Lecture Eleven – Exception Handling

- Introduction
- Compiler Errors
- SyntaxErrors program
- Runtime Errors
- Exceptions
- SimpleCalc3 program
- Catching Exceptions
- Finally
- Throwing Exceptions
- Exception Responsibility
- Drill

Read Chapter 9
Introduction

- So far we have written a lot of programs in this class. However, chances are that you have probably written code that didn’t compile on your first attempt. Similarly you may have discovered that your program is not free of bugs even if it does compile. For example, on your calculator programs what happens if the user doesn’t enter a number into the text fields?

- Being human programmers we are bound to make mistakes. This lesson will deal with the ways we can discover and handle our errors.
The first type of errors that we may encounter when writing code are called **syntax errors**.

**Syntax errors** occur when we violate the syntax standards of a programming language.

These errors are typically found by the compiler and so they may also be called **compiler errors**.

**Syntax errors** are typically the easiest to spot and correct because we can use the Java compiler to help us figure out where our syntax problems occur.
SyntaxErrors program

- The next program, `SyntaxErrors.java`, is a short program that is simply supposed to print the following message:
  
  2
  4
  6
  8

  Who do we appreciate?
  We appreciate Java

- However, the program is littered with syntax errors. Type the following program exactly as is appears. You may notice some of the syntax errors just when typing in the code. Do not fix these initially. If it helps, we will allow you to copy and paste this program just this once!
// SyntaxErrors.java
// There are 16 syntax errors in this program
public class SyntaxError
{
    public static void main (String args[])
    {
        int[] even = {"2", "4", "6", "8"};
        for (int i = 0, j < 4, i++)
            System.out.println(even(i));
        System.out.println("Who do we appreciate?");
        weAppreciate(Java);
    }

    public void weAppreciate(String input)
    {
        System.out.println("We appreciate + input");
    }
}
SyntaxErrors program

- If you attempt to compile this program you will likely receive the following message from the Java compiler:
  
  SyntaxErrors.java:5: <identifier> expected
  public static void main (String args[])
  ^
  SyntaxErrors.java:16: unclosed string literal
  System.out.println("We appreciate + input);
  ^
  SyntaxErrors.java:18: ';' expected
  ^
  3 errors

- Even though our code contains 16 syntax errors, only 3 are caught initially by the compiler. Others could potentially be found once these errors are fixed.
When fixing bugs it is best to start with the first error and make your way down the list (because sometimes your initial error may be the cause of your other errors).

This is the typical format for a Java compiler error:

```
filename : line #: error message
  erroneous line of code
```

Let’s look at the first error we received:

```
SyntaxErrors.java:5: <identifier> expected
 public static void main (String args[])
  ^
```

If we look on line 5 of our code, it looks like we misspelled the identifier “static”
SyntaxErrors program

- For the most part the error messages given by the Java compiler will typically help you fix your syntax errors.

- However, there are times when the error message does not always make sense. This is when Google becomes a programmer's good friend!

- There are still 15 other syntax errors in our program, but we will leave that as a drill for you at the end of the lesson
Let’s now turn our attention to the other type of errors: **runtime errors**

- Sometimes our code will compile but it may encounter some error at **runtime** (i.e. when the program is executing)

- For example, a user may be prompted to enter an integer but may enter a String, or a number may be divided by zero.

- Java handles **runtime errors** through the use of **exceptions** which are said to be “thrown” by methods
Exceptions

- There are literally hundreds of different exceptions that can be thrown when writing Java programs.
- The exceptions that are the most common (and the ones you will likely see in this class) are:
  - Exception – the generic exception that all other exceptions are extended from
  - NumberFormatException – an exception that occurs when the program encounters a String or other non-numeric value when it expects a numeric value
  - ArrayIndexOutOfBoundsException – occurs when you attempt to access part of an array that does not exist. For example, if you have an array of size 5 and try to access the element at index 7 (which does not exist)
  - ArithmeticException – occurs when there is an arithmetic error (typically a division by zero)
  - NullPointerException – occurs when you attempt to access an object that has not been created or was defined to be null
  - IllegalArgumentException – occurs when a method was called with an inappropriate argument
SimpleCalc3 program

- We will now re-visit our calculator program once again – as always type it in and get it working!

- For this version, we will allow addition, subtraction, multiplication, and division

- We also want to handle any exceptions that may be thrown

- Some problems we foresee occurring with our calculator would be
  - The user does not input a number into the text fields
  - Division by zero
  - Any other unforeseen error
SimpleCalc3 program

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

public class SimpleCalc3 extends JFrame implements ActionListener, WindowListener {
    private JLabel enterALabel;
    private JTextField AField;
    private JLabel enterBLabel;
    private JTextField BField;
    private JLabel resultLabel;
    private JTextField resultField;
    private JButton addButton;
    private JButton subButton;
    private JButton mulButton;
    private JButton divButton;
```
public static void main(String args[]) {
    JFrame frame = new SimpleCalc3();
    frame.setSize(250, 150);
    frame.setTitle("Simple Calculator 3");
    frame.setVisible(true);
}

public SimpleCalc3() {
    enterALabel = new JLabel("Enter first value: a = ");
    AField = new JTextField("0", 6);
    enterBLabel = new JLabel("Enter second value: b = ");
    BField = new JTextField("0", 6);
    resultLabel = new JLabel("The Result is = ");
    resultField = new JTextField("0", 6);
SimpleCalc3 program

addButton = new JButton("Add");
subButton = new JButton("Sub");
mulButton = new JButton("Mul");
divButton = new JButton("Div");

FlowLayout layout = new FlowLayout();
Container window = getContentPane();
window.setLayout(layout);

window.add(enterALabel);
window.add(AField);
window.add(enterBLabel);
window.add(BField);
window.add(resultLabel);
window.add(resultField);
SimpleCalc3 program

```java
window.add(addButton);
window.add(subButton);
window.add(mulButton);
window.add(divButton);

addButton.addActionListener(this);
subButton.addActionListener(this);
mulButton.addActionListener(this);
divButton.addActionListener(this);
addWindowListener(this);
}

public void actionPerformed(ActionEvent e)
{
    String tempString;
    int a, b, result = 0;
```
```java
SimpleCalc3 program

try {
    tempString = AField.getText().trim();
a = (new Integer(tempString)).intValue();
tempString = BField.getText().trim();
b = (new Integer(tempString)).intValue();

    if (e.getSource() == addButton)
        result = add(a, b);
    else if (e.getSource() == subButton)
        result = sub(a, b);
    else if (e.getSource() == mulButton)
        result = mul(a, b);
    else if (e.getSource() == divButton)
        result = div(a, b);

    resultField.setText(tempString.valueOf(result));
}
```
catch (NumberFormatException e1)
{
    JOptionPane.showMessageDialog(null, "You did not enter a valid number!");
}
catch (Exception e1)
{
    JOptionPane.showMessageDialog(null, e1.getMessage());
}
finally
{
    JOptionPane.showMessageDialog(null, "You pressed a button!");
}
SimpleCalc3 program

public void windowClosing(WindowEvent e)
{
    System.exit(0);
}

public void windowActivated(WindowEvent e){}

public void windowClosed(WindowEvent e){}

public void windowDeactivated(WindowEvent e){}

public void windowIconified(WindowEvent e){}

public void windowDeiconified(WindowEvent e){}

public void windowOpened(WindowEvent e){}
public static int add(int num1, int num2) {
    return num1 + num2;
}
public static int sub(int num1, int num2) {
    return num1 - num2;
}
public static int div(int num1, int num2) throws IllegalArgumentException {
    if (num2 == 0)
        throw new IllegalArgumentException("Cannot divide by zero!");
    return num1 / num2;
}
public static int mul(int num1, int num2) {
    return num1 * num2;
}
SimpleCalc3 program

- Most of this program should be familiar. The main changes occurred in our `actionPerformed()` and `div()` methods so we will focus primarily on them.

- Let’s start by looking at the changes we made to our `actionPerformed()` method
Catching Exceptions

- Before we analyze our new calculator, go back and re-run an old calculator program (like SimpleCalc2) and enter the letter “a” in the first value textbox and hit the “Add” button. You will probably see something like this printed in the console:

  ```
  Exception in thread "AWT-EventQueue-0" java.lang.NumberFormatException: For input string: "a"
  at java.lang.NumberFormatException.forInputString(NumberFormatException.java:48)
  at java.lang.Integer.parseInt(Integer.java:447)
  at java.lang.Integer.<init>(Integer.java:620)
  at SimpleCalc2.actionPerformed(SimpleCalc2.java:60)
  ... 
  ```

- This is essentially telling us that a `NumberFormatException` was thrown in the `parseInt()` method called in our `actionPerformed()` method
Catching Exceptions

- This exception was printed to our console screen because in was thrown but not caught.

- In order to prevent this we must catch the exceptions using a try/catch block.

- This first step is to surround the code that may throw an exception in a try block.

- Since we know that we may get an exception in our actionPerformed() method, we surround the code in that method with a try block. Note that it does not necessarily hurt to contain code that may not cause an exception in our try block.
Catching Exceptions

- The next step is to catch the various exceptions that may occur.

- Each catch block contains a parameter in parenthesis that specifies the type of exception we want to catch.
  
  ```java
catch (NumberFormatException e1) { . . . }
  ```

- A try block can be followed by as many catch blocks that you want. The code in the catch blocks will be executed only when an exception is thrown.

- In our example we attempt to catch a NumberFormatException and a generic Exception (which will essentially allow us to catch any other exception that is not a NumberFormatException)
Catching Exceptions

- Something else to point out is the use of the `getMessage()` method which is a member of the `Exception` class.

- Since all exceptions extend `Exception`, they have this `getMessage()` method.

- The `getMessage()` method returns a String that gives a brief summary of the cause of the exception.

- In our example we display this message to the user when an exception (other than `NumberFormatException`) occurs.

```java
    catch (Exception e1)
    {
        JOptionPane.showMessageDialog(null,
            e1.getMessage());
    }
```
Finally

- After we define our catch blocks, we can also define an OPTIONAL finally block.

- The code in our finally block will always be executed, regardless of whether or not an exception was thrown in the try block.

- In our example, we display a message dialog stating that a button was pressed, regardless of whether or not an error occurred in user input.

- finally blocks are normally used when dealing with file I/O to ensure files that get opened are closed even in the event of an exception.
Throwing Exceptions

- In addition to catching exceptions that may be thrown automatically by other Java methods, we can actually throw our own exceptions.

- Not only can we throw our own exceptions, but we can also write our own exception classes (but we will not cover that in this class)

- In our example, we threw an IllegalArgumentException in our div() method when the user attempts to divide by zero
Throwing Exceptions

- The format used to throw our own exception is to type the keywords `throw new`, followed by the constructor of the type of exception we want to throw

  ```java
  throw new Exception();
  ```

- Most exceptions have a default constructor and a constructor that takes a String for the message. Using this second constructor allows us to set the text for the message that can be displayed with the `getMessage()` method we talked about earlier.

  ```java
  throw new Exception("Error Message!");
  ```
public static int div(int num1, int num2) throws IllegalArgumentException
{
    if (num2 == 0)
        throw new IllegalArgumentException("Cannot divide by zero!");
    return num1 / num2;
}

- In our example we choose to throw an IllegalArgumentException (we could have chosen another type of exception if we wanted) and we give it the error message “Cannot divide by zero!”
Exception Responsibility

```java
public static int div(int num1, int num2) throws IllegalArgumentException {
    if (num2 == 0)
        throw new IllegalArgumentException("Cannot divide by zero!");
    return num1 / num2;
}
```

- A final thing to note about this function is the use of the `throws` keyword in the function header.

- Adding this to the header is essentially telling the compiler that the function is not taking the responsibility for the exception.
Exception Responsibility

- Basically, since we are not handling the exception (using a try/catch block) in our div() method, we advertise that our div() method can throw an IllegalArgument Exception that needs to be taken care of by the calling function (which would be actionPerformed() in our example).

- Since the div() method is not handling its own exception, we say that it is passing the responsibility of that exception to actionPerformed().

- Notice that we are able to handle this exception in our actionPerformed() method because we catch Exception which is a superclass to IllegalArgument Exception.
Drill 11

1. Fix all of the syntax errors in the SyntaxErrors program. Get it to compile and run.

2. Modify the div() method of SimpleCalc3. Instead of passing responsibility of the IllegalArgumentException to actionPerformed(), handle the exception from within the div() method using a try/catch block.

3. Recall the second drill from Lesson 7 (when you created a GUI to calculate the roots of a quadratic equation). Add exception handling to this program. You should make sure the user enters valid numbers for a, b, and c. You should also make sure to check for division by zero. Also, your program should be able to handle any other unknown exceptions.