IS311 Programming Concepts

GUI Building with Swing

Agenda

- Java GUI Support
  - AWT
  - Swing
- Creating GUI Applications
- Layout Managers

ต้องมีพื้นความรู้ต่อไปนี้ก่อน

- Classes / Objects
- Method Overloading
- Inheritance
- Polymorphism
- Interfaces
- How to read the Java2 API Documents

Java Graphical User Interfaces

- AWT
  - Abstract Window Toolkit
  - GUI for Java 1.0 and 1.1
- Swing or Java Foundation Classes (JFC)
  - Java 2
  - Extension to AWT

AWT (Abstract Window Toolkit)

- Present in all Java implementations
- Described in (almost) every Java textbook
- Adequate for many applications
- Uses the controls defined by your OS (whatever it is)
  - therefore it's "least common denominator"
- Difficult to build an attractive GUI
- **`import java.awt.*;`**
  `import java.awt.event.*;`

Some AWT Components

`Java.awt` package

- **`Button`** – A graphical push button
- **`Label`** – Displays a single line of read only text
- **`Menu`** – A single pane of a pull-down menu
- **`MenuItem`** – A single item within a menu
- **`TextField`** – Displays a single line a text and allows user to enter and change the text
- **`Table`** – Displays a list of items

Swing

- Extension to `AWT`
- Available with Java 2 and as an extension to Java 1.1
- Provides more advanced components than `AWT`
- Be careful when using with Applets
  - Not all browsers support Swing

Swing Components

- Available in the `javax.swing` package
  - **`JButton`**
  - **`JLabel`**
  - **`JMenu`**
  - **`JMenuItem`**
  - **`JTextField`**
  - **`JTable`**
  - **`JSlider`** - Simulates a slider control
  - **`JProgressBar`** – Displays the progress of a time consuming operation

Advantages of Swing

- Swing components are implemented with absolutely no native code
  - More portable
  - Not restricted to least common denominator
- Swing buttons and labels can display images instead of, or in addition to, text
- You can easily add or change the borders drawn around most Swing components. For example, it's easy to put a box around the outside of a container or label.
- You can easily change the behavior or appearance of a Swing component by either invoking methods on it or creating a subclass of it.
- Swing components don't have to be rectangular.
  - Buttons, for example, can be round.

Swing Components

- Include everything from buttons to split panes to tables
- **Pluggable Look and Feel Support**
- **Accessibility API**
  - Enables assistive technologies such as screen readers and Braille displays to get information from the user interface
- **Java 2D API (Java 2 Platform only)**
  - Enables developers to easily incorporate high-quality 2D graphics, text, and images in applications and in applets
- **Drag and Drop Support (Java 2 Platform only)**
  - Provides the ability to drag and drop between a Java application and a native application

Import Swing

- Present in all modern Java implementations (since 1.2)
- More controls, and they are more flexible
- Gives a choice of “look and feel” packages
- Much easier to build an attractive GUI
- ```
  import javax.swing.*;
  import javax.swing.event.*;
  ```
  and
  ```
  import java.awt.*;
  import java.awt.event.*;
  ```
- You may not need *all* of these packages

Swing vs. AWT

- Swing is built “on top of” AWT, so you need to import AWT and use a few things from it
- Swing is bigger and slower
- Swing is more flexible and better looking
- Swing and AWT are *incompatible*--you can use either, but you can’t mix them
  - ความจริงสามารถใช้ผสมกันได้ แต่ไม่ควรทำ
  - Basic controls are practically the same in both
  - AWT: `Button b = new Button ("OK");`
  - Swing: `JButton b = new JButton("OK");`
- Swing gives *far more* options for everything (buttons with pictures on them, etc.)
  ```

To build a GUI...

1. Make somewhere to display things (a Container)
   - Usually you would use a `JFrame` or a `JApplet`
2. Create some Components (buttons, text areas, panels, etc.)
   - It's usually best to *declare* Components as instance variables, and
   - *Define* them in your applet's `init()` method or in some application method
3. Add your Components to your display area
   - Choose a layout manager
   - Add your Components to your `JFrame` or `JApplet` according to the rules for your layout manager
4. Attach Listeners to your Components
   - Interacting with a Component causes an `Event` to occur
   - A Listener gets a message when an interesting event occurs, and executes some code to deal with it

Container

The `Container` class is an abstract subclass of `Component`, which has additional methods that allow other `Components` to be nested inside of it.

Other `Container` objects can be stored inside of a `Container` (since they are themselves `Components`), which makes for a fully hierarchical containment system.

A container can also have a layout manager that controls the visual placement of components in a container.

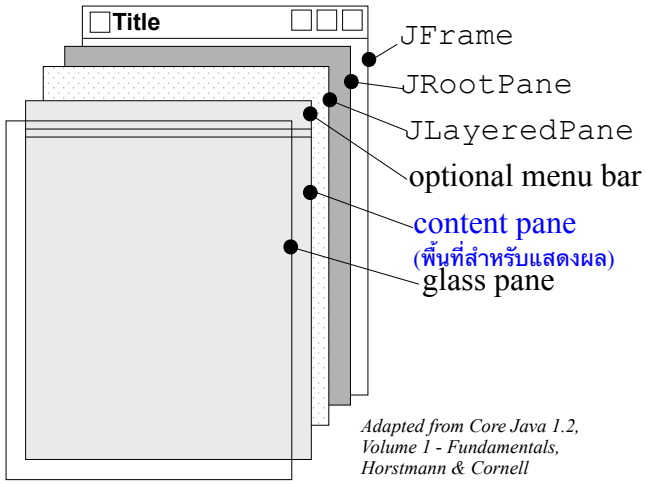
Containers

`Container`	The superclass of containers.
`JApplet`	Swing version of Applet
`JFrame`	A top-level window with a title, menu bar, and borders.
`JPanel`	A container that can be embedded in other containers
`JWindow`	A top-level window without a title, menu bar, or borders.

Containers and Components

- A GUI is built by putting components into containers
- The job of a `Container` is to hold and display Components
- Some frequently used types (subclasses) of `Component` are `JButton`, `JCheckbox`, `JLabel`, `JTextField`, and `JTextArea`
- A `Container` is also a `Component`
  - This allows Containers to be nested
- Important `Container` classes are `JFrame`, `JApplet`, and `JPanel`
  - `JFrame` and `JApplet` both contain other containers; use `getContentPane()` to get to the container you want
  - You typically create and use `JPanel`s directly

Structure of a `JFrame` object



What is a pane?

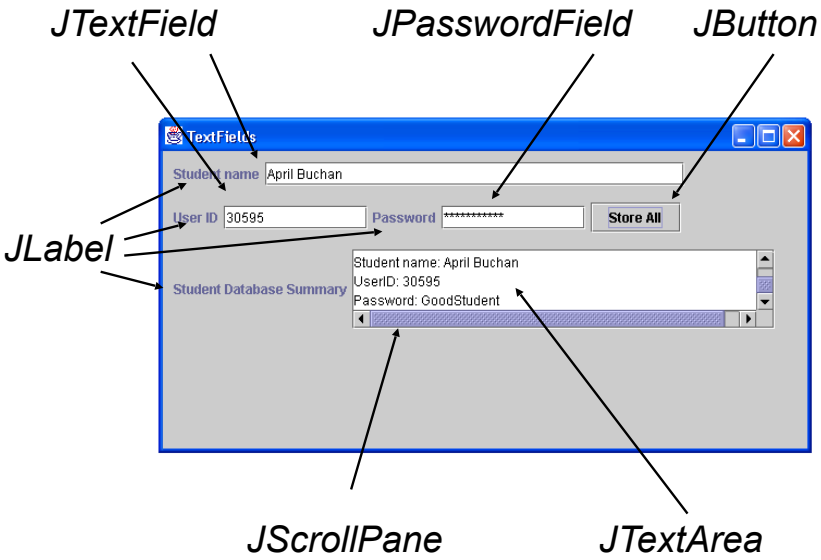
pane ในความหมายทั่วไป หมายถึงกระจกหนึ่งแผ่นบนหน้าต่างหรือประตู

ในคอมพิวเตอร์หมายถึงพื้นที่บนหน้าต่างที่ใช้แสดงผล

Adapted from Core Java 1.2, Volume 1 - Fundamentals, Horstmann & Cornell

Java top-level containers (`JFrame`, `JApplet`, ...) have several layers (*panes*): *root*, *content*, *layered*, and *glass*. Programs normally reference only the content pane.

Example



JButton

Declaration in GUI class

Example:

```
private JButton computeButton;
```

```
computeButton = new JButton("COMPUTE");
```

Instantiation in GUI class constructor

JLabel

Example:

```
JLabel nameLabel = new JLabel("Student name");
```

Creation in GUI class constructor since labels do not generate events

JTextField

Declaration in GUI class

Example:

```
private JTextField nameField;
```

```
nameField = new JTextField(35);
```

Instantiation in GUI class constructor

JPasswordField

Declaration in GUI class

Example:

```
private JPasswordField userPassField;
```

```
userPassField = new JPasswordField(12);
```

Instantiation in GUI class constructor

JTextArea

Declaration in GUI class

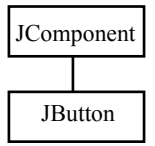
Example:

```
private JTextArea studentArea;
```

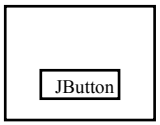
```
studentArea = new JTextArea(4,35);
```

Instantiation in GUI class constructor

Creating an Application



- Import classes
  - `import javax.swing.*;`
  - `import java.awt.*;`
  - `import java.awt.event.*;`
- All Swing components extend `JComponent` class
  - Set size
  - Change color
  - Define fonts
  - Hover help



- Before components can be displayed, they must be added to a container
  - Containers can be placed in other Containers
  - Layout managers arrange components

Creating an GUI

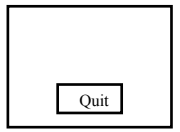
- Create a class that represents the GUI
  - Serves as the container
    - `JApplet, JFrame, JWindow`
- `public class Buttons extends JFrame {`
  - Call constructor method to handle setup
  - Set the size of the frame (in pixels)
  - Handle window closing
  - Display the frame

Adding Components to a Container

- Simplest container is a panel (extend JPanel)

```
JButton quit = new JButton("Quit");
JPanel panel = new JPanel();
panel.add(quit);
```

- Other containers (ie Frames, Windows, Applets)
  - Broken down into panes
  - Components are added to container's content pane

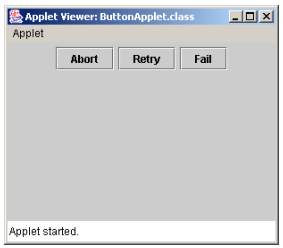


Japplet Example

```
import javax.swing.*;

public class ButtonApplet extends JApplet {
    JButton abort = new JButton("Abort");
    JButton retry = new JButton("Retry");
    JButton fail = new JButton("Fail");

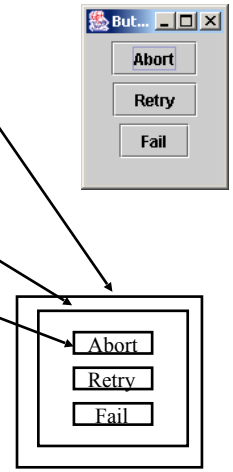
    public void init() {
        JPanel pane = new JPanel();
        pane.add(abort);
        pane.add(retry);
        pane.add(fail);
        setContentPane(pane);
    }
}
```



JFrame Examle

```
import javax.swing.*;
public class Buttons extends JFrame {
    JButton abort = new JButton("Abort");
    JButton retry = new JButton("Retry");
    JButton fail = new JButton("Fail");
    public Buttons() {

        JPanel pane = new JPanel();
        pane.add(abort);
        pane.add(retry);
        pane.add(fail);
        setContentPane(pane);
    }
    public static void main(String[] args) {
        Buttons rb = new Buttons();
        rb.show();
    }
}
```



Layout Management

Each container has a *layout manager* that directs the arrangement of its components

Java API has many layout managers but the most three useful layout managers are:

- border layout
- flow layout
- grid layout

Layout Managers

`LayoutManager`	The interface that a layout manager must implement.
`BorderLayout`	place components along each edge and in center
`CardLayout`	displays one component at a time
`FlowLayout`	places components left-to-right, top-to-bottom
`GridBagConstraints`	Used to specify constraints in a GridBagLayout object.
`GridLayout`	places components in a rigid grid with fixed sized cells.
`GridBagLayout`	places components in a grid with flexible sized cells.

`FlowLayout`

The components are arranged in the container from left to right in the order in which they were added. When one row becomes filled, a new row is started.

`FlowLayout` Constructors

- `public FlowLayout()`
  Constructs a new `FlowLayout` with a default center alignment and a default gap of five pixels for both horizontal and vertical.
- `public FlowLayout(int alignment)`
  Constructs a new `FlowLayout` with a specified alignment and a default gap of five pixels for both horizontal and vertical.
- `public FlowLayout(int align, int hGap, int vGap)`
  Constructs a new `FlowLayout` with a specified alignment, horizontal gap, and vertical gap. The *gaps* are the distances in pixel between components.

`GridLayout` Manager

- `GridLayout` arranges components into a grid of rows and columns.
- You can specify the number of rows and columns, or the number of rows only and let the layout manager determine the number of columns, or the number of columns only and let the layout manager determine the number of rows.
- The cells in the grid are equal size.
- Look at the API for the `add` method and constructor.

`GridLayout` Constructors

- `public GridLayout(int rows, int columns)`

  Constructs a new `GridLayout` with the specified number of rows and columns.

- `public GridLayout(int rows, int columns, int hGap, int vGap)`

  Constructs a new `GridLayout` with the specified number of rows and columns, along with specified horizontal and vertical gaps between components.

เมท็อดสำหรับ add component ลงใน container

- `public add(Component comp)`
  เพิ่ม component ลงใน container ตามลำดับ
- `public add(Component comp, int index)`
  เพิ่ม component ลงใน container โดยระบุตำแหน่งที่ต้องการเพิ่ม ถ้าเป็น -1 หมายถึงต่อท้าย

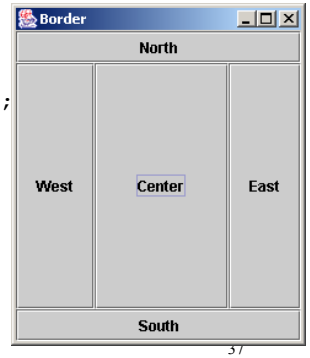
Border Layout

- A BorderLayout manager can place a component into any of five regions.
- Regions which are unused give up their space to `BorderLayout.CENTER`.

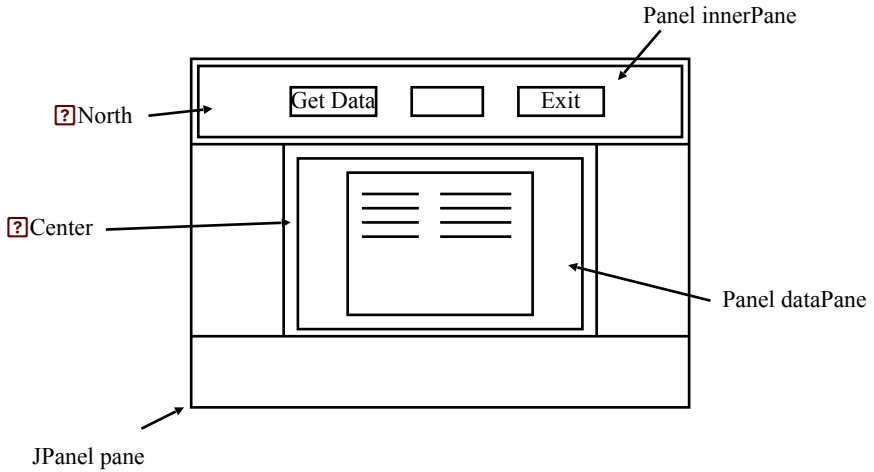
BorderLayout Demo

```
class Border extends JFrame {
    JButton north = new JButton("North");
    JButton south = new JButton("South");
    JButton east = new JButton("East");
    JButton west = new JButton("West");
    JButton center = new JButton("Center");

    Border() {
        super("Border");
        setSize(240, 280);
        JPanel pane = new JPanel();
        pane.setLayout(new BorderLayout());
        pane.add("North", north);
        pane.add("South", south);
        pane.add("East", east);
        pane.add("West", west);
        pane.add("Center", center);
        setContentPane(pane);
    }
```



Border Layout Example



The BorderLayoutDemo class

```
public class BorderLayoutDemo extends JFrame {
    static DefaultTableModel dataTable = new
        DefaultTableModel();
    static  JTable table = new JTable(dataTable);
    static JButton exitButton = new JButton( "Exit" );
    static JPanel pane = new JPanel();
    static JPanel innerPane = new JPanel();
    static JPanel dataPane = new JPanel();
    static JButton getButton = new JButton ("Get Data");
    static JTextField inputData = new JTextField(6);
```


The BorderLayoutDemo class

```
pane.setLayout(new BorderLayout());
innerPane.add(getButton);
innerPane.add(inputData);
innerPane.add(exitButton);
pane.add("North",innerPane);
.
.
.
.
JScrollPane scrollPane = new JScrollPane(table);
dataPane.add("Center",scrollPane);
pane.add("Center",dataPane);
```

Summary

- Abstract Windowing Toolkit provides GUI support for Java 1.0 and Java 1.1
- Swing (Java Foundation Classes) provides GUI support for Java 2
- Swing is an extension of the Java 1.1 AWT
- Swing components are found in the `javax.swing` package
- Layout Managers provide a consistent form for a GUI

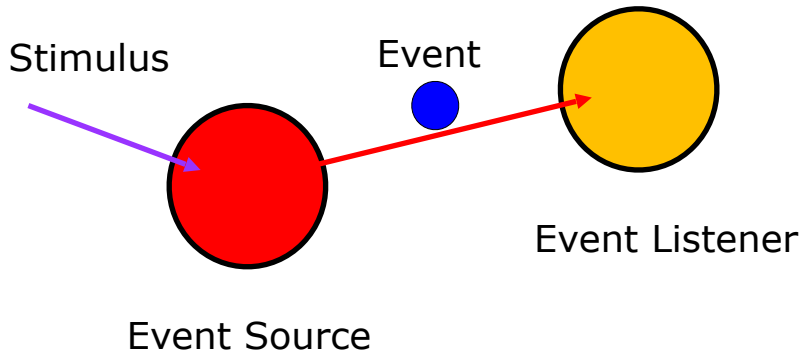
Event Handling

Components & Events

A component generates events when a user interacts with it. The `Event` class encapsulates all the states in an event. Each component generates a different set of events.

Event Delegation Model

An **event source** produces **events** and dispatches them to all registered **event listeners**



source & listener

- A **single class** can implement **multiple listener interfaces**
- A **single listener** object can register with **multiple event sources**
- **Multiple listener** objects can register with a **single event** source (multicast)

Events

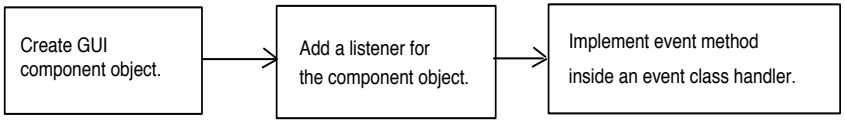
- An Event is the encapsulation of some user input delivered to the application **asynchronously**
- Events correspond to
  - **Physical actions**, (e.g., mouse button down, key press/release)
  - **Logical events**, (e.g., button press, got focus)
- The `java.awt.AWTEvent` is the root class of all AWT events (a subclass `of java.util.EventObject`)
- From any `AWTEvent` you can get the object that was the event source by invoking `getSource()`
- AWTEvent is subclassed as: `ActionEvent, WindowEvent, ItemEvent, KeyEvent, MouseEvent, TextEvent, etc.`

GUI Events Defined by Java

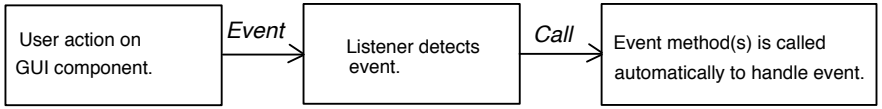
Event Class	Source Object	User Action
WindowEvent	A frame	Window opened/closed
ActionEvent	A button	Click button
	A text field	Press Enter
	A text area	Press Enter
	A menu item	Select menu item
	A combo box	Select/deselect item
ItemEvent	A checkbox	Select/deselect item
	A radio button	Select/deselect item
ListSelection	A list box	Select/deselect item
KeyEvent	A key	Key pressed or released
MouseEvent	A mouse	Move or click mouse

Event Programming and Processing

Programming Steps:



Java Processing Steps:



Event Listener Interfaces

- Event handling is achieved through the use of listener interfaces, defined in `java.awt.event`, e.g.
  - `ActionListener` - Button, MenuItem, List
  - `WindowListener` - Window
  - `ItemListener` - Choice, List, Checkbox
  - `KeyListener` - Component
  - `MouseListener` - Commponent
  - `MouseMotionListener` -Component
  - `TextListener` - TextComponent

Event Listener Interfaces (cont.)

Inheritors of the interface have to **implement one or more methods** in order to respond to certain types of events, e.g.

- `ActionListener - actionPerformed`
- `WindowListener - windowOpen, WindowClosing,` etc.

Event Registration

Three things must be done to receive events:

- The object's class must **inherit from an *`XXX``Listener` interface**
- Each interface requires the class to **implement one or more interface (abstract) methods** that receive an event and process it
- Prior to receiving any events, an object must **register itself with the event source** by invoking its `add`*`XXX`*`Listener` method

Event Delivery and Decoding

- When an event occurs, the event source sends the event to the appropriate listening methods on all registered event listeners
- If necessary the listener can get the event source (`getSource()`) from the event
- For ActionEvents, the action command (`getActionCommand()`) can also be gotten
- Learn about events and event processing by implementing event listener and calling `System.out.println(e)` in the listening methods

การถอน Event Listener

An event listener may be removed from a component's list of listeners by calling a **`removeXXXListener()`** method, passing in the listener to be removed.

eg.,

```
btn.removeActionListener(al);
```

removes action listener `al` from button `btn`

การจัดการ Event ด้วย subclass ของ component

```
class MyBtn extends Button {
  public MyBtn(String label) {
    super(label);
    enableEvents(AWTEvent.ACTION_EVENT_MASK);
  }

  public void processActionEvent(ActionEvent ae) {
    System.out.println(
                  "Processing an action event.");
    super.processActionEvent(ae);
  }
}
```


Adding an ActionListener to the Bottons class

```
abort.addActionListener(
         new ActionListener() {
          public void actionPerformed( ActionEvent event )
               {

                        System.exit(0);
               }
        }
   );
```

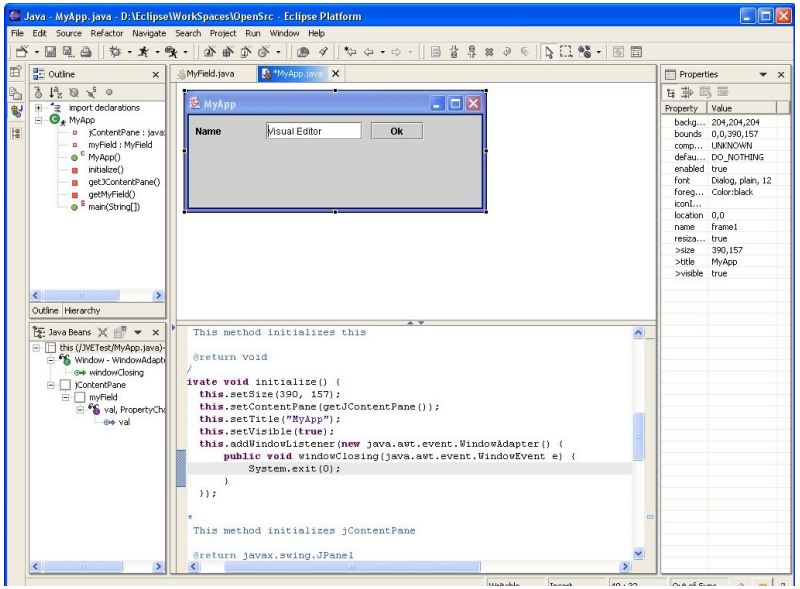
An ActionListener allows you to add the lines of code that will be executed when the end user clicks the button

IDE with Visual Editor

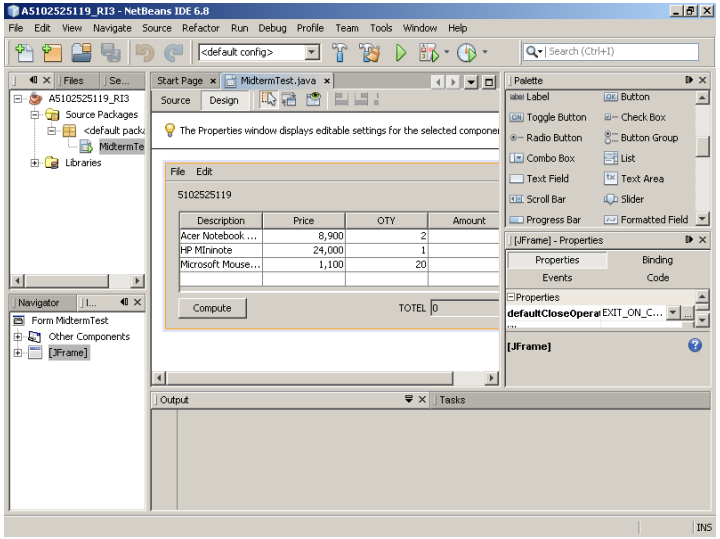
- JBuilder (Borland ขายกิจการไปแล้ว)
- Eclipse (IBM and others)
  http://www.eclipse.org/
- Netbeans (Sun->Oracle)
  https://netbeans.org/
- IntelliJ IDEA (JetBrains)
  https://www.jetbrains.com/idea/

IDE = Integrated Development Environment

Eclipse Visual Editor



Visual Programming



A Visual Programming Environment