

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.



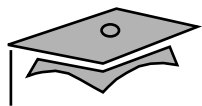
Reading From Standard Input

```
1  import java.io.*;
2
3  public class KeyboardInput {
4      public static void main (String args[]) {
5          String s;
6          // Create a buffered reader to read
7          // each line from the keyboard.
8          InputStreamReader ir
9              = new InputStreamReader(System.in);
10         BufferedReader in = new BufferedReader(ir);
11
12         System.out.println("Unix: Type ctrl-d to exit." +
13                             "\nWindows: Type ctrl-z to exit");
```



Reading From Standard Input

```
14     try {
15         // Read each input line and echo it to the screen.
16         s = in.readLine();
17         while ( s != null ) {
18             System.out.println("Read: " + s);
19             s = in.readLine();
20         }
21
22         // Close the buffered reader.
23         in.close();
24     } catch (IOException e) { // Catch any IO exceptions.
25         e.printStackTrace();
26     }
27 }
28 }
```



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:

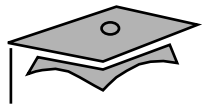
```
1  import java.io.*;
2  import java.util.Scanner;
3  public class ScanTest {
4      public static void main(String [] args) {
5          Scanner s = new Scanner(System.in);
6          String param = s.next();
7          System.out.println("the param 1" + param);
8          int value = s.nextInt();
9          System.out.println("second param" + value);
10         s.close();
11     }
12 }
```




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 isHidden();
```



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:

```
1  import java.io.*;
2  public class ReadFile {
3      public static void main (String[] args) {
4          // Create file
5          File file = new File(args[0]);
6
7          try {
8              // Create a buffered reader
9              // to read each line from a file.
10             BufferedReader in
11                 = new BufferedReader(new FileReader(file));
12             String s;
13
```



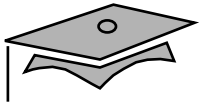
Printing a File

```
14      // Read each line from the file and echo it to the screen.
15      s = in.readLine();
16      while ( s != null ) {
17          System.out.println("Read: " + s);
18          s = in.readLine();
19      }
20      // Close the buffered reader
21      in.close();
22
23      } catch (FileNotFoundException e1) {
24          // If this file does not exist
25          System.err.println("File not found: " + file);
26
27      } catch (IOException e2) {
28          // Catch any other IO exceptions.
29          e2.printStackTrace();
30      }
31  }
32 }
```



File Output Example

```
1  import java.io.*;
2
3  public class WriteFile {
4      public static void main (String[] args) {
5          // Create file
6          File file = new File(args[0]);
7
8          try {
9              // Create a buffered reader to read each line from standard in.
10             InputStreamReader isr
11                 = new InputStreamReader(System.in);
12             BufferedReader in
13                 = new BufferedReader(isr);
14             // Create a print writer on this file.
15             PrintWriter out
16                 = new PrintWriter(new FileWriter(file));
17             String s;
```

File Output Example

```
18
19     System.out.print("Enter file text.  ");
20     System.out.println("[Type ctrl-d to stop.]");
21
22     // Read each input line and echo it to the screen.
23     while ((s = in.readLine()) != null) {
24         out.println(s);
25     }
26
27     // Close the buffered reader and the file print writer.
28     in.close();
29     out.close();
30
31     } catch (IOException e) {
32     // Catch any IO exceptions.
33         e.printStackTrace();
34     }
35 }
36 }
```