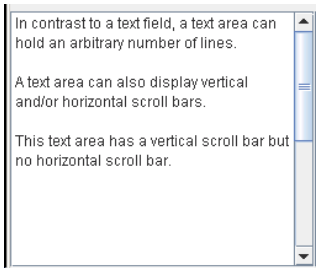
Class 22 Notes
Scrollbars

Scrollbars are often a necessary addition to a text area. Here is a Java text area with a vertical scroll bar. A text area can also have a horizonal scroll bar



You can add a scrollbar to a text area by placing a text area in a ***scroll pane.***

Think of lt like this
Here are two sheets of glass --

And here is a wooden window pane

We can place the glass into the window pane and make a nice useful window that can be raised and closed etc.
But the glass is still what we use to look outside or get sun etc.

OK How we get the scroll bars on a text areas

1. The JTextArea object is like the piece of glass
2. The JScrollPane object is like the window pane
3. We place the JTextArea (glass)into the JScroll Pane (window frame) and we can get scroll bars
4. And we use the text area as before..it just comes with scroll bars now

Here's how you can add scroll bars to a text area.

If you prefer that horizontal and vertical scroll bars appear **only when necessary** (the default), pass a JTextArea reference to JScrollPane and that's all:

```
private JTextArea textArea = new JTextArea();                  // make a text area
JScrollPane scrollpane= new JScrollPane(textArea);             // make a JScrollPane & add  the text area
```

This is the easiest to do and you get the scroll bars as needed by your text area.

Or, you can set the "scroll bar policy" with the following segment:

```
private JTextArea textArea = new JTextArea();
JScrollPane scrollpane = new JScrollPane(textArea, int verticalPolicy, int horizontalPolicy);
```

where verticalPolicy is one of these constants:

- ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED
- ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS
- ScrollPaneConstants.VERTICAL_SCROLLBAR_NEVER

and horizontalPolicy is one of:

- ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED
- ScrollPaneConstants.HORIZONTAL_SCROLLBAR_ALWAYS
- ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER

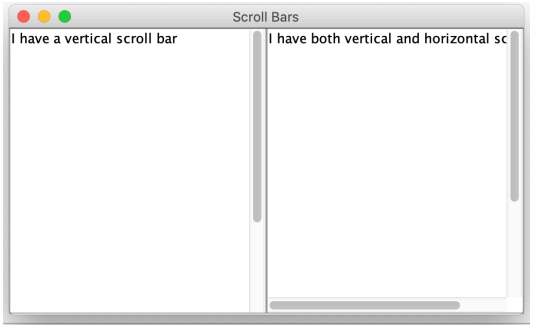
Here's an example:

```
private JTextArea textArea = new JTextArea();
JScrollPane scrollpane= new JScrollPane(textArea,
                                        ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS,
                                        ScrollPaneConstants.HORIZONTAL_SCROLLBAR_ALWAYS);
```

In this case, the scroll bars are always present.
Using the default is easiest and you don't have to use these long constants.

Here is an example that produces a JFrame with two text areas. The first one always has a vertical scroll bar and the second always has both. The text areas are side by side and were put on a panel with a 1 x 2 GridLayout manager. To produce this:

- The text areas are placed in the scroll panes
- the scroll panes are placed in the panel
- the panel is added to thein the JFrame.

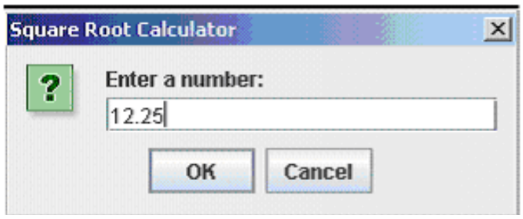


```
import javax.swing.*;
import java.awt.*;
public class Scrolls extends JFrame
{
 JTextArea area1;
 JTextArea area2;

 public  Scrolls()
 {
  super("Scroll Bars");
  setBounds(0,0,500,300);


  area1= new JTextArea("I have a vertical scroll bar",25,20);
  area2 = new JTextArea("I have both vertical and horizontal scroll bars",25,20);

  // Get Scroll Panes for the TextAreas

  JScrollPane pane1 = new JScrollPane(area1,
                            ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS,
                            ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);

  JScrollPane pane2 = new JScrollPane(area2,
                              ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS,
                              ScrollPaneConstants.HORIZONTAL_SCROLLBAR_ALWAYS);

// create  a panel
   JPanel panel = new JPanel(new GridLayout(1,2));

// place the scroll panes – not the text areas--in the panel
   panel.add(pane1);
   panel.add(pane2);

  //add the panel to the frame
   add(panel);

   setVisible(true);
  }
  public static void main(String[] args)
  {
   JFrame f = new Scrolls();
  }
}
```

Dialog Boxes

A *dialog box* is a pop-up window that is used for both input and output.
Here is a dialog box that just gives information--output

Here is one that accepts input



Swing's **JOptionPane** (in swing) class provides the dialog boxes.
We will start with the most simple dialog box: The **message dialog box**

The Message Dialog Box

A message dialog box displays a message and nothing else
Here is a message dialog box. The message displayed is "Password Incorrect."

To incorporate a message dialog box into an application, use the **static** method; of the **JOptionPane** class

public static void showMessageDialog(Component parent, Object message, String title,
int messageType) ;

(This is no different than Math.random(…) or Math.power(..) or SelectionSort.sort(). You are just calling a static method using the class name -- JOptionPane.)

Wow! This method requires has four parameters (I color coded them)

- parent is the parent component of the dialog box. **We will always use null to signify the default component**. So that is easy,

- message is the object that the dialog box displays. For us, message is a string.
  In the above dialog box the message is "Password Incorrect"

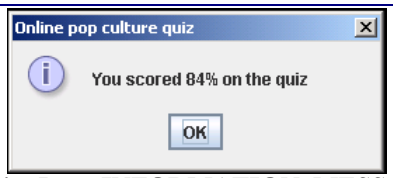
- title is the text displayed on the title bar.
  In the above dialog box the title is "We sell Everything"

- messageType is one of the following constants :
  - JOptionPane.ERROR_MESSAGE (numerical value: 0)
  - JOptionPane.INFORMATION_MESSAGE (numerical value: 1)
  - JOptionPane.PLAIN_MESSAGE (numerical value: –1)
  - JOptionPane.WARNING_MESSAGE (numerical value: 2)
  - JOptionPane.QUESTION_MESSAGE (numerical value: 3)

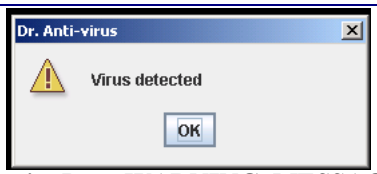
  (You can use the name of the constant or its numerical value. For example, you can use the constant name JOptionPane.ERROR_MESSAGE) or its value -- 0)

  Depending on the message type you choose Java will display a different icon (Picture).

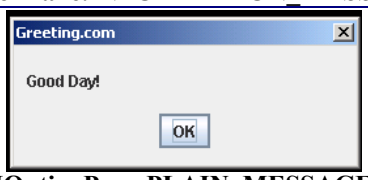
Here are the icons that java uses for messages (on a Windows system, Apple looks different):



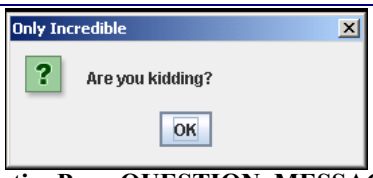
JOptionPane.INFORMATION_MESSAGE



JOptionPane.WARNING_MESSAGE



JOptionPane.PLAIN_MESSAGE



JOptionPane.QUESTION_MESSAGE

Four message dialog boxes

Here is an example of a program using a message dialog box. To display a message box, you just call the method.
THERE ARE NO EVENTS HERE. Just call the method, with its four parameters.
You can call the method from any program – graphical or text based. In the following example
I use a try-catch:

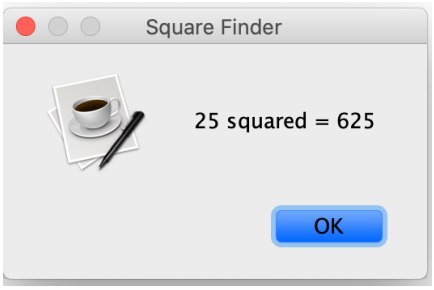
- 'the scanner reads an integer as a string
- parseInt returns its numerical value
- the number is squared and displayed in a dialog box

but

- if the input is bad (e.g. 345Y7) the catch block displays an error dialog box.
  Notice each of the parameters (color coded)

```
import java.util.*;
import javax.swing.*;

public class MessageBox
{
 public static void main(String[] args)
 {
  Scanner input = new Scanner(System.in);
  String num="";
  try
  {
   System.out.print("Enter an  integer ");
    num = input.next();
   int value = Integer.parseInt(num);
   int square = value*value;

   // notice the four color coded parameters
   JOptionPane.showMessageDialog(null,     num +" squared = "+square,
                                 "Square Finder",     JOptionPane.INFORMATION_MESSAGE);
  }

  catch( NumberFormatException e)
  {
    JOptionPane.showMessageDialog(null,    "Bad input: "+num,
                                  "Square Finder",     JOptionPane.ERROR_MESSAGE);
  }
 }
}
```

The first time I ran the program I entered 25 and this message popped up:

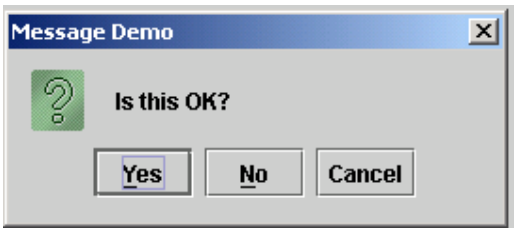


The second time I ran the program I entered 23XY5 (catch block executed) and this box popped up:

The pictures displayed depend on your system. I ran the program on a MacBook

Confirmation Dialog Box

A *confirmation dialog box* displays a question, expects a simple reply such as yes, no or ok, and returns the reply an integer.



A confirmation dialog box

Like a message dialog box, you can use a confirmation dialog box by calling a static method of JOptionPane:

```
public static int showConfirmDialog(Component parent, Object message, String title,
                                    int optionType, int messageType);
```

Here there are five parameters that you supply to the method. They are the same as the message dialog except for the new one –**optionType.**

The optionType parameter determines the options that appear as buttons. The parameter is one of the following JOptionPane constants – or just the numerical value:

- YES_NO_OPTION (numerical value: 0)
- YES_NO_CANCEL_OPTION (numerical value: 1) **//as in the picture above**
- OK_CANCEL_OPTION (numerical value: 2)

Like the message dialog box the messageType parameter can be one of the following constants:

- ERROR_MESSAGE
- INFORMATION_MESSAGE
- PLAIN_MESSAGE
- WARNING_MESSAGE
- QUESTION_MESSAGE

The **return value, an integer, is one of the following constants -- just a named integer:**

- CANCEL_OPTION (numerical value: 2) // if user clicked Cancel
- CLOSED_OPTION (numerical value: –1) // dialog closed without choosing one of the options)
- NO_OPTION (numerical value: 1) // user clicked no button
- OK_OPTION (numerical value: 0) // user clicked OK
- YES _OPTION (numerical value: 0) // user clicked yes

So if in the above picture the user clicked the Yes button the method would return 0 (Yes_Option has value 0); If the user closed the dialog box by clicking the x in the upper right hand corner the method returns CLOSED_OPTION (that is -1)

The following program uses the confirmation dialog box and the message dialog box to ask a question and get a reply. The figures after the program shows the results after running the program twice. The first time, I clicked yes and the second time I clicked no

Important: The Confirmation box **RETURNS A VALUE. That value is one of the constants above, an int**

```
import java.util.*;
import javax.swing.*;

public class ConfirmationMessage
{
 public static void main(String[] args)
 {
  Scanner input = new Scanner(System.in);

 // returns an int that is stored in answer
   int answer = JOptionPane.showConfirmDialog
                (null,     "Are you having a good day?",
                "Greeting",      JOptionPane.YES_NO_CANCEL_OPTION,
                  JOptionPane.QUESTION_MESSAGE);


    if (answer == JOptionPane.NO_OPTION)                          // if answer ==1
     JOptionPane.showMessageDialog(null, "Sorry about that", "Greeting",
                          JOptionPane.PLAIN_MESSAGE );

     else if (answer == JOptionPane.YES_OPTION)                   // if answer == 0
      JOptionPane.showMessageDialog(null,"I'm glad to hear that!",
             "Greeting", JOptionPane.PLAIN_MESSAGE );
 }
}
```

The first time I ran the program I clicked the yes button, The second time "no."
Notice I used a PLAIN option on the responses—no icon

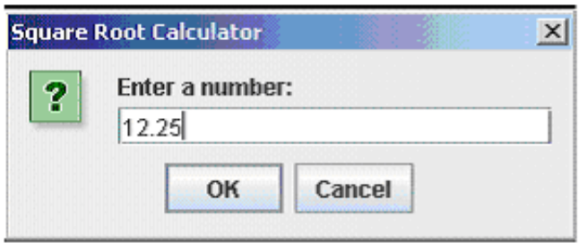
The method (with all its glorious parameters) returns an int that is stored in the variable answer.
I could use

```
if (answer ==1) or
if (answer == NO_OPTION)
```

no difference

Input Dialog Box

An *input dialog box* can be used to obtain **String input** from a user. Even though there is a double in the picture below it is a String. So



The available JOptionPane method in vibrant technicolor is:

public static **String** showInputDialog(Component parent, Object message,
String title, int messageType)

The parameters have the same meaning as the parameters of the message dialog box and the confirmation dialog box.

In the above picture

- the parent is null,
- the message is "Enter a number" ,
- the title is "Square Root Calculator,
- and the message type is OK_OPTION
- the returned value is the String "12.25"

You can substitute an input dialog box for a Scanner object in any of the interactive programs that we have written before. However, unlike the Scanner methods nextInt() or nextDouble(), **an input dialog box always returns a String,** which can be converted to a numerical value, if necessary.

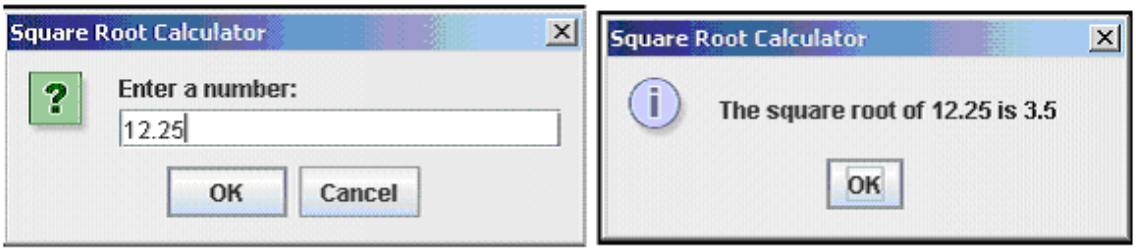
The string returned by an input dialog box is either the string supplied by the user, or null if the user chooses *Cancel* or closes the dialog box.

Here is a short program that uses the input dialog box to get a number from the user and displays the square root of the number in a message dialog box. If the input is bad it is handled by a catch block.

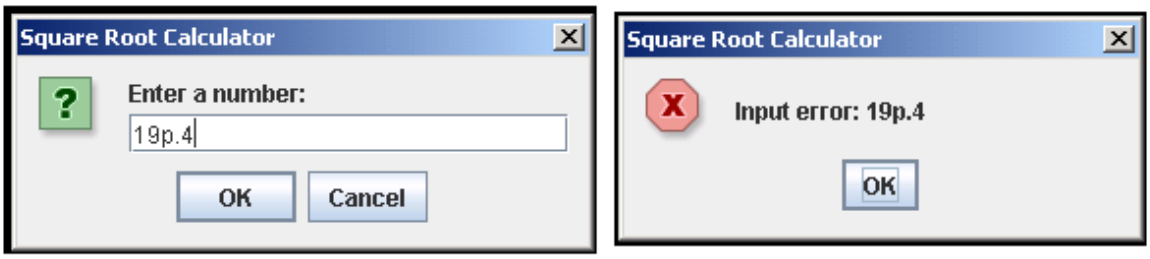
```
import javax.swing.*;
public class SquareRootCalculator
{
 public static void main(String[] args)
 {
  // get input  from the user via an input dialog box; input is returned as a String reference

    String numberString = JOptionPane.showInputDialog(null,
                                "Enter a number:", "Square Root Calculator",
                                   JOptionPane.QUESTION_MESSAGE);



    if (numberString != null)        // Cancel or Close option returns null
    {
          // convert  numberString to double
          try    // if parseDouble() throws an exception
          {
                double number = Double.parseDouble(numberString);

                // display result with a message dialog box
                 JOptionPane.showMessageDialog( null,
                                          "The square root of " + number + " is " + Math.sqrt(number),
                                           "Square Root Calculator",
                                            JOptionPane.INFORMATION_MESSAGE);
          }
         catch ( NumberFormatException e)
         {
                JOptionPane.showMessageDialog( null, "Input error: " + numberString,
                  "Square Root Calculator",  JOptionPane.ERROR_MESSAGE);
          }
    }
}
```

Notice the dialog box methods take four parameters. It looks intimidating but it is always the same.

Here are the results of running the program twice—once with good data and then with bad data



(a) Correct Input



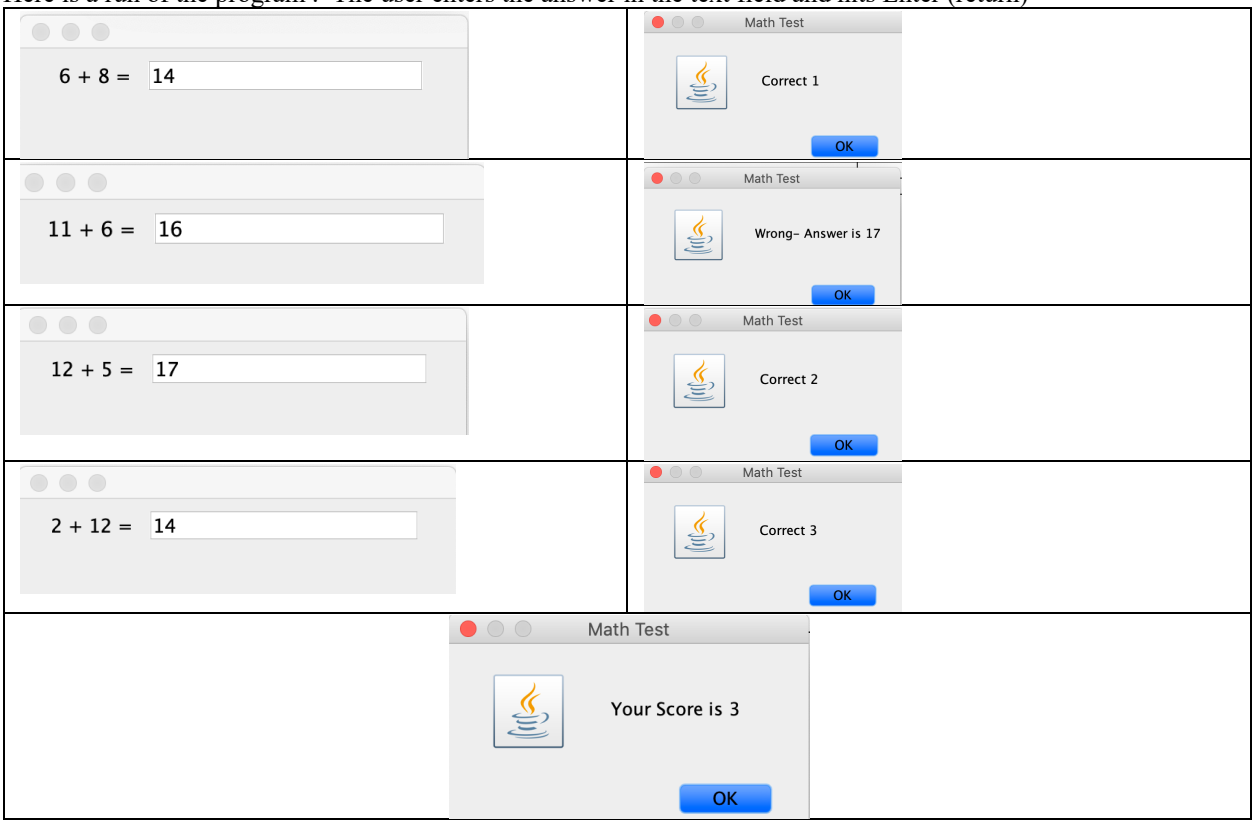
(b) Incorrect Input

}

The dialog boxes do not involve events. There are no listeners. They are used only for input and output. Th
You can use an input dialog box in a program where you might use a scanner.

Example: Here is a small addition quiz for a child. There are four questions. Each answer is marked correct or wrong and after four questions, a final grade (out of 4) is given.

Here is a run of the program . The user enters the answer in the text field and hits Enter (return)



```
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
public class MathTest extends JFrame
{
   private JTextField answer;
   private JLabel lb;
   private int score = 0;
   private int question = 1;
   private Random r = new Random();
   int num1, num2;
 public MathTest()
 {
        setBounds (200,200, 300,100);
        JPanel p = new JPanel();
        answer =new JTextField(15);

        // make up the addition problem
        num1 = r.nextInt(12) + 1;
        num2 = r.nextInt(12) + 1;
        // put the problem on a label
         lb = new JLabel(""+num1+ " + "+ num2+ " = ");

        p.add(lb);              // add the label to the panel
        p.add(answer);          // add the text field to the panel
         add(p);                // add the panel to the frame
         setVisible(true);
         answer.addActionListener(new TextFieldListener()); // register the text field
 }
 private class TextFieldListener implements ActionListener
 {
  public void actionPerformed(ActionEvent e)
  {
   if (e.getSource() ==answer)  // The "Enter/Return key" was hit in the text field
   {
          try
          {
            int ans = Integer.parseInt(answer. getText());  // must convert to number
            if (ans == (num1+num2))
            {
               score++;
             JOptionPane.showMessageDialog(null,"Correct "+ score,
             "Math Test",JOptionPane.INFORMATION_MESSAGE );
            }
```

```
            else
            {
              JOptionPane.showMessageDialog(null,"Wrong- Answer is "+(num1+num2),
               "Math Test",JOptionPane.INFORMATION_MESSAGE );
              }
            if (question < 4)  // the quiz has four questions
            {
                answer.setText("");   // for the next question
                num1 = r.nextInt(12) + 1;
                num2 = r.nextInt(12) + 1;
                lb.setText(""+num1+ " + "+ num2+ " = ");
                question++;
            }
           else
           {
                    JOptionPane.showMessageDialog(null,"Your Score is "+ score,
                   "Math Test",JOptionPane.INFORMATION_MESSAGE );
                    System.exit(0);
          }
  }
  catch (NumberFormatException ex)
  {
        tf.setText("Illegal number format");
  }
  }
  }
}
 public static void main(String[] args)
 {
     MathTest mt = new MathTest();
 }
}
```

Another type of input dialog box

.

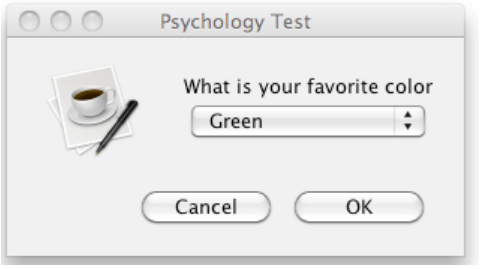
Input dialog box with a list of options

The JOptionPane method that displays an option dialog is:

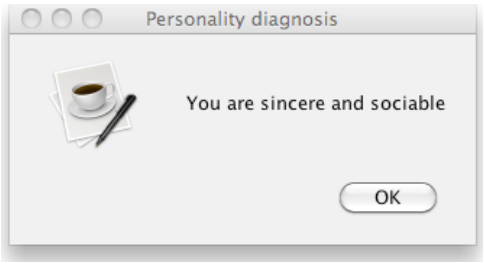
```
public static Object showInputDialog( Component parent,
                                      Object message,
                                      String title,
                                      int messageType,
                                      Icon icon,
                                      Object [] options,
                                      Object selected)
```

- The array, ***options***, is a list of choices that appears in the drop-down box.
- This array can be an array of references to objects of any class, but is usually an array of String references.
- The parameter selected gives the values that initializes the input. For the dialog box shown , selected is *Green*. The value of selected can be null.
- The value of icon can also be null.
- The return value belongs to Object and is usually cast to String.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class Mood
{

 public static void main(String[] args)
 {

  String[] colors = {"Yellow", "Green", "Blue", "Red", "Orange"};   // options array

  String choice = (String)JOptionPane.showInputDialog
                                  (      null,
                                        "What is your favorite color",
                                         "Psychology Test",
                                          JOptionPane.QUESTION_MESSAGE,
                                         null,
                                         colors,
                                        "Green"      );

  if (choice !=null)                // can be null if user cancels or closes dialog box
  {
     String personality= new String();
     if ( choice .equals("Blue"))
        personality = " You are calm and compassionate";
     else if ( choice .equals("Green"))
        personality = " You are sincere and sociable";
     else if ( choice .equals("Yellow"))
        personality = " You are wise with a good business sense";
     else if ( choice .equals("Red"))
        personality = " You are outgoing and ambitious";
     else if ( choice .equals("Orange"))
        personality = " You are flamboyant and dramatic";
     JOptionPane.showMessageDialog(
                                             null,
                                            personality,
                                            "Personality  diagnosis",
                                             JOptionPane.INFORMATION_MESSAGE);
  }
 }
}
```



Input dialog (with a dropdow menu)



Message