Files
File Input and Output

File Input and Output

- Files can be *input files* or *output files*.
- File Input:
  - Reentering data all the time could get tedious for the user. The data can be saved to a file that can be used as an input file.
- File Output:
  - Files have to be opened.
  - Data is then written to the file.
  - The file must be closed prior to program termination.
- In general, there are two types of files:
  - binary
  - text

Creating a File
Specify a File Location

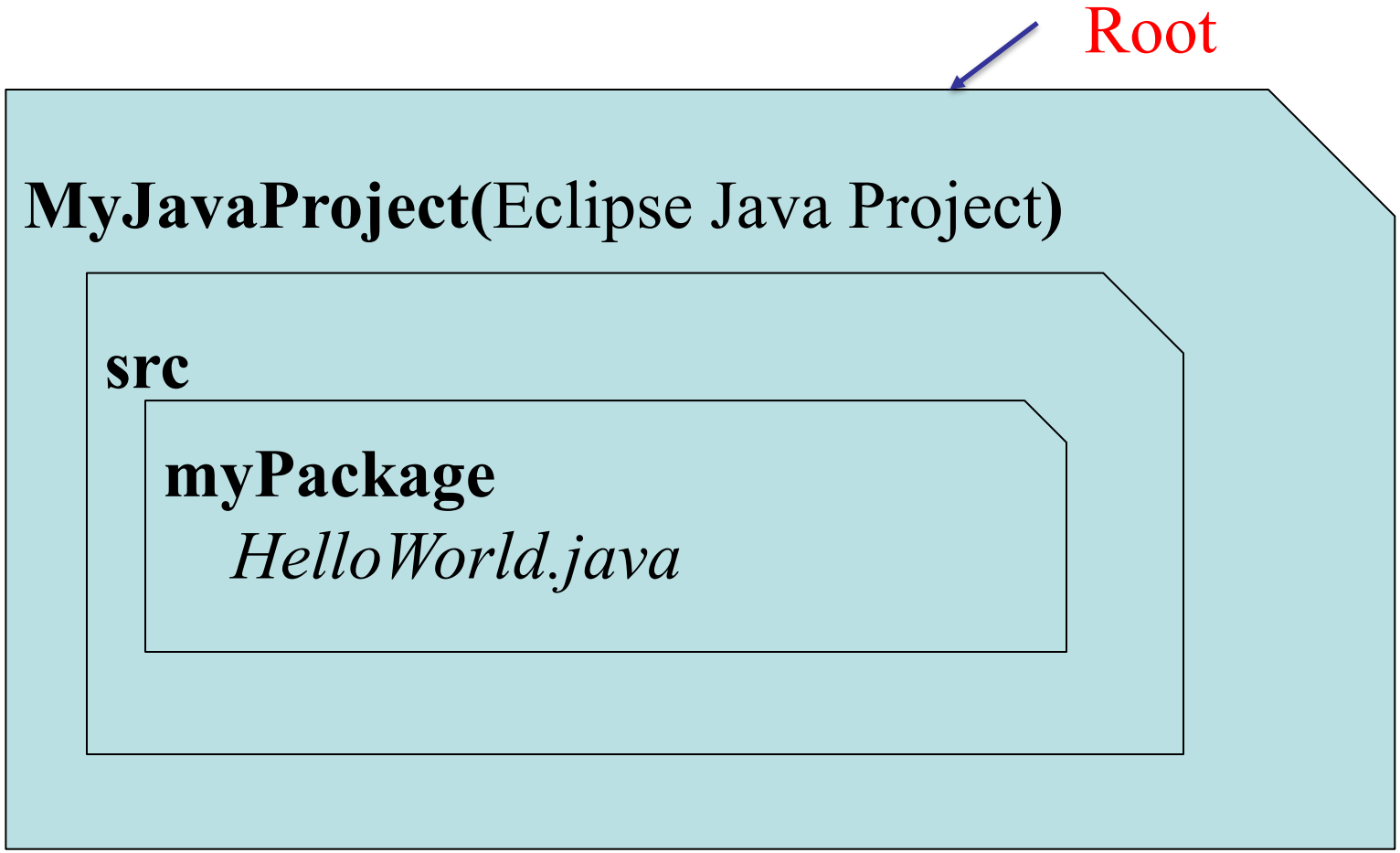
- You use the `File` class constructor to create a `File` object by passing a relative path (`pathname`) to the file into the constructor.

```
public File(String pathname)
```

- A relative path is the file hierarchy from the root directory to the file.
- Java uses the forward slash (`/`) to separate directories.

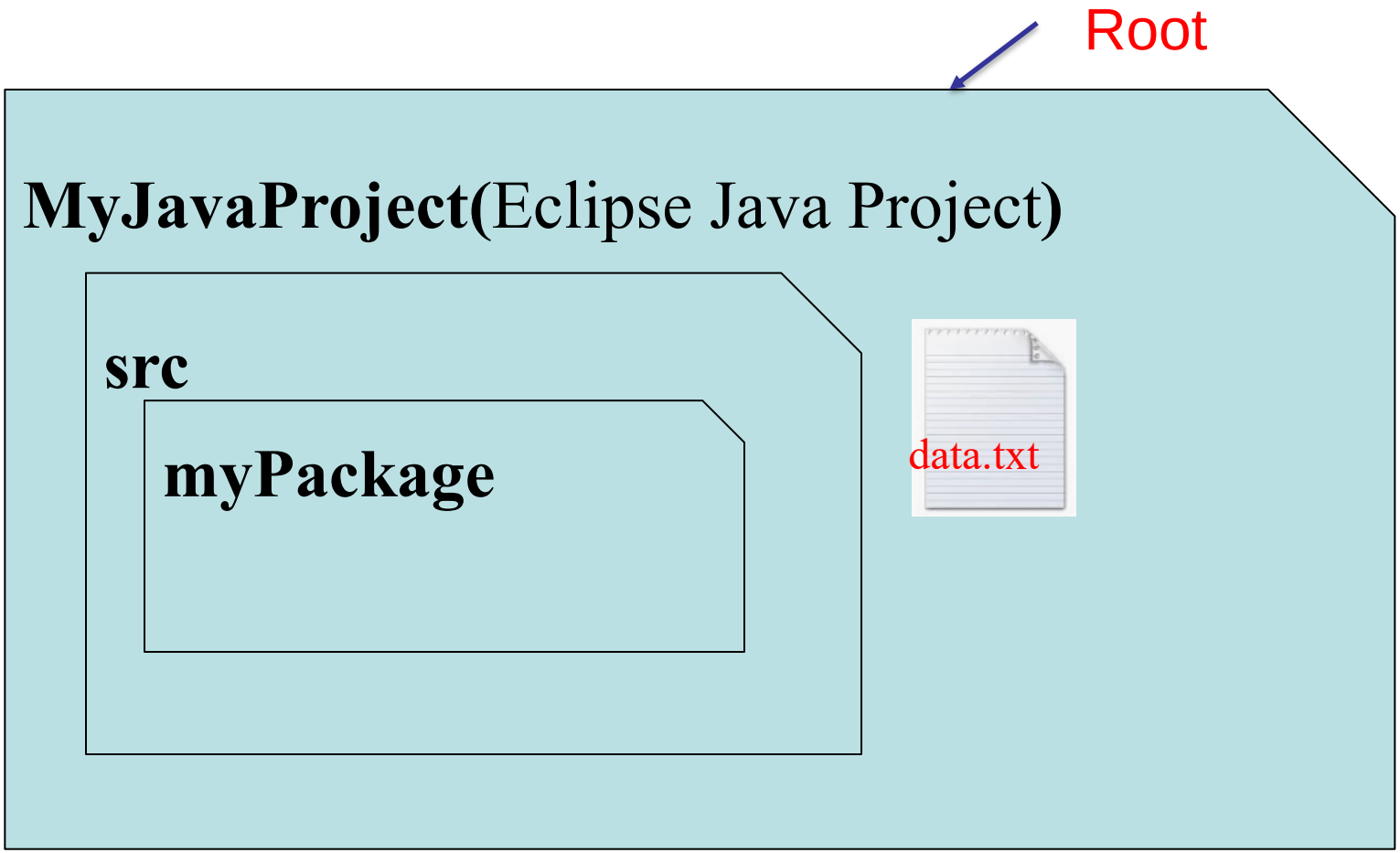
Specify a File Location

- A relative path is the file hierarchy to the file.



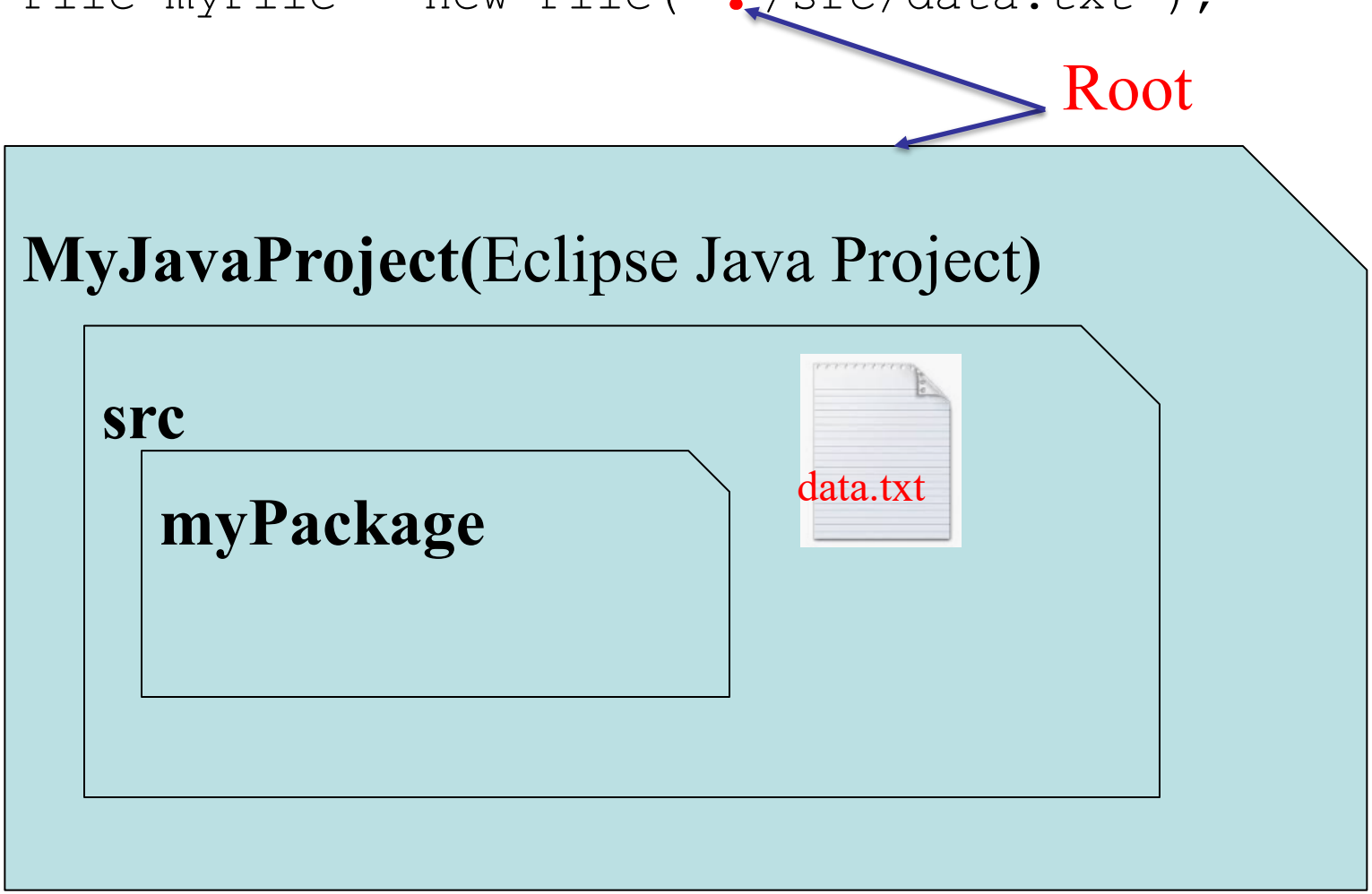
Specify a File Location

```
File myFile = new File("data.txt");
```



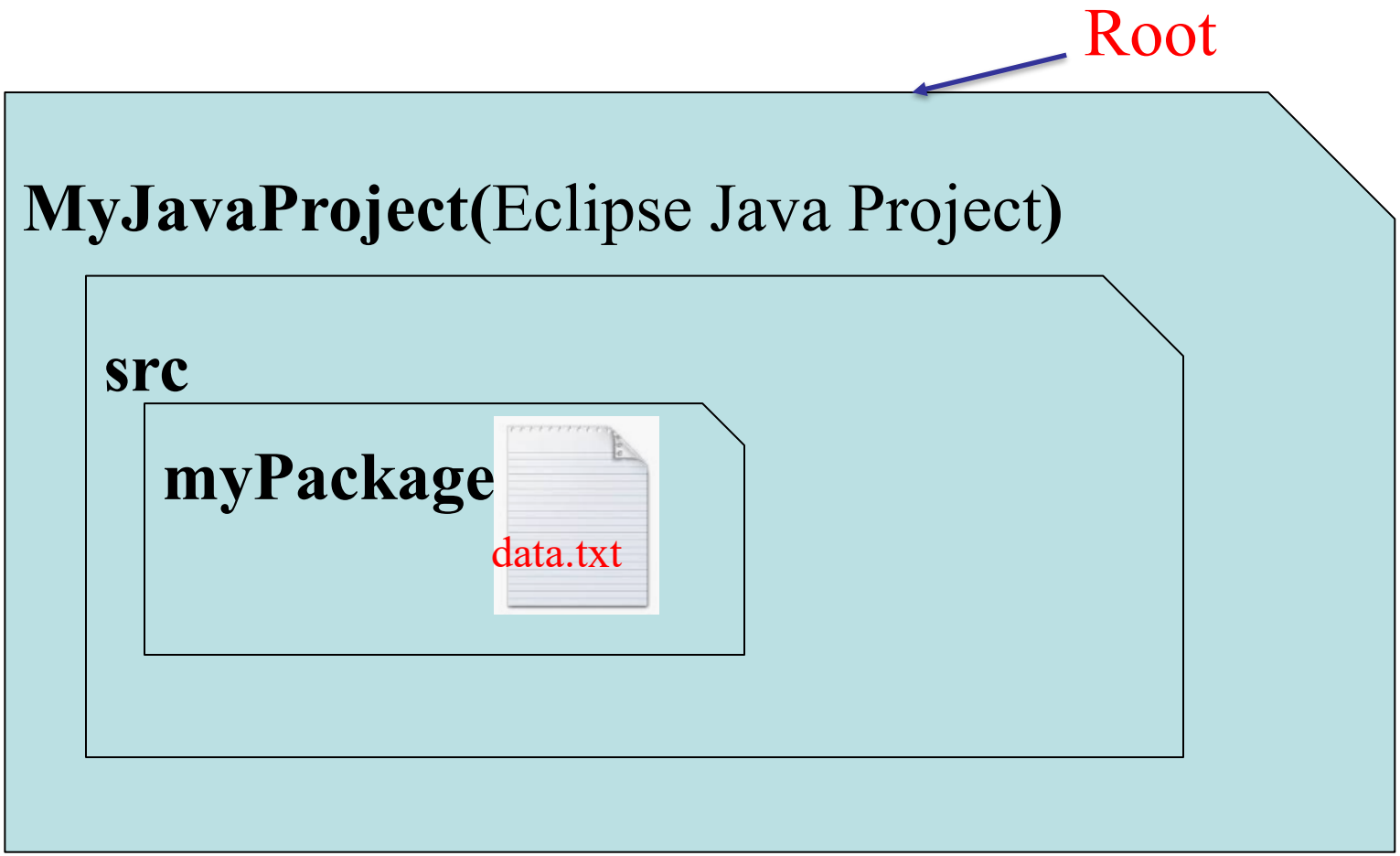
Specify a File Location

```
File myFile = new File(" ./src/data.txt");
```



Specify a File Location

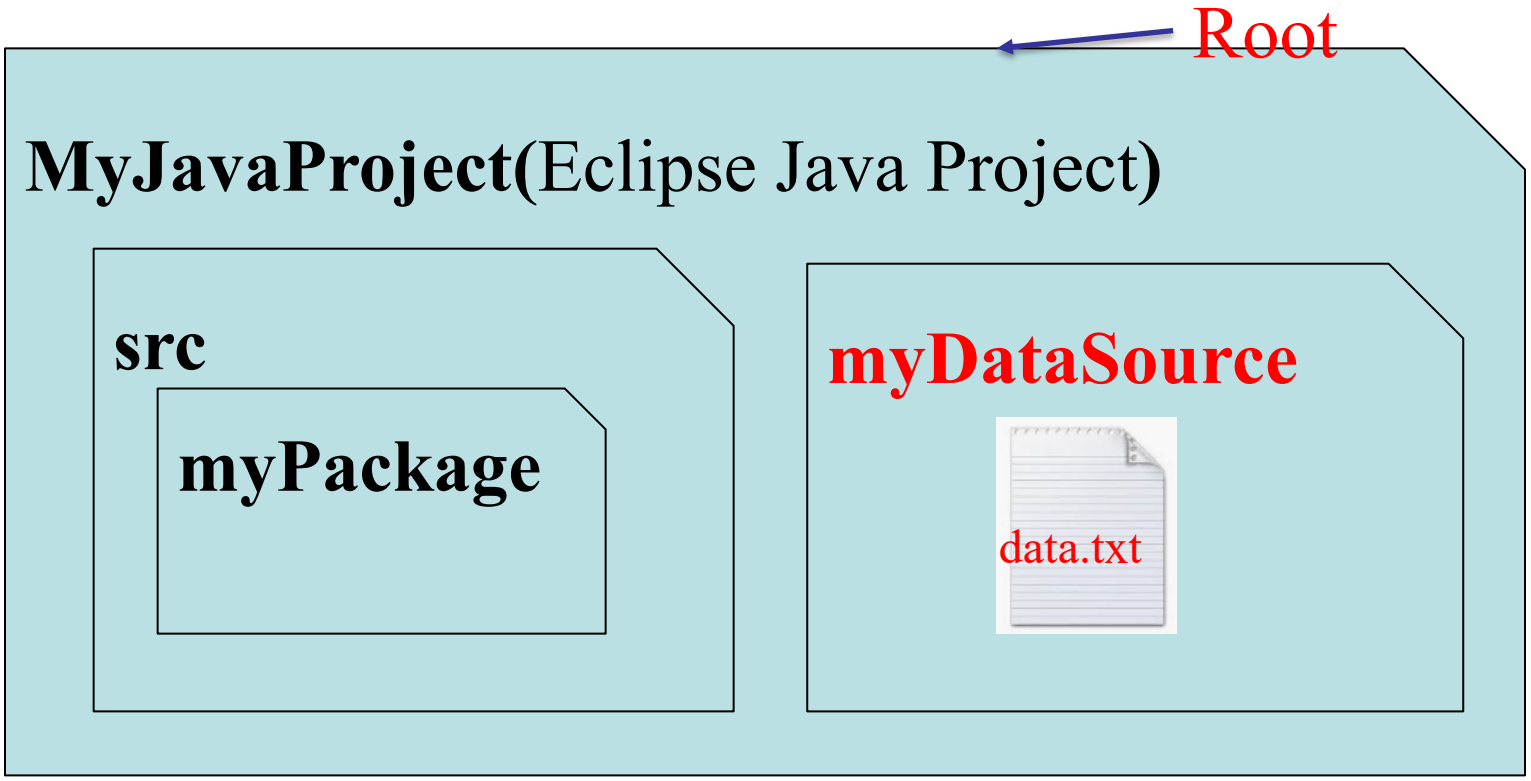
```
File myFile = new File("./src/myPackage/data.txt");
```



Specify a File Location

```
File myFile = new File("./myDataSource/data.txt");
```

- Data should be stored in their own folder.



Read from a File

Reading Data From a File

- You use the `File` class and the `Scanner` class to read data from a file:

Pass the name of the file as an argument to the `File` class constructor.

```
File myFile = new File("Customers.txt");
Scanner inputFile = new Scanner(myFile);
```

Pass the `File` object as an argument to the `Scanner` class constructor.

Reading Data From a File

- Create an instance of `Scanner` for reading from a file.

```
File file = new File("Names.txt");
Scanner inputFile = new Scanner(file);
```

- Data can be read using the same methods used to read keyboard input (`nextLine`, `nextInt`, `nextDouble`, etc).

```
// Read a line from the file.
String str = inputFile.nextLine();
```

Exceptions

- The `Scanner` class can throw an `IOException` when a `File` object is passed to its constructor.
- So, we put a `throws IOException` clause in the header of the method that instantiates the `Scanner` class.

Detecting The End of a File

- The `Scanner` class's `hasNext()` method will return true if another item can be read from the file.

```
// Open the file.
File file = new File(filename);
Scanner inputFile = new Scanner(file);

// Read until the end of the file.
while (inputFile.hasNext()){
   String str = inputFile.nextLine();
   System.out.println(str);
}
```

```java
package chapter04;

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

/**
 * Reads data from a file.
 * @author Qi Wang
 * @version 1.0
 */
public class ReadFileDemo{
    /**
     * Reads friends' information from a file.
     * File name: MyFriends.txt. File default folder: the Java project folder.
     * File contains
     * Joe
     * Rose
     * Greg
     * Kirk
     * Renee
     * @param args A String array that can hold command-line arguments
     */
    public static void main(String[] args) throws IOException  {
      File file = new File("MyFriends.txt");
      Scanner inputFile = new Scanner(file);
      String line;

      //Check if there is more lines in file.
      while(inputFile.hasNext()){
          //Read a line and print.
          line = inputFile.nextLine();
          System.out.println(line);
      }

      inputFile.close();
   }
}
```

Write into a File

Writing Text To a File

- To open a file for text output you create an instance of the `PrintWriter` class.

```
PrintWriter outputFile = new PrintWriter("StudentData.txt");
```

Pass the name of the file that you wish to open as an argument to the `PrintWriter` constructor.

Warning: if the file already exists, it will be erased and replaced with a new file.

The `PrintWriter` Class

- The `PrintWriter` class allows you to write data to a file using the `print` and `println` methods, as you have been using to display data on the screen.
- Just as with the `System.out` object, the `println` method of the `PrintWriter` class will place a newline character after the written data.
- The `print` method writes data without writing the newline character.

The `PrintWriter` Class

1. Open the file.

```
PrintWriter outputFile = new PrintWriter("Names.txt");
outputFile.println("Chris");
outputFile.println("Kathryn");
outputFile.println("Jean");
outputFile.close();
```

2. Write data to the file.

3. Close the file.

The `PrintWriter` Class

- To use the `PrintWriter` class, put the following `import` statement at the top of the source file:

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

```java
package chapter04;

import java.util.Scanner;
import java.io.PrintWriter;
import java.io.IOException;

/**
 * Writes data to a file.
 * @author Qi Wang
 * @version 1.0
 */
public class FileWriteDemo{
    /**
     * Writes friends' information into a file.
     * @param args A String array that can hold command-line arguments
     */
   public static void main(String[] args) throws IOException{
      String fileName, friendName;
      int numberOfFriends;
      PrintWriter outputFile;
      Scanner input = new Scanner(System.in);

      // Read the number of friends.
      System.out.print("How many friends do you have? ");
      numberOfFriends = input.nextInt();
      // Skip the remaining newline character.
      input.nextLine();

      // Read the filename.
      System.out.print("Enter the filename: ");
      fileName = input.nextLine();

      // Open the file.
      outputFile = new PrintWriter(fileName);
```

```java
        // Get data and write it to the file.
        for (int i = 0; i < numberOfFriends; i++){
            // Get the name of a friend.
            System.out.print("Enter the name of friend " + "number " + (i + 1) + ": ");
            friendName = input.nextLine();
            // Write the name to the file.
            outputFile.println(friendName);
        }
        // Close the file.
        outputFile.close();
        input.close();
        System.out.println("Data written to the file.");
    }
}
```

```java
package chapter04;

import java.util.Scanner;
import java.io.PrintWriter;
import java.io.IOException;
import java.io.File;
/**
 * Writes data to a file.
 * @author Qi Wang
 * @version 1.0
 */
public class FileWriteDemo2{
    /**
     * Writes friends' information into a file.
     * @param args A String array that can hold command-line arguments
     */
   public static void main(String[] args) throws IOException {
      String fileName, friendName;
      int numberOfFriends;
      PrintWriter outputFile;
      Scanner input = new Scanner(System.in);

      // Read the number of friends.
      System.out.print("How many friends do you have? ");
      numberOfFriends = input.nextInt();

      // Skip the remaining newline character.
      input.nextLine();

      // Read the filename.
      System.out.print("Enter the filename: ");
      fileName = input.nextLine();
```

Checking for file existence first .

```java
    // Make sure the file does not exist.
    File file = new File(fileName);
    if (file.exists()){
       System.out.println("The file " + fileName + " already exists.");
       System.exit(0);
    }

    // Open the file.
    outputFile = new PrintWriter(file);

    // Get data and write it to the file.
    for (int i = 0; i < numberOfFriends; i++){
       // Get the name of a friend.
       System.out.print("Enter the name of friend " +  "number " + (i+1) + ": ");
       friendName = input.nextLine();

       // Write the name to the file.
       outputFile.println(friendName);
    }

    // Close the file.
    outputFile.close();
    input.close();
    System.out.println("Data written to the file.");
  }
}
```

Exceptions

- When something unexpected happens in a Java program, an *exception* is thrown.
- The method that is executing when the exception is thrown must either handle the exception or pass it up the line.
- Handling the exception will be discussed later.
- To pass it up the line, the method needs a `throws` clause in the method header.

Exceptions

- To insert a `throws` clause in a method header, simply add the word *throws* and the name of the expected exception.
- `PrintWriter` objects can throw an `IOException`, so we write the `throws` clause like this:

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

Appending Text to a File

- To avoid erasing a file that already exists, create a `FileWriter` object in this manner:

```
FileWriter fw =
     new FileWriter("names.txt", true);
```

- Then, create a `PrintWriter` object in this manner:

```
PrintWriter fw = new PrintWriter(fw);
```

Specifying a File Location

- On a Windows computer, paths contain backslash (\) characters.
- Remember, if the backslash is used in a string literal, it is the escape character so you must use two of them:

```
PrintWriter outFile =
     new PrintWriter("A:\\PriceList.txt");
```

Specifying a File Location

- This is only necessary if the backslash is in a string literal.
- If the backslash is in a `String` object, then it will be handled properly.
- Fortunately, Java allows Unix style filenames using the forward slash (`/`) to separate directories:

```
PrintWriter outFile = new
    PrintWriter("/home/rharrison/names.txt");
```

Generating Random Numbers with the `Random` Class

- Some applications, such as games and simulations, require the use of randomly generated numbers.
- The Java API has a class, `Random`, for this purpose. To use the `Random` class, use the following `import` statement and create an instance of the class.

```
import java.util.Random;

Random randomNumbers = new Random();
```

Some Methods of the `Random` Class

Method	Description
nextDouble()	Returns the next random number as a double. The number will be within the range of 0.0 and 1.0.
nextFloat()	Returns the next random number as a float. The number will be within the range of 0.0 and 1.0.
nextInt()	Returns the next random number as an int. The number will be within the range of an int, which is –2,147,483,648 to +2,147,483,648.
nextInt(int n)	This method accepts an integer argument, n. It returns a random number as an int. The number will be within the range of 0 to n.

```java
package chapter04;

import java.util.Scanner;
import java.util.Random;

/**
 * Rolls two dice.
 * @author Qi Wang
 * @version 1.0
 */
public class RollDice{
    /**
     * Rolls six-sided dice.
     * @param args A String array that can hold command-line arguments
     */
    public static void main(String[] args)   {
      String again = "y";
      int die1;
      int die2;
      Scanner input = new Scanner(System.in);
      Random rand = new Random();


      // Simulate rolling the dice.
      while (again.equalsIgnoreCase("y")){
         System.out.println("Rolling the dice...");
         die1 = rand.nextInt(6) + 1;
         die2 = rand.nextInt(6) + 1;

         System.out.println("Their values are: " + die1 + " " + die2);
         System.out.print("Enter \'y\' to roll again, \'n\' to quit: ");
         again = input.nextLine();
      }

      input.close();
   }
}
```

Summary

- **Repetition Statements (Loops)**
  - **The `while, do-while, for` Statements**
- **Files**
  - **Write into a file**
  - **Read from a file**