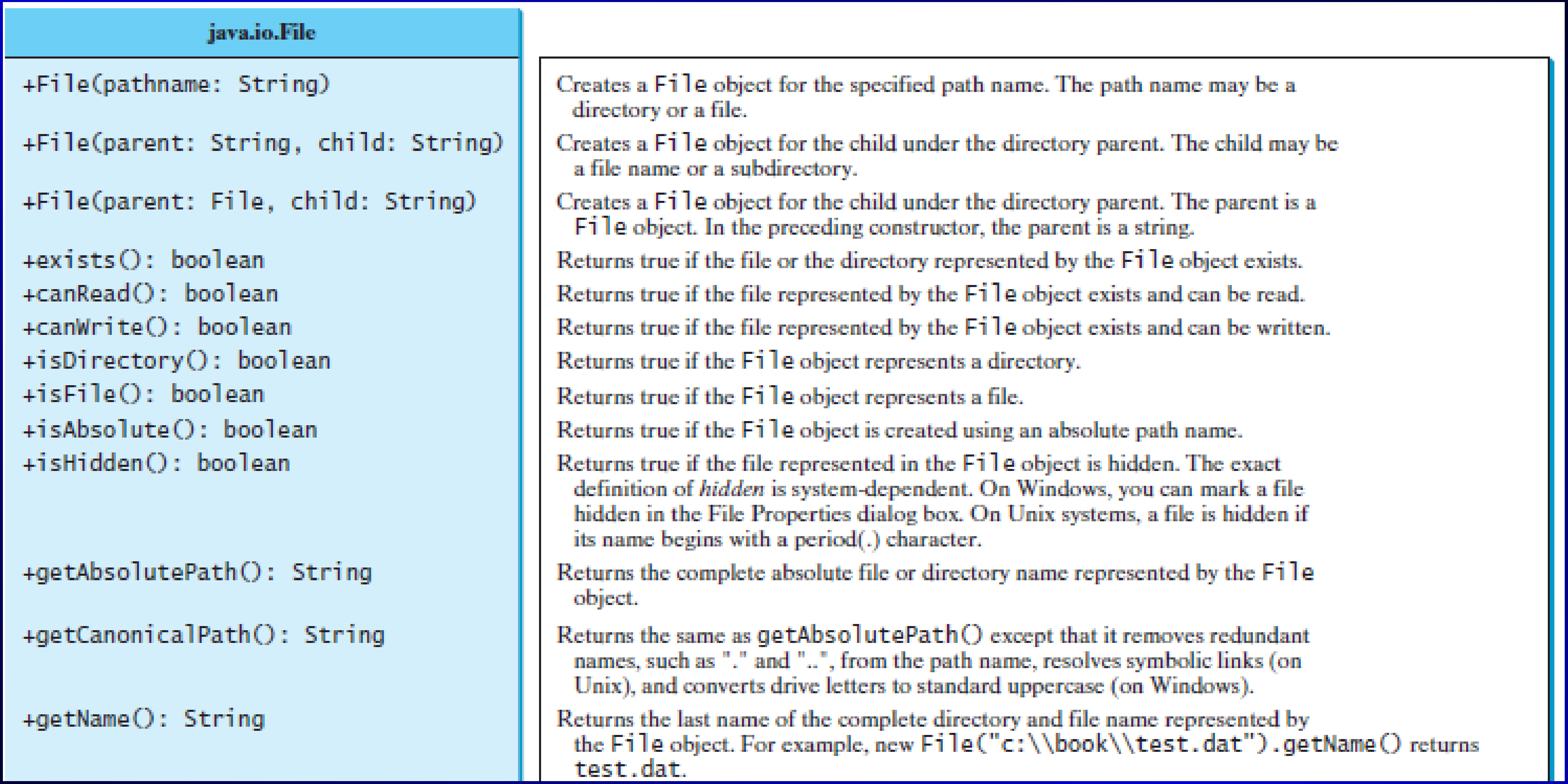
File Class and JFileChooser Class (12.10 – 12.13)

The File Class

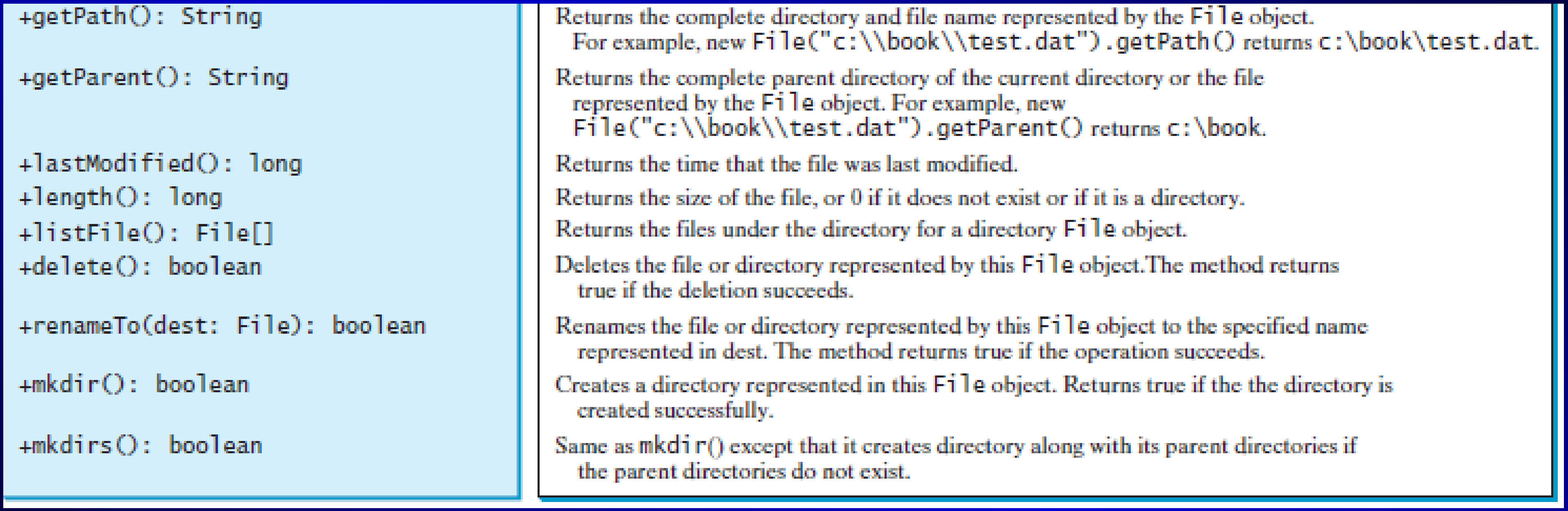
- The File class is intended to provide an abstraction that deals with most of the machine-dependent complexities of files and path names in a machine-independent fashion.
- It is a wrapper class for the file name and its directory path.
- The class contains the methods for obtaining the properties of a file/directory and for renaming and deleting a file/directory.

File Class Methods



java.io.File	
+File(pathname: String)	Creates a File object for the specified path name. The path name may be a directory or a file.
+File(parent: String, child: String)	Creates a File object for the child under the directory parent. The child may be a file name or a subdirectory.
+File(parent: File, child: String)	Creates a File object for the child under the directory parent. The parent is a File object. In the preceding constructor, the parent is a string.
+exists(): boolean	Returns true if the file or the directory represented by the File object exists.
+canRead(): boolean	Returns true if the file represented by the File object exists and can be read.
+canWrite(): boolean	Returns true if the file represented by the File object exists and can be written.
+isDirectory(): boolean	Returns true if the File object represents a directory.
+isFile(): boolean	Returns true if the File object represents a file.
+isAbsolute(): boolean	Returns true if the File object is created using an absolute path name.
+isHidden(): boolean	Returns true if the file represented in the File object is hidden. The exact definition of *hidden* is system-dependent. On Windows, you can mark a file hidden in the File Properties dialog box. On Unix systems, a file is hidden if its name begins with a period(.) character.
+getAbsolutePath(): String	Returns the complete absolute file or directory name represented by the File object.
+getCanonicalPath(): String	Returns the same as getAbsolutePath() except that it removes redundant names, such as "." and "..", from the path name, resolves symbolic links (on Unix), and converts drive letters to standard uppercase (on Windows).
+getName(): String	Returns the last name of the complete directory and file name represented by the File object. For example, new File("c:\\book\\test.dat").getName() returns test.dat.

File Class Methods Cont.



`+getPath(): String`	Returns the complete directory and file name represented by the `File` object. For example, new `File("c:\\book\\test.dat").getPath()` returns `c:\book\test.dat`.
`+getParent(): String`	Returns the complete parent directory of the current directory or the file represented by the `File` object. For example, new `File("c:\\book\\test.dat").getParent()` returns `c:\book`.
`+lastModified(): long`	Returns the time that the file was last modified.
`+length(): long`	Returns the size of the file, or 0 if it does not exist or if it is a directory.
`+listFile(): File[]`	Returns the files under the directory for a directory `File` object.
`+delete(): boolean`	Deletes the file or directory represented by this `File` object.The method returns true if the deletion succeeds.
`+renameTo(dest: File): boolean`	Renames the file or directory represented by this `File` object to the specified name represented in dest. The method returns true if the operation succeeds.
`+mkdir(): boolean`	Creates a directory represented in this `File` object. Returns true if the the directory is created successfully.
`+mkdirs(): boolean`	Same as `mkdir()` except that it creates directory along with its parent directories if the parent directories do not exist.

Notes on Files

- Constructing a File instance does not create a file on the machine.
- You can create a File instance for any file name regardless whether it exists or not.
- You can invoke the exists() method on a File instance to check whether the file exists.
- The directory separator for Windows is a backslash (\ ). The backslash is a special character in Java and should be written as \\ in a string literal (such as a file name).

Notes on Files

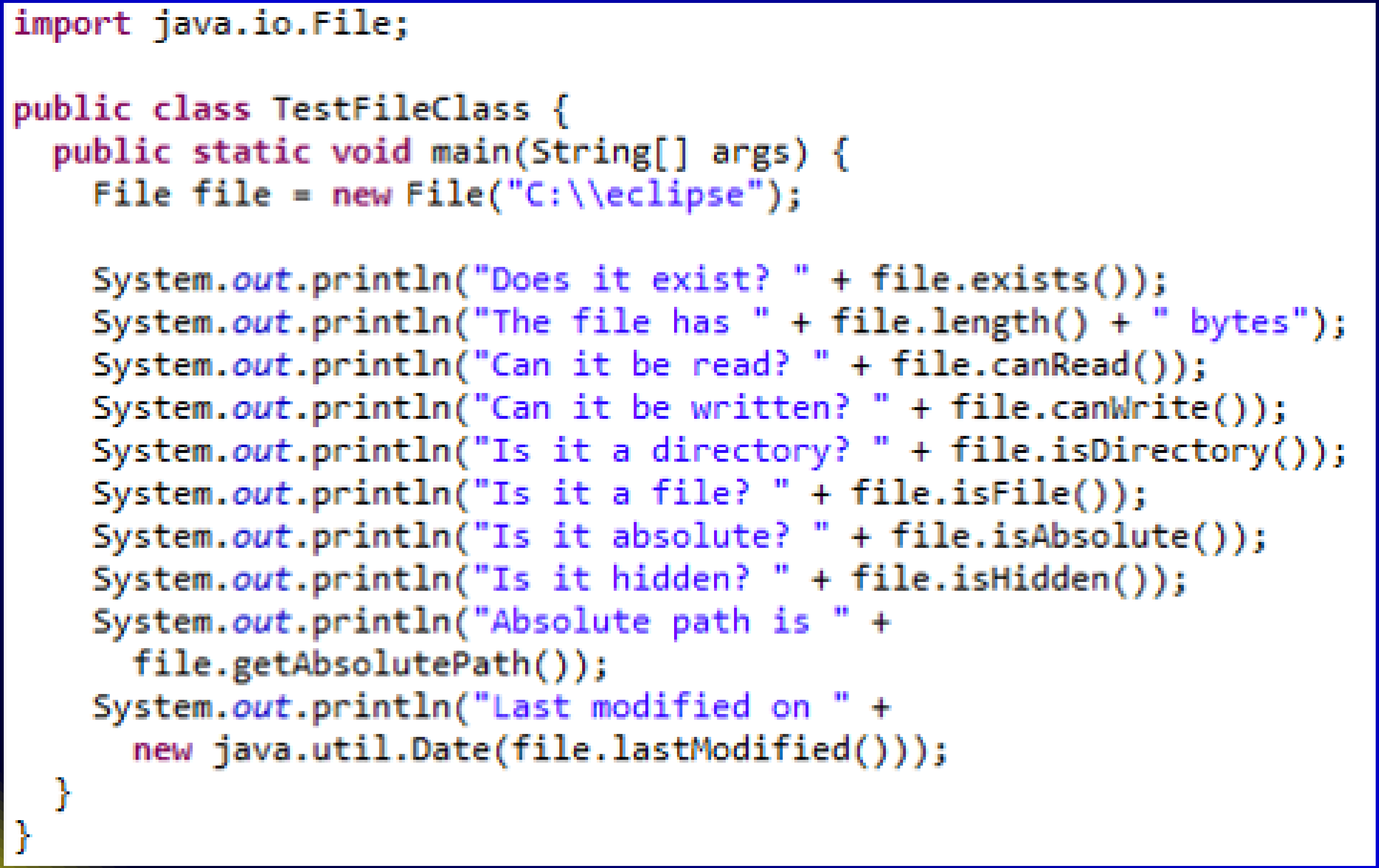
- Do not use absolute file names in your program. If you use a file name such as c:\\book\\Welcome.java, it will work on Windows but not on other platforms.
- You should use a file name relative to the current directory. For example, you may create a File object using new File("Welcome.java") for the file Welcome.java in the current directory.
- Or use System.getProperty("file.separator") to get the appropriate file separator for each platform.

Notes on Files

```
public class FileSeparatorExample {
    public static void main(String[] args) {
        // file.separator system property return the correct file separator
        //for each different platform (Windows = \), (Linux = /)
        String dataFolder = System.getProperty("user.dir") +
                System.getProperty("file.separator") + "data";

        System.out.println("Data Folder = " + dataFolder);
    }
}

// System.getProperty("user.dir")  is the current working directory
```

Example: Explore File Properties

Write a program that demonstrates how to create a file and use the methods in the File class to obtain its properties.



```java
import java.io.File;

public class TestFileClass {
  public static void main(String[] args) {
    File file = new File("C:\\eclipse");

    System.out.println("Does it exist? " + file.exists());
    System.out.println("The file has " + file.length() + " bytes");
    System.out.println("Can it be read? " + file.canRead());
    System.out.println("Can it be written? " + file.canWrite());
    System.out.println("Is it a directory? " + file.isDirectory());
    System.out.println("Is it a file? " + file.isFile());
    System.out.println("Is it absolute? " + file.isAbsolute());
    System.out.println("Is it hidden? " + file.isHidden());
    System.out.println("Absolute path is " +
      file.getAbsolutePath());
    System.out.println("Last modified on " +
      new java.util.Date(file.lastModified()));
  }
}
```

Text I/O

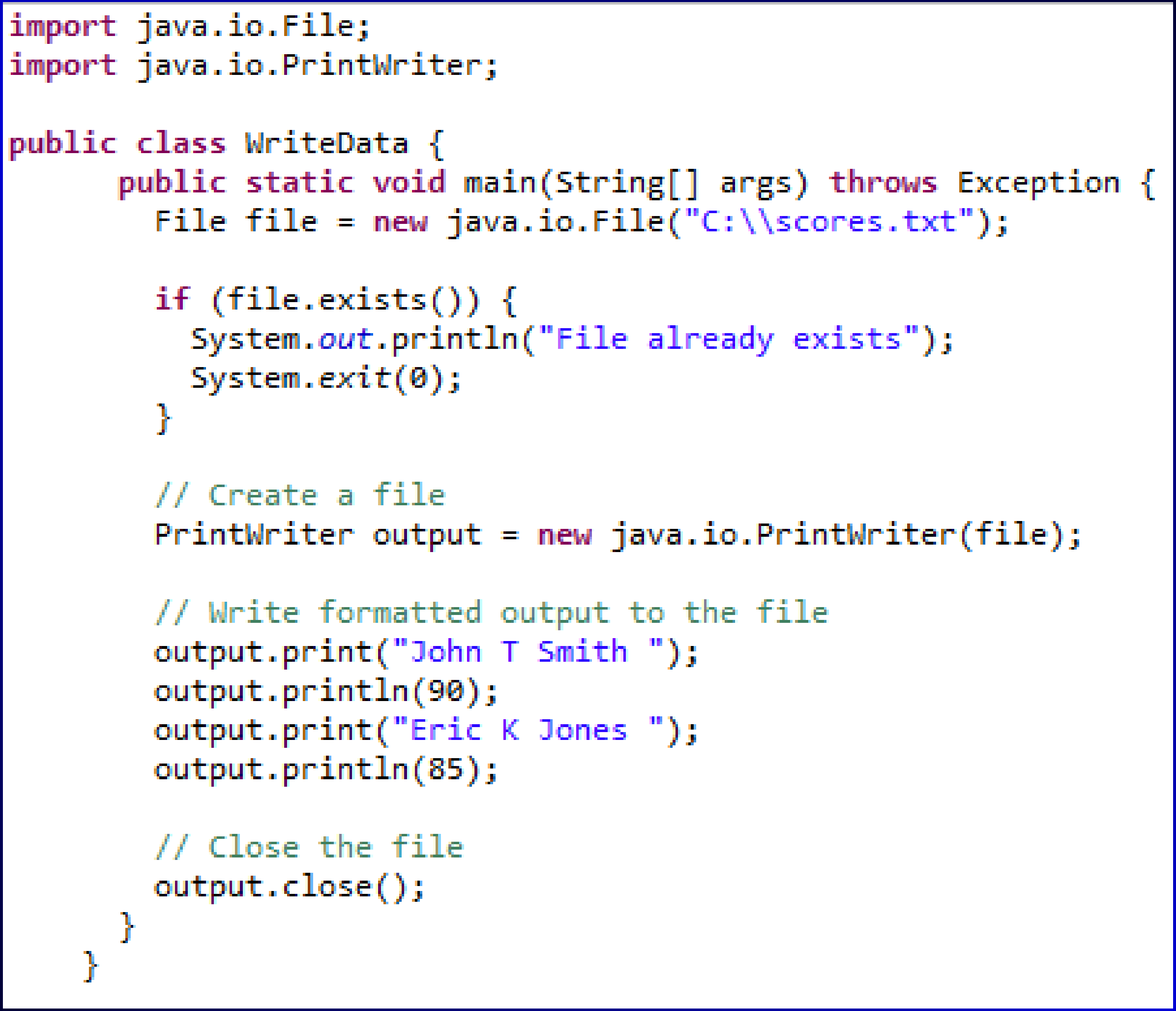
- A File object encapsulates the properties of a file or a path, but does not contain the methods for reading/writing data from/to a file.
- In order to perform I/O, you need to create objects using appropriate Java I/O classes (Scanner and PrintWriter).

Writing Data Using PrintWriter

java.io.PrintWriter	
+PrintWriter(filename: String)	Creates a PrintWriter for the specified file.
+print(s: String): void	Writes a string.
+print(c: char): void	Writes a character.
+print(cArray: char[]): void	Writes an array of character.
+print(i: int): void	Writes an int value.
+print(l: long): void	Writes a long value.
+print(f: float): void	Writes a float value.
+print(d: double): void	Writes a double value.
+print(b: boolean): void	Writes a boolean value.
Also contains the overloaded println methods.	A println method acts like a print method; additionally it prints a line separator. The line separator string is defined by the system. It is \r\n on Windows and \n on Unix.
Also contains the overloaded printf methods.	The printf method was introduced in §3.6, "Formatting Console Output and Strings."

See appendix at end of presentation

Writing Data Using PrintWriter

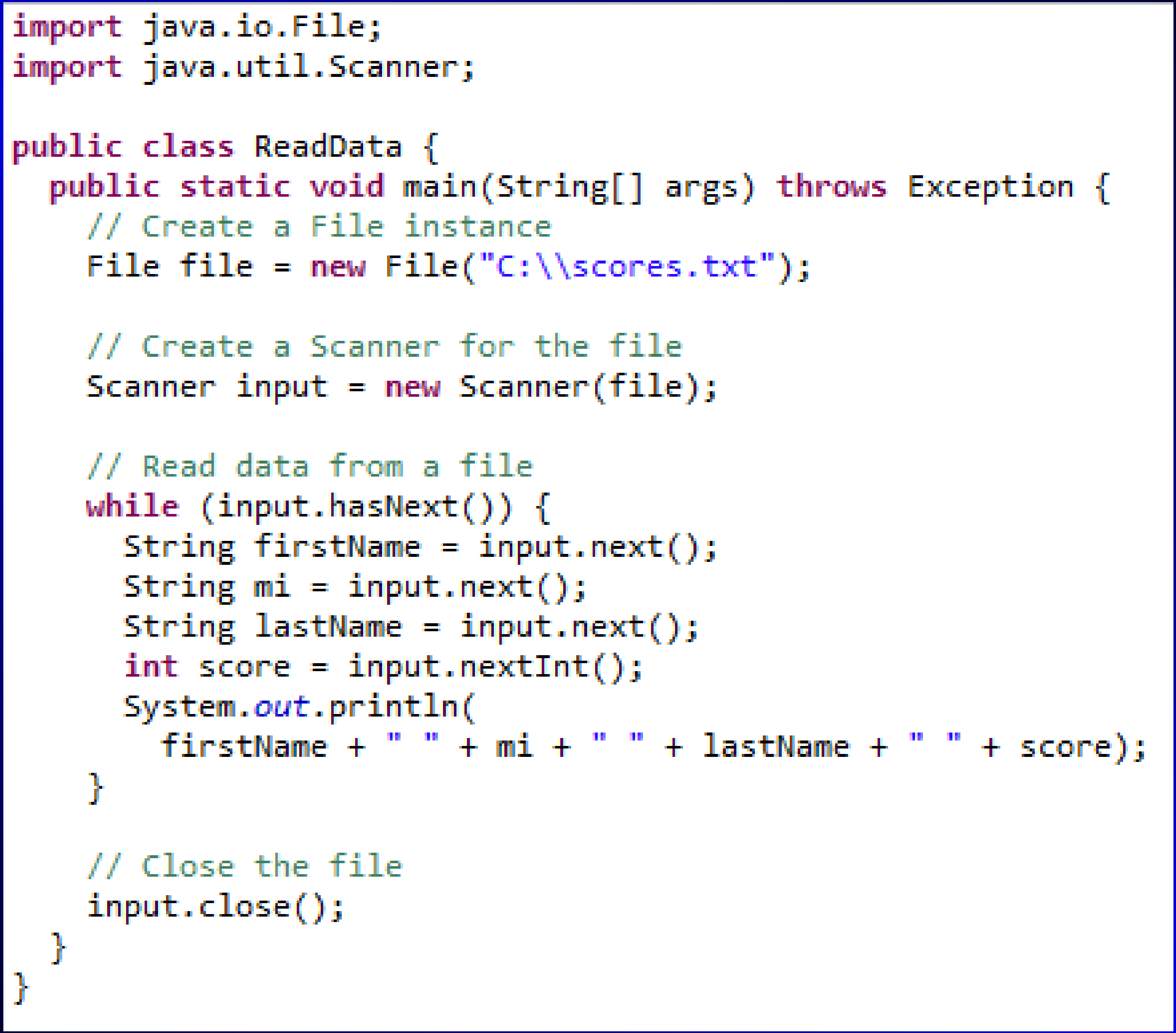


```
import java.io.File;
import java.io.PrintWriter;

public class WriteData {
      public static void main(String[] args) throws Exception {
        File file = new java.io.File("C:\\scores.txt");

        if (file.exists()) {
          System.out.println("File already exists");
          System.exit(0);
        }

        // Create a file
        PrintWriter output = new java.io.PrintWriter(file);

        // Write formatted output to the file
        output.print("John T Smith ");
        output.println(90);
        output.print("Eric K Jones ");
        output.println(85);

        // Close the file
        output.close();
      }
    }
```

Reading Data Using Scanner

java.util.Scanner	
+Scanner(source: File)	Creates a Scanner that produces values scanned from the specified file.
+Scanner(source: String)	Creates a Scanner that produces values scanned from the specified string.
+close()	Closes this scanner.
+hasNext(): boolean	Returns true if this scanner has another token in its input.
+next(): String	Returns next token as a string.
+nextByte(): byte	Returns next token as a byte.
+nextShort(): short	Returns next token as a short.
+nextInt(): int	Returns next token as an int.
+nextLong(): long	Returns next token as a long.
+nextFloat(): float	Returns next token as a float.
+nextDouble(): double	Returns next token as a double.
+useDelimiter(pattern: String): Scanner	Sets this scanner’s delimiting pattern.

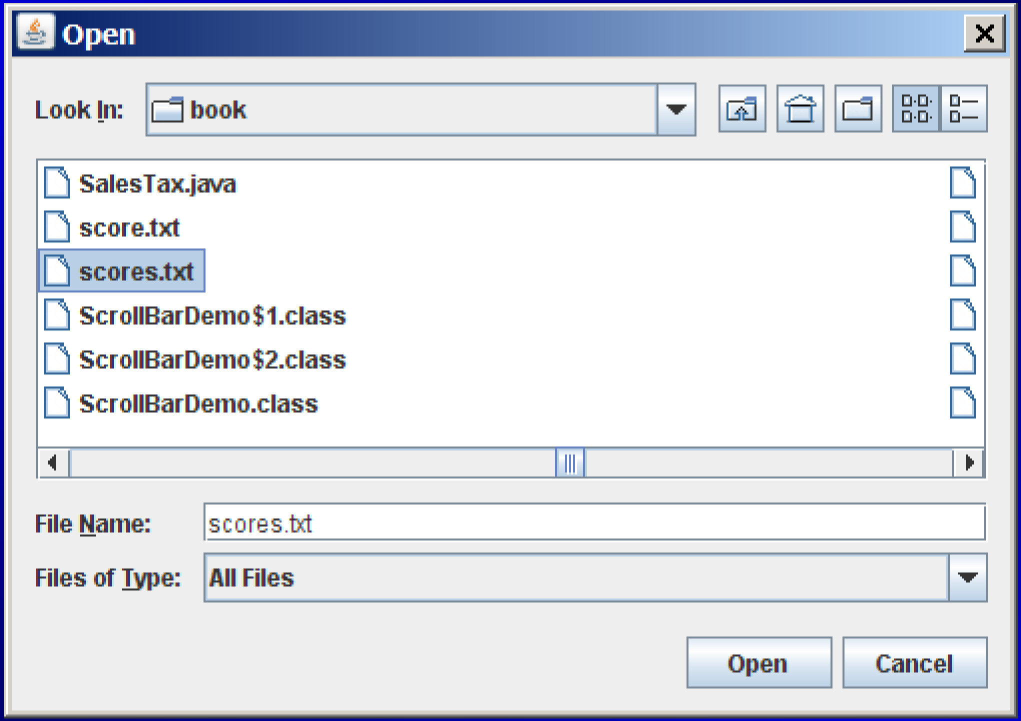
Reading Data Using Scanner



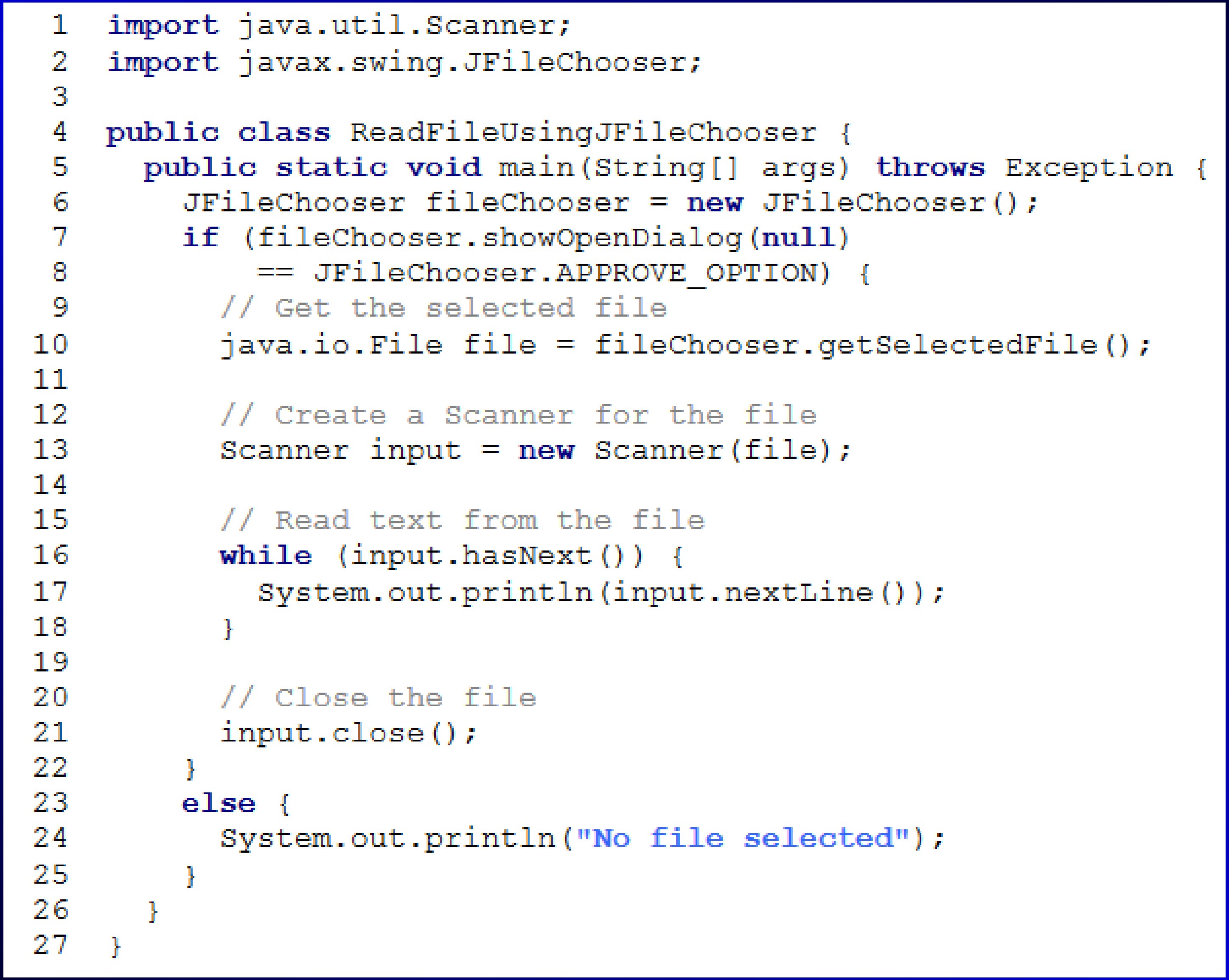
```
import java.io.File;
import java.util.Scanner;

public class ReadData {
  public static void main(String[] args) throws Exception {
    // Create a File instance
    File file = new File("C:\\scores.txt");

    // Create a Scanner for the file
    Scanner input = new Scanner(file);

    // Read data from a file
    while (input.hasNext()) {
      String firstName = input.next();
      String mi = input.next();
      String lastName = input.next();
      int score = input.nextInt();
      System.out.println(
        firstName + " " + mi + " " + lastName + " " + score);
    }

    // Close the file
    input.close();
  }
}
```

JFileChooser

JFileChooser *is a GUI component for displaying a file dialog.*



JFileChooser



```java
import java.util.Scanner;
import javax.swing.JFileChooser;

public class ReadFileUsingJFileChooser {
  public static void main(String[] args) throws Exception {
    JFileChooser fileChooser = new JFileChooser();
    if (fileChooser.showOpenDialog(null)
        == JFileChooser.APPROVE_OPTION) {
      // Get the selected file
      java.io.File file = fileChooser.getSelectedFile();

      // Create a Scanner for the file
      Scanner input = new Scanner(file);

      // Read text from the file
      while (input.hasNext()) {
        System.out.println(input.nextLine());
      }

      // Close the file
      input.close();
    }
    else {
      System.out.println("No file selected");
    }
  }
}
```

Reading Data from the Web

- Just like you can read data from a file on your computer, you can read data from a file on the Web if you know the file’s URL (Uniform Resource Locator— the unique address for a file on the Web).

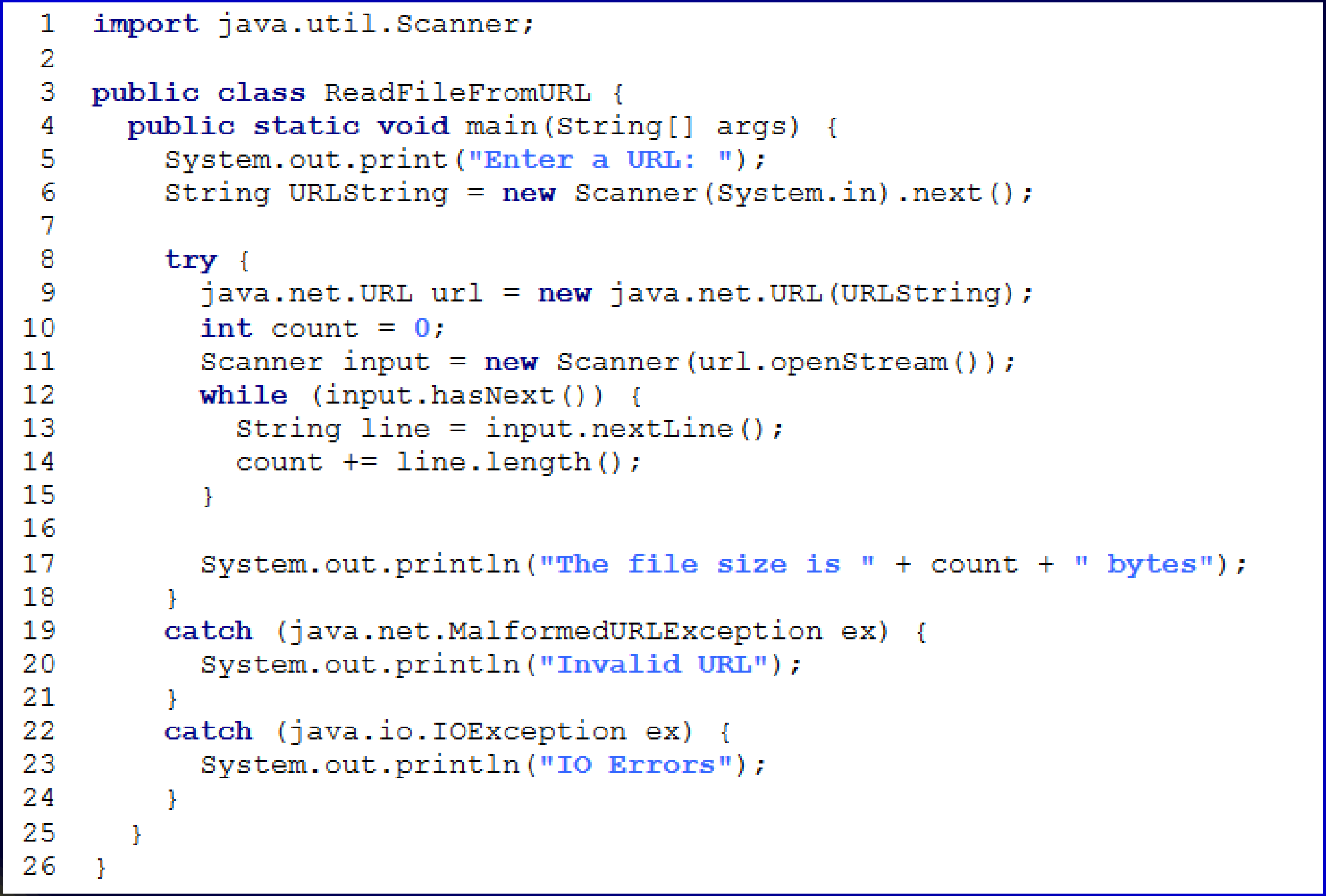
```
URL url = new URL("http://memory.loc.gov/ammem/index.html");
```

After a **URL** object is created, you can use the **openStream()** method defined in the **URL** class to open an input stream and use this stream to create a **Scanner** object as follows:

```
Scanner input = new Scanner(url.openStream());
```

Reading Data from the Web

Use “http://cs.armstrong.edu/liang/data/Lincoln.txt” as URL



```java
import java.util.Scanner;

public class ReadFileFromURL {
  public static void main(String[] args) {
    System.out.print("Enter a URL: ");
    String URLString = new Scanner(System.in).next();

    try {
      java.net.URL url = new java.net.URL(URLString);
      int count = 0;
      Scanner input = new Scanner(url.openStream());
      while (input.hasNext()) {
        String line = input.nextLine();
        count += line.length();
      }

      System.out.println("The file size is " + count + " bytes");
    }
    catch (java.net.MalformedURLException ex) {
      System.out.println("Invalid URL");
    }
    catch (java.io.IOException ex) {
      System.out.println("IO Errors");
    }
  }
}
```

Programming Challenge

Modify the program in the previous slide to count the number of letters, words, and lines in the file. Assume that words are separated by whitespace characters (Hint: Use split method in the string class).

Note: See http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html#sum for regular expressions

Programming Challenge

Write a program that removes all the occurrences of a specified string from a text file. Ask the user for the name of the source file (JFileChooser),, the name of the destination file (JFileChooser), and the text to remove (JOptionPane). The destination file will contain all the text in the source file without the text to remove.

Programming Challenge

Suppose that a text file contains an unspecified number of scores separated by blanks.
Write a program that prompts the user to enter the file (JFileChooser), reads the scores from the file, and displays their total and average (JOptionPane).

Scores are separated by blanks.

For Example:

100 78 98 34 56 79 …

Appendix
printf function

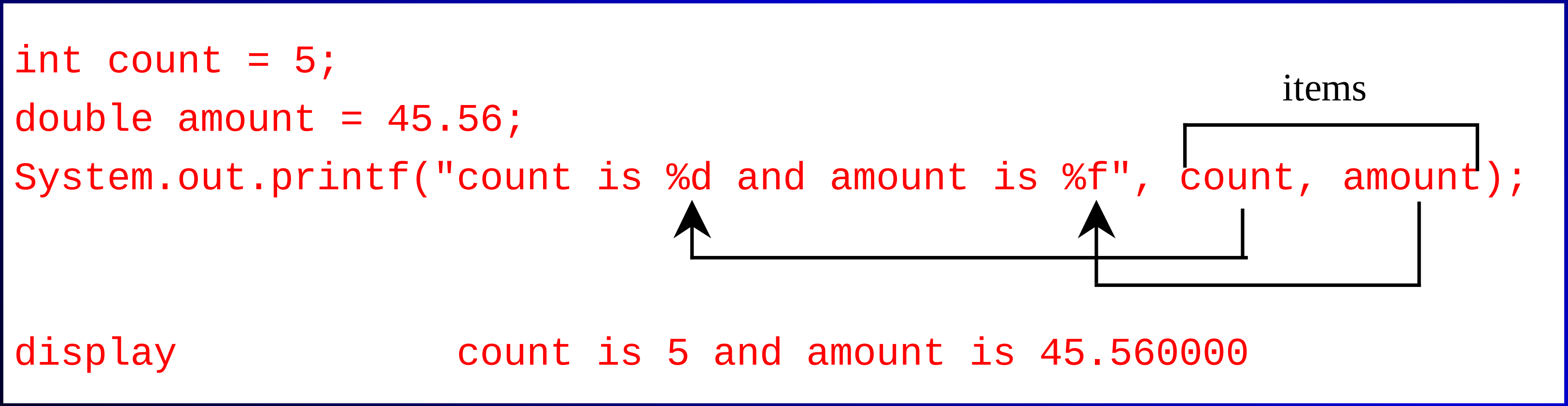
Use the printf statement.

```
System.out.printf(format, items);
```

Where format is a string that may consist of substrings and format specifiers. A format specifier specifies how an item should be displayed. An item may be a numeric value, character, boolean value, or a string. Each specifier begins with a percent sign.

Frequently-Used Specifiers

Specifier	Output	Example
%b	a boolean value	true or false
%c	a character	'a'
%d	a decimal integer	200
%f	a floating-point number	45.460000
%e	a number in standard scientific notation	4.556000e+01
%s	a string	"Java is cool"



```
int count = 5;
double amount = 45.56;
System.out.printf("count is %d and amount is %f", count, amount);


display            count is 5 and amount is 45.560000
```