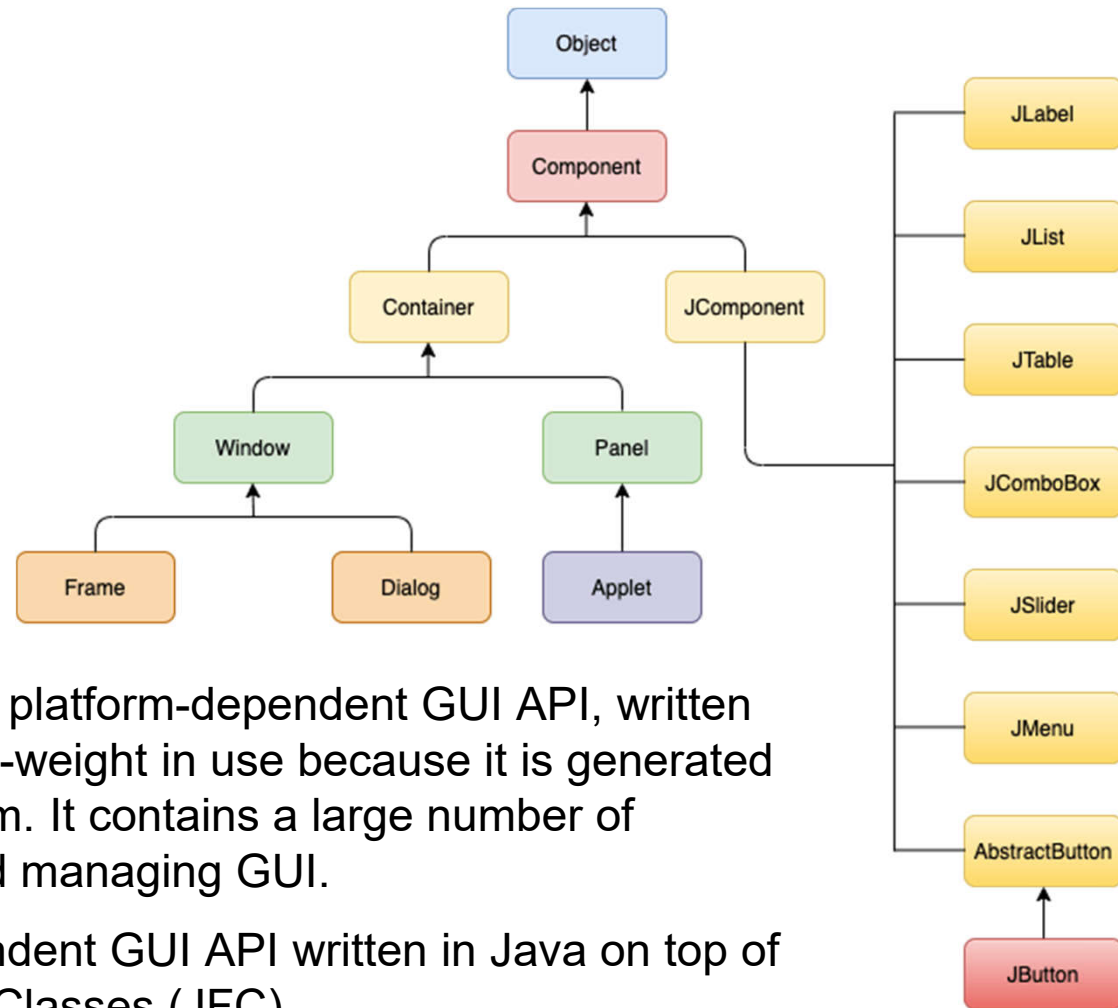


Graphic User Interface (GUI)

- ☐ AWT and Swing
- ☐ Swing Components
- ☐ Event-Driven Programming
- ☐ Layout

Introduction

- Command-line user interface is boring, modern applications use graphical components such as: windows, buttons, text boxes, and menus,...
- Java provides several API for the GUI development:
 - AWT (Abstract Windowing Toolkit): a platform-dependent GUI API, written in C and released in 1995. It is heavy-weight in use because it is generated by the system's host operating system. It contains a large number of classes and methods for creating and managing GUI.
 - Swing: a lightweight platform-independent GUI API written in Java on top of AWT. It is a part of Java Foundation Classes (JFC).



AWT Hello World

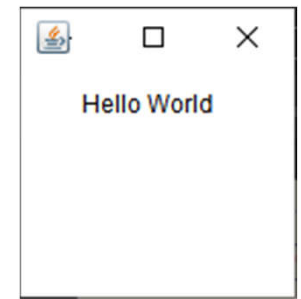
```
import java.awt.*;
import java.awt.event.*;

public class AwtExample extends Frame {
    public AwtExample() {
        setTitle("Hello World AWT");
        setLayout(new FlowLayout());
        setSize(150, 150);
        setVisible(true);

        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent we) {
                dispose();
            }
        });

        Label label = new Label("Hello World");
        add(label);
    }

    public static void main(String args[]) {
        new AwtExample();
    }
}
```



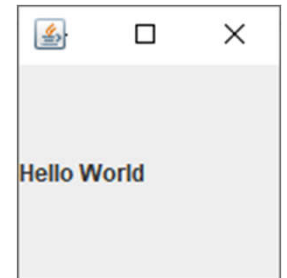
Swing Hello World

```
import javax.swing.*;

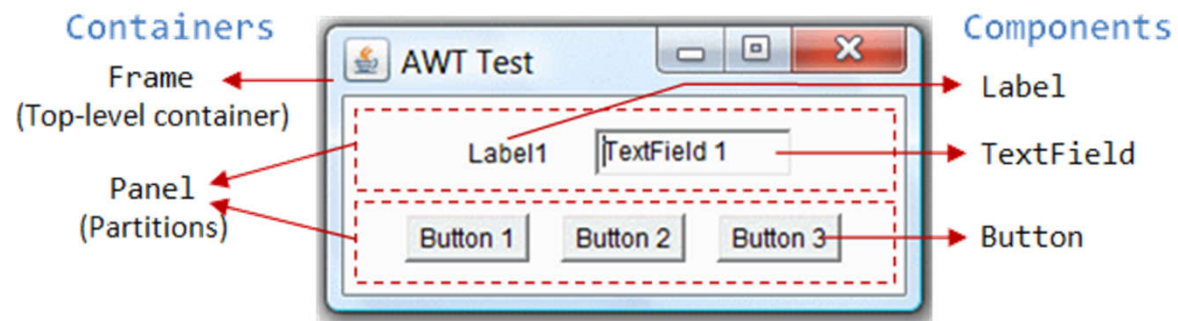
public class HelloWorldSwing {
    private static void createAndShowGUI() {
        JFrame frame = new JFrame("Hello World Swing");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(150, 150);
        frame.setVisible(true);

        JLabel label = new JLabel("Hello World");
        frame.getContentPane().add(label);
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                createAndShowGUI();
            }
        });
    }
}
```



Terminology



- Window: a first-class citizen of the graphical desktop
 - Also called a top-level container
 - Examples: frame, dialog box
- Component: a GUI widget that resides in a window
 - Also called controls in many other languages
 - Examples: button, text box, label
- Container: a logical grouping for storing components
 - Examples: panel, box



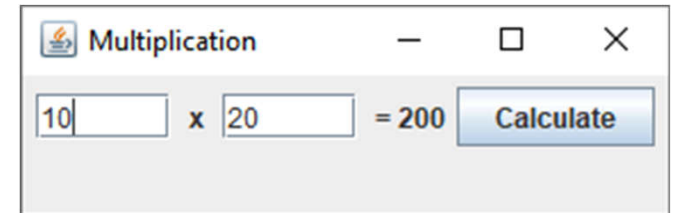
Basic Components

Creating Basic Components

```
Container ctn = frame.getContentPane();
JTextField xField = new JTextField("10"),
    yField = new JTextField("20");
JLabel result = new JLabel(" = 200");
xField.setColumns(5);
yField.setColumns(5);
```

```
ctn.add(xField);
ctn.add(new JLabel(" x "));
ctn.add(yField);
ctn.add(result);
```

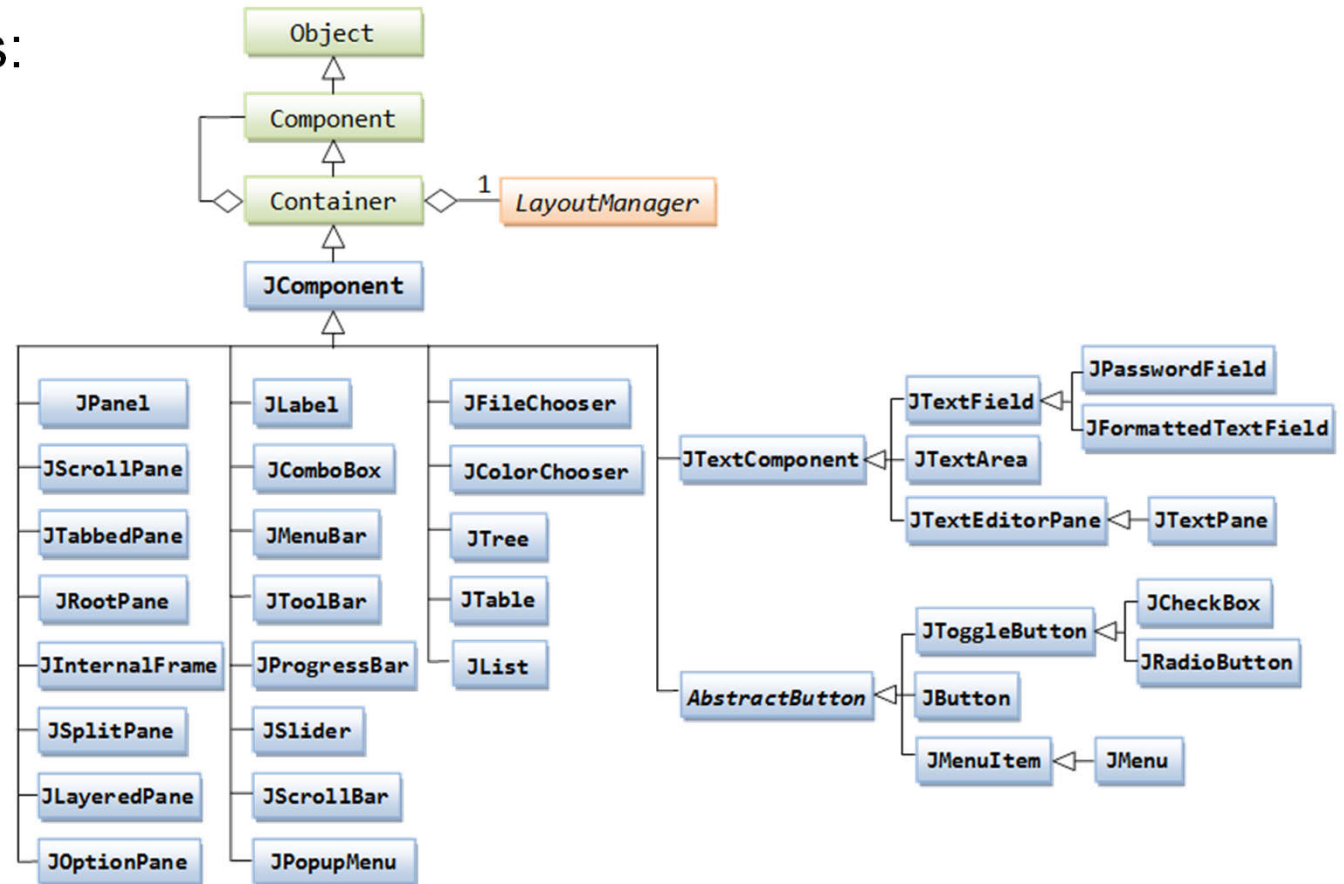
```
JButton calculateBtn = new JButton("Calculate");
ctn.add(calculateBtn);
```



Other Components

- Most common ones:

- JLabel
- JButton
- JCheckBox
- JRadioButton
- JPasswordField
- JFormattedTextField
- JTextArea
- JList
- JComboBox
- JSlider



Using Image Icons

- Many components (e.g., `JLabel`, `JButton`) support a text label and an image icon



```
URL url = HelloWorldSwing.class.getClassLoader()
    .getResource("image/calculator.png");
ImageIcon icon = new ImageIcon(url);
ImageIcon scaledIcon = new ImageIcon(icon.getImage()
    .getScaledInstance(32, 32, Image.SCALE_SMOOTH));
JButton calculateBtn = new JButton("Calculate", scaledIcon);
ctn.add(calculateBtn);
```

Component Appearance

- `JComponent`
 - `setBackground(Color bgColor)`
 - `setForeground(Color fgcolor)`
 - `setFont(Font font)`
 - `setBorder(Border border)`
 - `setPreferredSize(Dimension dim)`
 - `setMaximumSize(Dimension dim)`
 - `setMinimumSize(Dimension dim)`
- `JLabel` and buttons
 - `setText(String strText)`
 - `setIcon(Icon defaultIcon)`
 - `setHorizontalAlignment(int alignment)`
 - `setVerticalAlignment(int alignment)`
 - `setHorizontalTextPosition(int textPosition)`
 - `setVerticalTextPosition(int textPosition)`



Event Handling

Event-Driven Programming

- In a console program, prompts are written to the console, and the user responds to the prompts → The order of the prompts is determined by the program
- In a GUI program, the order of the events is determined by the user, not the program
 - Event-driven programming is a paradigm in which the flow of the program is determined by events such as user actions (mouse clicks, key presses), sensor outputs, or message passing from other programs or threads

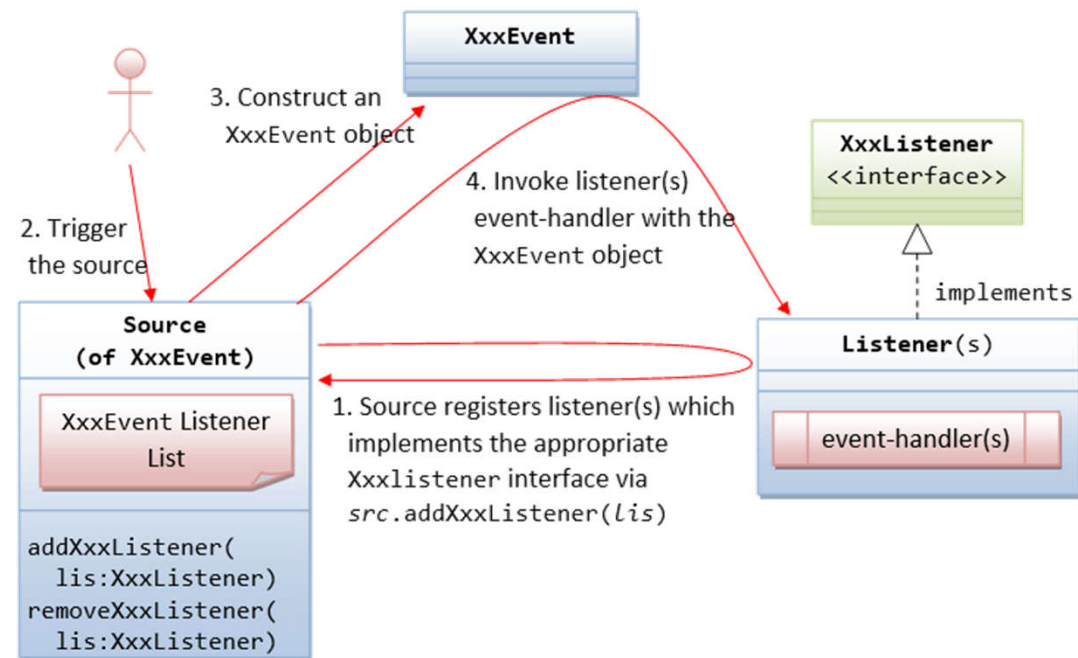
Events

- The user may interact in many different ways:

- Clicking on a button to choose a program option
- Making a choice from a menu
- Entering text in a text field
- Dragging a scroll bar
- ...

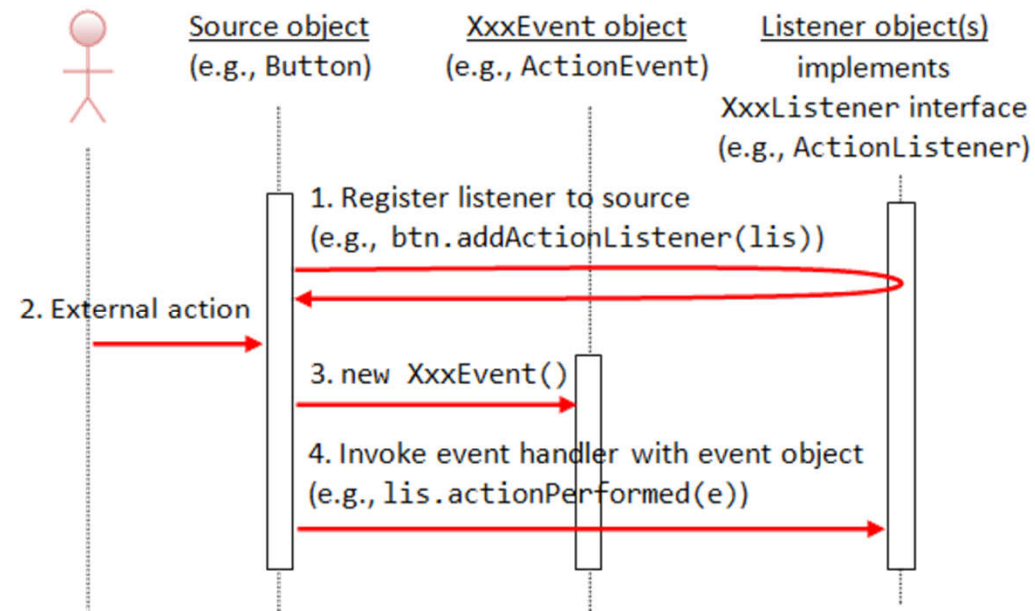
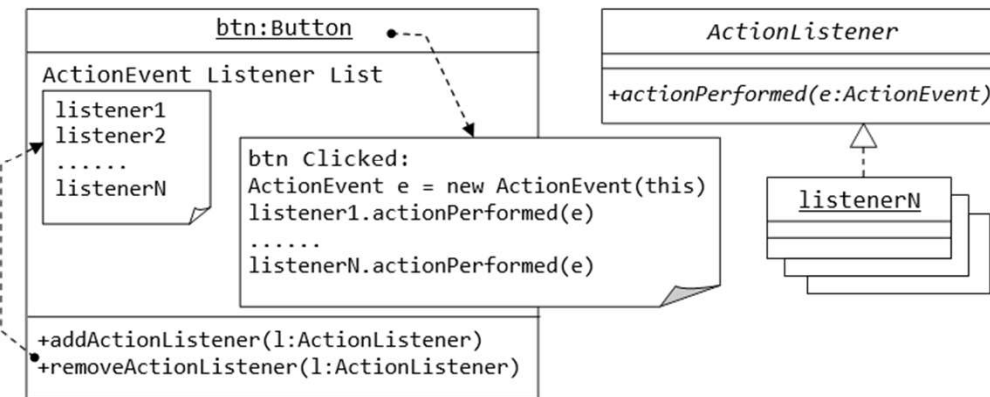
- An action like that is called an event

- The application needs to listen to these events and response to them by using event handlers



Event Handling

```
calculateBtn.addActionListener(new ActionListener() {  
    @Override  
    public void actionPerformed(ActionEvent event) {  
        System.out.println("Calculate button clicked");  
    }  
});
```



Getting and Setting Values

```
public void actionPerformed(ActionEvent event) {  
    try {  
        double x = Double.parseDouble(xField.getText());  
        double y = Double.parseDouble(yField.getText());  
        String resultStr = new DecimalFormat("0.###").format(x * y);  
        result.setText(" = " + resultStr);  
  
    } catch (NumberFormatException e) {  
        result.setText("Incorrect value(s)");  
    }  
}
```

Swing Timers

- A Swing timer (`javax.swing.Timer`) fires one or more action events after a specified delay
 - It works in EDT (Event Dispatch Thread – to be later explained) and use the event mechanism, so it is safe to update UI
 - It helps to avoid creating EDT-unfriendly threads
- Example:
 - ```
new Timer(10000, (ActionListener) new ActionListener() {
 @Override
 public void actionPerformed(ActionEvent e) {
 // can update UI from here!
 }
}).start();
```



# Custom Events

- ```
class MyEvent extends EventObject {  
    public MyEvent(Object source) {  
        super(source);  
    }  
}
```
- ```
interface MyEventListener extends EventListener {
 public void myEventOccurred(MyEvent evt);
}
```
- ```
class MyClass {  
    protected EventListenerList listenerList = new EventListenerList();  
  
    void addMyEventListener(MyEventListener listener) {  
        listenerList.add(MyEventListener.class, listener);  
    }  
    void removeMyEventListener(MyEventListener listener) {  
        listenerList.remove(MyEventListener.class, listener);  
    }  
    void fireMyEvent(MyEvent evt) {  
        Object[] listeners = listenerList.getListenerList();  
        for (int i = 0; i < listeners.length; i = i+2) {  
            if (listeners[i] == MyEventListener.class)  
                ((MyEventListener) listeners[i+1]).myEventOccurred(evt);  
        }  
    }  
}
```

Exercise

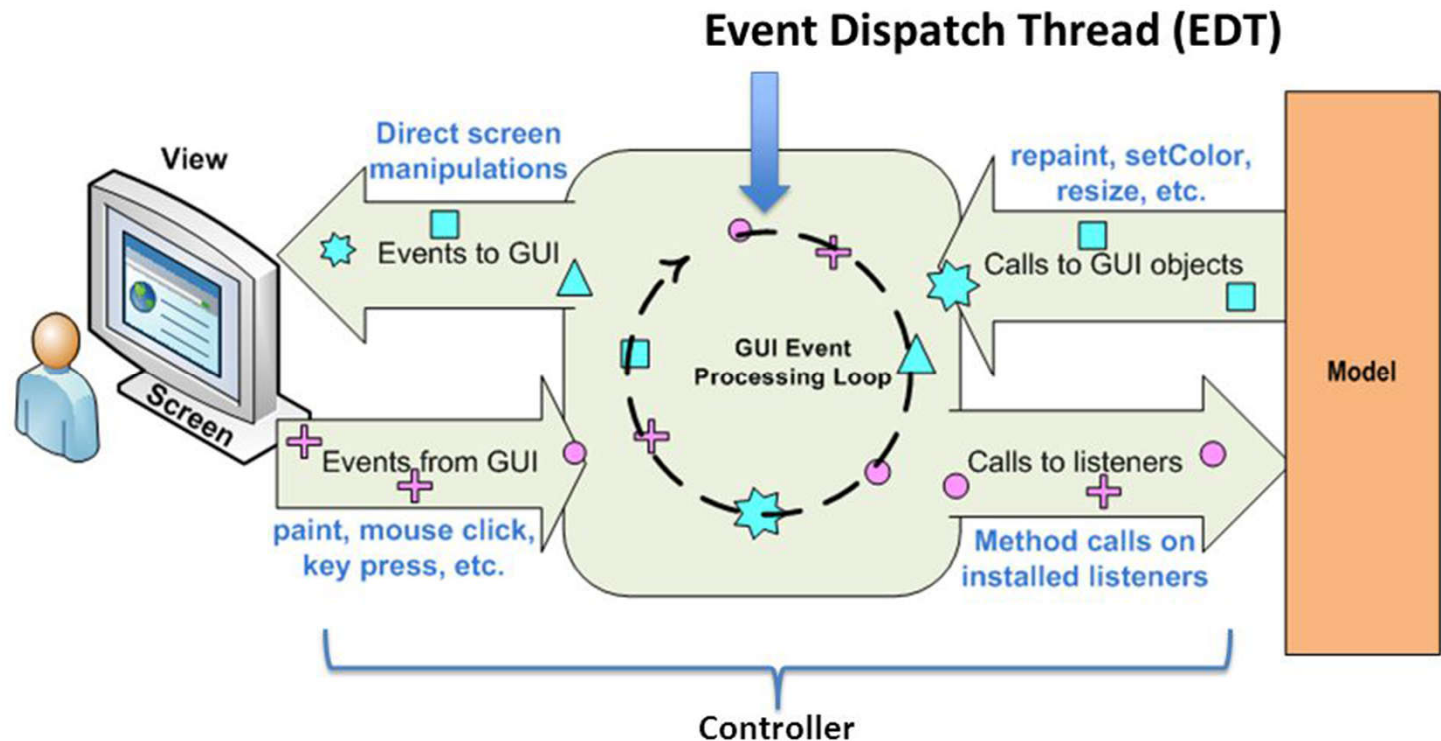
- Try to work with other Swing components
 - Create component instances
 - Listen to events



Multithreaded UI

Event Dispatch Thread (EDT)

- Swing event handling code runs on a special thread known as the event dispatch thread
- Event handlers must finish quickly, or the UI becomes unresponsive



Long Tasks

- A long task should be run asynchronously in a separate thread so that it does not block the UI
- Example:

```
○ JButton startTask = new JButton("Start");
startTask.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        Thread newThread = new Thread() {
            @Override
            public void run() {
                // Long task to do...
            }
        };
        newThread.start();
    }
});
```

- Problem: What if we now want to make update to the UI from the new thread (e.g., updating a progress bar)?

Thread Safety

- Most Swing methods are not thread safe, and must be invoked only from EDT
 - Only a few methods in Swing are thread safe
- Some thread-safe methods:
 - `JComponent`:
 - `repaint()`
 - `invalidate()`
 - `revalidate()`
 - `JTextField`:
 - `setText()`
 - `JTextArea`:
 - `setText()`
 - `insert()`
 - `append()`
 - `replaceRange()`

SwingUtilities.invokeLater() Method

- How to request a non thread-safe update for a component from some thread other than EDT?
- `invokeLater()` allows to perform a task on EDT asynchronously:

```
◦ Thread newThread = new Thread() {  
    @Override  
    public void run() {  
        for (int i = 0; i < n; i++) {  
            // Do a sub-task...  
  
            SwingUtilities.invokeLater(new Runnable() {  
                public void run() {  
                    // Update the progress bar... (will be executed in EDT)  
                }  
            });  
        }  
    }  
};
```

- Remind that the Swing startup code is put inside `invokeLater()` method

SwingUtilities.invokeLaterAndWait() Method

- `invokeAndWait()` allows to perform a task on EDT synchronously:

```
    Thread newThread = new Thread() {
        @Override
        public void run() {
            for (int i = 0; i < n; i++) {
                // Do a sub-task...

                SwingUtilities.invokeLaterAndWait(new Runnable() {
                    public void run() {
                        // Update the progress bar...
                        // This will be executed from the EDT
                    }
                });
            }
        }
    };
```

- Unlike `invokeLater()` which can be called from EDT, doing so with `invokeAndWait()` will result in a deadlock

Exercise

- Complete the long-task example: make the child thread updates the GUI every 3 seconds with some new information (such as the current time)



Layouts

Layout Manager

- A layout manager is a Java object associated with a particular container component ([JFrame](#) or [JPanel](#)), and controls the contained components
 - Example: If a frame holds a panel, and the panel holds a button, the panel's layout manager control the size and placement of the button, and the frame's layout manager controls the size and placement of the panel, and the button does not need a layout manager
- Different background components have different types of layout managers, different layout managers have different layout policies
- Commonly used layout managers:
 - [BorderLayout](#): default for a [JFrame](#)
 - [FlowLayout](#): default for a [JPanel](#)
 - [BoxLayout](#)
 - [GridLayout](#)
 - [GridBagLayout](#)
 - [CardLayout](#)
 - [SpringLayout](#)

BorderLayout

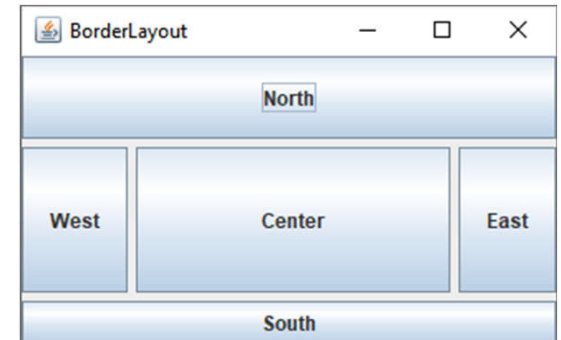
- The container is divided into 5 zones: **WEST** (or **LINE_START**), **EAST** (or **LINE_END**), **SOUTH** (or **PAGE_END**), **NORTH** (or **PAGE_START**), and **CENTER**
- One needs not to place components to all the 5 zones, as others will fill any space left over:
 - The **NORTH** and **SOUTH** components may be stretched horizontally
 - The **EAST** and **WEST** components may be stretched vertically
 - The **CENTER** component may stretch both horizontally and vertically

- Example:

```
BorderLayout layout = new BorderLayout(5, 5);  
frame.setLayout(layout);
```

```
Container ctn = frame.getContentPane();  
JButton btnNorth, btnSouth, btnCenter, btnEast, btnWest;  
btnNorth = new JButton("North");   ctn.add(btnNorth, BorderLayout.NORTH);  
btnSouth = new JButton("South");   ctn.add(btnSouth, BorderLayout.SOUTH);  
btnCenter = new JButton("Center"); ctn.add(btnCenter, BorderLayout.CENTER);  
btnEast = new JButton("East");     ctn.add(btnEast, BorderLayout.EAST);  
btnWest = new JButton("West");     ctn.add(btnWest, BorderLayout.WEST);
```

```
btnNorth.setPreferredSize(new Dimension(100, 50));
```



FlowLayout

- Behavior is similar to a text editor, except it works with components instead of words
 - Each component has the size it wants to be, and they are laid out left to right in the order that they are added
 - When a component won't fit horizontally, it drops to the next row in the layout
- Example:

```
FlowLayout layout = new FlowLayout();  
layout.setAlignment(FlowLayout.RIGHT);  
layout.setHgap(10);  
layout.setVgap(20);  
frame.setLayout(layout);
```



GridLayout

- Components are arranged in a grid (matrix) of rows and columns inside the container
- Components are added in a left-to-right, top-to-bottom manner in the order they are added
- Example:
 - `// rows, cols, hgap, vgap`
`GridLayout layout = new GridLayout(3, 4, 10, 15);`
`frame.setLayout(layout);`



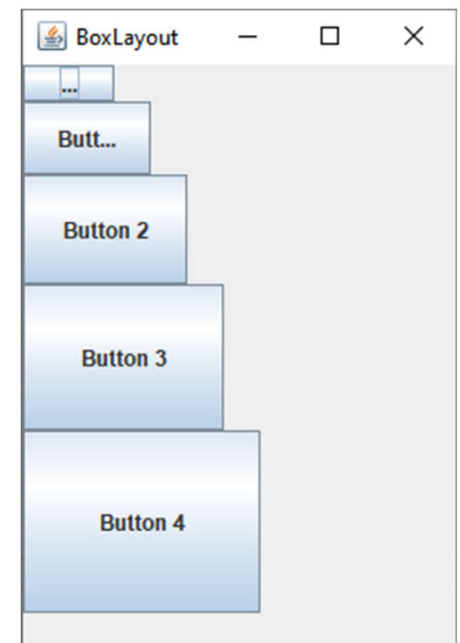
BoxLayout

- Components are put in a single row or column, respecting the components' requested minimum, maximum and preferred sizes

- Example:

```
Container ctn = frame.getContentPane();
BoxLayout layout = new BoxLayout(ctn,
    BoxLayout.Y_AXIS);
frame.setLayout(layout);

for (int i = 0; i < 5; i++) {
    JButton btn = new JButton(
        String.format("Button %d", i));
    btn.setMinimumSize(new Dimension(0, 0));
    btn.setMaximumSize(new Dimension
        (i * 20 + 50, i * 20 + 20));
    ctn.add(btn);
}
```



Absolute Positioning

- It's possible to use absolute position instead of a layout manager by invoking method `setLayout(null)`, then position the components by:

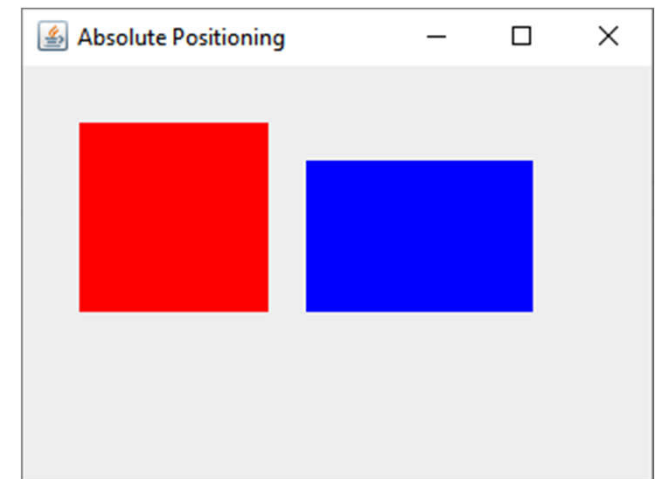
- `setBounds(int xTopLeft, int yTopLeft, int width, int height)`

- Example:

- `Container cp = frame.getContentPane();`
`cp.setLayout(null);`

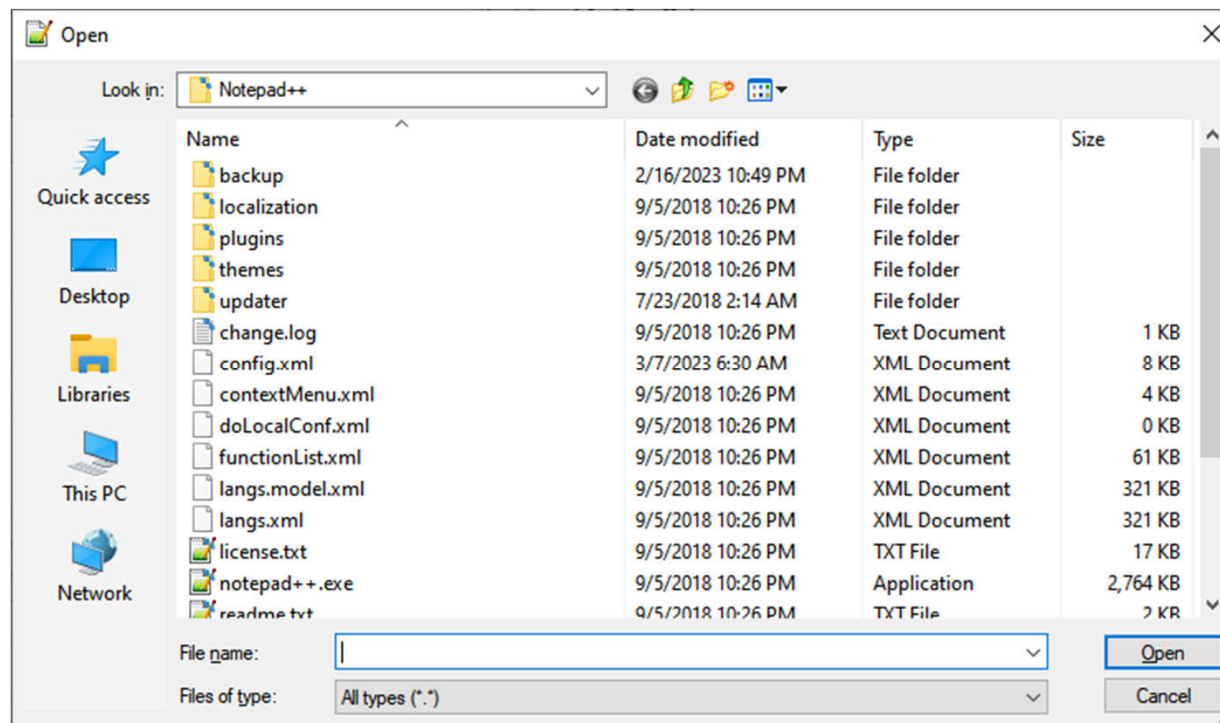
```
JPanel p1 = new JPanel();  
p1.setBounds(30, 30, 100, 100);  
p1.setBackground(Color.RED);  
cp.add(p1);
```

```
JPanel p2 = new JPanel();  
p2.setBounds(150, 50, 120, 80);  
p2.setBackground(Color.BLUE);  
cp.add(p2);
```



Mixed Layouts

- It's usually necessary to combine multiple layouts together with containers (such as `JPanel`) for a good GUI



Exercise

- Relayout the components of the calculator example

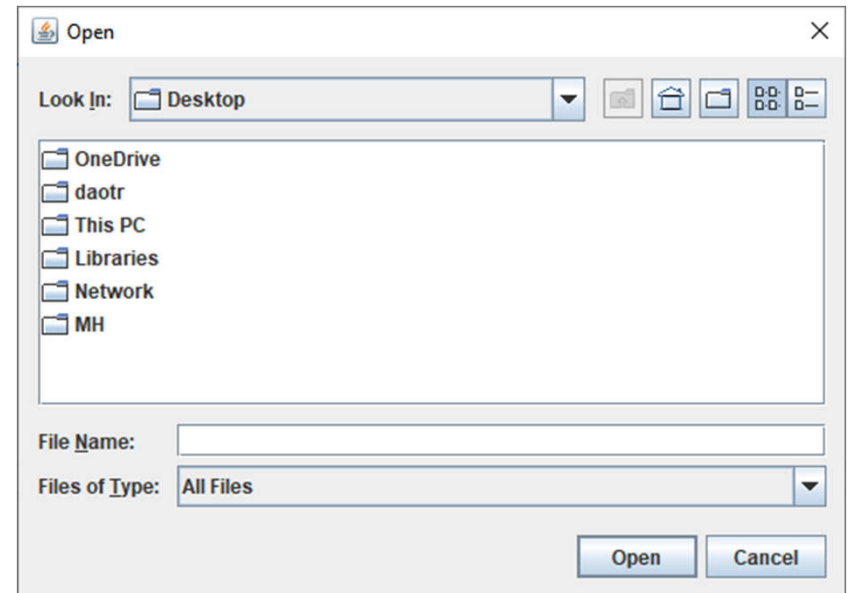


Dialogs

Dialogs

- Dialog is an independent sub window meant to carry temporary notice apart from the main window
- Example:
 - ```
JDialog dlg = new JDialog(frame, "Dialog Example");
dlg.setLayout(new FlowLayout());
dlg.add(new JLabel("This is a dialog box"));
dlg.add(new JButton("Button"));
dlg.setSize(100, 100);
dlg.setVisible(true);
```

# JFileChooser



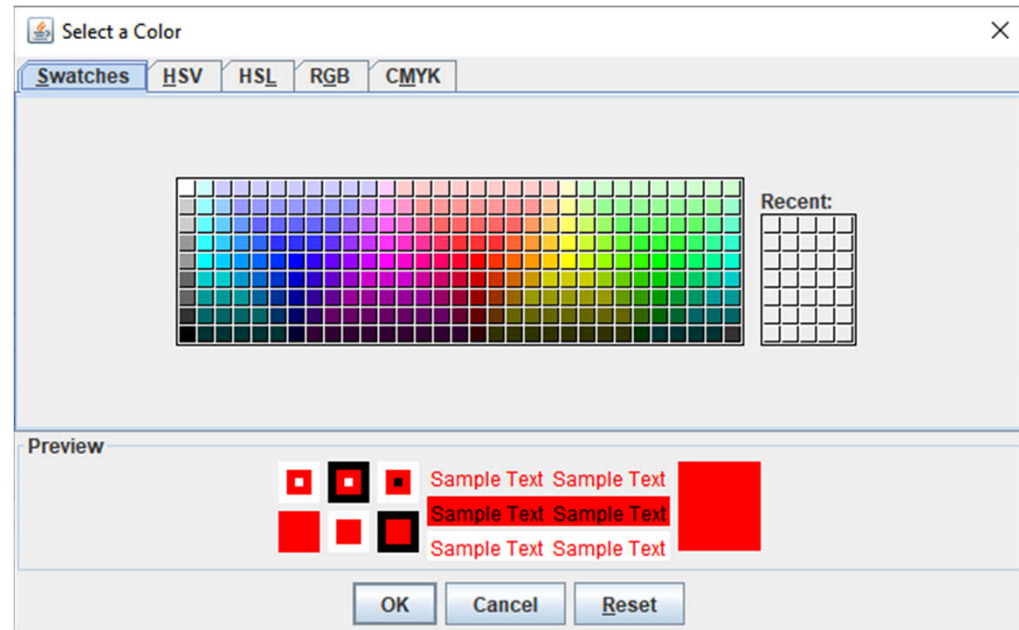
- Example

```
String homeDir = FileSystemView.getFileSystemView().getHomeDirectory();
JFileChooser jfc = new JFileChooser(homeDir);

int returnValue = jfc.showOpenDialog(null); // or: showSaveDialog

if (returnValue == JFileChooser.APPROVE_OPTION) {
 File selectedFile = jfc.getSelectedFile();
 System.out.println(selectedFile.getAbsolutePath());
}
```

# JColorChooser



- Example:
  - ```
Color initialColor = Color.RED;  
Color color = JColorChooser.showDialog(  
    frame, "Select a Color", initialColor);  
System.out.println(color);
```



Option Panes

JOptionPane

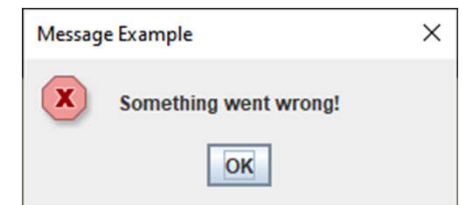
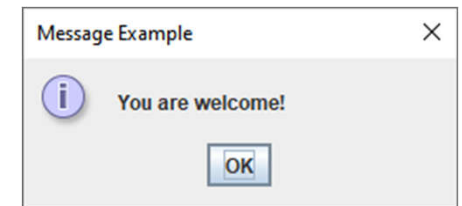
- `JOptionPane` makes it easy to pop up a standard dialog box that prompts users for a value or informs them of something
- Most common dialogs:

Method	Description
<code>showMessageDialog()</code>	Tells the user about something that has happened
<code>showConfirmDialog()</code>	Asks a confirming question, like yes/no/cancel
<code>showInputDialog()</code>	Prompts for some input
<code>showOptionDialog()</code>	The Grand Unification of the above three

- These methods are blocking!

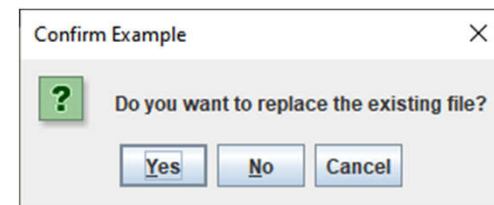
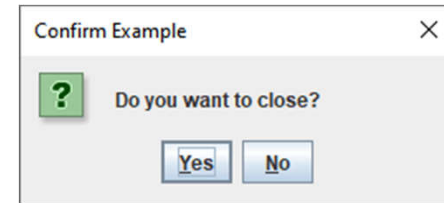
Message Dialogs

- Used to show a message to the user without getting any information back
- Examples:
 - `JOptionPane.showMessageDialog(frame, "You are welcome!", "Message Example", JOptionPane.INFORMATION_MESSAGE);`
 - `JOptionPane.showMessageDialog(frame, "Something went wrong!", "Message Example", JOptionPane.ERROR_MESSAGE);`



Confirm Dialogs

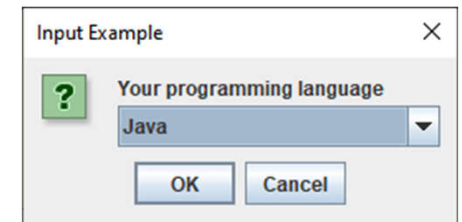
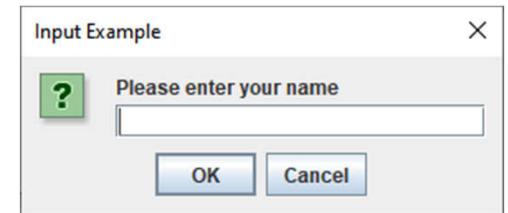
- Used to show message with the options to choose one: OK/cancel, yes/no, yes/no/cancel
 - Returned value is one of: `YES_OPTION`, `NO_OPTION`, `CANCEL_OPTION`, `OK_OPTION`
- Examples:
 - ```
int ret = JOptionPane.showConfirmDialog(frame, "Do you want to close?", "Confirm Example", JOptionPane.YES_NO_OPTION);
```
  - ```
int ret = JOptionPane.showConfirmDialog(frame, "Do you want to overwrite?", "Confirm Example", JOptionPane.YES_NO_CANCEL_OPTION);
```



Input Dialogs

- Used to show a message dialog requesting input from the user
- Examples:

- ```
String name = JOptionPane.showInputDialog(
 frame, "Please enter your name",
 "Input Example",
 JOptionPane.QUESTION_MESSAGE);
```
- ```
String progLang = (String)JOptionPane.showInputDialog(
    frame, "Your programming language",
    "Input Example",
    JOptionPane.QUESTION_MESSAGE, null,
    new String[]{"C++", "Java", "JavaScript"},
    "Java");
```

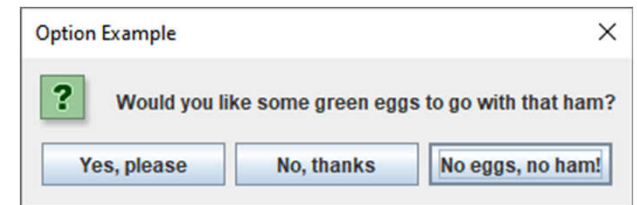


Generalized Option Dialogs

- Used to show a modal dialog with the specified buttons, icons, message, title, and so on
 - Possible to change the text that appears on the buttons of standard dialogs, or perform many other kinds of customization

- Example:

```
◦ Object[] options = {  
    "Yes, please",  
    "No, thanks",  
    "No eggs, no ham!"  
};  
int choice = JOptionPane.showOptionDialog(frame,  
    "Would you like some green eggs to go with that ham?",  
    "Option Example",  
    JOptionPane.YES_NO_CANCEL_OPTION,  
    JOptionPane.QUESTION_MESSAGE,  
    null, options, options[2]);
```





Menu-Bars

Overview

- The menu-bar is at the same level as the content pane (of the top-level container `JFrame`)
- It is set via the `JFrame`'s `setJMenuBar()` method (similar to `setContentPane()`)
- To create a menu-bar, construct a `JMenuBar`
 - A menu-bar (`JMenuBar`) contains menu (`JMenu`)
 - A menu contains menu-item (`JMenuItem`)
 - `JMenuItem` is a subclass of `AbstractButton`, and fires `ActionEvent` upon activation to all its registered `ActionListener`

Example

```
JMenuBar menuBar = new JMenuBar();
frame.setJMenuBar(menuBar);

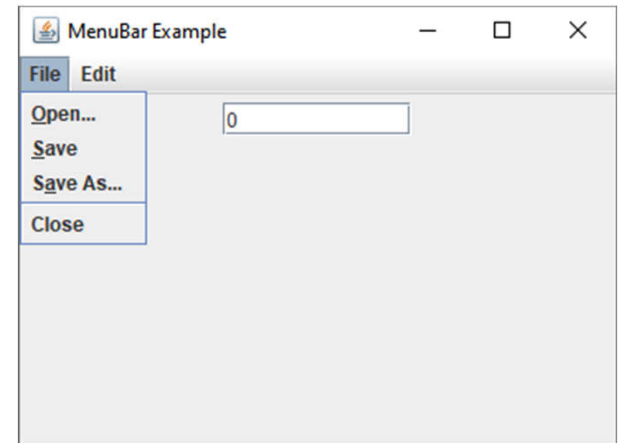
JMenu menuFile = new JMenu("File");
menuBar.add(menuFile);

JMenuItem itemOpen = new JMenuItem("Open", KeyEvent.VK_O);
menuFile.add(itemOpen);
itemOpen.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        //...
    }
});

// ...

menuFile.addSeparator();

// ...
```



Exercise

- Create a complete example menu



Look and Feel

Overview

- Swing is designed to allow to change the “look and feel” (L&F) of the application’s GUI
- Available L&F:
 - `CrossPlatformLookAndFeel`: also called Metal, looks the same on all platforms. It is part of the Java API and is the default that will be used.
 - `SystemLookAndFeel`: the L&F that is native to the system it is running on. It is determined at runtime, where the application asks the system to return the name of the appropriate L&F.
 - Synth: the basis for creating user-defined L&F with an XML file.
 - Multiplexing: a way to have the UI methods delegate to a number of different L&F implementations at the same time.

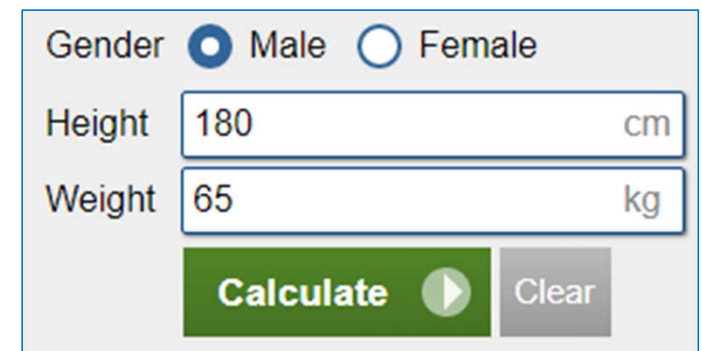
Changing the L&F

- Change to the `SystemLookAndFeel`:

```
○ try {  
    String lnf = UIManager.getSystemLookAndFeelClassName();  
    UIManager.setLookAndFeel(lnf);  
} catch (ClassNotFoundException | InstantiationException  
        IllegalAccessException | UnsupportedLookAndFeelException e)  
{  
    // ...  
}
```

Exercises


1. Write an application asking the user to enter day, month and year values and check if they make a valid date (use `JTextField`)
2. Write an BMI calculator application
 - $BMI = m/h^2$ (kg/m²)
 - < 18.5: Underweight
 - 18.5 ÷ 24.9: Normal weight
 - 25 ÷ 29.9: Overweight
 - ≥ 30: Obesity
3. Extend the calculator application with 3 more buttons for add, subtract and divide operations
4. Add a menu-bar to the calculator application with the same functionality



Gender ☒ Male ☐ Female

Height cm

Weight kg

Calculate  **Clear**



Internationalization

Introduction

- Internationalization is to display text messages in the appropriate language
- Two technical problems:
 - Using Unicode and different encoding systems
 - Customizing input/output values depending on the user's language

Unicode

- History:
 - 1963 – ASCII (American Standard Code for Information Interchange): 7 bits
 - 1981 – Extended ASCII: 8 bits
 - 1980..1990 – A lot of encoding standards for specific languages, e.g.:
 - ISO 8859-1 for Western European Language
 - GB18030 and BIG-5 for Chinese
 - TCVN 5712 for Vietnamese
 - 1991 – Unicode 1.0 defines UCS-2: 16 bits, obsolete
 - 2006 – Unicode 5.0 defines UTF-32 (aka UCS-4): 32 bits
 - 2022 – Unicode 15.0
- Java was introduced in 1996, and adopted UCS-2 standard for 16-bit characters and strings

Unicode Encoding Standards

- UTF-8: encoding a character from the U+0000..U+10FFFF (the UTF-16 accessible) range by a sequence of 1 to 4 bytes
 - 7-bit characters: 0xxxxxxx
 - 8- to 11-bit characters: 110xxxxx 10xxxxxx
 - 12- to 16-bit characters: 1110xxxx 10xxxxxx 10xxxxxx
 - 17- to 21-bit characters: 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
- UTF-16
 - Codepoints from U+0000 to U+D7FF, and U+E000 to U+FFFF: by one 16-bit code unit
 - Codepoints from U+010000 to U+10FFFF: by two 16-bit code units called a surrogate pair
 - UCS-2 is a subset of UTF-16 for the U+0000 to U+FFFF range
- UTF-32: so far, it's equivalent to UCS-4



What You Should be Careful

- Unicode character \Leftrightarrow codepoint \neq Java character
 - `String.codePointAt()`
 - `String.charCodeAt()`
- Unicode string length \neq Java string length
 - `String.length()`
 - `String.codePointCount()`
- Examples:

Unicode character	Codepoint	UTF-32	UTF-16	UTF-8
a	U+61	00000000 00000061	0061	61
ă	U+1eb7	00000000 00001eb7	1eb7	e1 ba b7
❤	U+2764	00000000 00002764	2764	e2 9d a4
🚫	U+1f6b3	00000000 0001f6b3	d83d deb3	f0 9f 9a b3

Java String ⇔ UTF-8 Conversion

- `String` to UTF-8
 - `byte[] utf8 = string.getBytes(StandardCharsets.UTF_8);`
 - `ByteBuffer utf8 = StandardCharsets.UTF_8.encode(string);`
- UTF-8 to `String`
 - `String string = new String(utf8, StandardCharsets.UTF_8);`
 - `String string = StandardCharsets.UTF_8.decode(utf8).toString();`
- Printing Unicode strings to console:
 - Encode strings as UTF-8 when writing to console:
 - `PrintStream out = new PrintStream(System.out, true, "UTF-8");`
`out.println("Trời đẹp quá");`
 - Change terminal encoding to UTF-8:
 - `chcp 65001`

Locale

- Locale: consisting of
 - A language code
 - A country code
- Properties file:
 - A file that stores information about the characteristics of a program or environment
 - In plain-text format

Example

- `String language, country;`

```
// obtain language and country codes...
```

```
Locale locale = new Locale(language, country);
ResourceBundle bundle = ResourceBundle.getBundle("Communication", locale);
System.out.println(bundle.getString("message.greetings"));
System.out.println(bundle.getString("message.farewell"));
System.out.println(bundle.getString("question.inquiry"));
```

- Create properties files:

- `Communication_en_US.properties`:
 - `message.greetings = Hello.`
`message.farewell = Goodbye.`
`question.inquiry = How are you?`
- `Communication_vi_VN.properties`
- `Communication_fr_FR.properties`
- `Communication.properties` (default file)

Formatting Date

- Date formatting can be customized with locale information
- Two ways to do:
 - ```
DateFormat dateFormat =
 DateFormat.getDateInstance(DateFormat.FULL, locale);
Date today = new Date();
String formattedDate = dateFormat.format(today);
```
  - ```
ZonedDateTime zoned = ZonedDateTime.now();  
DateTimeFormatter pattern =  
    DateTimeFormatter.ofLocalizedDate(FormatStyle.FULL)  
        .withLocale(locale);  
String formattedDate = zoned.format(pattern);
```

Formatting Numbers, Currency

- Example:

- `Locale locale = new Locale("vi", "VN");`

```
NumberFormat nf = NumberFormat.getInstance(locale);  
out.println(nf.format(123.456));
```

```
NumberFormat cf = NumberFormat.getCurrencyInstance(locale);  
out.println(cf.format(1000));
```

Collation

- Defines how to compare, convert values for a given locale
 - Common use-cases are: sorting, comparing, converting to uppercase/lowercase
- Example:
 - ```
Locale locale = new Locale("vi", "VN");
String s1 = "óğ mựợt", s2 = "thong thả";

Collator col = Collator.getInstance(locale);
System.out.println(s1.compareTo(s2) > 0);
System.out.println(col.compare(s1, s2) > 0);

System.out.println(s1.toUpperCase(locale));
```