File Input/Output (I/O)

Tuesday, November 27, 2007

CS111 Computer Programming

Department of Computer Science
Wellesley College

What is File I/O?

A file is an abstraction for storing information (text, images, music, etc.) on a computer.

Today we will explore how to read information from (Input) and write information to (Output) files in Java. Input/Output is often abbreviated I/O.

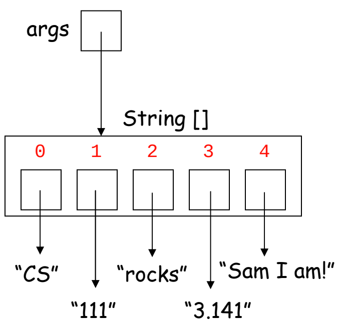
We will focus on files with textual information, but the techniques we'll learn generalize to any kind of information.

Writing to Standard Output

System.out.println writes a string to the Java console (standard output file).

```
public class WriteToConsole {
   public static void main (String [ ] args) {
      for (int i = 0;  i < args.length; i++) {
         System.out.println(args[i]);
      }
   }
}
```

```
> java WriteToConsole CS 111 rocks 3.141 "Sam I am!"
CS
111
rocks
3.141
Sam I am!
```



Writing to a File

FileWriter, BufferedWriter, and IOException live in the java.io package

File operations can generate IOExceptions

```
import java.io.*;

public class WriteToFile {

   public static void main (String [ ] args) throws IOException {
      String filename = args[0];
      BufferedWriter writer = new BufferedWriter(new FileWriter(filename));
      for (int i = 1;  i < args.length; i++) {
         writer.write(args[i]);
         writer.newLine(); // Same as writer.write("\n");
      }
      writer.close();
   }
}
```

Append string to end of file.

Idiom for creating a file writer.
If file doesn't exist, creates it.
If file exists, overwrites it.

Finish any output to file.
File is not guaranteed to
contain all output until closed.

```
> java WriteToFile out.txt CS 111 rocks 3.141 "Sam I am!"
```

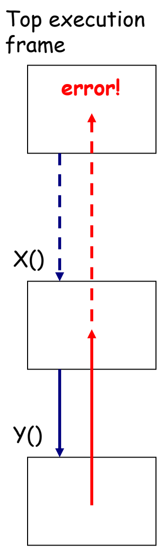
out.txt

```
CS
111
rocks
3.141
Sam I am!
```

What's An Exception?

- Java methods use **exceptions** to tell the calling code, "Something bad happened. I failed."
- Exceptions are used for **reliability**: they describe and allow recovery from errors that occur while a program is running.
- When an unusual condition arises, a method stops its normal execution and **throws** an exception describing the problem.
- If method X calls method Y and Y throws an exception, X may either handle the exception by **catching** it or allow the exception to **propagate** to X's caller. You will *not* learn how to catch exceptions in CS111 (but you will in CS230!)
- An exception that propagates all the way up to the top of the program results in an error message. E.g.

```
Exception in thread "AWT-EventQueue-0"
 BuggleException:  FORWARD: Can't move through wall!
at Buggle.forwardStep(BuggleWorld.java:2040)
at Buggle.forward(BuggleWorld.java:2034)
   ⋮
```



Declaring Exceptions in Method Headers

Java has two categories of exceptions:

- **checked exceptions** (such as IOException) must be declared in the header of any method that throws or propagates them.

```
// Java's method for writing a string to a file
public void write (String s) throws IOException
```

  Many methods of classes in the java.io package throw IOException.

- **unchecked exceptions** need not be explicitly declared.
  Examples of unchecked exceptions:

  NullPointerException

  ArrayIndexOutOfBoundsException

  Exceptions from all CS111 classes (Buggle, Turtle, IntList, etc.)

Reading From a File

```
import java.io.*;

// Display the contents of a file, line by line
public class DisplayFile {

 public static void main (String [ ] args) throws IOException {
   String filename = args[0];
   BufferedReader reader = new BufferedReader(new FileReader(filename));
   String line = reader.readLine(); // read the first line in file
   while (line != null) {
     System.out.println(line);
     line = reader.readLine(); // read next line from the file.
   }
   reader.close(); // Tidy up
 }
}
```

Idiom for creating a file reader. Throws an IOException if file doesn't exist.

readLine() returns next line from file as a string (without trailing newline character). It returns null if there are no more lines in the file.

```
> java DisplayFile out.txt
CS
111
rocks
3.141
Sam I am!
```

out.txt

```
CS
111
rocks
3.141
Sam I am!
```

File I/O 21-7

Transforming a File

Let's write a Java application that uppercases each line of a file (using the String toUpperCase() method).

```
> java UpperCaseFile cat4.txt cat4Upper.txt
```

input filename

output filename

cat4.txt

```
The sun did not shine.
It was too wet to play.
So we sat in the house
All that cold, cold, wet day.
```

cat4Upper.txt

```
THE SUN DID NOT SHINE.
IT WAS TOO WET TO PLAY.
SO WE SAT IN THE HOUSE
ALL THAT COLD, COLD, WET DAY.
```

File I/O 21-8

UppercaseFile

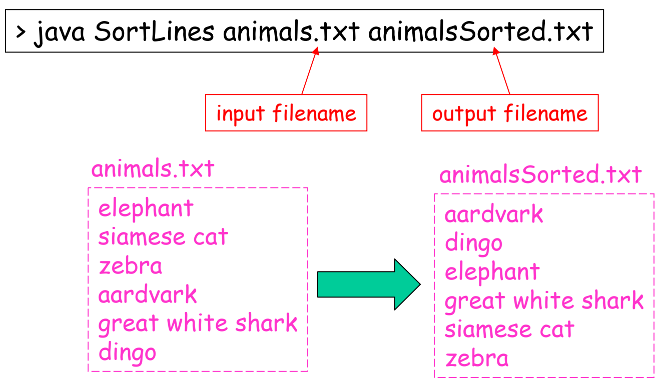
```java
import java.io.*;

// Application that writes an output file that uppercases each line
// of an input file
public class UpperCaseFile {

  public static void main (String [ ] args) throws IOException {
    String infile = args[0];
    String outfile = args[1];
    BufferedReader reader = new BufferedReader(new FileReader(infile));
    BufferedWriter writer = new BufferedWriter(new FileWriter(outfile));
    String line = reader.readLine(); // read the first line in file
    while (line != null) { // readLine() returns null when reaches end of file
      writer.write(line.toUpperCase());
      writer.newLine();
      line = reader.readLine(); // read next line from the file.
    }
    reader.close(); // Tidy up
    writer.close();
  }
}
```

Sorting the Lines of a File

Let's write a Java application that sorts the lines of a file.

Assume we can use the following method from the CS111 StringArray class:

```java
public static sort (String [] strings);
// Modifies strings so that its elements are in dictionary order.
```



```
> java SortLines animals.txt animalsSorted.txt
```

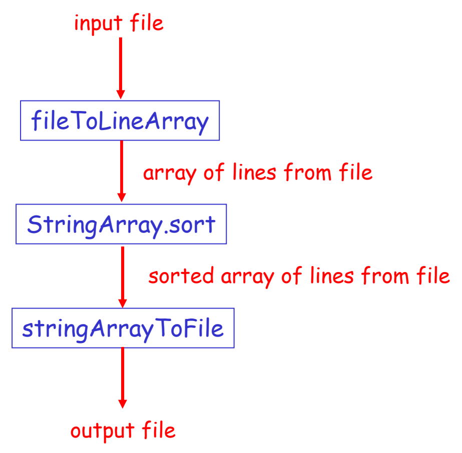
animals.txt

```
elephant
siamese cat
zebra
aardvark
great white shark
dingo
```

animalsSorted.txt

```
aardvark
dingo
elephant
great white shark
siamese cat
zebra
```

SortLines: A Modular Solution



SortLines: Coding The Modular Solution

```
import java.io.*;

// Application that writes an output file containing the
// sorted lines of the input file
public class SortLines {

  public static void main (String [ ] args) throws IOException {
    String infile = args[0];
    String outfile = args[1];
    String[] lines = fileToLineArray(infile);
    StringArray.sort(lines);
    stringArrayToFile(lines, outfile);
  }

  // Returns a string array containing the lines of the file named filename.
  public static String[] fileToLineArray(String filename)
                                     throws IOException {
    // flesh out this skeleton
  }

  // Writes each element of a string array to a line of a file named filename.
  public static void stringArrayToFile(String[] strings, String filename)
                                throws IOException {
      // flesh out this skeleton
  }
}
```

stringArrayToFile() is Easy

```
// Writes each element of a string array to a line of a file named filename.
public static void stringArrayToFile(String[] strings, String filename)
                        throws IOException {
    BufferedWriter writer = new BufferedWriter(new FileWriter(filename));
    for (int i = 0; i < strings.length; i++) {
      writer.write(strings[i]);
      writer.newLine();
    }
    writer.close();
  }
```

fileToLineArray() is Trickier

```
// Returns a string array containing the lines of the file named filename.
public static String[] fileToLineArray(String filename) throws IOException {
    BufferedReader reader = new BufferedReader(new FileReader(filename));

    int numLines = ???

    String [] lines = new String[numLines];
    for (int i = 0; i < numLines; i++) {
        lines[i] = reader.readLine();
    }
    reader.close();
    return lines;
}
```

How do we know the number of lines?

fileToLineArray(): Solution 1

Idea: Assume the first line of every file is an integer specifying the number of lines in the rest of the file.

This is sensible for specialized files (like the image files in PS9), but not all files will have this format.

animalsWithHeader.txt

```
6
elephant
siamese cat
zebra
aardvark
great white shark
dingo
```

```
// Returns a string array containing the lines of the file named filename.
public static String[] fileToLineArray(String filename) throws IOException {
  BufferedReader reader = new BufferedReader(new FileReader(filename));
  String header = reader.readLine(); // Read integer header in first line
  int numLines = Integer.parseInt(header);
  String [] lines = new String[numLines];
  for (int i = 0; i < numLines; i++) {
    lines[i] = reader.readLine();
  }
  reader.close();
  return lines;
}
```

fileToLineArray(): Solution 2

Idea: Count the lines in a file first in a separate pass over the file. This works for any file, but processes all the lines twice.

```
// Returns a string array containing the lines of the file named filename.
public static String[] fileToLineArray(String filename) throws IOException {
  BufferedReader reader = new BufferedReader(new FileReader(filename));
  int numLines = countLines(filename); // Count the lines in the file.
  String [] lines = new String[numLines];
  for (int i = 0; i < numLines; i++) {
    lines[i] = reader.readLine();
  }
  reader.close();
  return lines;
}
// Returns the number of lines in a file.
public static int countLines (String filename) throws IOException {
  BufferedReader reader = new BufferedReader(new FileReader(filename));
  int count = 0;
  String line = reader.readLine();
  while (line != null) {
   count++;
   line = reader.readLine();
  }
  reader.close();
  return count;
 }
```

fileToLineArray(): Solution 3

Idea: Read the lines of the file into a string list and then convert the list to an array. This works for any file and only reads the file once, but uses extra storage for list.

```
// Returns a string array containing the lines of the file named filename.
// This version reads the lines into a list first, and then converts the list to an array.
public static String[] fileToLineArray(String filename) throws IOException {
   BufferedReader reader = new BufferedReader(new FileReader(filename));
   StringList lines = StringList.empty(); // lines is a list of file lines in reverse order
   String line = reader.readLine();
   while (line != null) {
     lines = StringList.prepend(line, lines); // prepend lines to list in reverse order
     line = reader.readLine();
   }
   return listToArray(StringListOps.reverse(lines)); // reverse lines to get original order
 }

 // Convert a list of strings to an array of strings
  public static String[] listToArray (StringList list) {
   int len = StringListOps.length(list);
   String [] array = new String[len];
   for (int i = 0; i < len; i++) {
     array[i] = StringList.head(list);
     list = StringList.tail(list);
   }
   return array;
  }
```

Sorting the Words of a File

Let's write a Java application that sorts the words of a file.

```
> java SortWords animals.txt animalsSortedWords.txt
```

input filename

output filename

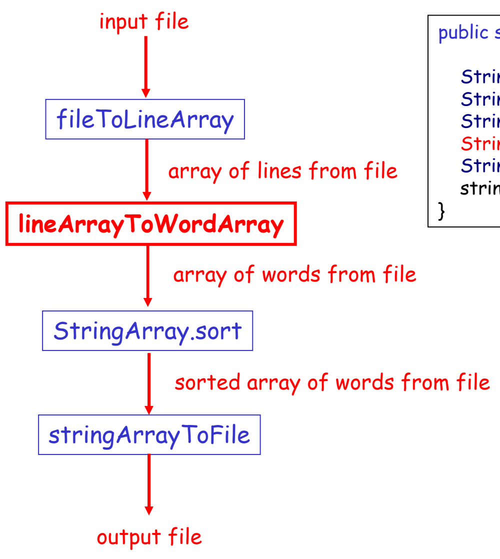
animals.txt

```
elephant
siamese cat
zebra
aardvark
great white shark
dingo
```

animalsSortedWords.txt

```
aardvark
cat
dingo
elephant
great
shark
siamese
white
zebra
```

SortWords: Just A Modification To Sort Lines



```
public static void main (String [ ] args)
                   throws IOException {
   String infile = args[0];
   String outfile = args[1];
   String[] lines = fileToLineArray(infile);
   String[] words = lineArrayToWordArray(lines);
   StringArray.sort(words);
   stringArrayToFile(words, outfile);
}
```

How to Write lineArrayToWordArray?

Hint: Use split(" ") from the String class to break lines into "words"
(character sequences separated by a space).

```
public static String[] lineArrayToWordArray(String[] lines)  {
    // Since we don't know how many words there will be,
    // we create a StringList of all the words and convert
    // this back to an array.
    StringList wordList = StringList.empty();
    for (int i = 0; i < lines.length; i++) {
      // Splits line into "words" separated by space, " ".
      String[] words_in_line = lines[i].split(" ");
      wordList = StringListOps.append(wordList,
                                  arrayToList(words_in_line));
    }
    return listToArray(wordList);
}
```

Reading Input from the Java Console

The Java console (e.g., the Dr. Java Interaction Pane) can be viewed as a special kind of file (known as the **standard input/output file**).

We already know how to write to standard output using System.out.println.

We can read from it as well by creating a reader for it using this idiom:

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

Here's an example of interacting with the InteractiveUpperCase application defined on the next slide:

```
> java InteractiveUpperCase
Type a line to convert to upper case (or quit to exit)>
This is a test; it is only a test.
THIS IS A TEST; IT IS ONLY A TEST.
Type a line to convert to upper case (or quit to exit)>
another example
ANOTHER EXAMPLE
Type a line to convert to upper case (or quit to exit)>
quit
>
```

InteractiveUpperCase Application

```
import java.io.*;

// Application that interactively prompts the user for a string,
// which is then displayed in upper case form.
public class InteractiveUpperCase {

  public static void main (String [ ] args) throws IOException {
    BufferedReader reader =
          new BufferedReader(new InputStreamReader(System.in));
    System.out.println("Type a line to convert to upper case (or quit to exit)> ");
    String line = reader.readLine();
    while (! line.equals("quit")) {
      System.out.println(line.toUpperCase());
      System.out.println("Type a line to convert to upper case (or quit to exit)> ");
      line = reader.readLine();
    }
    reader.close();
  }
}
```