This Program is due on Date Specified.

Comments are **REQUIRED**; flow charts and pseudocode are **NOT REQUIRED**.

Directions	Points
The file must be called <**LastInitialFirstInitialUnit3.java**> *Proper coding conventions required the first letter of the class start with a capital letter and the first letter of each additional word start with a capital letter.* Only submit the **.java** file needed to make the program run. Do not submit the **.class** file or any other file.	5%
Style Components Include properly formatted prologue, comments, indenting, and other style elements as shown in Chapter 2 starting page 64 and Appendix 5 page 881-892.	5%
LiFiUnit3.java - Main Method Program should read in a file called "sales.txt". The contents of the text file are below. Read in all data from the file. (refer to pages 102-103) • Sore units in integers and store sales in doubles. • The first integer is the first unit sales. The first double is the first $ of all the sales combined. • This continues through all five units. Utilize a constant called taxRate to hold the tax rate of 8.5%. Output the following items per the sample below: • Total Unit Sales (the sum of all units) • Total Sales (the sum of all sales) • Total Tax (rounded) (sales times taxRate) ◦ Type cast to int * 100, then double / 100. • Total Sale (not rounded) (sales plus tax) • Output results in table format using the escape character for a tab as shown in the sample. Round the sales tax using basic math to round it. Do **NOT** use printf or any rounding function to perform this.	90%

The `nextLine` method is the only method that processes leading whitespace. The other methods skip it. Suppose you have a `nextInt` method call and the user types `25` and then presses the enter key. The `nextInt` method call reads the 25 and returns it. The `nextInt` method call does not read in the enter key's newline character. Suppose that after the `nextInt` method call, you have a `nextLine` method call. The `nextLine` method call does not skip leading whitespace, so it's stuck with reading whatever is left over from the previous input call. In our example, the previous input call left the enter key's newline character. Thus, the `nextLine` call is stuck with reading it. Uh oh.

What happens if the `nextLine` method reads a newline character? It quits because it's done reading a line (the newline character marks the end of a line, albeit a very short line). So the `nextLine` method call doesn't get around to reading the next line, which is probably the line that the programmer intended it to read.

One solution to this `nextLine` problem is to include an extra `nextLine` method call whose sole purpose is to read in the leftover newline character. Another solution is to use one `Scanner` reference variable for `nextLine` input (e.g., `stdIn1`) and another `Scanner` reference variable for other input (e.g., `stdIn2`). But for the most part, we'll try to steer clear of the problem altogether. We'll try to avoid `nextLine` method calls that follow one of the other `Scanner` method calls.

3.24 Simple File Input for Repetitive Testing During Program Development

As you progress through the book, you'll see that input from the computer keyboard and output to the computer screen is all the I/O you need to solve a vast array of complex problems. But if you have a large amount of input, it might be easier and safer to use a simple text processor to write that input once into a file and then reread it from that file each time you rerun the program.

Figure 3.16 shows an updated version of Figure 3.15's `PrintInitials` program. This new version shows the original keyboard-input code commented out and replaced with code that reads the initials from the first name in a `names.txt` file. Note this line from the `PrintInitials2` program:

```
Scanner stdIn = new Scanner(new File("names.txt"));
```

The `new File("names.txt")` code causes input to come from a file named `names.txt`. For comparison purposes, look at the commented-out `Scanner` object code, where `System.in` is used for keyboard input.

In the `PrintInitials2` program's `new File("names.txt")` code, note the word `File`. The `File` class is not part of the core Java language, so if you use it, you need to import it at the top of your program. More specifically, you need to do this:

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

The attempt to create a `Scanner` object with `new Scanner(new File("names.txt"))` might not work. For example, it won't work if the operating system is unable to create the specified file. The Java compiler is a pessimist by nature, so, by default, it will generate a compilation error if you use the command `new Scanner(new File("names.txt"))` without some kind of "protective gear." The simplest way to avoid the compilation error is to append `throws Exception` to the `main` method heading, like this:

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

```
/*********************************************************************
* PrintInitials2.java
* Dean & Dean
*
* This prints leading character of first two words in a file.
*********************************************************************/

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

public class PrintInitials2
{
  public static void main(String[] args) throws Exception
  {
//    Scanner stdIn = new Scanner(System.in);
    Scanner stdIn = new Scanner(new File("names.txt"));
    String first; // first name
    String last;  // last name

//    System.out.print (
//      "Enter your first and last name separated by a space: ");
    first = stdIn.next();
    last = stdIn.next();
    System.out.println("The initials are " +
      first.charAt(0) + last.charAt(0) + ".");
  } // end main
} // end PrintInitials2 class
```

new code to create a Scanner object that reads input from a file

Figure 3.16 Updated version of the PrintInitials program that reads input from a file

The `throws Exception` clause tells the Java compiler that you're aware that the JVM might not be able to do what you want. More specifically, you're telling the compiler that an error (exception) might be generated (thrown). The `throws Exception` clause won't magically guarantee that the file will be created successfully, but if you don't append it to the `main` method heading, your program won't even be able to try. Why? Because it won't compile. Appending `throws Exception` to the `main` method header is a "simple and dirty" way to force successful compilation.

Chapter 15 describes a more responsible treatment of exceptions. If you want to use this more responsible treatment when you write to or read from a file, look at the HTMLGenerator program in Section 16.2. It contains code that you can use as a template. That code is more complicated, however, than just appending `throws Exception` to `main`.

At this point, some readers might want to apply what they've learned to an object-oriented programming (OOP) context. OOP is the idea that programs should be organized into objects. You're not required to learn about OOP just yet, but if you can't wait, you can find such details in Chapter 6, Sections 6.1 through 6.8.