



Lab 9

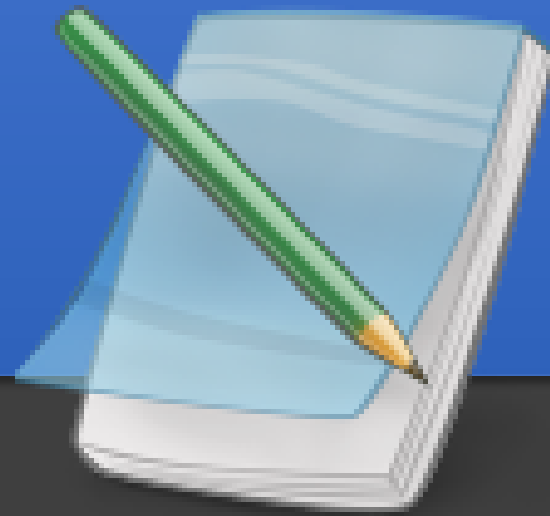
Java Exceptions

Goals

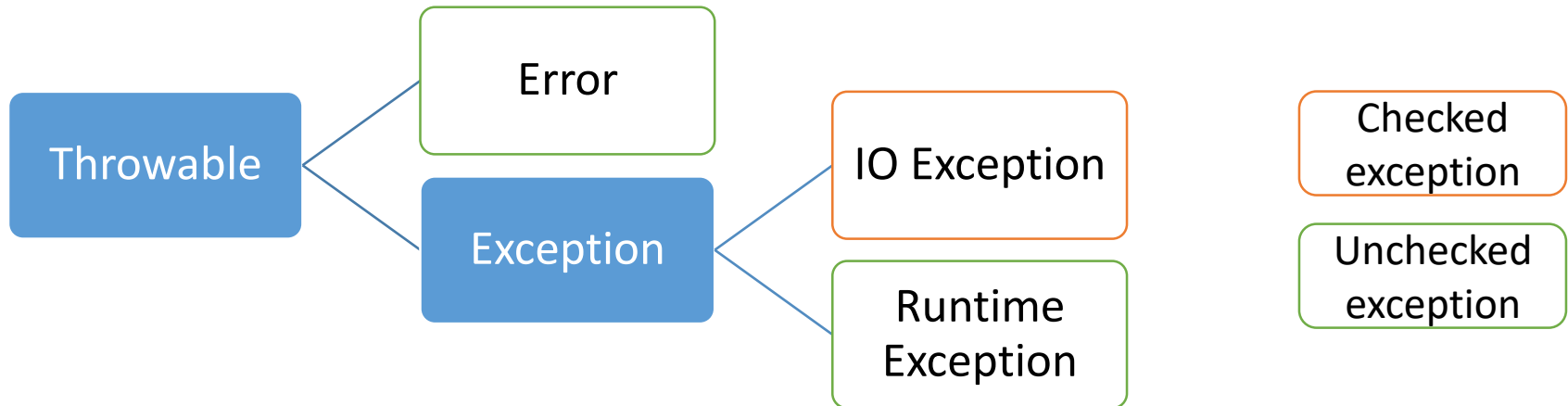


- In this lesson we will learn:
 - To use **try, throw and catch** to detect, indicate and handle exceptions, respectively.
 - To use the **finally block** to release resources.
 - How exceptions are arranged in an **exception class hierarchy**.
 - To **declare new** exception classes.

KEYNOTE



Exception Hierarchy



Some common built-in exception:

- **ArithmeticException** (e.g., divide by zero)
- **ClassCastException** (e.g., attempt to cast a String Object to Integer)
- **IndexOutOfBoundsException**
- **NullPointerException**
- **FileNotFoundException** (e.g., attempt to open a non-existent file for reading)

Catching Exceptions



- A **try/catch** block is placed around the code that might generate an exception
- A **finally** block of code that follows a **try** block, always executes, whether or not an exception has occurred.
- Example:

```
try {  
    //Protected code  
} catch (ExceptionType1 e1) {  
    //Catch block  
} catch (ExceptionType2 e2) {  
    //Catch block  
} catch (ExceptionType3 e3) {  
    //Catch block  
} finally {  
    //The finally block always executes.  
}
```

The throws/throw Keywords



- If a method does not handle a *checked exception*, the method must declare it using the **throws** keyword.
- You can **throw** an exception, either a newly instantiated one or an exception that you just caught, by using the **throw** keyword
- Example:

```
import java.io.*;
public class className
{
    public void deposit(double amount) throws RemoteException
    {
        // Method implementation
        throw new RemoteException();
    }
    //Remainder of class definition
}
```

Declaring you own Exception



- Keep the following points in mind when writing your own exception classes:
 - All exceptions must be a child of **Throwable**.
 - If you want to write a **checked exception**, extend the **Exception** class.
 - If you want to write a **runtime exception**, extend the **RuntimeException** class.

```
public class MyException extends Exception
{
    private double field1;
    public MyException (double input)
    {
        this.field1 = input;
    }
    public double getField1 ()
    {
        return field1;
    }
}
```

Exceptions Methods



Method	Description
String getMessage()	Returns a detailed message about the exception that has occurred. This message is initialized in the Throwable constructor.
synchronized Throwable getCause()	Returns the cause of the exception as represented by a Throwable object.
String toString()	Returns the name of the class concatenated with the result of getMessage()
void printStackTrace()	Prints the result of toString() along with the stack trace to System.err, the error output stream.

When to throw an exception



- “An exception is thrown when a **fundamental assumption** of the current code block is found to be false.”
- “The other side of this equation is: if you find your functions throwing exceptions frequently, then you probably need to **refine** their assumptions.”

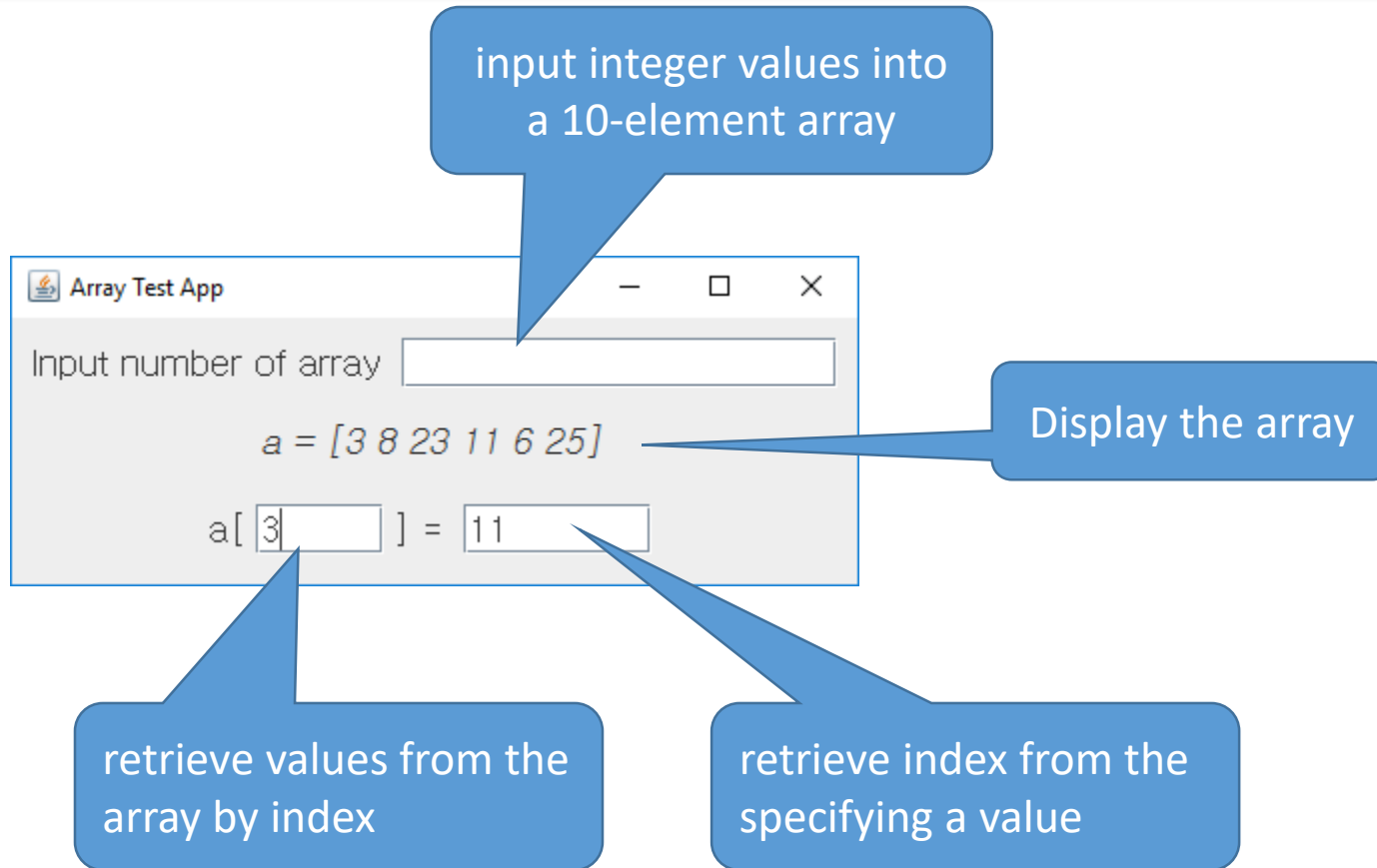
<http://stackoverflow.com/questions/77127/when-to-throw-an-exception>

PRACTICE

Practice makes perfect



1. Problem



The screenshot shows a Java application window titled "Array Test App". It contains a text input field labeled "Input number of array", a line of code `a = [3 8 23 11 6 25]`, and an assignment statement `a[3] = 11`. The input field is empty. The code is displayed in a monospaced font. The assignment statement has a small input field containing the number 3 for the index and another input field containing the number 11 for the value. Five blue callout boxes with white text provide annotations: "input integer values into a 10-element array" points to the input field; "Display the array" points to the code line; "retrieve values from the array by index" points to the index input field; "retrieve index from the specifying a value" points to the value input field; and "Display the array" also points to the code line.

input integer values into a 10-element array

Input number of array

`a = [3 8 23 11 6 25]`

`a[] =`

Display the array

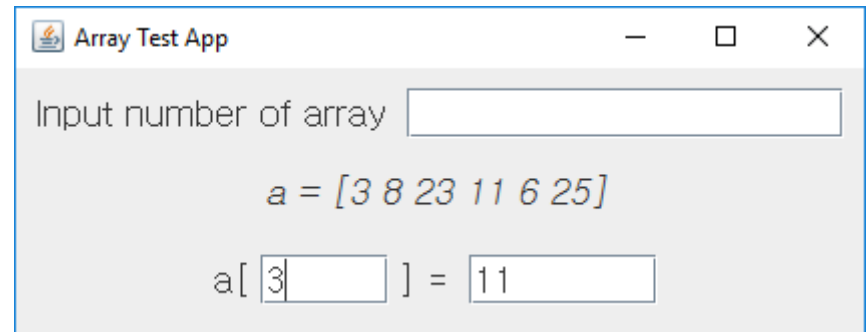
retrieve values from the array by index

retrieve index from the specifying a value

2. Design - MVC pattern



- Model:
 - `int index = 0;`
 - `int array[] = new int[10];`
- View:
 - JLabel lblArray and other 3 labels
- Controller:
 - JTextField txtInputField;
 - JTextField txtNumber;
 - JTextField txtIndex;



3. Implements

Create the components



The screenshot displays the Java Swing IDE interface. On the left, the **Components** palette shows a tree structure for the 'Array Test App' window, including a **contentPane** with sub-components like **txtInputField**, **lblNewLabel**, **lblArray**, and a **panel** with **lblNewLabel_1**, **txtIndex**, **lblNewLabel_2**, and **txtNumber**. Below this, the **Properties** window shows the **Layout** property for **contentPane** set to **(java.awt.GridBagLayout)**, with a **Class** of **java.awt.GridBagLayout** and a **Variable** of **gbl_contentPane**. The **Class** property for the **contentPane** is set to **javax.swing.JPanel**. On the right, a preview of the 'Array Test App' window is shown, featuring a text input field labeled 'Input number of array', a label 'a = []', and a text area 'a[] = '.

Components

- (javax.swing.JFrame) - "Array Test App"
 - contentPane
 - txtInputField
 - lblNewLabel - "Input number of array"
 - lblArray - "a = []"
 - panel
 - lblNewLabel_1 - "a["
 - txtIndex
 - lblNewLabel_2 - "]" = "
 - txtNumber

Properties

Variable	contentPane
Layout	(java.awt.GridBagLayout)
Class	java.awt.GridBagLayout
Variable	gbl_contentPane
columnWidths	0 0
rowHeights	0 0 0 0
Class	javax.swing.JPanel
background	240,240,240
border	EmptyBorder
foreground	0,0,0
tab order	
toolTipText	

Palette

- System
 - Selection
 - Choose co...
- Containers
 - JPanel
 - JScrollPane
 - JSplitPane
 - JTabbedPane
 - JToolBar
 - JLayeredPa...
 - JDesktopPa...
 - JInternalFra...
- Layouts
 - Absolute la...
 - FlowLayout
 - BorderLayo...
 - GridLayout
 - GridBagLay...
 - CardLayout
 - BoxLayout
 - SpringLayout
 - FormLayout
 - MigLayout
 - GroupLayout
- Struts & Springs
- Components
 - JLabel
 - JTextField
 - JComboBox
 - JButton
 - JCheckBox
 - JRadioButton
 - JToggleBut...
 - JTextArea
 - JFormatted...
 - JPasswordField
 - JTextPane
 - JEditorPane
 - JSpinner
 - JList
 - JTable
 - JTree
 - JProgressBar
 - JScrollBar
 - JSeparator
 - JSlider
- Swing Actions
- Menu
- AWT Components
- IGoodies

Array Test App

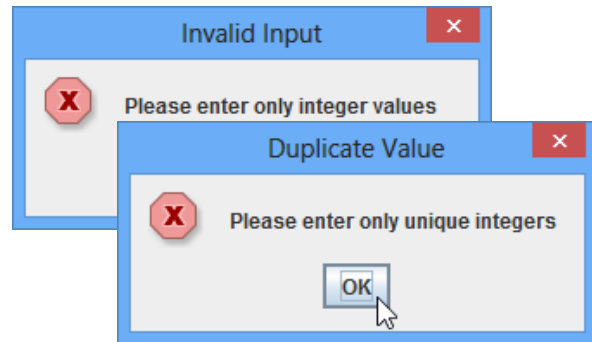
Input number of array

a = []

a[] =

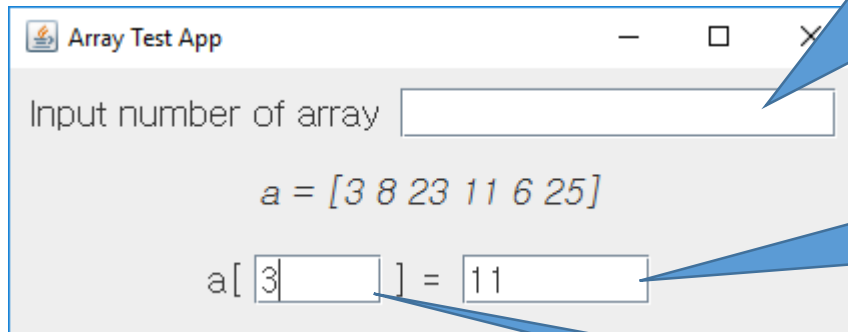
3. Implements

Define the exceptions



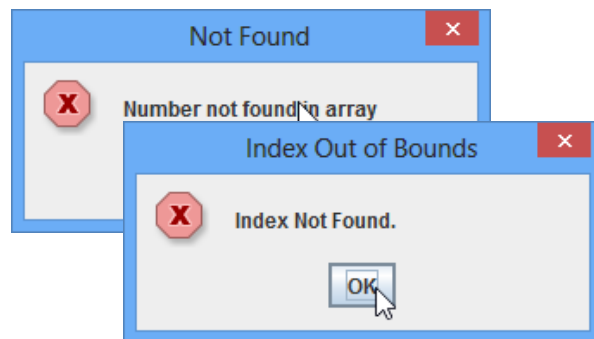
Write catch handlers that catch:

- `NumberFormatException`
- `ArrayIndexOutOfBoundsException`
- `DuplicateValueException` (user define)



Write catch handlers that catch:

- `NumberFormatException`
- `NumberNotFoundException` (user define)



Write catch handlers that catch:

- `NumberFormatException`
- `ArrayIndexOutOfBoundsException`

3. Implements

Add action listener for txtInput



- Create try block and catch the exception

```
txtInputField.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        /*  
        * Create a try block in which the application reads the number  
        * entered in the txtInputField and assigns it to the array.  
        */  
  
        /* Write catch handlers that catch 3 types of exceptions that  
        * the previous try block might throw (NumberFormatException,  
        * ArrayIndexOutOfBoundsException and DuplicateValueException)  
        * and display appropriate messages in error message dialogs  
        * using JOptionPane.showMessageDialog() method  
        */  
    }  
});
```

3. Implements

Add action listener for txtNumber



- Create try block and catch the exception

```
txtNumber.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        /*  
        * Create a try block to reads from txtNumber  
        * the number the user wants to find in the array,  
        * then searches the current array contents for 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 catch handlers that catch the two types  
        * of exceptions that the try block might throw  
        * (NumberFormatException and NumberNotFoundException),  
        * and display appropriate messages in error  
        * message dialogs.  
        */  
    }  
});
```


3. Implements

Add action listener for txtIndex



- Create try block and catch the exception

```
txtIndex.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        /*  
        * Create a try block to reads from txtIndex the index of the array,  
        * then displays the value at that index in the txtNumber.  
        * 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 ArrayIndexOutOfBoundsException should  
        * be thrown.  
        */  
  
        /*  
        * Write catch handlers that catch the two types of exceptions  
        * the try block might throw (NumberFormatException and  
        * ArrayIndexOutOfBoundsException), and display appropriate  
        * messages in error message dialogs.  
        */  
    }  
});
```

QUIZ



Find the 4 errors in this code



```
1. import java.io.IOException;
2. public class SpecialIOException throws IOException {
3.     public SpecialIOException() {
4.         super("Special IO Exception Occurred");
5.     }
6.     public SpecialIOException(String message) {
7.         this(message);
8.     }
9. } // end class SpecialIOException

10. public class DebugException {
11.     public static void main(String args[]) {
12.         try {
13.             throw new SpecialIOException();
14.         } catch (Exception exception) {
15.             System.err.println(exception.toString());
16.         } catch (IOException ioException) {
17.             System.err.println(ioException.toString());
18.         } catch (SpecialIOException specialIOException) {
19.             specialIOException.toString();
20.         }
21.     }
22. } // end class DebugException
```