import java.io.*;
public class CheckParens {
    public static void main (String[] args) throws IOException{
        BufferedReader fileIn = new BufferedReader
            (new FileReader(args[0]));

        String line;

        while ((line = fileIn.readLine()) != null) {
            int count = 0;
            while(count < line.length()) {
                if (line.charAt(count).equals('('));
                ...

            } // end inner while
        } // end outer while
        fileIn.close(); // close input file
    } // end main
    <nested classes here>
} // end class
Lab10: Defining Exceptions

```java
static class UnMatchedLeftParenException extends RuntimeException {
    public UnMatchedLeftParenException(String msg) {
        super(msg); // call constructor of parent class
    }
}
```

- Our text distinguishes between what it calls “nested” classes and “inner classes”.
- Nested classes are static class definitions inside the body of another class.
- Inner classes are non-static class definitions inside body of another class.
Lab10: Defining Exceptions

```java
static class UnMatchedRightParenException extends RuntimeException {

    public UnMatchedRightParenException(String msg) {
        super(msg);  // call constructor of parent class
    }

} // end constructor
} // end nested class
```

- You will be instructed to use the nested class shown above in your solution to Lab 10.
Fundamentals of Graphical User Interfaces (GUIs)

- Imports: `java.awt.*`, `java.awt.event.*`, `javax.swing.*`

- Class can extend `JFrame`, the basic window on a particular operating system.

- Main method starts by creating an object of its own class. The main method is not really part of an instance of a class…it is only used by the JVM as a starting point for execution.

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

public class BasicGUI extends JFrame {
    public static void main(String[] args) {
        BasicGUI bGUI = new BasicGUI();
    }

    ...
Fundamentals of Graphical User Interfaces (GUIs)

- Components defined in package javax.swing:
  JButton, JLabel, JTextField, JTextArea

- Components defined in package java.awt:
  Container, Graphics

```java
public class BasicGUI extends JFrame {
    private JButton quit, button1, button2;
    private JLabel instructions;
    private Container holder;
    private JTextField getInputJTextField;

    BasicGUI() {
        holder = this.getContentPane();
        holder.setLayout(null);
        holder.setBackground(Color.white);
        this.setLocation(100,50);
        ...
    }

    public static void main(String[] args) {
        BasicGUI bGUI = new BasicGUI();
    }
}
```

Constructor can instantiate components and sets up the user interface.

Main creates object of BasicGUI type.
Fundamentals of Graphical User Interfaces (GUIs)

- Constructor creates the components of the GUI:

```java
BasicGUI() {
    holder = this.getContentPane();
    holder.setLayout(null);
    holder.setBackground(Color.white);
    this.setLocation(100,50);
    // set size of JFrame
    this.setPreferredSize(new Dimension(600,450));

    // set size of JFrame
    button1 = new JButton("ONE");
    button1.setBounds(275,350,50,20);
    button1.addActionListener(SEE NEXT SLIDE);
    holder.add(button1); // add JButton to Container
    ...}
```

Components are visual elements, subclasses of class `java.awt.Component`
Every component is responsible for drawing itself.
Swing Components

• JButton:
  Constructor takes String parameter which is text on button
  Generates ActionEvent when button clicked
  Registration of Listener – addActionListener in constructor
  Events: can call getActionCommand or getSource() on event
  Changes: can setText to be different, can disable button

• JLabel:
  Constructor takes String which is text written
  No events generated
  Changes: can setText to be different

• JTextField:
  Constructor takes int which is preferred number of characters, String, String and int, or nothing
  Generates an ActionEvent when return is pressed
  Changes: can setText to be different
Adding ActionListeners

• There are at least 3 ways to add an ActionListener to a component that generates an ActionEvent:

  1. Have class implement ActionListener and write an actionPerformed method in class:

     This usually requires multiple decisions in method actionPerformed to choose between actions taken from different sources. Not a big disadvantage.

  2. Create a private inner class that implements ActionListener

     private class ButtonHandler implements ActionListener {
         public void actionPerformed(ActionEvent e) {
             if (e.getSource().equals(panic)) {
                 notice.setText("<html><h1>DON'T PANIC!!!!</h1></html>");
             }
         }
     }

     Then create a new ButtonHandler object in the constructor, use the constructor in the addActionListener call on components that generate ActionEvents.

     Creates BasicGUI$ButtonHandler.class
Adding ActionListeners

3. Add an **anonymous inner class** to component that generates ActionEvent that implements ActionListener and write an actionPerformed method where JButton is instantiated:

```java
quit = new JButton("Quit");
quit.setBounds(275, 50, 50, 20);
quit.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e){
        System.exit(0);}});
holder.add(quit);
```

*Has the advantage of putting the code for the generated action with the instantiation of the action generator.*

*Creates many weird-looking .class methods in the directory, eg: BasicGUI$1.class*
Components that generate ActionEvents

JButton, JTextField, JCheckBox, JCheckBoxMenuItem, JComboBox, JMenu, JMenuItem, JRadioButton, JRadioButtonMenuItem

```java
public class JButton {
    public void actionFileEntered() {
        try {
            fileIn = new BufferedReader(new FileReader(getInputJTextField.getText()));
        } catch (IOException ioe) {
            System.out.println(ioe.getMessage());
            System.exit(0);
        }
        resultsJTextArea.setText("Press the " +"Get Next Line" button");
    }
}
```

```java
getInputJTextField = new JTextField();
getInputJTextField.setBounds(10, 145, 100, 20);
getInputJTextField.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent event) {
        actionFileEntered(); // method written below
    }
});
holder.add(getInputJTextField);
```

```
getInputJTextField = new JTextField();
getInputJTextField.setBounds(10, 145, 100, 20);
getInputJTextField.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent event) {
        actionFileEntered(); // method written below
    }
});
holder.add(getInputJTextField);
```

```java
public void actionFileEntered() {
    try {
        fileIn = new BufferedReader(new FileReader(getInputJTextField.getText()));
    } catch (IOException ioe) {
        System.out.println(ioe.getMessage());
        System.exit(0);
    }
    resultsJTextArea.setText("Press the " +"Get Next Line" button");
}
```
Ending of Constructor

End the BasicGUI constructor with lines needed to display JFrame.

```java
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
this.pack();
this.setTitle("BasicGUI");
this.setSize(470, 500);
this.setVisible(true); }
//end of constructor
```

When you run program, an interactions thread is started, so be sure you have a `System.exit(0);` and/or a `setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)`

Stops program if user chooses to close the JFrame using red dot or X
Graphics and Painting

- To create a drawing surface, you should define a subclass of JPanel and override the paintComponent() method.

The parameter g is a Graphics object that is provided by the system when it calls paintComponent. You need a graphics context to do any drawing. Basic shapes are done through the Graphics object.

```java
public class SimpleDraw extends JPanel{
    //definition of paintComponent
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D)g;
        g2.setRenderingHint(
            RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON);
        /* Add a gray rectangle */
        g.setColor(Color.GRAY);
        g.fillRect(100, 100, 100, 100);
        // x,y coord and width and height of rect
```
main and constructor in SimpleDraw

```java
JFrame window;
 JButton quit;
 JLabel words;

public static void main(String[] args) {
    SimpleDraw sd = new SimpleDraw();
}

SimpleDraw() {
    window = new JFrame("Simple!");
    JPanel content = this; //set content to "this" JPanel
    content.setLayout(null);
    window.add(content); // add content to JFrame
    words = new JLabel("Change me"); // blank JLabel
    words.setBounds(220, 150, 100, 40);
    content.add(words);
    quit = new JButton("Quit");
    quit.setBounds(50, 20, 75, 30);
    quit.addActionListener(<anonymous inner class>);
    content.add(quit);
}
```

You can create a GUI that has a drawing component by adding a constructor to set up all the JComponents.
Drawing in GUIs (cont)

• The `paintComponent` method is only called once by the system. If you want to re-call it, use the “repaint()” method.

• To set the color of text, use `name.setForeground(Color)`. This can be called on any component.

• The shapes you can draw are listed in our on-line textbook, in section 6.2.4.

• In a Graphics context, you need to do

  `g.setColor(Color.BLUE);` // or some other color before you draw the object. It can only be set to one color at a time.
Limitations of JPanels

• A JPanel is not a component that can be displayed on its own. That’s why I created the main method in SimpleDraw…to call the zero-parameter constructor of SimpleDraw and create a JFrame to hold the components, including the JPanel.

• Notice that the JFrame and the constructor have no connection to the JPanel where the drawing takes place, but they can occupy the same space.
Adding MouseListeners

• In order for your program to respond to mouse actions, somewhere in the class you need to implement MouseListener or use an anonymous inner class.

• MouseListener requires about 6 different methods to be implemented. You can choose to do only one by adding a MouseAdapter.