Review Session for

EXCEPTIONS & GUI

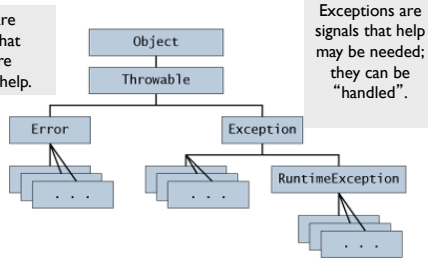
-Deepak Bapat

Adapted from Previous Review Slides

Exception: event that disrupts the normal flow of program execution

Throwable Class and Its Subclasses

Errors are signals that things are beyond help.



Exceptions are signals that help may be needed; they can be "handled".

How do you know if a method throws an exception?

- Execution generates an error if a method throws an exception and you have not handled it yet. You may catch the exception and handle it.
- Refer to Javadoc API specifications.

  Eg : method charAt inclass String

  public char charAt(**int** index)

  Return the character at the specified index. An index ranges from 0 to length() - 1. ….

  Throws: IndexOutOfBoundsException - if the index argument is negative or not less than the length of this string.

Writing an exception class

```
class MyException extends Exception {
    public MyException( ) {
       super();
    }
    public MyException(String msg) {
       super(msg);
    }
}
```

Probably best to extend RuntimeException

```
public class Test {
    public void testMethod( ){
       throw new MyException( );
    }
}
```

Error: Unhandled exception type MyException in testMethod()


```
class MyException extends
    Exception{
    public MyException() {
    }
    public MyException(String msg) {
       super(msg);
    }
}
```

```
class Test {
    public void testMethod( ) {
        try {
           throw new MyException( );
        } catch (MyException e) {
           e.printStackTrace();
           …
        }
    }
}
```


try/catch statements

- What you just saw on the previous page was a try/ catch block
- Sometimes voluntary, sometimes java requires you to try/catch or throw (get to throw in a minute)
- We "try" a series of commands, but if we get an exception we "catch" the exception and do something else

```
int x=0;
String s = "java";
try {
   x = Integer.parseInt(s);
   x=2;
} catch (NumberFormatException e) {
   x=1;
}
```

Throwing exceptions

- To throw an exception, you use the command
  - throw <exception>
    - throw new MyException()
- When an exception is thrown, normal flow of code stops
- The exception is propagated up function calls
- If no catch statement is found, java will exit and the error will be displayed

```
/** Illustrate exception handling */
public class Ex {
public static void first() {
        second();
        }
public static void second() {
        third();
        }
public static void third() {
        throw new
        MyException("mine");
        }
        }
```

7

The "throws" clause

```
/** Class to illustrate exception handling */
public class Ex {
   public static void first() throws MyException {
      second();
   }
   public static void second() throws MyException {
      third();
   }
   public static void third() throws MyException {
      throw new MyException("mine");
   }
```

Output of Ex.first()

Call Output

```
Ex.first();
ArithmeticException: mine
  at Ex.third(Ex.java:14)
  at Ex.second(Ex.java:9)
  at Ex.first(Ex.java:5)
  at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
  at sun.reflect.NativeMethodAccessorImpl.invoke(…)
  at sun.reflect.DelegatingMethodAccessorImpl.invoke(…)
  at java.lang.reflect.Method.invoke(Method.java:585)
```

9

Some Java Exception classes

ApplicationException
ArithmeticException
ArrayStoreException
FileNotFoundException
IndexOutOfBoundsException
IllegalArgumentException
IllegalStateException
InvalidOperationException
InvalidParameterException

10

Which is better?

Using exceptions

```
public static Object get(Vector v, int i) {
    try {
       return v.get(i);
    } catch (Exception e) {
       return null;
    }
}
```

Using an if-statement

```
public static Object get(Vector v, int i) {
    if (i < 0 || v.size() <= i)
        return null;
    return v.get(i);
}
```

11

What is wrong with this?

```
try {
        int bricksInRow= Integer.valueOf(b[0]);
        int brickRows= Integer.valueOf(b[1]);
        if (bricksInRow <= 0 || brickRows <= 0)
            return;

} catch (NumberFormatException nfe) {
}
BRICKS_IN_ROW= bricksInRow;
BRICK_ROWS= brickRows;
BRICK_WIDTH= WIDTH / BRICKS_IN_ROW - BRICK_SEP_H;
```

12

```
/** If b is null, doesn't have exactly two elements, or the elements are not
      positive integers, DON'T CHANGE ANYTHING.
      If b is non-null, has exactly two elements, and they are positive
      integers with no blanks surrounding them, then:
      Store the first int in BRICKS_IN_ROW, store the second int in BRICK_ROWS,
      and recompute BRICK_WIDTH using the formula given in its declaration.
       */
   private static void fixBricks(String[] b) {
      /** Hint. You have to make sure that the two Strings are positive integers.
          The simplest way to do that is to use the calls Integer.valueOf(b[0]) and
          Integer.valueOf(b[1]) within a try-statement in which the catch block is
          empty. Don't store any values in the static fields UNTIL you are sure
          that both array elements are positive integers. */
      if (b == null || b.length != 2)
         return;
      try {
         int bricksInRow= Integer.valueOf(b[0]);
         int brickRows= Integer.valueOf(b[1]);
         if (bricksInRow <= 0 || brickRows <= 0)
            return;
         BRICKS_IN_ROW= bricksInRow;
         BRICK_ROWS= brickRows;
         BRICK_WIDTH= WIDTH / BRICKS_IN_ROW - BRICK_SEP_H;
      } catch (NumberFormatException nfe) {
      }
   }
```

BAD

```
if (b.length != 2 || b == null)
    return
```

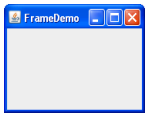

GUIs

- Three things are a must know
  - JFrame
  - JPanel
  - Box
- Each has its own default LayoutManager
  - JFrame – BorderLayout
  - JPanel – FlowLayout
  - Box – BoxLayout

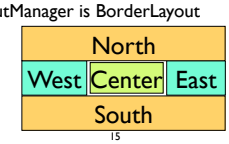
GUIs – JFrame

Extend a JFrame implement its functionality or just call a JFrame

- JFrame frame = new JFrame("FrameDemo");
- **public class** ComponentExample **extends** JFrame {
  public ComponentExample(String t) {
  super("FrameDemo");
  }
  }



The default LayoutManager is BorderLayout



GUIs – JFrame

Components in a JFrame

- java.awt: Old package
- javax.swing: New package
- Components

JButton, Button: Clickable button
JLabel, Label: Line of text
JTextField, TextField: Field into which the user can type:
JTextArea, TextArea: Many-row field into which user can type
JPanel, Panel: Used for graphics; to contain other components
JCheckBox: Checkable box with a title
JComboBox: Menu of items, one of which can be checked
JRadioButton: Same functionality as JCheckBox
Container: Can contain other components
Box: Can contain other components

Basic Components

Component
  Button, Canvas
  Checkbox, Choice
  Label, List, Scrollbar
  TextComponent
    TextField, TextArea
  Container
    JComponent
      AbstractButton
        JButton
        JToggleButton
          JCheckBox
          RadioButton
      JLabel, JList
      JOptionPane, JPanel
      JPopupMenu, JScrollBar, JSlider
      JTextComponent
      JTextField, JTextArea

Component: Something that can be placed in a GUI window. These are the basic ones that one uses in a GUI

Note the use of subclasses to provide structure and efficiency. For example, there are two kinds of JToggleButtons, so that class has two subclasses.

Components that can contain other components

Component
  Box
  Container
    JComponent
    JPanel
    Panel
      Applet
  Window
    Frame
      JFrame
    JWindow

java.awt is the old GUI package.

javax.swing is the new GUI package. When they wanted to use an old name, they put J in front of it.

(e.g. Frame and JFrame)

When constructing javax.swing, the attempt was made to rely on the old package as much as possible.

So, JFrame is a subclass of Frame.

But they couldn't do this with JPanel.

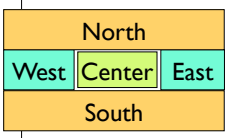
GUIs – BorderLayout

```
Container cp= getContentPane();

JButton jb= new JButton("Click here");
JLabel jl= new JLabel( "label 2");

cp.add(jb, BorderLayout.EAST);
cp.add(jl, BorderLayout.WEST);

pack();
setVisible(true);
```

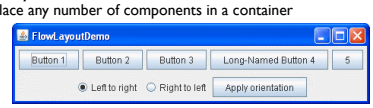


You can pack up to 5 things, so you might nest JPanels within a JFrame

GUIs – JPanel

- This is another type of container
- We nest these inside of other windows
- The default LayoutManager is FlowLayout

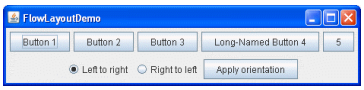
FlowLayout
- Place any number of components in a container



http://download.oracle.com/javase/tutorial/uiswing/examples/layout/FlowLayoutDemoProject/src/layout/FlowLayoutDemo.java

GUIs – FlowLayout

```
JPanel compsToExperiment = new JPanel();
compsToExperiment.add(new JButton("Button 1")); compsToExperiment.add
(new JButton("Button 2")); compsToExperiment.add(new JButton("Button 3"));
compsToExperiment.add(new JButton("Long-Named Button 4"));
compsToExperiment.add(new JButton("5"));
JPanel controls = new JPanel();
controls.add(new JRadioButton("Left to right"));
controls.add(new JRadioButton("Right to left"));
controls.add(new JButton("Apply orientation"));
```



http://download.oracle.com/javase/tutorial/uiswing/examples/layout/FlowLayoutDemoProject/src/layout/FlowLayoutDemo.java

GUIs – Box

```
public class BoxDemo extends JFrame {
   /** horizontal Box with 4 buttons in center. */
   public BoxDemo() {
      super("Box demo");

      Box b= new Box(BoxLayout.X_AXIS);
      b.add(new JButton("0"));    b.add(new JButton("1"));
      b.add(new JButton("2"));    b.add(new JButton("3"));
      getContentPane().add(b);
   }
}
```

Boxes use a BoxLayout in which you add components along an axis