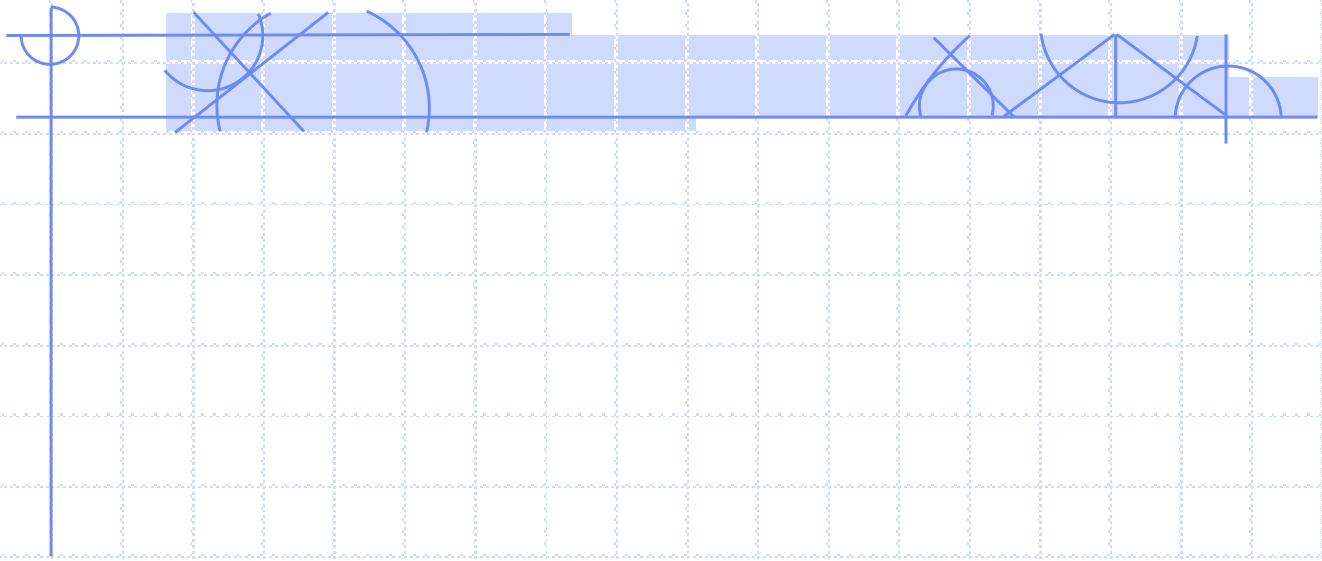
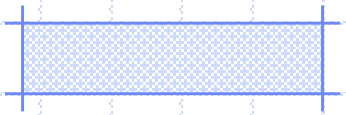
Lecture 7

File Input/Output





Files

- Storage of data in variables and arrays is temporary—the data is lost when a local variable goes out of scope or when the program terminates.
- Computers use files for long-term retention of large amounts of data, even after programs that create the data terminate. We refer to data maintained in files as persistent data, because the data exists beyond the duration of program execution.
- Computers store files on secondary storage devices such as magnetic disks, optical disks and magnetic tapes.

Files

There are two general types of files you need to learn about: ***text*** files and ***binary*** files…

- A **text**, or character-based, file stores information using ASCII character representations. Text files can be viewed with a standard editor or word processing program but cannot be manipulated arithmetically without requiring special conversion routines.

- A **binary** file stores numerical values using the internal numeric binary format specified by the language in use. A Java program can read a binary file to get numeric data, manipulate the data arithmetically, and write the data to a binary file without any intermediate conversions.

File Operations

There are three basic operations that you will need to perform when working with disk files:

- Open the file for input or output.
- Process the file, by reading from or writing to the file.
- Close the file.

The Class File

- Class `File` useful for retrieving information about files and directories from disk
- Objects of class `File` do not open files or provide any file-processing capabilities
- File objects are used frequently with objects of other `java.io` classes to specify files or directories to manipulate.

Creating File Objects

- To operate on a file, we must first create a File object (from java.io).

Class `File` provides constructors:

1. Takes `String` specifying name and path (location of file on disk)

```
File filename = new File("sample.dat");
```

Opens the file sample.dat in the current directory.

```
File filename = new File("C:/SamplePrograms/test.dat");
```

Opens the file test.dat in the directory C:\SamplePrograms using the generic file separator / and providing the full pathname.

2. Takes two Strings, first specifying path and second specifying name of file

```
File filename = new File(String pathToName, String Name);
```

File Methods

Method	Description
`boolean canRead()`	Returns `true` if a file is readable by the current application; `false` otherwise.
`boolean canWrite()`	Returns `true` if a file is writable by the current application; `false` otherwise.
`boolean exists()`	Returns `true` if the name specified as the argument to the `File` constructor is a file or directory in the specified path; `false` otherwise.
`boolean isFile()`	Returns `true` if the name specified as the argument to the `File` constructor is a file; `false` otherwise.
`boolean isDirectory()`	Returns `true` if the name specified as the argument to the `File` constructor is a directory; `false` otherwise.
`boolean isAbsolute()`	Returns `true` if the arguments specified to the `File` constructor indicate an absolute path to a file or directory; `false` otherwise.
`String getAbsolutePath()`	Returns a string with the absolute path of the file or directory.
`String getName()`	Returns a string with the name of the file or directory.
`String getPath()`	Returns a string with the path of the file or directory.
`String getParent()`	Returns a string with the parent directory of the file or directory (i.e., the directory in which the file or directory can be found).
`long length()`	Returns the length of the file, in bytes. If the `File` object represents a directory, `0` is returned.
`long lastModified()`	Returns a platform-dependent representation of the time at which the file or directory was last modified. The value returned is useful only for comparison with other values returned by this method.
`String[] list()`	Returns an array of strings representing the contents of a directory. Returns `null` if the `File` object does not represent a directory.

Some File Methods

```
if (filename.exists( ) ) {
```

To see if `filename` is associated to a real file correctly.

```
if (filename.isFile() ) {
```

To see if `filename` is associated to a file or not. If false, it is a directory.

Textfile Input and Output

- Instead of storing primitive data values as binary data in a file, we can convert and store them as a string data.
  - This allows us to view the file content using any text editor
- To output data as a string to file, we use a **PrintWriter** object.
- To input data from a textfile, we use **FileReader** and **BufferedReader** classes
  - From Java 5.0 (SDK 1.5), we can also use the Scanner class for inputting textfiles

Sample Textfile Output

A test program to save data to a file using PrintWriter for high-level IO

```
import java.io.*;
class TestPrintWriter {
   public static void main (String[] args) throws IOException {

          //set up file and stream
          File outFile = new File("sample3.data");

          PrintWriter PF = new PrintWriter(outFile);

          //write values of primitive data types to the stream
          PF.println(987654321);
          PF.println("Hello, world.");
          PF.println(true);

          //output done, so close the stream
          PF.close();
   }
}
```

We use println and print with PrintWriter. The print and println methods convert primitive data types to strings before writing to a file.

Sample Textfile Input

To read the data from a text file, we use the FileReader and BufferedReadder objects.

To read back from a text file:

- we need to associate a BufferedReader object to a file,

```
File inF = new File("sample3.data");
FileReader FR = new FileReader(inF);
BufferedReader BFR = new BufferedReader(FR);
```

- read data using the readLine method of BufferedReader,

```
String str;
    str = BFR.readLine();
```

- convert the string to a primitive data type as necessary.

```
int i = Integer.parseInt(str);
```

Sample Textfile Input

```
import java.io.*;
class TestBufferedReader {

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

  //set up file and stream
  File inF = new File("sample3.data");
  FileReader FR = new FileReader(inF);
  BufferedReader BFR = new BufferedReader(FR);
  String str;
  //get integer
  str = BFR.readLine();
  int i = Integer.parseInt(str);

   //get long
    str = BFR.readLine();
    long l = Long.parseLong(str);

   //get float
    str = BFR.readLine();
    float f = Float.parseFloat(str);

   //get double
    str = BFR.readLine();
    double d = Double.parseDouble(str);

    //get char
    str = BFR.readLine();
    char c = str.charAt(0);

    //get boolean
    str = BFR.readLine();
    Boolean boolObj = new Boolean(str);
    boolean b = boolObj.booleanValue( );

    System.out.println(i);
    System.out.println(l);
    System.out.println(f);
    System.out.println(d);
    System.out.println(c);
    System.out.println(b);

    //input done, so close the stream
    BFR.close();
  }
}
```