



# JAVA Textbook

## *Chapter 7*

# File Handling in Java Programs



# Objectives

In this chapter, you will learn about:

- File operations in Java.



# File Handling

- Business applications often need to work with data stored in files.
- To work with data stored in a file, the following operations are commonly needed:
  - Open a file
  - Read data from a file
  - Write data to a file
  - Close a file
- Prewritten classes in the Java Standard Edition Development Kit (JDK) are used to accomplish the file operations.

# Importing Packages and Classes

- The JDK contains many classes that are prewritten by the Java development team, which you can use to simplify your work.
- Prewritten classes must be imported into a Java program that uses them.
  - Use the *import* keyword to include a class from a Java package.
- The classes needed for file operations are part of the package *java.io*.
  - A package is a group of related classes.
  - To import the *BufferedReader* class  
***import java.io.BufferedReader;***
  - To import all classes from the *java.io* package  
***import java.io.\*;***

# Opening a File for Reading

- First, create a *FileReader* object and specify the name of the file to associate with the object.
  - If the file is located in the same folder as the Java program:  
***FileReader fr = new FileReader("inputFile.txt");***
  - If not in the same folder, need to include its complete path:  
***FileReader fr = new FileReader("C:\myJavaPrograms\Chapter7\inputFile.txt");***
- Then, create a *BufferedReader* object to read more efficiently.
  - A *FileReader* object reads one character at a time, whereas a *BufferedReader* object reads a line at a time.
  - Create a *BufferedReader* object by decorating the *FileReader* object.
  - Decorating is a way of adding functionality to objects in

# Reading Data from an Input File

Tim

Moriarty

4000.00

```
String firstName, lastName, salaryString;  
double salary;  
firstName = br.readLine();  
lastName = br.readLine();  
salaryString = br.readLine();
```

```
salary = Double.parseDouble(salaryString);
```

```
salary = Double.parseDouble(br.readLine());
```

- The *BufferedReader* class reads data from a file with the *readLine()* method.
  - The *readLine()* method reads a line from an input file.
  - A line is defined as all of the characters up to a newline character or up to the End Of File (EOF) marker.
  - The newline character is generated when the Enter key on the keyboard is pressed.
  - The EOF marker is automatically placed at the end of a file<sup>6</sup> when it is saved.

# Reading Data Using a Loop and EOF

```
while((firstName = br.readLine()) != null)
{
    // body of loop
}
```

- Use a loop to read large amounts of data from a file.
- The *readLine()* method returns a null value when EOF is reached.

# Opening a File for Writing

- First, create a *FileWriter* object and specify the name of the file to associate with the object.
  - If the file is in the same folder as the Java program:  
***FileWriter fw = new FileWriter("outputFile.txt");***
  - Need to include the file's complete path if not in the same folder:  
***FileWriter fw = new FileWriter("C:\\myJavaPrograms\\Chapter7\\outputFile.txt");***
- Then, create a *PrintWriter* object to decorate the *FileWriter* object for more efficient operations.
  - The *PrintWriter* class provides the ability to flush (that is, empty) and close an output file.
  - In Java, a write operation is not complete until the buffer associated with an output file is flushed (emptied) and closed, thus being made unavailable for further output. 8

***PrintWriter pw = new PrintWriter(fw);***



# Writing Data to an Output File

Tim

Moriarty

4000.00

```
final double INCREASE = 1.15;  
double newSalary;  
newSalary = salary * INCREASE;
```

```
FileWriter fw = new FileWriter("newSalary2015.txt");  
PrintWriter pw = new PrintWriter(fw);  
pw.println(lastName);  
pw.println(firstName);  
pw.println(newSalary);  
pw.flush();  
pw.close();
```

- The *PrintWriter* class writes a line to an output file with the *println()* method.
- Example above assumes that data needed have already been read from the file.

# Reading/Writing: A Complete Example

- There may be potential problems with file operations.
  - May try to open a nonexistent file, to read beyond EOF, etc.
- Such a problem will generate an exception.
  - An exception is an event that disrupts the normal flow of execution, and is handled by an exception handler.
- Java compiler won't compile programs with such operations unless

```
// EmployeeRaise.java - This program reads employee first
// and last names and salaries from an input file,
// calculates a 15% raise, and writes the employee's first
// and last name and new salary to an output file.
// Input:  employees.txt.
// Output: newSalary2015.txt

import java.io.*; // Import class for file input.

public class EmployeeRaise
{
    public static void main(String args[]) throws Exception
    {

        String firstName, lastName, salaryString;
        double salary, newSalary;
        final double INCREASE = 1.15;

        // Open input file.
        FileReader fr = new FileReader("employees.txt");
        // Create BufferedReader object.
        BufferedReader br = new BufferedReader(fr);

        // Open output file
        FileWriter fw = new FileWriter("newSalary2015.txt");
        PrintWriter pw = new PrintWriter(fw);

        // Read records from file and test for EOF.
        while((firstName = br.readLine()) != null)
        {
            lastName = br.readLine();
            salaryString = br.readLine();
            salary = Double.parseDouble(salaryString);
            newSalary = salary * INCREASE;
            pw.println(lastName);
            pw.println(firstName);
            pw.println(newSalary);
            pw.flush();
        }

        br.close();
        pw.close();
        System.exit(0);
    } // End of main() method.
} // End of EmployeeRaise class.
```



Thank You!