# *Programming II (420-B20-HR)*

# *Lab 2 – Class Variables and Methods and Outputting To a Data File*

Date assigned: Wednesday, January 27, 2016

Date due: **Wednesday, January 27, 2016**

**Learning Objectives**

1. Learn how to declare and use class variables.
2. Learn how to create and call class methods.
3. Review how to read from a data file.
4. Learn how to write to a data file.
5. Learn how to add data to the end of an existing data file.
6. Learn how to use the **WindowListener** interface to perform tasks when a window closes.

**Methods Used:**

*File constructors:*

**File(String***fileName***)** .

*FileWriter constructors:*

**FileWriter(File***file***)** .

FileWriter(File *file*, boolean *append*)

*FileWriter methods:*

**void write(String***str***)**

**void close()**

***WindowListener methods:***

**void windowClosing(WindowEvent** *e***)**

***String methods:***

**String valueOf(int** *integerVariable***)**

**To Be Handed In:**

1. All java files must be formatted using the Eclipse **format** command.
2. The ***username*\_B20\_L02\_Project** folder should be zipped and uploaded to **Moodle**.

**To Start:**

1. Download and unzip the **B20\_L02\_Project** folder from **Moodle** into your **420-B20\Labs** folder. Rename it to ***username*\_B20\_L02\_Project**.
2. Start **Eclipse** and select your **420-B20\Labs** folder as your workspace.
3. Create a new **Java Project** called ***username*\_B20\_L02\_Project**.

# Class Variables and Methods

***Purpose:*** Learn how to use class variables and class methods to store and initialize class data.

***Terminology:***

***Class Variable***: If a variable is declared static, there is exactly one instance of that variable created no matter how many times its class is instantiated.

***Class Method***: Methods declared static are class methods, and can be invoked through the class (as well as through any instance of the class).

***To Do:***

## Open the **StaticVariable** class in the **B20\_L02** package. Which methods change the **num** instance variable? \_\_The class methods and the\_\_\_\_\_\_\_ incrementNumbers method \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Run **TestStatic**. It instantiates 3 objects of the **StaticVariable** class. The **StaticVariable** class contains a single integer variable. When you increment **num** in object **s1**, does **num** in **s2** or **s3** change? \_\_No\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Add a **static** private integer variable called **staticNum** to the **StaticVariable** class. Initialize it to 0.

## Increment **staticNum** in the two constructor methods and in the **incrementNumbers()** method.

## Add an accessor for **staticNum** called **getStaticNum()** to the **StaticVariable** class. (To automatically add an accessor, right-click, select **Source** and then **Generate Getters and Setters**. Expand **staticNum**, check the **getStaticNum** check box and click **OK**.)

## In the **TestStatic** class, comment the first **printf()** statement in the **showHeadings()** method and uncomment the second **printf()** statement. Repeat in the **showNumbers()** method.

## Run the **TestStatic** again. When you increment the **staticNum** in object **s1**, does **staticNum** in **s2** or **s3** change? \_\_Yes\_\_\_\_\_\_\_\_\_\_\_\_

***Explanation:* num** is an instance variable, **staticNum** is a class variable. This means that there is a new copy of **num** created for each object that is instantiated, but there is only one copy of **staticNum** for all instantiated objects. Therefore, when you increment **num**, you are incrementing the copy of **num** in the active object and when you increment **staticNum** you are incrementing the single copy of **staticNum**.

## Create a mutator for **staticNum** called **setStaticNum()** in the **StaticVariable** class. Make it a **static** method:

public static void setStaticNum(int newStatic)

## Try adding the following statement to the **setStaticNum()** method:

num = newStatic;

What happens? \_\_\_Error\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

***Explanation:* num** is an instance variable, **setStaticNum** is a class method. A class method can only use static variables. It **cannot** use instance variables.

## Delete the line added in the previous step.

## In **TestStatic**, uncomment the second last set of displays in **main()**. Replace the comment to add a call to **setStaticNum()** tochange the value of **staticNum** to 15 for object **s1** with:

s1.setStaticNum(15);

## Run the program. What is the last value of **staticNum** for **s1**? 5

for **s2**? 5

for **s3**? 5

When you change a static variable for an object, what happens to it in the other objects of the same class?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Uncomment the last set of displays in **main()**. Replace the comment to add a call to **setStaticNum()** tochange the value of **staticNum** to 125 for the class to:

StaticVariable.setStaticNum(125);

## Run the program. What is the last value of **staticNum** for s1? 5\_\_\_ \_\_

for s2? 5

for s3? 5

## Try changing one of the calls to **incrementNumbers()** to:

StaticVariable.incrementNumbers();

and compile the program. What happens? Error

***Java Rule:*** A class method may be called using **either** the class or an object of the class. An instance method **must** be called using an **object** of the class.

## Change the line changed in the previous step to its previous contents.

***The Video Store System:***

The **Customer** and **AddCustomerFrame** classes that we looked at in class are in the **videoStoreSystem** package. Run **AddCustomerFrame** and test it on the test cases we developed in class. (See the appendix.) At this point, the customer number is correctly created and the customer data is not added to the customer.txt file. The data for the successful customers is displayed in the console display area.

Now we want to add the code to generate the customer number for a new customer automatically. The starting value (the next customer number) will be input once at the beginning of the program. Every time a new customer is added, its customer number will be set to the next new customer number and the next new customer number will be incremented.

## Add a class integer variable called **nextCustomerNumber** to the **Customer** class. This variable will contain the customer number to use for the next new customer. Because it needs to change every time a customer is added, we are using a **static** variable that has the same value for all instances of the class.

## Add the following class method to read the **nextCustomerNumber** class variable from the user:

public static void initializeNextCustomerNumber()

{

Scanner in = new Scanner(System.in);

System.out.print ("What is the next customer number?");

nextCustomerNumber = in.nextInt();

} // initializeNextCustomerNumber ()

## Add a call to **initializeNextCustomerNumber()** for the **Customer** class after the open brace ({) of the **AddCustomerFrame** class constructor.

## Run **AddCustomerFrame** to test the **initializeNextCustomerNumber ()** method. You should be prompted to enter the first customer number.

## In the **Customer** class, add a private void method called **setCustomerNumber()**. It should not have any parameters. In the body, set the **customerNumber** instance variable to **nextCustomerNumber** and then increment **nextCustomerNumber**.

## Replace the initialization of customer number with a call to **setCustomerNumber()** in each of the **Customer** constructors.

## Run **AddCustomerFrame** to test the **customerNumber** assignment. Set the next customer number to 1000 and run test cases 1, 6, 7, 8 and 9.

# Review Reading from a data file.

***Purpose:*** Review how to read data from a text file.

***To Do:***

## Now we want to read the next customer number from a data file instead of entering it each time we run the program. In the **Customer** class, modify the **initializeNextCustomerNumber ()** method as follows:

1. Add **throws Exception** to the end of the method header
2. Create a **File** object called **customerNumberFile** and pass it the filename “customerNumber.txt”.
3. Change the **Scanner** instantiation to use **customerNumberFile** instead of **System.in**.
4. Delete the prompt for customer number.
5. Add **in.close()** at the end of the method.

## What is the error in the **AddCustomerFrame**? \_The call to\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ initialiseNextCustomerNumber() has an error because the name of the method in the customer class has changed. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## To correct the error and to let the user know why the program has stopped, we need to add a try/catch:

### Before the call to **initializeNextCustomerNumber ()**:

try{

### After the call to **initializeNextCustomerNumber ()**:

} catch (FileNotFoundException e)

{

System.exit(-1);

}

catch (Exception e)

{

System.exit(-2);

}

### Add a **JOptionPane** error message titled "Missing customer number file" that states that the *customerNumber.txt* file could not be found before the **System.exit(-1)** statement in the first **catch** block.

### Add a **JOptionPane** error message titled "Error" that states that the an error was encounteredand display the Java message (**e.getMessage()**) before the **System.exit(-2)** statement in the second **catch** block.

## Run **AddCustomerFrame** to test your error message.

## Create a text file in your **project** folder called **customerNumber.txt**. It should contain only the number 1000.

## Run **AddCustomerFrame** to test reading the **customerNumber.txt** file. Use test cases 1, 6, 7, 8 and 9 to test it.

# Writing Output to a Text File

***Purpose:*** Learn to save records in an output data file.

***The Video Store System:***

We want to store the customer information in a text file called **customers.txt**. The data file contains the following fields:

| **Field** | **Description** | **Datatype** | **Delimiter** |
| --- | --- | --- | --- |
| customerNumber  (required) | A unique number that is assigned by the system to the particular customer | int | ~ |
| lastName (required) | The customer's last name | String | ~ |
| firstName (required) | The customer's first name | String | ~ |
| phoneNumber | The customer's phone number (must be 10 digits long) | String | ~ |
| outstandingBalance | The balance owing on late rentals | double | \r\n |

***To Do:***

## Create a new class called **CustomerFile** in the **videoStoreSystem** project.

## Add a static **write()** method to the **CustomerFile** class with the following header:

**public static void write(Customer customer) throws IOException**

This method is going to connect to the "customers.txt" file and add a customer record to it.

The four steps for writing to a data file are:

1. Instantiate a **File** object that defines the text file.

2.Instantiate a **FileWriter** object and connect it to the File object.

3. Use the **FileWriter** **write()** method to write a **String** containing all the fields you wish to include in the record. Repeat this for each record.

4. Close the data file.

### Instantiate a **File** object called **custFile** that connects to the "customers.txt" file in the same way that you create a **File** object when you are reading from a text file.

### Create a **FileWriter** object called **out** that connects to the **custFile** object in the same way that you connect a **File** object to a **Scanner** object.

### Code the **write()** statement to add the customer data in the format described above to the text file. Use the accessor methods in the **Customer** class to get the values of each field. Be sure to include a "\r\n" at the end of the string that you write.

### Close the **out** **FileWriter**.

### Add a new private method called **addCustomer ()** to **AddCustomerFrame**. Add the following try/catch block to the new method:

try

{

CustomerFile.write(customer);

JOptionPane.showMessageDialog(

this,

customer.getFirstName() + " " + customer.getLastName()

+ " has been added with customer number "

+ customer.getCustomerNumber(), "Customer Added",

JOptionPane.INFORMATION\_MESSAGE); }

catch (IOException e1)

{

JOptionPane.showMessageDialog(this, "ERROR: " + e1.getMessage(),

"I/O Error",

JOptionPane.ERROR\_MESSAGE);

}

## In the **actionPerformed()** method in **AddCustomerFrame** replace the **JOptionPane** message to display successful creation with a call to **addCustomer()**.

## Run **AddCustomerFrame** and add the customers from test cases 1, 6, 7, 8 and 9.

## Open "customers.txt". (You may have to refresh the Navigator (F5) to see it in Eclipse.) How many records are there? \_\_\_\_\_\_\_\_\_\_\_\_

## Delete "customers.txt" from disk.

# Appending Data to the End of a File

***Purpose:*** Learn to add data to the end of an existing file.

***To Do:***

The **FileWriter** constructors are:

**FileWriter(File**file**)** Constructs a **FileWriter** object given a **File** object.

**FileWriter(FileDescriptor fd)** Constructs a **FileWriter** object associated with a file descriptor.

**FileWriter(String**fileName**)** Constructs a **FileWriter** object given a file name.

**FileWriter(File**file**, boolean**append**)** Constructs a **FileWriter** object given a file name with a boolean indicating whether or not to append the data written.

## We have used the first **FileWriter** constructor in programs for this lab. The fourth constructor contains a second parameter called **append**. If the value of this parameter is **true**, data is appended to the end of the file instead of replacing the contents of an existing file. If the file does not already exist, it will still create it. Add a second argument to the call to the **FileWriter** constructor in the **write()** method of the **CustomerFile** class to indicate that you want to append to the "customers.txt" file.

## Run the program again using test cases 1, 6, 7, 8 and 9 and open the "customers.txt" file. Is the data for all the customers included? \_\_\_\_\_

# Using a WindowListener

***Purpose:*** Learn how to use a WindowListener to execute clean-up activities when a window closes.

***The Video Store System:***

We want to rewrite the **customerNumber.txt** file with the latest **nextCustomerNumber** when the **AddCustomerFrame** closes.

***To Do:***

## Add a **static rewriteNextCustomerNumber()** method to the **Customer** class. It should:

### Throw **IOException**.

### Create a **File** object called **customerNumberFile** that connects to the "customerNumber.txt" file.

### Instantiate a **FileWriter** object called **out** that connects to the **customerNumberFile** object.

### Write **nextCustomerNumber** to "customerNumber.txt" using the **out FileWriter**.

### Close the **out FileWriter**.

Now we need to tell the **AddCustomerFrame** to call the **rewriteNextCustomerNumber ()** method when the window closes. To do this we're going to use the **WindowListener** interface. The **WindowListener** interface operates in a manner similar to the **ActionListener** interface. While **ActionListener** only has one method that must be implemented (the **actionPerformed()** method), there are seven methods that must be implemented with a **WindowListener** interface:

|  |  |  |
| --- | --- | --- |
| ListenerInterface | Event Generated | Listener Methods |
| ActionListener | ActionEvent | actionPerformed() |
| WindowListener | WindowEvent | windowActivated() |
|  |  | windowClosed() |
|  |  | windowClosing() |
|  |  | windowDeactivated() |
|  |  | windowIconified() |
|  |  | windowOpened() |

## Add the **WindowListener** after **ActionListener** on the **AddCustomerFrame** class header. (Separate the two listeners with a comma.) Click the error icon in the left margin and select **Import WindowListener (java.awt.event)** to add the import statement that is required. Click the error icon again and select **Add** **unimplemented methods**. This will automatically add stub methods for all the required methods to the end of the class.

## We want to add our code in the **windowClosing()** method. Go to the **windowClosing()** method and add a call to the **rewriteNextCustomerNumber ()** method for the **Customer** class in a try catch block. Catch **IOException** and display the I/O message (**e.getMessage()** ina **JOptionPane**.)

## Test your change by running **AddCustomerFrame**. Open "customerNumber.txt". Has it changed? \_\_\_\_\_\_\_\_\_\_\_\_\_

This is because the frame wasn't "listening" for the close.

## Add

this.addWindowListener(this);

at the end of the constructor.

## Test your change again by adding two customers.

## Run **AddCustomerFrame** again. What happens? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Open "customerNumber.txt". What does it contain? \_\_\_\_\_\_\_\_\_\_\_\_\_

***Explanation:* nextCustomerNumber** is an integer. The **write()** method writes an **int** with its binary value, rather than its ASCII value. For the number to be stored as an ASCII, we must convert it to a **String**. Use the **String.valueOf()** method to do this.

## Change the write statement in **rewriteNextCustomerNumber ()** to:

out.write(String.valueOf(nextCustomerNumber));

## Replace the data in "customerNumber.txt" with 1000.

## Delete "customers.txt" and run **AddCustomerFrame** for test cases 1, 6 and 7.

## Open "customerNumber.txt". What does it contain now? \_\_\_\_\_\_\_\_\_\_\_\_\_

## Run **AddCustomerFrame** again for test cases 8 and 9. What is the first number assigned this time? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Don’t forget!**

Zip your ***username*\_B20\_L02\_Project** folder and upload it to **Moodle**

# Homework

## Complete the **Week 2 Quiz** on Moodle by Jan. 31.