LAB 7 – Exception Handling

Objectives:

At the end of this lab, the students are able to:

i. Demonstrate the basic concepts of exception handling.
ii. Implementing runtime exception.
iii. Implementing exception using throw statement.
iv. Using Throwable class to retrieve detail information about the exception.

7.1 Activity 1

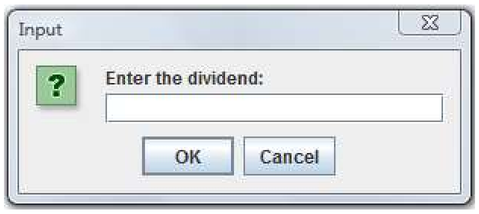
7.1.1 Objective

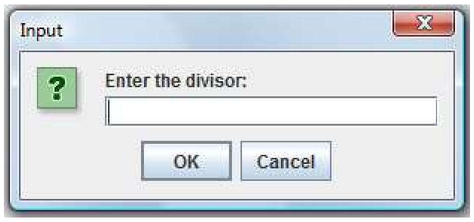
To demonstrate the basic concepts of exception handling.

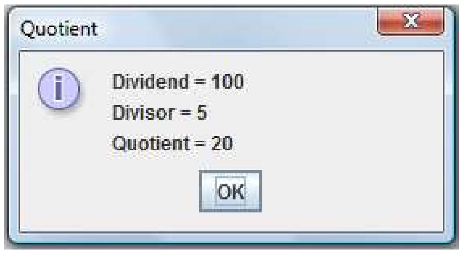
7.1.2 Problem Description

Write a program called Division.java and DivisionTest.java to accomplish the followings:

- Accept two input values from end users for dividend and divisor.
- Calculate the quotient.
- Display the quotient as well as the input values.
- Compile and run the program.
- Run the program by entering value 0 for the divisor. Explain the output.
- Run the program again using values other than integer. Explain the output.







```
Dividend = 100
Divisor = 5
Quotient = 20
```

[Estimated Time: 20 minutes]

7.2 Activity 2

7.2.1 Objective

Implement a runtime exception.

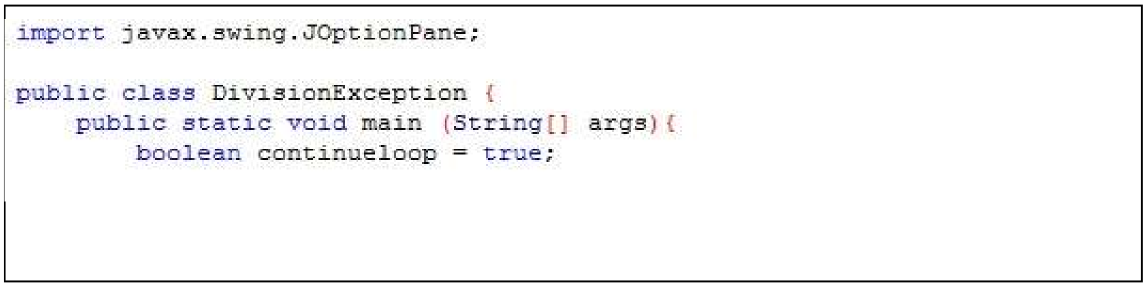
7.2.2 Problem Description

Based on the class you created in Activity 1, catch and handle the two exceptions you found in Activity 1 using the try-catch blocks.

7.2.3 Solution Activity 2

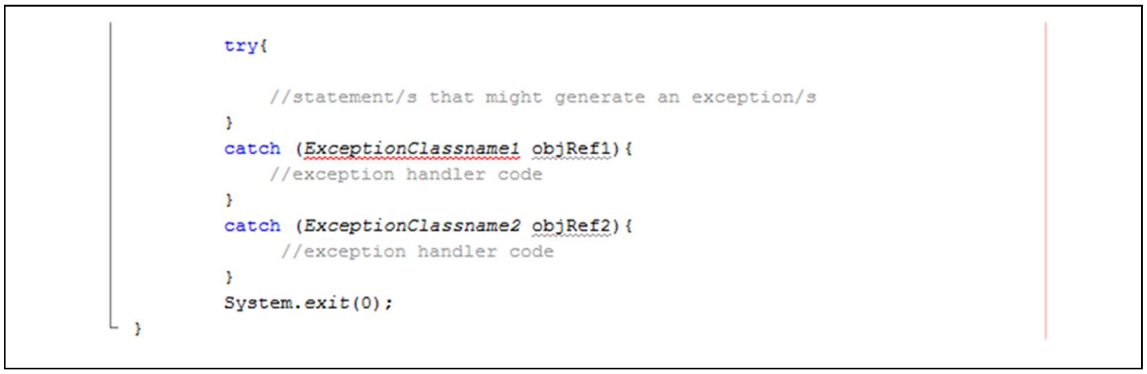
<u>Algorithms</u>

Step 1: Go to CSF3043 folder and create sub-folder called Lab – Chapter 8.
Step 2: Go to CSF3043 –> Lab – Chapter 8 folder and create Sub-working folder called Activity 2.
Step 3: Open IDE NetBeans or JCreator.
Step 4: Create new file/class called DivisionWithException.java
Step 5: Import javax.swing.JOptionPane.
Step 6: Define the class DivisionException() and instance variables as below:

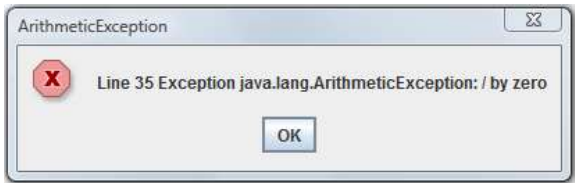


```
import javax.swing.JOptionPane;

public class DivisionException {
    public static void main (String[] args){
        boolean continueloop = true;
```

Step 7: Place the statement/s that might generate an exception/s in the try block, followed by a number of catch blocks to handle the exception/s. A sample run is shown below which returns the line containing the statement that catch the exception and the exception class.



```
        try{

            //statement/s that might generate an exception/s
        }
        catch (ExceptionClassname1 objRef1){
            //exception handler code
        }
        catch (ExceptionClassname2 objRef2){
             //exception handler code
        }
        System.exit(0);
}
```


ArithmeticException

Line 35 Exception java.lang.ArithmeticException: / by zero

Step 8: Compile the DivisionException.java.

[Estimated Time: 40 Minutes]

7.3 Activity 3

7.3.1 Objective

Implement an exception using throw statement.

7.3.2 Problem Description

- Create your own division by zero exception class by extending the Exception class.
- Modify your program in Activity 2 to use the newly created exception class. You may need to add the following codes:

```
try  {
        //statements that might generates an exception
        if (divisor == 0) throw new MyDivisionByZeroException();
        //other statements
}

catch (MyDivisionByZeroException mdbzRef){
        //exception handler codes
}
catch (ExceptionClassName  objRef){
        //exception handler codes
```

[Estimated Time: 30 Minutes]

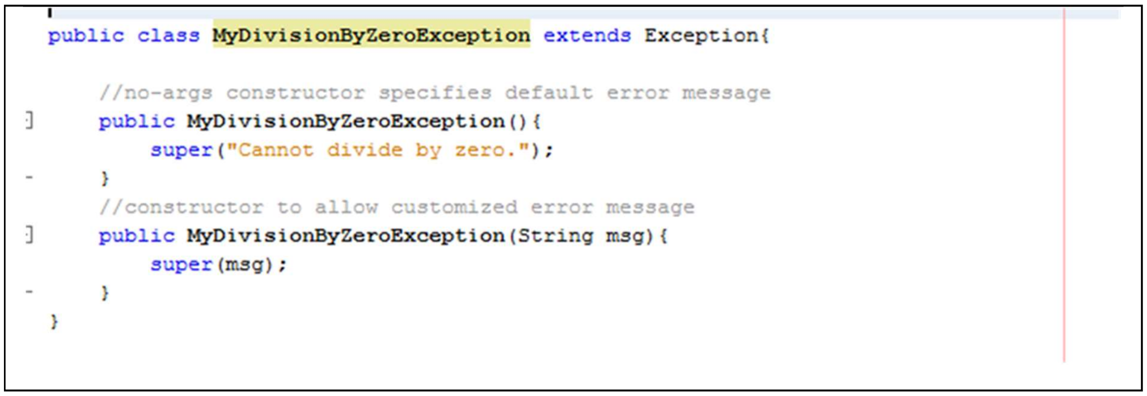
Lab Exercise

Objective

Implement an exception using throws statement.

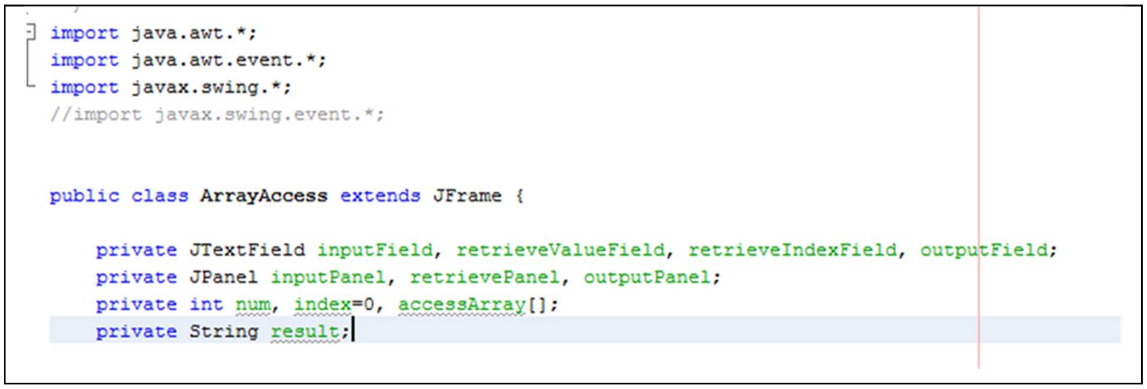
Problem Description

- Write an application that will allow a user to input integer values into an array of size 10. The program should allow for retrieving values from the array by index or by specifying a value greater than 0 to search for in the array. The application should handle any exceptions that might arise when inputting values into the array or accessing array elements. The application should also make use of the exception class NumberNotFoundException, defined below. If an attempt is made to access an element outside the array bounds, catch the ArrayIndexOutOfBounds exception and display an appropriate error message.

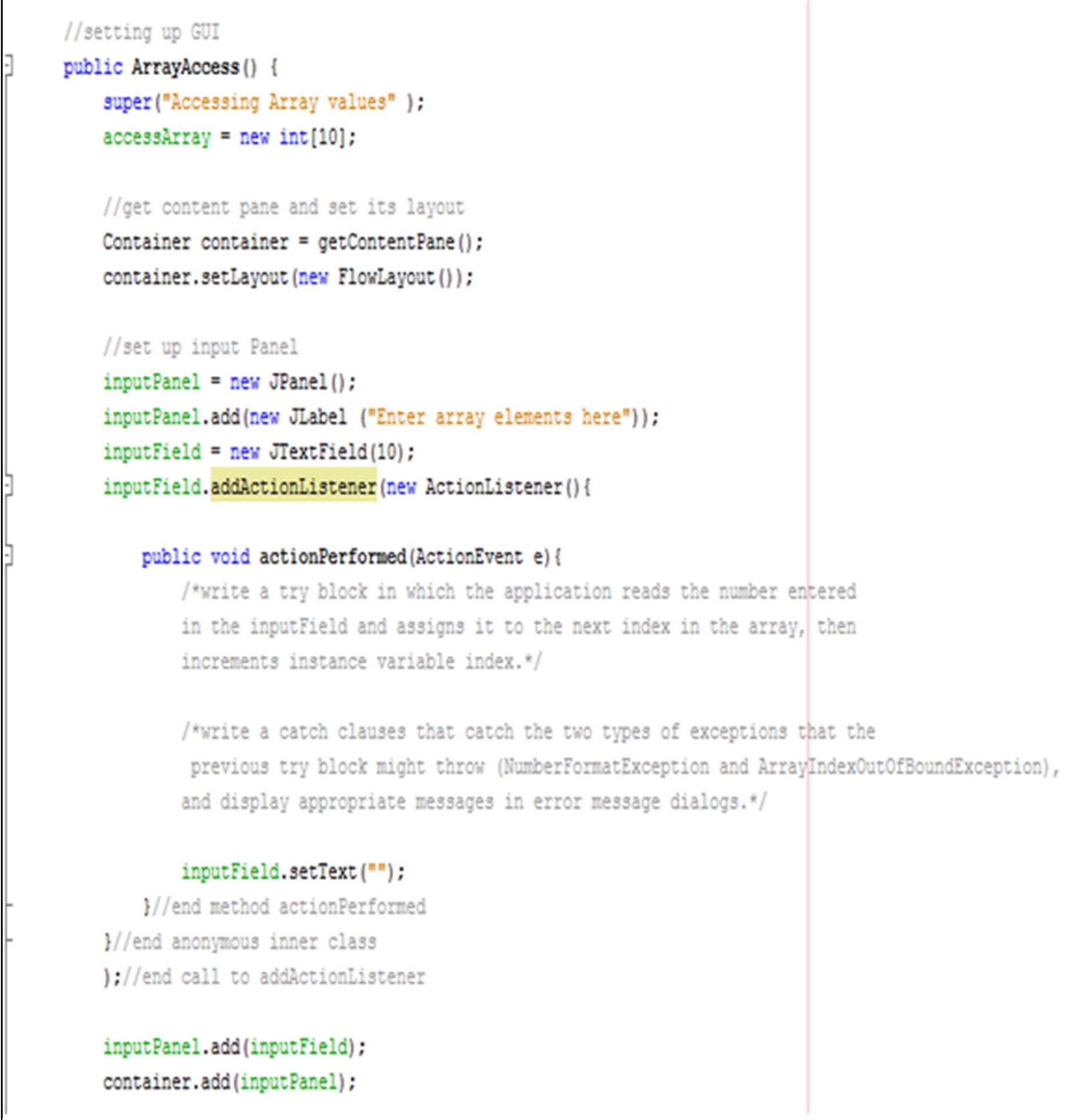


```java
public class MyDivisionByZeroException extends Exception{

    //no-args constructor specifies default error message
    public MyDivisionByZeroException(){
        super("Cannot divide by zero.");
    }
    //constructor to allow customized error message
    public MyDivisionByZeroException(String msg){
        super(msg);
    }
}
```

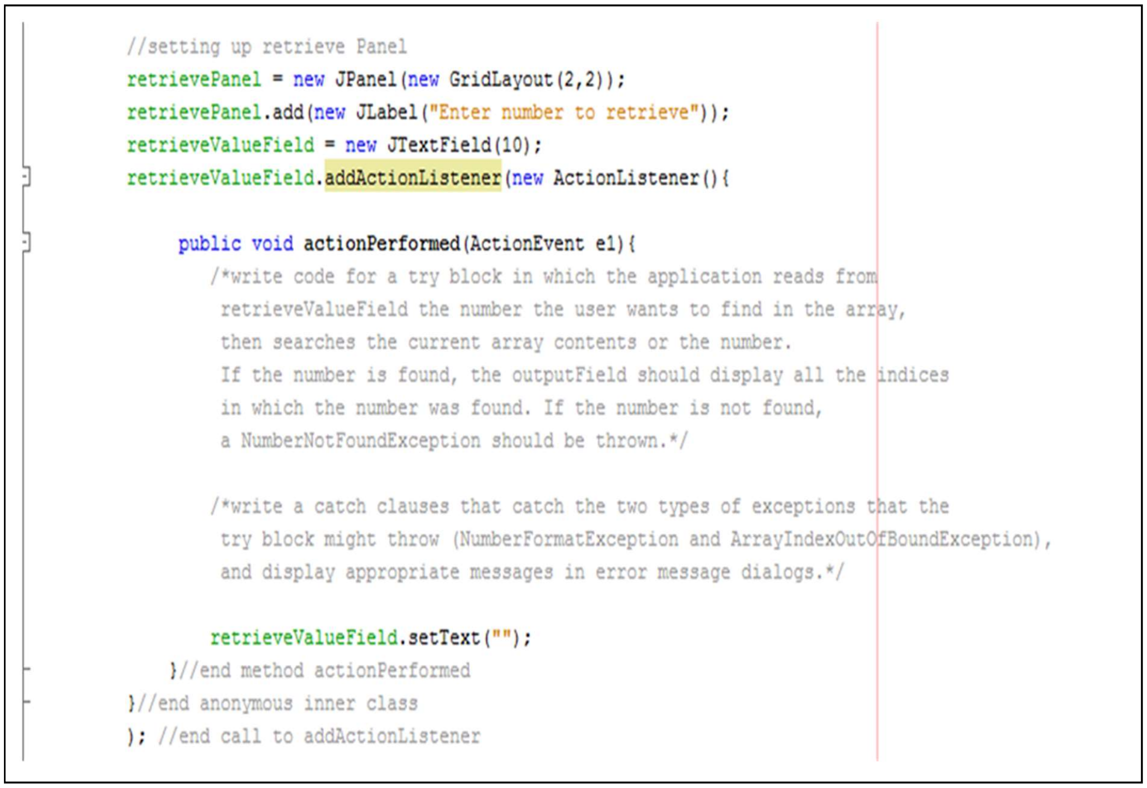
- Modify the program by creating another exception class called DuplicateValueException for detecting whether the user inputs a duplicate number. A DuplicateValueException should be thrown if the user inputs a value that already resides in the array. Also, display an appropriate error message. The program should continue normal execution after handling the exception.



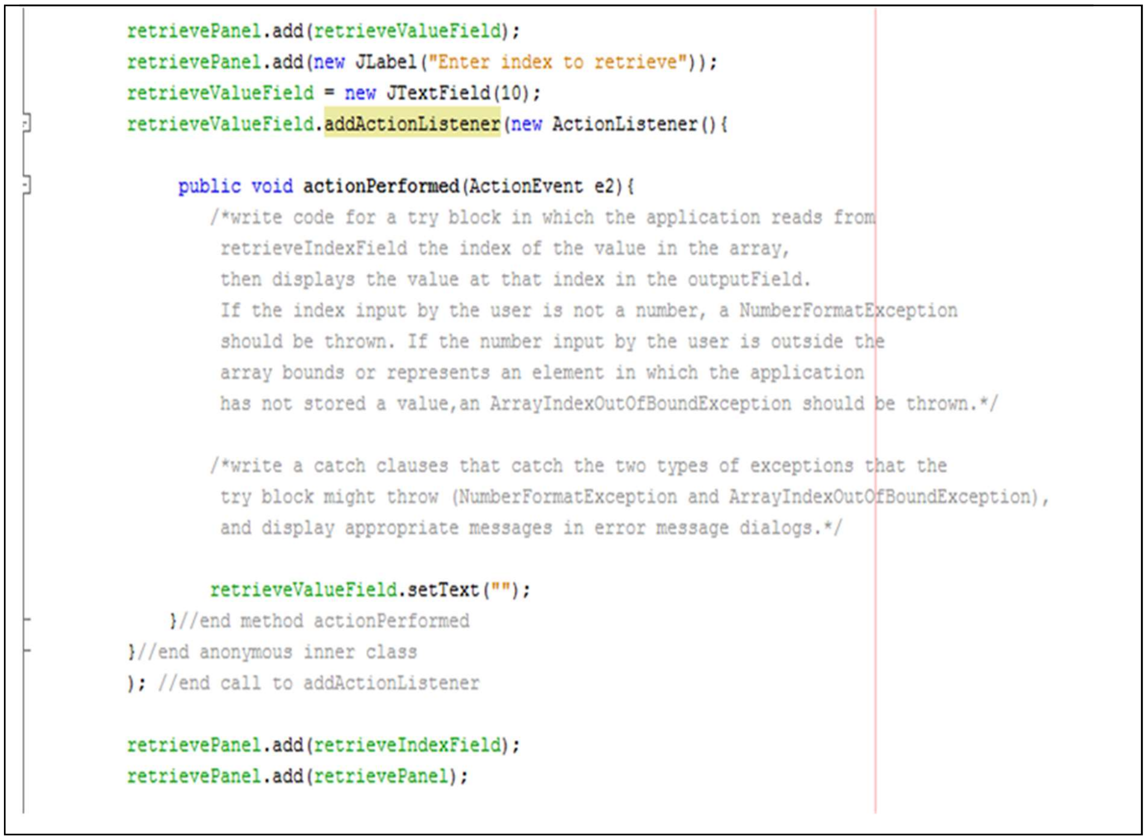
```java
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
//import javax.swing.event.*;


public class ArrayAccess extends JFrame {

    private JTextField inputField, retrieveValueField, retrieveIndexField, outputField;
    private JPanel inputPanel, retrievePanel, outputPanel;
    private int num, index=0, accessArray[];
    private String result;
```



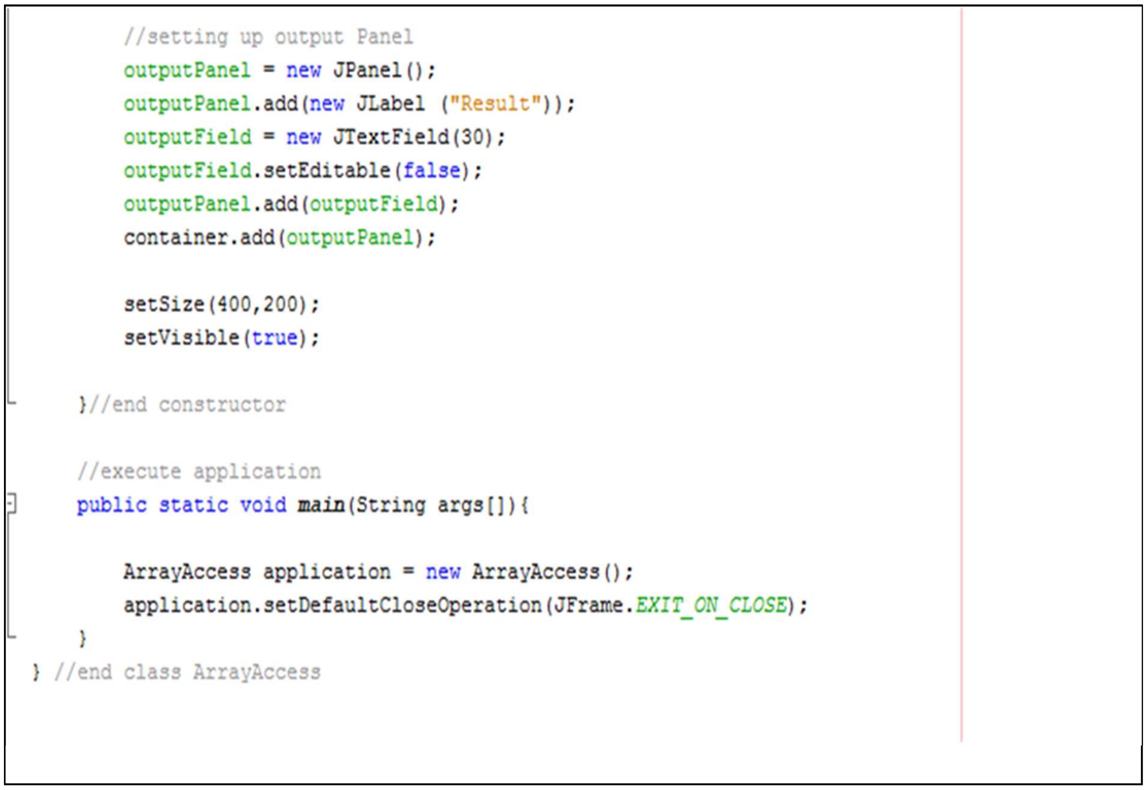
```
//setting up GUI
public ArrayAccess() {
    super("Accessing Array values" );
    accessArray = new int[10];

    //get content pane and set its layout
    Container container = getContentPane();
    container.setLayout(new FlowLayout());

    //set up input Panel
    inputPanel = new JPanel();
    inputPanel.add(new JLabel ("Enter array elements here"));
    inputField = new JTextField(10);
    inputField.addActionListener(new ActionListener(){

        public void actionPerformed(ActionEvent e){
            /*write a try block in which the application reads the number entered
            in the inputField and assigns it to the next index in the array, then
            increments instance variable index.*/

            /*write a catch clauses that catch the two types of exceptions that the
             previous try block might throw (NumberFormatException and ArrayIndexOutOfBoundException),
            and display appropriate messages in error message dialogs.*/

            inputField.setText("");
        }//end method actionPerformed
    }//end anonymous inner class
    );//end call to addActionListener

    inputPanel.add(inputField);
    container.add(inputPanel);
```



```
//setting up retrieve Panel
retrievePanel = new JPanel(new GridLayout(2,2));
retrievePanel.add(new JLabel("Enter number to retrieve"));
retrieveValueField = new JTextField(10);
retrieveValueField.addActionListener(new ActionListener(){

    public void actionPerformed(ActionEvent e1){
      /*write code for a try block in which the application reads from
       retrieveValueField the number the user wants to find in the array,
       then searches the current array contents or the number.
       If the number is found, the outputField should display all the indices
       in which the number was found. If the number is not found,
       a NumberNotFoundException should be thrown.*/

      /*write a catch clauses that catch the two types of exceptions that the
       try block might throw (NumberFormatException and ArrayIndexOutOfBoundException),
       and display appropriate messages in error message dialogs.*/

      retrieveValueField.setText("");
    }//end method actionPerformed
}//end anonymous inner class
); //end call to addActionListener
```



```
retrievePanel.add(retrieveValueField);
retrievePanel.add(new JLabel("Enter index to retrieve"));
retrieveValueField = new JTextField(10);
retrieveValueField.addActionListener(new ActionListener(){

    public void actionPerformed(ActionEvent e2){
      /*write code for a try block in which the application reads from
       retrieveIndexField the index of the value in the array,
       then displays the value at that index in the outputField.
       If the index input by the user is not a number, a NumberFormatException
       should be thrown. If the number input by the user is outside the
       array bounds or represents an element in which the application
       has not stored a value,an ArrayIndexOutOfBoundException should be thrown.*/

      /*write a catch clauses that catch the two types of exceptions that the
       try block might throw (NumberFormatException and ArrayIndexOutOfBoundException),
       and display appropriate messages in error message dialogs.*/

      retrieveValueField.setText("");
    }//end method actionPerformed
}//end anonymous inner class
); //end call to addActionListener

retrievePanel.add(retrieveIndexField);
retrievePanel.add(retrievePanel);
```



```
        //setting up output Panel
        outputPanel = new JPanel();
        outputPanel.add(new JLabel ("Result"));
        outputField = new JTextField(30);
        outputField.setEditable(false);
        outputPanel.add(outputField);
        container.add(outputPanel);

        setSize(400,200);
        setVisible(true);

    }//end constructor

    //execute application
    public static void main(String args[]){

        ArrayAccess application = new ArrayAccess();
        application.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
} //end class ArrayAccess
```

[Estimated Time: 90 Minutes]