# *Programming I (420-B10-HR)*

# *Lab 12 – Reading from a Data File and Introduction to using a Frame*

Date assigned: Tuesday, November 17, 2015

Date due: **Tuesday, November 17, 2015**

**Learning Objectives**

Upon successful completion of this lab exercise, the student will be able to:

1. read numeric data from a data file;
2. read data with delimiters from a data file;
3. read data from a file location other than the default.
4. write a simple Java **JFrame**.
5. use the inheritance principle in Java by extending a class.
6. add GUI objects to an frame.
7. implement an interface.
8. override the **actionPerformed()** method of the **ActionListener** interface.

**To Be Handed In:**

1. The **Lab 12 Review Quiz** should be completed in Moodle.
2. The ***username*\_B10\_L12** folder should be zipped and uploaded to Moodle. Make sure that you have reformatted all your Java files to make them properly indented.

**To Start:**

1. Download and unzip the **B10\_L12\_File\_Reading** folder to your **H:\420-B10\Labs** folder. Rename it to ***username*\_B10\_L12\_File\_Reading**.
2. Start **Eclipse**. Use your **H:\420-B10\Labs** folder as the workspace.
3. Create a **New Java Project** called ***username*\_B10\_L12\_File\_Reading**.

# Reading Numeric Data from a Data File

***Purpose:*** Learn to read numbers from a data file.

To Do:

## Create a class called **ReadTest** in the **b10\_l12** package. It should contain a **main()** method.

## Import **java.io.File** and **java.util.Scanner**.

## Add the following at the beginning of the **main()** method in order to define the objects necessary to read from ***someIntegers.dat*** file instead of the keyboard:

### Declare a **File** object called **integerFile**:

**File integerFile;**

### Declare a **Scanner** object called **inFile**:

**Scanner inFile = null;**

### Instantiate **integerFile** with a constructor identifying the ***someIntegers.dat*** data file:

**integerFile = new File("someIntegers.dat ");**

### Instantiate the **inFile** Scanner object to refer to the **integerFile** object:

**inFile = new Scanner(integerFile);**

What is the syntax error? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

You need to add a try and catch construct to handle I/O errors.

## Add:

**try**

**{**

before the **integerFile** instantiation.

Add:

**}**

**catch (FileNotFoundException e)**

**{**

**System.err.println("someIntegers.dat could not be found.");**

**System.exit(-1);**

**} // catch (FileNotFoundException e)**

**catch (IOException e)**

**{**

**System.err.println("I/O error. " + e.getMessage());**

**System.exit(-2);**

**} // catch (IOException)**

after the instantiation of the **inFile** Scanner.

## Add import statements for the Exceptions:

**import java.io.FileNotFoundException;**

**import java.io.IOException;**

## Run the program. What happens? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Where did the message come from? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Select **File 🡪 New 🡪 File** to create a new text file called **someIntegers.dat**. Save it in your ***username*\_B10\_L12\_File\_Reading** folder. Type in the following 3 lines:

**500 23**

**420 7**

**790 85**

## Save the file.

## Add the following code after the end of the catch block. It reads the first line from the data file and displays the data that was read.

**int num1, num2;**

**num1 = inFile.nextInt();**

**num2 = inFile.nextInt();**

**System.out.println("num1 is " + num1);**

**System.out.println("num2 is " + num2);**

## Run the program. What is output? \_\_\_num1 is 500­­\_\_\_\_\_\_\_\_\_\_\_\_\_

## \_\_\_num2 is 23\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Where did the output come from? \_\_\_someIntegers.dat\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

After the **nextInt()** method completes reading a number, is the next read from the next line of the file or from the same line? \_same line\_\_

## Add the following line after the println for **num2**:

**System.out.println("inFile.hasNext() is " + inFile.hasNext());**

## Run the program again. What is the value of **hasNext()** after the last number is displayed? \_true\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## The **hasNext()** method returns **true** if there is more data in the file and **false** if there is no more data. We can use the value of **hasNext()** to end a read loop. Add the following loop after the println you just added:

**int count = 3;**

**while (inFile.hasNext())**

**{**

**num1 = inFile.nextInt();**

**System.out.println("The count is " + count**

**+ ". The number read is " + num1);**

**++count;**

**System.out.println("inFile.hasNext() is " + inFile.hasNext());**

**} // while (inFile.hasNext())**

**inFile.close();**

## Run the program again. What is the last value of **hasNext()**? \_\_false\_\_\_\_\_\_\_\_

## Create a second data file called ***someReals.dat***. Add 4 double precision numbers to it and save it.

## Create a second **File** object called **realFile** and a second **Scanner** called **inFile2**. Use these objects to read in and display the numbers from ***someReals.dat***. Your program should work for any number of numbers in ***someReals.dat***.

## Modify the program to calculate and print the average of the numbers you read in from the **someReals.dat** file. Format the average with 2 decimal places. Your program should still work if you add more numbers or remove numbers from **someReals.dat** file.

# Reading from a Data File with delimiters

***Purpose:*** Learn to read records from a data file with delimiters.

***To Do:***

## Open **Course.java**. Run it.

***Questions:***

1. How many averages were calculated? \_\_3\_\_\_\_\_\_\_\_
2. Which program variable is counting the number of students? \_numStudents\_
3. How many marks were averaged together for each student? \_4\_\_\_\_\_\_\_\_\_\_\_
4. Which program variable is counting the number of marks? \_grade\_\_\_\_\_\_\_\_\_

***To Do:***

## Change the program to input 5 marks for two students. Test your changes.

## Add an instance variable to the **Student** class for the student name. Initialize the name to "Unknown" in the constructor and in the **newStudent()** method. Add methods to set and get the student name.

## Modify the **main()** method in **Course** to input a name for each student and use the mutator to set it in the **Student** class. Use the student name accessor in the **Student** class to print the student's name along with his/her average. Test your program. Your output should look similar to:

What is student 1's name? Jane

Enter a grade for test 1 for student 1 > 88

Enter a grade for test 2 for student 1 > 99

Enter a grade for test 3 for student 1 > 89

Enter a grade for test 4 for student 1 > 98

Enter a grade for test 5 for student 1 > 78

The average for Jane is 90.4

What is student 2's name? George

Enter a grade for test 1 for student 2 > 88

Enter a grade for test 2 for student 2 > 68

Enter a grade for test 3 for student 2 > 87

Enter a grade for test 4 for student 2 > 45

Enter a grade for test 5 for student 2 > 63

The average for George is 70.2

We’re going to modify the **Course** class to read the student data from a data file called **students.txt**.

## At the beginning of the **main()** method, declare a **File** object called **studentFile** and instantiate it with the ***students.txt*** data file.

## Change the instantiation of the **input** Scanner object to refer to the **studentFile** object instead of the keyboard.

## Add a try and catch construct to handle I/O errors.

## Delete the **System.out.println()** method calls in **main()** that prompt for user input. Since the program will now read the data from the ***students.txt*** file, no prompts are required.

## Open the ***students.txt*** file to see what data is stored there and how it is stored. What character is used to separate the fields? \_\_\_~\_\_\_\_\_\_\_\_\_ This is called a delimiter.

## Run the program. What happens? \_\_\_ I get the error shown below\_\_\_\_\_\_\_\_\_\_\_\_

You should get a stack dump similar to the following:

Exception in thread "main" java.util.InputMismatchException

at java.util.Scanner.throwFor(Unknown Source)

at java.util.Scanner.next(Unknown Source)

at java.util.Scanner.nextDouble(Unknown Source)

at b10\_l12.Course.main(Course.java:43)

Click on the underlined **Course.java***.* The statement causing the error will be highlighted. What statement caused the error?

\_\_\_\_grade = input.nextDouble(); \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Explanation:** The **input.next()** statement to read the student name read *Chester* and not *Chester Field* because it assumed that the field ended when it encountered a space. Then when the **input.nextDouble()** statement to input the next mark was executed, it tried to read *Field*. Since *Field* is not a double precision number, an *InputMismatchException* occurred.

You want to read both *Chester* and *Field* into the **studentName** field. You need to tell the Scanner object that the field ends when a ~ is encountered instead of a space.

## Add the following line after the **input** instantiation to declare the delimiters to be the ~ and the end of line characters.

**input.useDelimiter("~|\r\n");**

## Run the program again. Notice that for the second student there is a new line between “for” and the student name.

**Explanation:** When Scanner object reads the second name, it begins immediately after the ~. This means that it reads the new line characters as part of the name.

## Add the following line after the *while* loop to read the marks (immediately before the line that prints the student average). It will read the line end characters.

**input.nextLine(); // read line end characters**

## Run the program again. How many students are displayed? \_\_2\_\_\_\_\_\_\_\_ How many students are in the ***students.txt*** file? \_\_\_4\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Which statement determines how many student records are read?

## \_\_While (numStudents <= 2) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## We want to read all the students in the file into the program, regardless of how many there are. Modify your class to use the **input.hasNext()** method to keep looping as long as there is more data in the file.

## Run the program again. How many students are displayed this time? \_3\_\_

## Add yourself to the *students.txt* file and run the program again.

## Add a Scanner to read from the keyboard. Ask the user for the filename to use. Use the filename that the user provides for the studentFile.

## Run the program again and enter *students.txt* for the filename. The output should be the same as last time you ran it.

## Modify the program to calculate the average for the course. Print out the number of students and the class average after reading all the students. Your last line of output should look similar to:

There are 4 students in the class. The class average is 82.6

## There is another file of students called ***b10\_students.txt*** in the **b10\_l12** package. Run the program again and enter ***b10\_students.txt*** for the filename. What happens? \_\_Nothing has changed\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Since ***b10\_students.txt*** is not in the project folder, we have to specify the full path name. We'll use relative addressing for the path. This time enter the following for the filename:

**src/b10\_l12/ *b10\_students.txt***

## We want to be able to detect whether a file exists before attaching it to a Scanner. If the file doesn't exist, we want to allow the user to reenter the filename. Add a loop which uses **studentFile.exists()** to determine whether a user entered a valid filename. Loop until the user enters a valid filename or chooses to exit the program. Use **System.exit(0)** to exit from the program. Two sample runs are shown here:

*Example 1:*

What is the filename for the students in the course? b10\_students.txt

b10\_students.txt is not in the project folder. Enter the proper path or enter X to exit: src/b10\_l12/b10\_students.txt

The full path is src/b10\_l12/b10\_students.txt

The average for Sue Metu is 89.6

The average for Sue Yu is 72.4

The average for Dwight Wong is 73.0

The average for Mahatma Coate is 81.4

The average for Amanda Linn is 74.2

The average for Moe Bilhome is 77.2

The average for Sal Monella is 80.0

The average for Peter Piper is 83.0

There are 8 students in the class. The class average is 78.85

*Example 2:*

What is the filename for the students in the course? c10\_students.txt

c10\_students.txt is not in the project folder. Enter the proper path or enter X to exit: x

# The JFrame class

***Purpose:*** Learn how to create a subclass of the **JFrame** class.

***To Do:***

## Create a new class called **MyFirstFrame** in the **b10\_l12** package. Generate a main method and a default constructor.

## Add **extends JFrame** to the class header. What is the syntax error?

## \_Syntax error on token “extends”, @ expected\_\_\_\_\_\_\_

**Explanation**: *You need to import the package containing the JFrame class.*

## Add the following line before the class header.

**import javax.swing.\*;**

**In the main() method:**

## Instantiate an instance of the class in the main() method:

MyFirstFrame frame = new MyFirstFrame();

## Add the following lines to your main method after the instantiation of **frame**:

frame.setSize(300, 200);

frame.setLocation(100, 150);

frame.setDefaultCloseOperation( JFrame.EXIT\_ON\_CLOSE );

frame.setVisible(true);

## Add the following statement to the default constructor. Substitute your firstname for name (i.e. I would code "Anne's First Frame"):

setTitle("*name's* First Frame");

## Run the program. Where does it display your title? \_At the top of the container\_

You have created a subclass of the **JFrame** class. At the moment it does nothing because you have not defined what it is supposed to do, but it runs successfully and creates an empty frame.

# Adding GUI Components to an JFrame

***Purpose:*** Learn how to add a button to a JFrame.

***To Do:***

## Add the following instance variable to the class:

**private JButton when;**

## Try running the program again. Nothing should have changed because you have not yet instantiated the **JButton** or added it to the frame.

## Instantiate the **when JButton** in the constructorto display "NOW!":

**when = new JButton("NOW!");**

## Run the program. Has anything changed? \_\_Not yet\_\_\_\_\_\_\_\_\_\_

**Explanation**: *You have created a JButton object, but you have not added it to the frame.*

## To add the **JButton** to the frame, add the following statement to the constructor:

**getContentPane().add(when, BorderLayout.CENTER);**

## There is a red line under **BorderLayout**. That is because **BorderLayout** is not in Swing package. It is in the awt package. Import **java.awt.\***

## Run the program. What has changed? \_There’s a button that says NOW!\_\_\_\_\_ Click the button. Does anything happen? \_\_No\_\_\_\_\_\_\_\_\_\_

# Implementing the ActionListener Interface

***Purpose:*** Learn how respond to mouse clicks on a frame using the **ActionListener**.

***To Do:***

## We must define what it is we want to happen when we click on the button we added. The **actionPerformed()** method of the **ActionListener** interface is executed anytime any mouse or keyboard activity happens in the frame. Add the following to your **MyFirstFrame** class:

**import java.awt.event.\*;**

## Add implements ActionListener to the class header after extends JFrame.

## What is the syntax error you get? \_\_\_\_\_There’s two errors\_\_\_\_\_\_\_\_

**Explanation**: **ActionListener** *is an interface and as such contains only abstract methods. An* ***abstract method*** *is a method that does not have a body. If you implement* **ActionListener***, you must override its* **actionPerformed** *method.*

## Add the following stub method to your class:

public void actionPerformed(ActionEvent e)

{

} // actionPerformed(ActionEvent)

## Run the program again. Has anything changed? \_\_Not yet\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## We have to define what we want the frame to do when we click the button. Add the following code to your **actionPerformed()** method:

String msg = when.getText();

if (msg.equals("NOW!"))

when.setText("THEN!");

## Run the program again. Click the button. Has anything changed? \_Nope\_\_\_\_\_\_

**Explanation**: *We have not specified what GUI object* **ActionListener** *is "listening" for actions on.*

## Add the following line to the constructor:

when.addActionListener(this);

## Run the program again. Click the button several times. What happens now? \_\_Once the button becomes then it doesn’t change again\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Modify the **actionPerformed()** method so that the labels rotate from "NOW!" to "THEN!" to "NEVER!" and back to "NOW!". Test your changes.

# Review Questions

Purpose: Review the file reading concepts and syntax.

To Do:

## Do the **Lab 12 Review** quiz on **Moodle**.