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

# *Lab 13 – Java Exceptions*

Date assigned: Wednesday, April 20, 2016

Date due: **Wednesday, April 20, 2016**

**Learning Objectives**

At the end of this lab, the student will be able to:

1. explain Java's exception handling mechanisms
2. use the Java try/catch statement
3. design effective exception handlers
4. create an exception
5. appreciate the importance that exception handling plays in program design

**Commands, Methods and Structures Used:**

***Throw an exception:***

**throw new** *ExceptionName***();**

**throw new** *ExceptionName***(***message***);**

***The try/catch/finally block:***

**try**

**{**

// Block of statements - at least one of which may throw an exception

**}**

**catch (***ExceptionName e***)**

**{**

// Block of statements to be executed if the *ExceptionName* exception is thrown

**}**

**catch (***ExceptionName2 e***)**

**{**

// Block of statements to be executed if the *ExceptionName2* exception is thrown

**}**

**. . .**

**finally**

**{**

// Block of statements that is executed whether or not an exception is thrown

**}**

**To Be Handed In:**

1. The ***username*\_B20\_L13\_Project** folder should be zipped and uploaded to **Moodle**.

**To Start:**

1. Download and unzip the B20\_L13\_Project folder in your **420-B20\Labs** folder. Rename it to ***username*\_B20\_L13\_Project**.
2. Start **Eclipse**.
3. Create a new Java Project called ***username*\_B20\_L13\_Project**.

# Java's Exception Handling Mechanisms

***Purpose:*** Learn to use Java's exception handling mechanisms to catch run time errors and correct them and continue or exit gracefully.

***To Do:***

## Create a new class called **IntField** inthe **exceptions** package. It should be a subclass of **JTextField.** (Type **javax.swing.JTextField** in the **Superclass** field of the **New Java Class** window.) Check the *Constructors from superclass* check box. It should not contain a **main()** method.

## Add a **getInt()** method. It should have no parameters and should return an **int**. In the body create a local integer variable called **num** and assign it to the value returned by **getText()** converted to an integer. The method should return **num**.

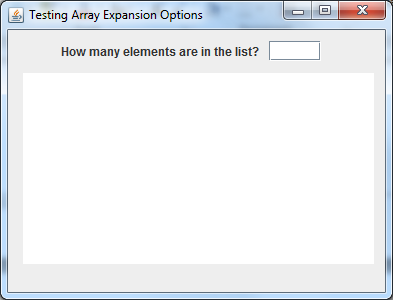
## Use **WindowBuilder** to create a Java Swing frame called **ExceptionTesterFrame** inthe **exceptions** package. The frame should be titled "Testing Array Expansion Options".

## Add the following components to **ExceptionTesterFrame**.

* a **JLabel** called **lblPrompt** with the text "How many elements are in the list?"
* an **JTextField** called **fldArraySize**
* a **JTextArea** called **areaMessage**. It should not be editable.
* a **JLabel** called **lblStatusBar**

## The **lblStatusBar** label is across the length of the bottom of the frame. It is not visible because at the moment it does not contain any text.

## Run the application. Your frame should look similar to:



## In the frame source, change **fldArraySize** from a **JTextField** to an **IntField**. (You will have to change the declaration and the instantiation.)

## Add an **actionListener** for **fldArraySize**. The **actionPerformed()** method should:

### Use the **getInt()** method to get the integer entered in the **fldArraySize** **IntField** and store it in an integer **n**.

### Display "*n* integers were inserted" in the **lblStatusBar**. (n is the integer entered in the **fldArraySize IntField**.)

## Run the application and test it by entering an integer, a real number and a word in the **IntField**. What exception is generated when you don't enter an integer in the field?

\_\_\_\_NumberFormatException\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

What method is executing when the exception occurs? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Instead of letting Java handle the exception we want to handle it ourselves. Recall that when you read from a data file, you learned to use a try-catch block to customize your error message when a **FileNotFoundException** was generated. Now we are going to do the same thing for a **NumberFormatException**.

## Modify the **getInt()** method in **IntField** to throw **NumberFormatException**. (i.e. add **throws NumberFormatException** to the method header for **getInt()** ).

## Add a **try**/**catch** block around the statements in your **actionPerformed()** method. In the catch block clear the **fldArraySize** **IntField** and display a **JOptionPane** error with the message "*You must enter an integer. Please re-enter*".

## Run the application again and try entering an integer, a real number and a word in **fldArraySize**. What happens this time?

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

Now we are going to compare the performance of creating a static array that is large enough to hold all the values and creating a dynamic array that increases in size each time a new element is added.

## Add a private integer array instance variable called **list** to **ExceptionTesterFrame**.

## Add a private void method called **testBigEnoughArray(int n)** to create to an array with the specified size:

### Add a local **long** variable called **startTime** that is initialized to the current System time:

long startTime = System.currentTimeMillis();

### Instantiate the instance variable array **list** with **n** elements

### Add a **for** loop that loops n times and adds an integer element to the **list** arrayeach time through. The value of the integer should be the loop counter.

### Add a local **long** variable called **endTime** that is set to the current System time (see a above.)

### Add a message to the end of the message text area that gives the time in milliseconds that it took to create the array. (The time in milliseconds to create the array is **endTime – startTime**)

## Add a call to **testBigEnoughArray(n)** in the **actionPerformed()** method after initializing **n**.

## Run the program and try creating an array that is large enough to be measurable.

Now we want to add a method that starts with an array of 0 elements and increases the array size by 1 each time a new element is added.

## In order to increase the size of an array, we are first going to write a private void method called **insertAt(int n)** :

### create a **try/catch** block to catch an **ArrayIndexOutOfBoundsException**

### in the **try** block assign **n** to **list[n]**.

### in the **catch** block :

#### create an new **int** array called **newList** that is one element longer than the length of **list**;

#### add a for loop to loop through all the elements in **list** and copy them to the corresponding indexes in **newList**

#### add the value **n** to the end of **newList**

#### replace **list** with **newList** (i.e. **list = newList**)

## Replace the assignment statement in the for loop to a call to **insertAt()** in **testBigEnoughArray()**. The parameter is the **for** loop counter.

## Code the **testArray(int n)** method:

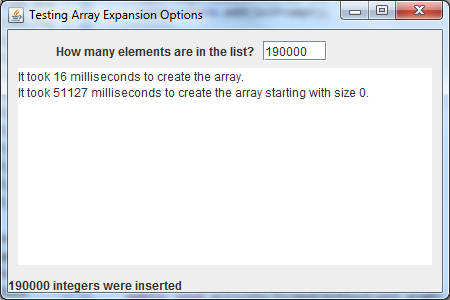
### Copy the code from **testBigEnoughArray()**.

### Change the initial array size to **0**.

### Change the message to state the time required to create an array starting with size 0.

## Add a call to **testArray(n)** in the **actionPerformed()** method after the call to **testBigEnoughArray(n)**.

## Run the program with different numbers until you find a collection size that is large enough to be measurable. (You will have to wait a bit in order for a large enough time to elapse.) Your frame should look similar to:



# Creating an Exception

***Purpose:*** Learn to use create a user-defined exception for a specialized error.

***To Do:***

## When you ran the frame in the previous question, you may have experienced a heap out of space exception with a very large array. We want to limit the size that our collection can be in order to prevent this from happening. First we will create a new **Exception** to handle this error:

### Create a subclass of **Exception** called **IntOutOfRangeException**.

### Add a constructor with a single integer parameter called **bound**. It should call the superclass constructor with the String parameter: "*The input value exceeds the bound " + bound*.

## Now we will modify the **IntField** class to throw the exception if necessary:

### add a private integer instance variable call **bound**. Initialize it to **Integer.MAX\_VALUE**.

### Add a constructor with two integer parameters: **size** and **max**. It should call **super(size)** and set **bound** to **max**.

### Add a **set()** and **get()** method for **bound**.

### In the **getInt()** method:

#### add **IntOutOfRangeException** to the method header after **NumberFormatException** (Separate them with a comma.)

#### After the num assignment statement: add code to throw an **IntOutOfRangeException** if **num** > **bound**. To throw an exception, you code throw followed by the exception instantiation:

**throw new IntOutOfRangeException(bound);**

## Inthe constructoro f **ExceptionTesterFrame**, use the **setBound()** method to set the upper bound of **fldArraySize** to100,000.

## Compile **ExceptionTesterFrame**. What happens?

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

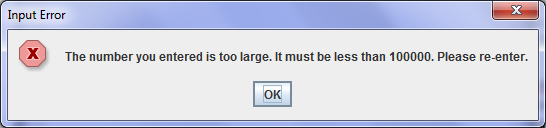
Since **IntOutOfRangeException** is a subclass of **Exception**, you must throw it or handle it. First we will see what happens when we throw it.

## Add **throws** **IntOutOfRangeException** to the **actionPerformed()** method header. Compile it. What happens?

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

Since the **actionPerformed()** method does not throw any exceptions, we cannot throw exceptions when we are overriding it.

## Let's handle the exception with a customized message. In the **actionPerformed()** method, add a second **catch** clause to catch **IntOutOfRangeException**. Use a **JOptionPane** to display an appropriate error message if the exception is caught. Your error message should look similar to:



## Run **ExceptionTester**. Enter 80,000. Enter 150,000. What error is generated?

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

Where did the error message come from?

# Homework

## Complete the **Week 13 Quiz** on Moodle by April 24.