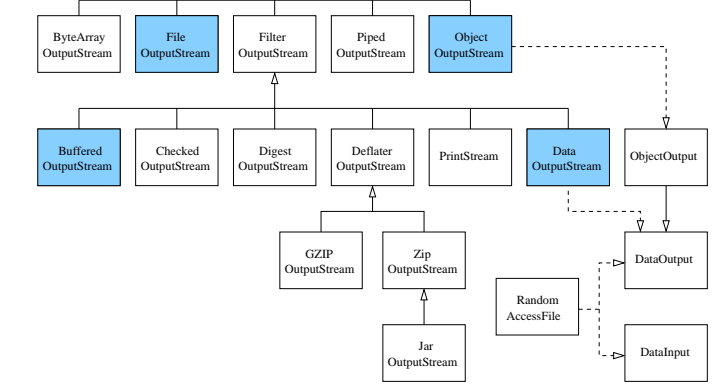
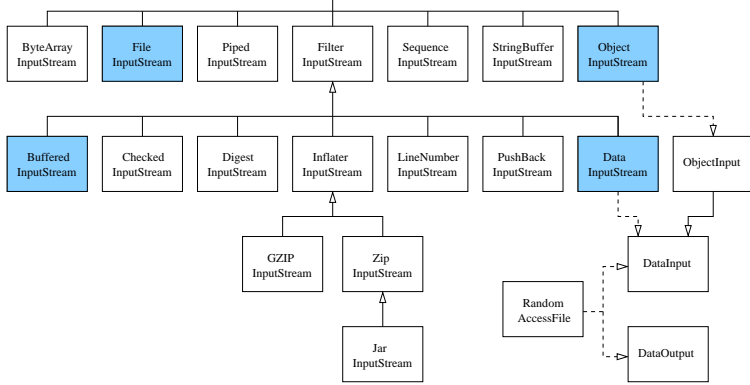
- Input and Output as a stream of raw bytes
  - Methods take **byte[]** rather than **char[]** arguments
- Base abstract classes: **InputStream** and **OutputStream**

```

abstract int read() // returns -1 at the end of stream
int read(byte[] b) // reads at most b.length bytes
int read(byte[] b, int off, int len)
int available()
void close()
long skip(long n)

abstract void write(int b)
void write(byte[] b)
void write(byte[] b, int off, int len)
void close()
void flush()

```



## Input & Output Streams

- **File{Input,Output}Stream**: stream of bytes from file
- **Buffered{Input,Output}Stream**: buffered stream of bytes
- **Data{Input,Output}Stream**: read/write primitive type values

```

FileOutputStream(File file)           || FileInputStream(File file)
FileOutputStream(String name)        || FileInputStream(String name)
FileOutputStream(File file, boolean append)
FileOutputStream(String name, boolean append)

void writeBoolean(boolean v)         || boolean readBoolean()
void writeByte(int v)                || byte readByte()
void writeChar(int v)                || char readChar()
void writeDouble(double v)           || double readDouble()
void writeFloat(float v)             || float readFloat()
void writeInt(int v)                 || int readInt()
void writeLong(long v)               || long readLong()
void writeShort(int v)               || short readShort()
void writeUTF(String str)            || String readUTF()

```

## Reading and Writing Objects

- Often you want objects to be **persistent** across program runs
- Java provides mechanism called **object serialization**
  - Class must implement **Serializable** interface
- Objects can then be read (written) from (to) ObjectStreams

```
class Employee implements Serializable { ... }
```

```

Employee fre = new Employee("Fre", 25000);

ObjectOutputStream out = new ObjectOutputStream(
    new FileOutputStream("fre.dat"));

out.writeObject(fre);
out.close();

ObjectInputStream in = new ObjectInputStream(
    new FileInputStream("fre.dat"));

Employee newFre = (Employee)in.readObject();
in.close();

```