Review Session for

EXCEPTIONS & GUI

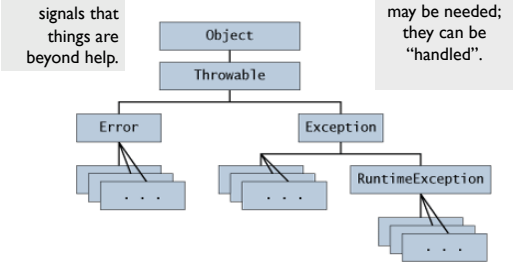
-Ankur Agarwal

An *Exception* is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.

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?

- The Java compiler will generate an error if a method throws an exception and you have not handled it yet. You are supposed to catch the exception to remove that error.
- By referring to Java Docs.

Eg : **charAt** method from the String class

public char **charAt**(int index)

Returns the character at the specified index. An index ranges from 0 to length() - 1. The first character of the sequence is at index 0, the next at index 1, and so on, as for array indexing.

Parameters: index - the index of the character.**Returns:** the character at the specified index of this string. The first character is at index 0.

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

Writing your exception class

```
class MyException extends Exception{
        public MyException( )
        {
        }
        public MyException(String msg)
        {
              super(msg);
        }
}
public class Test{
        public void testMethod( ){
              throw new MyException( );
        }
}
```

Error: Unhandled exception type MyException in testMethod()

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


Some Java Exception classes

- ApplicationException
- ArithmeticException
- ArrayStoreException
- FileNotFoundException
- IndexOutOfBoundsException
- IllegalArgumentException
- IllegalStateException
- InvalidOperationException
- InvalidParameterException

GUI

Listening to events: mouseclick, mouse movement into or out of a window, a keystroke, etc.

- An event is a mouseclick, a mouse movement into or out of a window, a keystroke, etc.
- To be able to "listen to" a kind of event, you have to
  1. Write a method that will listen to the event.
  2. Let Java know that the method is defined in the class.
  3. Register an instance of the class that contains the method as a *listener* for the event.

We show you how to do this for clicks on buttons, clicks on components, and keystrokes.

Listening to a Button

1. Write the procedure to be called when button is clicked:

```
/** Process click of button */
public void actionPerformed(ActionEvent ae) {
    ...
}
```

3. Have class implement interface ActionListener:

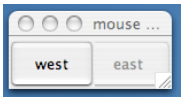
```
public class C extends JFrame  implements
                                ActionListener {
    ...
}
```

3. Add instance of this class as an "action listener" for button:

```
button.addActionListener(this);
```

Listening to a Button

```
/** An instance has two buttons. Exactly one is always enabled. */
public class ButtonDemo1 extends JFrame
                          implements ActionListener {
  /** Class invariant: exactly one of eastB and westB is enabled */
  private JButton westB= new JButton("west");
  private JButton eastB= new JButton("east");
  /** Constructor: frame with title t & two buttons */
  public ButtonDemo1(String t) {
    super(t);
    Container cp= getContentPane();
    cp.add(westB, BorderLayout.WEST);
    cp.add(eastB, BorderLayout.EAST);
    westB.setEnabled(false);
    eastB.setEnabled(true);
    westB.addActionListener(this);
    eastB.addActionListener(this);
    pack();
    setVisible(true);
  }
```



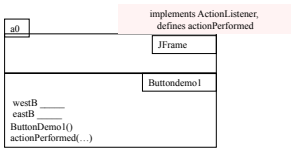

```
  /** Process a click of a button */
  public void actionPerformed
                (ActionEvent e) {
    boolean b= eastB.isEnabled();
    eastB.setEnabled(!b);
    westB.setEnabled(b);
  }
```

red: listening

blue: placing

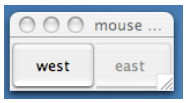
Listening to a Button

```
/** An instance has two buttons. Exactly one is always enabled. */
public class ButtonDemo1 extends JFrame
                          implements ActionListener {
  /** Class invariant: exactly one of eastB and westB is enabled */
  private JButton westB= new JButton("west");
  private JButton eastB= new JButton("east");
  /** Constructor: frame with title t & two buttons */
  public ButtonDemo1(String t) {
    super(t);
    Container cp= getContentPane();
    cp.add(westB, BorderLayout.WEST);
    cp.add(eastB, BorderLayout.EAST);
    westB.setEnabled(false);
    eastB.setEnabled(true);
    westB.addActionListener(this);
    eastB.addActionListener(this);
    pack();
    setVisible(true);
  }
/** Process a click of a button */
  public void actionPerformed
                (ActionEvent e) {
    boolean b= eastB.isEnabled();
    eastB.setEnabled(!b);
    westB.setEnabled(b);
  }
}
```

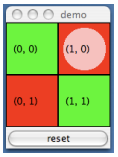


red: listening

blue: placing



A JPanel that is painted



- The content pane has a JPanel in its CENTER and a "reset" button in its SOUTH.
- The JPanel has a horizontal box b, which contains two vertical Boxes.
- Each vertical Box contains two instances of class Square.
- Click a Square that has no pink circle, and a pink circle is drawn. Click a square that has a pink circle, and the pink circle disappears. Click the rest button and all pink circles disappear.
- This GUI has to listen to:
  (1) a click on a Button
  (2) a click on a Square

these are different kinds of events, and they need different listener methods

Class Square

```
/** An instance is a JPanel of size (WIDTH,HEIGHT). Green
 or red depending on whether the sum of constructor parameters
 is even or odd. .. */
public class Square extends JPanel {
  public static final int HEIGHT= 70;  // height and
  public static final int WIDTH= 70;   // width of square
  private int x, y; // Coordinates of square on board
  private boolean hasDisk= false; // = "square has pink disk"
  /** Constructor: a square at (x,y) */
  public Square(int x, int y) {
    this.x= x;       this.y= y;
    setPreferredSize(new Dimension(WIDTH,HEIGHT));
  }

  /** Complement the "has pink disk" property */
  public void complementDisk() {
    hasDisk= ! hasDisk;
    repaint(); // Ask the system to repaint the square
  }
```

continued on next page

continuation of class Square

Class Square

```
/* paint this square using g. System calls
   paint whenever square has to be redrawn.*/
 public void paint(Graphics g) {
  if ((x+y)%2 == 0) g.setColor(Color.green);
  else g.setColor(Color.red);
  g.fillRect(0, 0, WIDTH-1, HEIGHT-1);
  if (hasDisk) {
   g.setColor(Color.pink);
   g.fillOval(7, 7, WIDTH-14, HEIGHT-14);
  }
  g.setColor(Color.black);
  g.drawRect(0, 0, WIDTH-1,HEIGHT-1);
  g.drawString("("+x+", "+y+")", 10, 5+HEIGHT/2);
 }
}
```

```
/** Remove pink disk
    (if present) */
public void clearDisk() {
  hasDisk= false;
  // Ask system to
  // repaint square
  repaint();
 }
```

A class that listens to a mouseclick in a Square

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

red: listening

blue: placing


```
/** Contains a method that responds to a
    mouse click in a Square */
public class MouseEvents
             extends MouseInputAdapter {
  // Complement "has pink disk" property
  public void mouseClicked(MouseEvent e) {
    Object ob= e.getSource();
    if (ob instanceof Square) {
      ((Square)ob).complementDisk();
    }
  }
}
```

This class has several methods (that do nothing) that process mouse events:

- mouse click
- mouse press
- mouse release
- mouse enters component
- mouse leaves component
- mouse dragged beginning in component

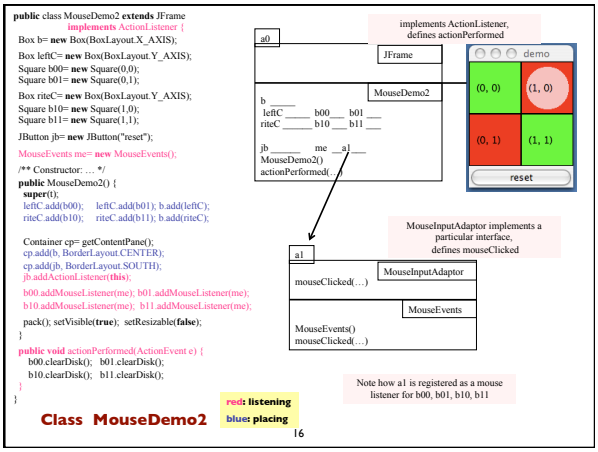
Our class overrides only the method that processes mouse clicks


```
public class MouseDemo2 extends JFrame
             implements ActionListener {
  Box b= new Box(BoxLayout.X_AXIS);
  Box leftC= new Box(BoxLayout.Y_AXIS);
  Square b00= new Square(0,0);
  Square b01= new Square(0,1);
  Box riteC= new Box(BoxLayout.Y_AXIS);
  Square b10= new Square(1,0);
  Square b11= new Square(1,1);
  JButton jb= new JButton("reset");
  MouseEvents me= new MouseEvents();
  /** Constructor: … */
  public MouseDemo2() {
    super(t);
    leftC.add(b00);    leftC.add(b01);
    riteC.add(b10);    riteC.add(b11);
    b.add(leftC);      b.add(riteC);
    Container cp= getContentPane();
    cp.add(b, BorderLayout.CENTER);
    cp.add(jb, BorderLayout.SOUTH);
    jb.addActionListener(this);
    b00.addMouseListener(me);
    b01.addMouseListener(me);
    b10.addMouseListener(me);
    b11.addMouseListener(me);
    pack(); setVisible(true);
    setResizable(false);
  }
  public void actionPerformed(
      ActionEvent e) {
    b00.clearDisk();  b01.clearDisk();
    b10.clearDisk();  b11.clearDisk();
  }
}
```

red: listening

blue: placing

Class MouseDemo2



Class MouseDemo2

red: listening

blue: placing

Listening to the keyboard

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

public class AllCaps extends KeyAdapter {
  JFrame capsFrame= new JFrame();
  JLabel capsLabel= new JLabel();

  public AllCaps() {
    capsLabel.setHorizontalAlignment(SwingConstants.CENTER);
    capsLabel.setText(":)");
    capsFrame.setSize(200,200);
    Container c= capsFrame.getContentPane();
    c.add(capsLabel);
    capsFrame.addKeyListener(this);
    capsFrame.show();
  }

  public void keyPressed (KeyEvent e) {
    char typedChar= e.getKeyChar();
    capsLabel.setText((""" + typedChar + """).toUpperCase());
  }
}
```

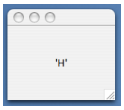
red: listening

blue: placing

1. Extend this class.

3. Add this instance as a key listener for the frame

2. Override this method. It is called when a key stroke is detected.



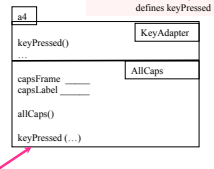
Listening to the keyboard

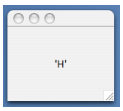
red: listening

blue: placing

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

public class AllCaps extends KeyAdapter {
  JFrame capsFrame= new JFrame();
  JLabel capsLabel= new JLabel();

  public AllCaps() {
    capsLabel.setHorizontalAlignment(SwingConstants.CENTER);
    capsLabel.setText(":)");
    capsFrame.setSize(200,200);
    Container c= capsFrame.getContentPane();
    c.add(capsLabel);
    capsFrame.addKeyListener(this);
    capsFrame.show();
  }

  public void keyPressed (KeyEvent e) {
    char typedChar= e.getKeyChar();
    capsLabel.setText((""" + typedChar + """).toUpperCase());
  }
}
```

KeyAdaptor implements a particular interface, defines keyPressed





Overriding method. Called when a key stroke is detected.