# Introduction to using text files with Java

**Prerequisite Knowledge**

You have used Java to create programs that use standard input and output. Essentially these are special case text files that have been associated for you with input devices (the keyboard) and output devices (the screen/terminal/monitor).

You have researched online how to open, close, read, and write text files using java and have hopefully found multiple ways to declare files, have possibly made sense out of class handouts, or have prior experience with programming file i/o with Java. Your prior experience with C should give you a general idea of basic file operations.

**Environment**

The examples in the file create the java programs in a root directory where the \*.java and \*.class files will be stored. Under this directory are two subdirectories created with the commands:

mkdir javainput

mkdir javaoutput

The javaoutput directory is where all output files will be created.

The javainput directory is where all input files will be expected. When an output file from one program is to be used as input to another program, it will be copied from javaoutput to javainput (using the Linux cp command).

**Java Topics**

The java programs in this file will provide examples of using file operations (open, close, read, write) and processing run time exceptions using **try/catch**.

## writeHello.java

The main program is kept intentionally brief.

1. create an instance of a writeHello object
2. call writeString to :
   1. declare a file
   2. open a file
   3. write single line of text
   4. close the file

Take a brief look at the program, start in main:

import java.io.BufferedWriter;

import java.io.File;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.OutputStreamWriter;

import java.io.Writer;

public class writeHello {

public void writeString() {

try {

//Whatever the file path is.

File statText = new File("javaoutput/writeHello.txt");

FileOutputStream is = new FileOutputStream(statText);

OutputStreamWriter osw = new OutputStreamWriter(is);

Writer w = new BufferedWriter(osw);

w.write("IB Computer Science Rocks");

w.close();

} catch (IOException e) {

System.err.println("Problem writing to writeHello.txt ");

}

}

public static void main(String[]args) {

writeHello write = new writeHello();

write.writeString();

}

}

### Sample program compilation and execution:

Compile the program:

~/workspace>javac writeHello.java

List the files in the output directory (none)

~/workspace>ls -l javaoutput/

total 0

Execute the program

~/workspace>java writeHello

List the files in the output directory (one)

~/workspace>ls -l javaoutput/

total 4

-rw-r--r-- 1 ubuntu ubuntu 25 Nov 30 10:12 writeHello.txt

Display the contents of the output file

~/workspace>cat javaoutput/writeHello.txt

## How writeHello Accesses Text Files

Declaring a file variable : this will not open or create the file

A try/catch block is not required because an i/o exception will not be thrown

File statText = new File("javaoutput/writeHello.txt");

Tell java that the File object statText is to be treated as an output stream using the FileOutputStream class: This statement will create the output file and can generate an i/o exception. Java would like this to be placed inside a try block.

FileOutputStream is = new FileOutputStream(statText);

The is object is passed to create an OutputStreamWriter object

OutputStreamWriter osw = new OutputStreamWriter(is);

Notice in the program that after statText, is, and osw are used to create objects, they are not referenced again. All of the file activity is through using variable w.

Writer w = new BufferedWriter(osw);

Notice that the import statements show how to include each of these needed classes.

import java.io.BufferedWriter;

import java.io.File;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.OutputStreamWriter;

import java.io.Writer;

All of these statements could be replaced with this single statement:

import java.io.\*;

public class writeHello {

public void writeString() {

try {

//Whatever the file path is.

File statText = new File("javaoutput/writeHello.txt");

FileOutputStream is = new FileOutputStream(statText);

OutputStreamWriter osw = new OutputStreamWriter(is);

Writer w = new BufferedWriter(osw);

w.write("IB Computer Science Rocks");

w.close();

} catch (IOException e) {

System.err.println("Problem writing to writeHello.txt ");

}

}

## Writing the file

The write operation occurs using the write method of the w object.

w.write("IB Computer Science Rocks");

## Closing the file

The write operation occurs using the write method of the w object.

w.close();

## writeHello2.java

In the version of the program below, the name of the output file has been changed so that the path is invalid. When the program tries to open the file an exception will be triggered. Rather than having the program crash or terminate with an error, the "catch" block will be called and a (more) polite message will be displayed.

import java.io.BufferedWriter;

import java.io.File;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.OutputStreamWriter;

import java.io.Writer;

public class writeHello2 {

public void writeString() {

try {

//Whatever the file path is.

File statText = new File("/invalid/javaoutput/writeHello.txt");

FileOutputStream is = new FileOutputStream(statText);

OutputStreamWriter osw = new OutputStreamWriter(is);

Writer w = new BufferedWriter(osw);

w.write("IB Computer Science Rocks");

w.close();

} catch (IOException e) {

System.err.println("Problem writing to writeHello.txt ");

}

}

public static void main(String[]args) {

writeHello2 write = new writeHello2();

write.writeString();

}

}

### Sample program compilation and execution

:~/workspace> javac writeHello2.java

~/workspace> java writeHello2

Problem writing to writeHello.txt