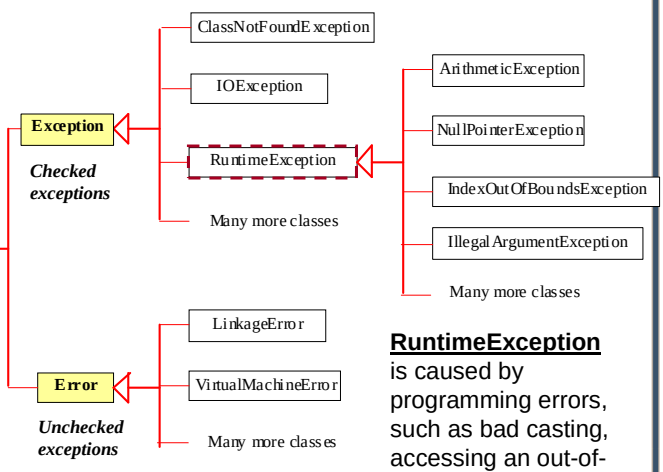
Java Exception Handling

- When a program runs into a runtime error, the program terminates abnormally
- Exception handling feature of Java allows programmer to handle exceptions gracefully so that the program can continue to run or terminate gracefully
- Unchecked Exceptions
  - RuntimeException
  - Error and their subclasses
- Checked Exceptions
  - All other exceptions
  - Compiler requires programmer to check and handle

 1

Exception Types

Exception describes errors caused by your program and external circumstances. These errors can be caught and handled by your program.



System errors are thrown by JVM and represented in the Error class. The Error class describes internal system errors. Such errors rarely occur. If one does, there is little you can do beyond notifying the user and trying to terminate the program gracefully.

RuntimeException is caused by programming errors, such as bad casting, accessing an out-of-bounds array, and numeric errors.

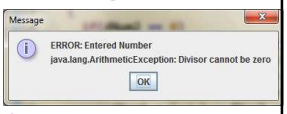
 2

Exception Handling

- Exception Handling Processing
  - Error occurs while method is running
  - Method creates *Information Object* about error
  - Information Object passed to Java Virtual Machine
  - JVM attempts to locate code to handle exception
  - This process is called *Throwing an Exception*
- Checked exception
  - `try`
    - Identifies start of exception handling block of code
    - Must be followed by one or more catch blocks
  - `catch`
    - Exception handler code
  - `finally`
    - Default set of instructions that is always executed whether or not any exception occurred

 3

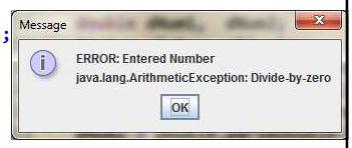
ArithmeticException Handling Example




```
1.  import javax.swing.*;
2.  public class Exception1
3.  {
4.    public static void main(String args[])
5.    {
6.      double dNum1,  dNum2;
7.      String sEntry, sDisplay;
8.      sEntry = JOptionPane.showInputDialog("Enter 1st Number:");
9.      dNum1 = Double.parseDouble(sEntry);
10.     sEntry = JOptionPane.showInputDialog("Enter 2nd Number:");
11.     dNum2 = Double.parseDouble(sEntry);
12.     try
13.     {
14.       if(dNum2 == 0)
15.         throw new ArithmeticException("Divisor cannot be zero");
16.       sDisplay = "Operators Output\nby Bob Laurie\n"
17.         + dNum1 + " + " + dNum2 + " = " + (dNum1+dNum2) + '\n'
18.         + dNum1 + " - " + dNum2 + " = " + (dNum1-dNum2) + '\n'
19.         + dNum1 + " / " + dNum2 + " = " + (dNum1/dNum2) + '\n'
20.         + dNum1 + " x " + dNum2 + " = " + (dNum1*dNum2);
21.       JOptionPane.showMessageDialog(null, sDisplay);
22.     }
23.     catch(ArithmeticException ex)
24.     {
25.       JOptionPane.showMessageDialog(null, "ERROR: Entered Number\n" + ex);
26.     }
27.     finally
28.     {
29.       JOptionPane.showMessageDialog(null, "Program Done");
30.     }
31.   }
32. }
```

ArithmeticException using Quotient method

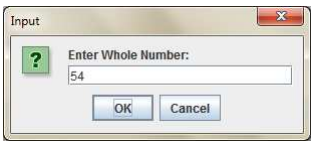
```
1.  import javax.swing.*;
2.  public class Exception2
3.  {
4.    public static String quotient(double dN1, double dN2) {
5.      if(dN2 == 0)
6.        throw new ArithmeticException("Divide-by-zero");
7.      return String.format("%.2d", dN1/dN2);
8.    }
9.    public static void main(String args[]) {
10.     double dNum1,  dNum2;
11.     String sEntry, sDisplay;
12.     sEntry = JOptionPane.showInputDialog("Enter 1st Number:");
13.     dNum1 = Double.parseDouble(sEntry);
14.     sEntry = JOptionPane.showInputDialog("Enter 2nd Number:");
15.     dNum2 = Double.parseDouble(sEntry);
16.     sDisplay = "Operators Output\nby Bob Laurie\n"
17.         + dNum1 + " + " + dNum2 + " = " + (dNum1+dNum2) + '\n'
18.         + dNum1 + " - " + dNum2 + " = " + (dNum1-dNum2) + '\n'
19.         + dNum1 + " x " + dNum2 + " = " + (dNum1*dNum2) + '\n';
20.     try {
21.       sDisplay += dNum1 + " / " + dNum2 + " = " + quotient(dNum2, dNum2) + '\n';
22.     }
23.     catch(ArithmeticException ex) {
24.       JOptionPane.showMessageDialog(null, "ERROR: Entered Number\n" + ex);
25.     }
26.     finally {
27.         JOptionPane.showMessageDialog(null, sDisplay);
28.      }
29.   }
30. }
```



Advantage of using exception handling is it allows a method to throw an exception to its caller.

JOptionPane NumberFormatException processing

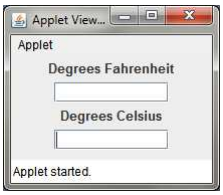
```
1.  import java.util.*;
2.  import javax.swing.*;
3.  public class InputMismatchExceptionDemo {
4.    public static void main(String[] args) {
5.      double nNum;
6.      String sEntry="";
7.      boolean continueInput = true;
8.      do {
9.        try {
10.         sEntry = JOptionPane.showInputDialog("Enter Whole Number:");
11.         nNum = Integer.parseInt(sEntry);
12.         JOptionPane.showMessageDialog(null,  "You entered is " + sEntry);
13.         continueInput = false;
14.       }
15.       catch (NumberFormatException ex) {
16.         JOptionPane.showMessageDialog(null, "Try again. [" + sEntry +
17.           "]  <--An Integer is required)");
18.       }
19.     } while (continueInput);
20.   }
21. }
```

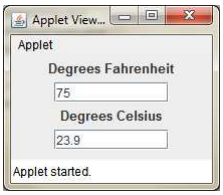


 6

GUI JFrameNumberFormatException

```java
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class TmpAppletApException extends JApplet {
  private JTextField txtFahr = new JTextField("", 10);
  private JTextField txtCelc = new JTextField("", 10);
  public TmpAppletApException() {
    setLayout(new FlowLayout(FlowLayout.CENTER, 6, 6));
    JLabel lblFahr = new JLabel("Degrees Fahrenheit");
    lblFahr.setFont(new Font("Arial", Font.BOLD, 13));
    add(lblFahr);
    add(txtFahr);
    JLabel lblCelsius = new JLabel("Degrees Celsius");
    lblCelsius.setFont(new Font("Arial", Font.BOLD, 13));
    add(lblCelsius);
    add(txtCelc);
    txtCelc.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent event) {
        try {
          double dFahr =  Double.parseDouble(txtCelc.getText()) ;
          txtFahr.setText(String.format("%.1f", dFahr * 9 / 5 + 32));
        }
        catch (NumberFormatException e) {
          JOptionPane.showMessageDialog(null, "ERROR: " +
            "Not a Number\n" + e.getMessage());
          txtFahr.setText("");
          txtCelc.setText("");
        }
      }
    });
    txtFahr.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent event) {
        try {
          double dCelc = Double.parseDouble(txtFahr.getText());
          txtCelc.setText(String.format("%.1f", (dCelc - 32 ) * 5 / 9));
        }
        catch (NumberFormatException e) {
          JOptionPane.showMessageDialog(null, "ERROR: " +
            "Not a Number\n" + e.getMessage());
          txtFahr.setText("");
          txtCelc.setText("");
        }
      }
    });
  }
}
```





Java Exception Handling Model

- Declaring Exceptions
  - Every method must state the types of checked exceptions it might throw which are not Error or Runtime exceptions
  - This is known as declaring exceptions
    public void myMethod() throws IOException
    public void myMethod() throws IOException, OtherException
- Throwing Exceptions
  - When the program detects an error, the program can create an instance of the exception type and throw it
  - Known as throwing an exception
    throw new TheException();
    TheException ex = new TheException();
    throw ex;

 8

Common Unchecked Exceptions

These Exception arise during run-time, due to invalid argument passed to method. The java Compiler does not check the program error during compilation. For Example when you divide a number by zero.

Exception	Reason for Exception
Arithmetic Exception	These Exception occurs, when you divide a number by zero causes an Arithmetic Exception
Class Cast Exception	These Exception occurs, when you try to assign a reference variable of a class to an incompatible reference variable of another class
Array Store Exception	These Exception occurs, when you assign an array which is not compatible with the data type of that array
Array Index Out Of Bounds Exception	These Exception occurs, when you assign an array which is not compatible with the data type of that array
Null Pointer Exception	These Exception occurs, when you try to implement an application without referencing the object and allocating to a memory
Number Format Exception	These Exception occurs, when you try to convert a string variable in an incorrect format to integer (numeric format) that is not compatible with each other
Negative ArraySizeException	These are Exception, when you declare an array of negative size.

 9

Circle Exception Handling

```
1.  public class TestCircle {
2.    public static void main(String[] args) {
3.      try {
4.        Circle c0 = new Circle(1.75);
5.        Circle c1 = new Circle(5);
6.        Circle c2 = new Circle(-2);
7.        Circle c3 = new Circle(3);
8.      }
9.      catch (IllegalArgumentException ex) {
10.       System.out.println(ex);
11.     }
12.     System.out.println("Objects Qty: " + Circle.getNumberOfObjects());
13.   }
14. }
15. class Circle {
16.   private double dRadius;
17.   private static int nQtyCircles = 0;
18.   public Circle() {
19.     this(1.0);
20.   }
21.   public Circle(double dNewRadius) {
22.     setRadius(dNewRadius);
23.     nQtyCircles++;
24.     System.out.println("Circle object created: radius = " + this.getRadius());
25.   }
26.   public double getRadius() {
27.     return dRadius;
28.   }
29.   public void setRadius(double dNewRadius) throws IllegalArgumentException {
30.     if (dNewRadius >= 0)
31.       dRadius =  dNewRadius;
32.     else
33.       throw new IllegalArgumentException("\nRadius cannot be negative");
34.   }
35.   public static int getNumberOfObjects() {
36.     return nQtyCircles;
37.   }
38.   public double findArea() {
39.     return dRadius * dRadius * Math.PI;
40. } }
```

```
Circle object created: radius = 1.75
Circle object created: radius = 5.0
java.lang.IllegalArgumentException:
Radius cannot be negative
Objects Qty: 2
```