Module 11

Console I/O and File I/O

Objectives

- Read data from the console
- Write data to the console
- Describe files and file I/O

Console I/O

- The variable `System.out` enables you to write to *standard output*.

  `System.out` is an object of type `PrintStream`.
- The variable `System.in` enables you to read from *standard input*.

  `System.in` is an object of type `InputStream`.
- The variable `System.err` enables you to write to *standard error*.

  `System.err` is an object of type `PrintStream`.

Writing to Standard Output

- The `println` methods print the argument and a newline character (`\n`).
- The `print` methods print the argument without a newline character.
- The `print` and `println` methods are overloaded for most primitive types (`boolean`, `char`, `int`, `long`, `float`, and `double`) and for `char[]`, `Object`, and `String`.
- The `print(Object)` and `println(Object)` methods call the `toString` method on the argument.

Reading From Standard Input

```
import java.io.*;

public class KeyboardInput {
  public static void main (String args[]) {
    String s;
    // Create a buffered reader to read
    // each line from the keyboard.
    InputStreamReader ir
      = new InputStreamReader(System.in);
    BufferedReader in = new BufferedReader(ir);

    System.out.println("Unix: Type ctrl-d to exit." +
                       "\nWindows: Type ctrl-z to exit");
```

Reading From Standard Input

```
      try {
        // Read each input line and echo it to the screen.
        s = in.readLine();
        while ( s != null ) {
          System.out.println("Read: " + s);
          s = in.readLine();
        }

        // Close the buffered reader.
        in.close();
      } catch (IOException e) { // Catch any IO exceptions.
        e.printStackTrace();
      }
    }
  }
```

Simple Formatted Output

- You can use the formatting functionality as follows:

```
out.printf("name count\n");
String s = String.format("%s %5d%n", user, total);
```

- Common formatting codes are listed in this table.

Code	Description
`%s`	Formats the argument as a string, usually by calling the `toString` method on the object.
`%d %o %x`	Formats an integer, as a decimal, octal, or hexadecimal value.
`%f %g`	Formats a floating point number. The `%g` code uses scientific notation.
`%n`	Inserts a newline character to the string or stream.
`%%`	Inserts the `%` character to the string or stream.

Simple Formatted Input

- The `Scanner` class provides a formatted input function.
- A `Scanner` class can be used with console input streams as well as file or network streams.
- You can read console input as follows:

```
import java.io.*;
import java.util.Scanner;
public class ScanTest {
  public static void main(String [] args) {
    Scanner s = new Scanner(System.in);
    String param = s.next();
    System.out.println("the param 1" + param);
    int value = s.nextInt();
    System.out.println("second param" + value);
    s.close();
  }
}
```

Files and File I/O

The `java.io` package enables you to do the following:

- Create `File` objects
- Manipulate `File` objects
- Read and write to file streams

Creating a New `File` Object

The `File` class provides several utilities:

- `File myFile;`
- `myFile = new File("myfile.txt");`
- `myFile = new File("MyDocs", "myfile.txt");`

Directories are treated like files in the Java programming language. You can create a `File` object that represents a directory and then use it to identify other files, for example:

```
File myDir = new File("MyDocs");
myFile = new File(myDir, "myfile.txt");
```

The `File` Tests and Utilities

- File information:

```
String getName()
String getPath()
String getAbsolutePath()
String getParent()
long lastModified()
long length()
```

- File modification:

```
boolean renameTo(File newName)
boolean delete()
```

- Directory utilities:

```
boolean mkdir()
String[] list()
```

The `File` Tests and Utilities

- File tests:

```
boolean exists()
boolean canWrite()
boolean canRead()
boolean isFile()
boolean isDirectory()
boolean isAbsolute();
boolean is Hidden();
```

File Stream I/O

- For file input:
  - Use the `FileReader` class to read characters.
  - Use the `BufferedReader` class to use the `readLine` method.
- For file output:
  - Use the `FileWriter` class to write characters.
  - Use the `PrintWriter` class to use the `print` and `println` methods.

File Input Example

A file input example is:

```java
import java.io.*;
public class ReadFile {
  public static void main (String[] args) {
    // Create file
    File file = new File(args[0]);

    try {
      // Create a buffered reader
      // to read each line from a file.
      BufferedReader in
        = new BufferedReader(new FileReader(file));
      String s;

```

Printing a File

```
        // Read each line from the file and echo it to the screen.
        s = in.readLine();
        while ( s != null ) {
          System.out.println("Read: " + s);
          s = in.readLine();
        }
        // Close the buffered reader
        in.close();

      } catch (FileNotFoundException e1) {
        // If this file does not exist
        System.err.println("File not found: " + file);

      } catch (IOException e2) {
        // Catch any other IO exceptions.
        e2.printStackTrace();
      }
    }
  }
```

File Output Example

```
import java.io.*;

public class WriteFile {
  public static void main (String[] args) {
    // Create file
    File file = new File(args[0]);

    try {
      // Create a buffered reader to read each line from standard in.
      InputStreamReader isr
        = new InputStreamReader(System.in);
      BufferedReader in
        = new BufferedReader(isr);
      // Create a print writer on this file.
      PrintWriter out
        = new PrintWriter(new FileWriter(file));
      String s;
```

File Output Example

```

        System.out.print("Enter file text.  ");
        System.out.println("[Type ctrl-d to stop.]");

        // Read each input line and echo it to the screen.
        while ((s = in.readLine()) != null) {
          out.println(s);
        }

        // Close the buffered reader and the file print writer.
        in.close();
        out.close();

      } catch (IOException e) {
      // Catch any IO exceptions.
        e.printStackTrace();
      }
    }
  }
```