Chương 5 : AWT

Sau khi học xong chương này, bạn có thể nắm được các nội dung sau:

- Hiểu về AWT
- Sử dụng các Component
- Sử dụng các Container
- Sử dụng các Layout Manager
- Xử lý sự kiện với các Component

5.1 Giới thiệu về AWT

Các ứng dụng phần mềm hiện nay vô cùng thân thiện vì được trình bày nhiều màn hình giao diện đồ họa đẹp mắt. Các ngôn ngữ lập trình hiện nay được cung cấp các đối tượng đồ họa, chúng có thể được điều khiển bởi người lập trình viên, hay bởi người sử dụng. Một trong số những kết quả quan trọng nhất chính là các ngôn ngữ hiện nay được dựa trên Giao diện người dùng đồ họa (Graphical User Interface - GUI). Trong chương này, ta sẽ thảo luận về Java hỗ trợ tính năng DUI cùng các sự thi hành của chúng.

GUI cung cấp chức năng nhập liệu theo cách thân thiện với người dùng. GUI biến đổi từ ứng dụng đến ứng dụng và có thể chứa nhiều điều khiển như textbox, label, listbox hay các điều khiển khác. Các ngôn ngữ lập trình khác nhau cung cấp nhiều cách khác nhau để tạo GUI. Các phần mềm giống như VB hay VC++ có thể cung cấp chức năng kéo và thả trong khi đó phần mềm giống như C++ yêu cầu người lập trình phải viết toàn bộ mã để xây dựng một GUI.

Một phần tử (element) GUI được thiết lập bằng cách sử dụng thủ tục sau:

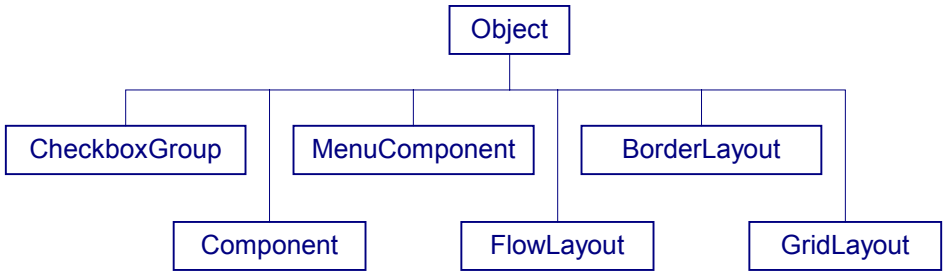
- Tạo element, instance, checkbox, label, hay listbox
- Xác định sự xuất hiện khởi đầu của các phần tử
- Quyết định xem phần tử đó có nên chiếm giữ vị trí được chỉ ra hay không
- Thêm phần tử vào giao diện trên màn hình

Một thành phần (component) GUI là một đối tượng trực quan. Người dùng tương tác với đối tượng này thông qua con trỏ chuột hay bàn phím. Các thành phần như là button, label v.v... có thể được nhìn thấy trên màn hình. Bất kỳ thao tác nào áp dụng đến tất cả các thành phần GUI đều được tìm thấy trong lớp đối tượng Component. Để tạo các thành phần GUI này, chúng ta cần sử dụng các lớp tồn tại trong gói **java.awt**.

AWT có nghĩa là Abstract Windowing Toolkit. AWT là một bộ các lớp trong Java cho phép chúng ta tạo một GUI và chấp nhận các nhập liệu của người dùng thông qua bàn phím và chuột. AWT cung cấp các item khác nhau để tạo một GUI hiệu quả và lôi cuốn người sử dụng. Các item này có thể là:

- Thùng chứa (**Container**)
- Thành phần (**Component**)
- Trình quản lý cách trình bày (**Layout manager**)
- Đồ họa (**Graphic**) và các tính năng vẽ (**draw**)
- Phông chữ (**Font)**
- Sự kiện (**Event**)

Gói AWT chứa các lớp, giao diện và các gói khác. Hình sau đây mô tả một phần nhỏ của hệ thống phân cấp lớp AWT.



Hình 5.1 Hệ thống cây phân cấp lớp AWT

5.2 Container

Container là vùng mà bạn có thể đặt các thành phần của bạn vào đó. Bất cứ vật gì mà kế thừa từ lớp Container sẽ là một container. Applet là một container, applet được dẫn xuất từ panel, lớp panel lại được dẫn xuất từ lớp Container.

Một container có thể chứa nhiều phần tử, các phần tử này có thể được vẽ hay được tô màu tuỳ thích. Bạn hãy xem container như một cửa sổ. Đã là cửa sổ thì phải có khung (frame), pane, latch, hook, và các thành phần có kích thước nhỏ hơn.

Gói java.awt chứa một lớp gọi là Container. Lớp này trực tiếp hay gián tiếp phát sinh ra hai container được sử dụng phổ biến nhất là Frame và Panel.

Frame và Panel là các container thường được sử dụng. Frame là các cửa sổ được tách riêng nhau nhưng ngược lại panel là các vùng được chứa trong một cửa sổ. Panel không có các đường viền, chúng được trình bày trong một cửa sổ do trình duyệt hay appletviewer cung

cấp. Appletviewer là một công cụ được JDK hỗ trợ để xem các applet. Frame là lớp con của Window. Chúng được trình bày trong một cửa sổ độc lập, cửa sổ này có chứa các đường viền xung quanh.

5.2.2 Frame

Frame không phụ thuộc vào applet và trình duyệt. Frame có thể hoạt động như một container hay như một thành phần (component). Bạn có thể sử dụng một trong những constructor sau để tạo một frame:

- **Frame()**: Tạo một frame vô hình (không nhìn thấy được)
- **Frame(String, title)**: Tạo một frame với nhan đề trống.

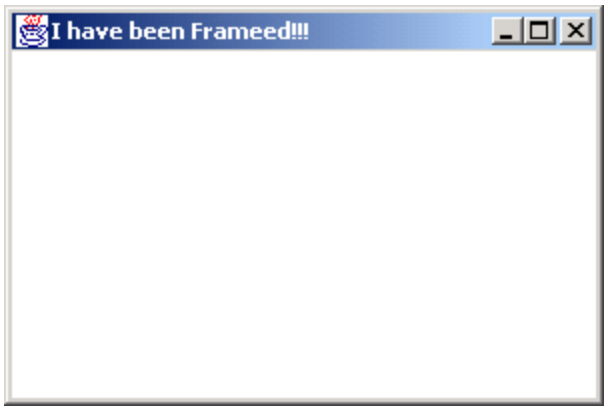
Chương trình 5.1 minh hoạ cách tạo một Frame.

Chương trình 5.1

```
import java.awt.*;
class FrameDemo extends Frame
{
        public FrameDemo(String title)
        {
                super(title);
        }
        public static void main(String args[])
        {
                FrameDemo f=new FrameDemo("I have been Frameed!!!");
                f.setSize(300,200);
                f.setVisible(true);
        }
}
```

Lớp được định nghĩa Framedemo là một lớp con của lớp Frame. Lớp FrameDemo này có một constructor, trong contructor này ta cho gọi hàm super(). Tiến trình này sẽ lần lượt gọi constructor của lớp con (trong trường hợp này là frame). Mục đích của super() là gọi constructor của lớp cha mẹ. Tiến trình này sẽ tạo một đối tượng của lớp con, lớp con này sẽ tạo frame. Thêm vào đó, nó cũng sẽ cho phép đối tượng frame nhìn thấy được thông qua phạm vi lớp. Tuy nhiên, frame vẫn không nhìn thấy được và không có kích thước. Để làm được điều này, ta sử dụng hai phương thức nằm trong phương thức main: setSize() và setVisible().

Kết xuất của chương trình giống như hình 5.2



Hình 5.2 Frame

5.2.2 Panel

Panel được sử dụng để nhóm một số các thành phần lại với nhau. Cách đơn giản nhất để tạo một panel là sử dụng hàm constructor của nó, hàm Panel().

Chương trình 5.2 chỉ ra cách tạo một panel:

Chương trình 5.2

```
import java.awt.*;
class Paneltest extends Panel
{
        public static void main(String args[])
        {
                Paneltest p=new Paneltest();
                Frame f=new Frame("Testing a Panel");
                f.add(p);
                f.setSize(300,200);
                f.setVisible(true);
        }
        public Paneltest()
        {
        }
}
```

Panel không thể được nhìn thấy trực tiếp. Do đó, chúng ta cần thêm panel đến một frame. Vì vậy ta cần tạo một frame mới và thêm Panel mới được tạo này vào nó. Tuy nhiên, frame sẽ không nhìn thấy được, và không có kích thước. Chúng ta sử dụng hai phương thức trong phương thức main – setSize() và setVisible() để thiết lập kích thước và hiển thị frame.

Kết xuất của chương trình:

Hình 5.3 Panel

5.2.3 Dialog

Lớp 'Dialog' tương tự như lớp Frame, nghĩa là Dialog là lớp con của lớp Window. Đối tượng dialog được tạo như sau:

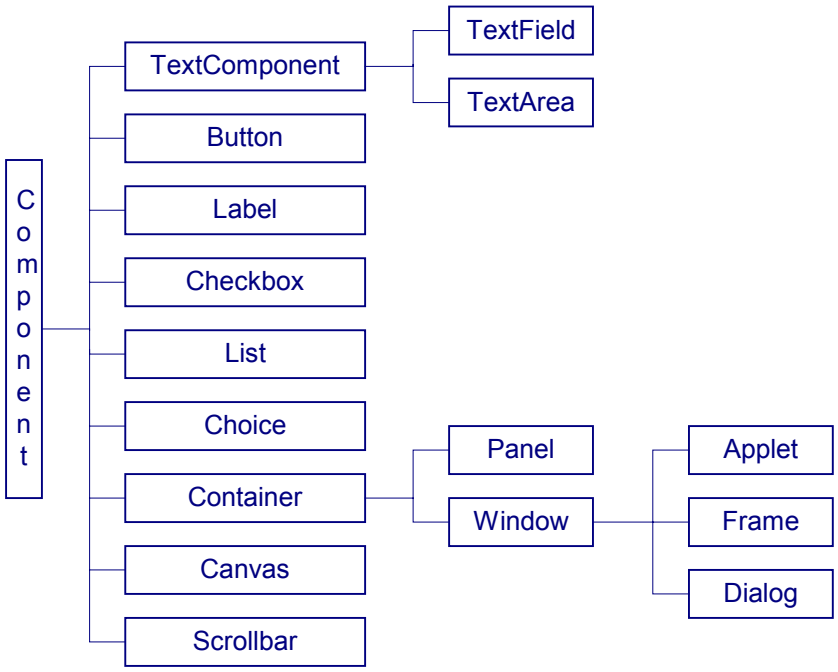
```
Frame myframe=new Frame("My frame"); // calling frame
String title = "Title";
boolean modal = true; // whether modal or not
Dialog dlg=new Dialog(myframe, title, modal);
```

Số hạng 'modal' chỉ ra rằng dialog sẽ ngăn chặn bất kỳ tương tác nào xảy đến với các cửa sổ được mở khác, trong khi dialog đang được hiển thị trên màn hình. Kiểu hộp thoại này ngăn chặn người dùng tương tác với các cửa sổ khác trên màn hình, cho tới khi dialog được đóng lại.

5.3 Thành phần (Component)

Một component có thể được đặt trên giao diện người dùng, có thể được thay đổi kích thước hay làm cho nhìn thấy được. Ví dụ được dùng phổ biến nhất là textfield, label, checkbox,

textarea v.v... Các thành phần cao cấp khác như scrollbar, scrollpane và dialog cũng tồn tại. Tuy nhiên chúng không được sử dụng thường xuyên.



Hình 5.4 Các lớp đối tượng thành phần

Bây giờ chúng ta hãy xét một số thành phần thường được sử dụng.

5.3.1 Nhãn (Label)

Lớp này được sử dụng để trình bày một String. Nó không thể được sửa đổi. Đây là một chuỗi chỉ đọc. Sử dụng một trong những constructor sau đây để tạo một label:

- **Label()**
  Tạo một Label trống.

- **Label(String labeltext)**
  Tạo một Label với văn bản được cho.

- **Label(String labeltext, int alignment)**
  Tạo một Label với một chế độ canh lề alignment được cho, alignment có thể là Label.LEFT, Label.RIGHT hay Label.CENTER.

Các phương thức được sử dụng phổ biến của label được trình bày ở bảng bên dưới:

Phương thức	Chức năng
setFont(Font f)	Thay đổi phông chữ đang được chọn của Label
setText(String s)	Thiết lập nhãn cho Label
getText()	Lấy nội dung hiện hành của Label

Bảng 5.1 Các phương thức của Label

Chương trình 5.3 chỉ ra cách sử dụng của Label:

Chương trình 5.3

```
import java.awt.*;
class Labeltest extends Frame
{
        Label label1=new Label("This is just a label");
        public Labeltest(String title)
        {
                super(title);
                add(label1);
        }
        public static void main(String args[])
        {
                Labeltest f=new Labeltest("Label");
                f.setSize(300,200);
                f.show();
        }
}
```

label1=new Label("Enter your details :");
Tạo đối tượng Label

add(label1);
Label sẽ hiển thị chỉ khi nó được thêm vào container. Ở đây, Frame là container mà thành phần Label được thêm vào. Việc này được thực hiện bằng cách sử dụng phương thức add().

Kết xuất của chương trình được chỉ ra ở hình 5.5

Hình 5.5 Label

5.3.2 Ô văn bản (TextField)

Một textfield là một vùng chỉ chứa một dòng đơn, trong đó văn bản có thể được trình bày hay được nhập vào bởi người dùng. Trong Java, một trong những constructor sau có thể được sử dụng để tạo một textfield:

- TextField(): Tạo một textfield mới.
- TextField(int columns): Tạo một textfield mới với số cột được cho.
- TextField(String s): Tạo một textfield mới với chuỗi văn bản được cho.
- TextField(String s, int columns): Tạo một textfield mới với nhãn và số cột được cho.

Các phương thức thường được sử dụng của đối tượng TextField được tóm tắt trong bảng sau:

Phương thức	**Chức năng**
setEchoChar(char)	Thiết lập các kí tự được trình bày trong dạng của một kí tự được cho.
setText(String s)	Thiết lập nhãn cho TextField.
getText()	Trả về nhãn của TextField.
setEditable(boolean)	Xác định trường có thể được soạn thảo hay không. Trường chỉ được soạn thảo khi giá trị này được đặt là True.
isEditable()	Xác định xem trường có đang trong mode soạn thảo hay không. Giá trị trả về kiểu Boolean.

Bảng 5.2 Các phương thức của TextField

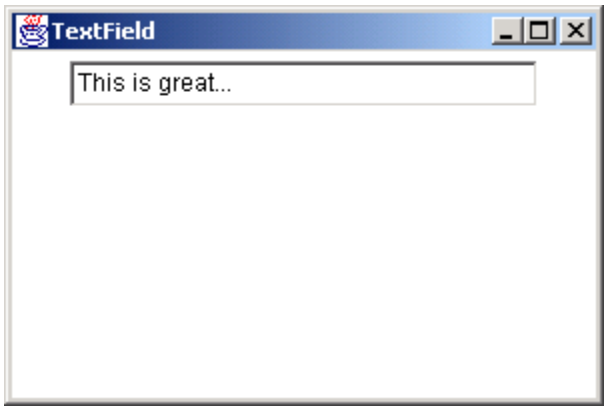
Chương trình 5.4 chỉ ra cách sử dụng của TextField:

Chương trình 5.4

```
import java.awt.*;
class TextFieldtest extends Frame
{
    TextField tf1=new TextField(30);
    public TextFieldtest(String title)
    {
        super(title);
        setLayout(new FlowLayout());
        add(tf1);
}
public static void main(String args[])
{
    TextFieldtest f=new TextFieldtest("TextField");
    f.setSize(300,200);
    f.show();
}
}
```

Trong chương trình này, chúng ta sử dụng phương thức setLayout() để thay đổi cách trình bày của các thành phần trên màn hình. Layout manager có chức năng xắp xếp các thành phần trong một container.

Kết xuất của chương trình được chỉ ra ở hình bên dưới:



Hình 5.6 TextField

5.3.3 Vùng văn bản (TextArea)

Một Textarea được sử dụng khi văn bản nhập vào trên hai hay nhiều dòng. Textarea có một scrollbar. Thành phần TextArea là một trường văn bản có thể được soạn thảo với đặc tính nhiều dòng.

Để tạo một Textarea, làm theo các bước sau:

1) Tạo một phần tử.
2) Chỉ ra số dòng hay số cột phần tử này cần có.
3) Bố trí phần tử này trên màn hình.

Trong Java, bạn có thể sử dụng các constructor sau để tạo TextArea:

- **TextArea()**: Tạo một TextArea mới.
- **TextArea(int rows, int cols)**: Tạo một TextArea mới với số lượng cột và dòng được cho.
- **TextArea(String text)**: Tạo một TextArea mới với nhãn được cho.
- **TextArea(String text, int rows, int cols)**: Tạo một TextArea mới với nhãn, số dòng và số cột được cho.

Các phương thức thường được sử dụng nhiều nhất của TextArea:

Phương thức	**Chức năng**
setText(String)	Thiết lập nhãn cho TextArea.
getText()	Trả về nhãn của TextArea.
setEdiable(boolean)	Xác định xem trường có thể được soạn thảo hay không. Trường có thể được soạn thảo khi giá trị này là True.
isEdiable()	Xác định xem trường có đang trong mode soạn thảo được không. Trả về giá trị là kiểu Boolean.
insertText(String, int)	Chèn String được cho vào vị trí index được cho.
replaceText(String, int, int)	Thay thế văn bản nằm giữa vị trí int, int được cho.

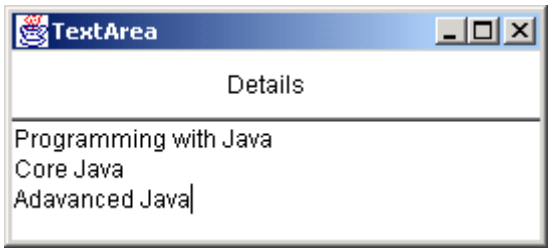
Bảng 5.3 Các phương thức của TextArea

Chương trình 5.5 chỉ ra cách sử dụng của TextArea:

Chương trình 5.5

```
import java.awt.*;
class TextAreatest extends Frame
{
        Label lbl=new Label("Details");
        TextArea ta1=new TextArea();
        public TextAreatest(String title)
        {
                super(title);
                setLayout(new FlowLayout());
                add(lbl);
                add(ta1);
        }
        public static void main(String args[])
        {
                TextAreatest t=new TextAreatest("TextArea");
                t.setSize(300,200);
                t.show();
        }
}
```

Kết xuất của chương trình được chỉ ra ở hình bên dưới:



Hình 5.7 TextArea

5.3.4 Button

Nút nhấn hay còn gọi là nút lệnh là một phần nguyên của bất kỳ GUI nào. Sử dụng button là cách dễ nhất để chặn các tác động của người dùng.

Để tạo một button, bạn làm theo các bước sau:

1) Tạo phần tử button với một nhãn chỉ ra mục đích của button.

2) Bố trí phần tử này trên màn hình.
3) Hiển thị phần tử trên màn hình.

Sử dụng một trong hai constructor sau để tạo các button trong Java:

- **Button()**
- **Button(String text)**

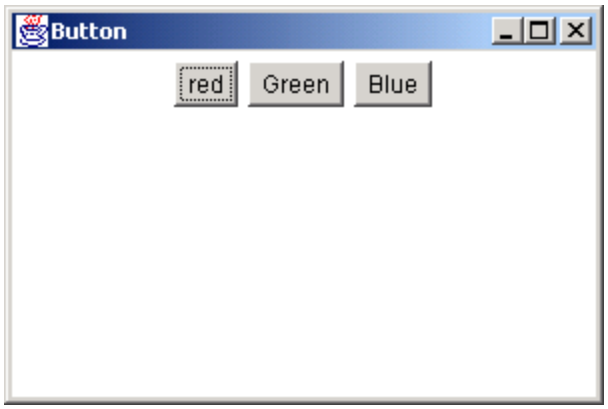
Sử dụng setLabel() và getLabel() để thiết lập và nhận về nhãn của button.

Ví dụ đơn giản sau đây sẽ tạo ra 3 button được trình bày trong chương trình 5.6:

Chương trình 5.6

```
import java.awt.*;
class Buttontest extends Frame
{
        Button b1 = new Button("red");
        Button b2 = new Button("Green");
        Button b3 = new Button("Blue");

        public Buttontest(String title)
        {
                super(title);
                setLayout(new FlowLayout());
                add(b1);
                add(b2);
                add(b3);
        }
        public static void main(String args[])
        {
                Buttontest t= new Buttontest("Button");
                t.setSize(300,200);
                t.show();
        }
}
```

Kết xuất của chương trình được chỉ ra ở hình 5.8.



Hình 5.8 Button

5.3.5 Checkbox và RadioButton

Checkbox được sử dụng khi người dùng tiến hành chọn một hay nhiều tùy chọn. Người dùng phải click trên các checkbox để chọn hay bỏ chọn chúng. Một radiobutton cũng tương tự như một checkbox. Nó được sử dụng như một option button để xác định các chọn lựa. Bạn có thể chỉ chọn một button trong nhóm các nút radiobutton, ngược lại bạn có thể chọn nhiều hơn một checkbox tại một thời điểm.

Làm theo các bước sau để tạo các checkbox hay radiobutton:

1) Tạo phần tử.
2) Quyết định trạng thái khởi đầu của phần tử (chọn hay không chọn).
3) Bố trí các phần tử trên màn hình.
4) Hiển thị các phần tử trên màn hình.

Thành phần checkbox có thể sử dụng một lớp phụ được gọi là CheckboxGroup để tạo ra các radiobutton.

Sử dụng các constructor sau để tạo các checkbox trong Java:

- **Checkbox()**: Tạo một checkbox trống.
- **Checkbox(String text)**: Tạo một checkbox với nhãn được cho.

Để tạo các radiobutton, đầu tiên chúng ta tạo đối tượng CheckboxGroup như sau:

CheckboxGroup cg=new CheckboxGroup();

Sau đó chúng ta tạo các button, như chỉ ra dưới đây:

Checkbox male=new Checkbox(“male”, cg, true);
Checkbox female=new Checkbox(“female”, cg, false);

Chúng ta sử dụng các phương thức setState() và getState() để thiết lập và nhận về trạng thái của checkbox.

Chương trình 5.7 minh họa cách sử dụng của các checkbox và các radiobutton:

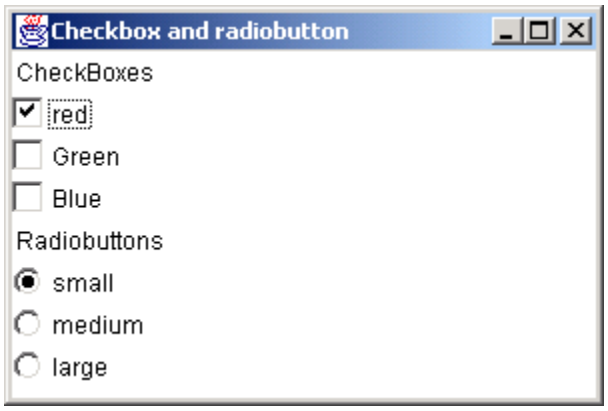
Chương trình 5.7

```
import java.awt.*;
class Checkboxtest extends Frame
{
        Label l1=new Label(“CheckBoxes”);
        Checkbox b1=new Checkbox(“red”,true);
        Checkbox b2=new Checkbox(“Green”,false);
        Checkbox b3=new Checkbox(“Blue”,false);
        Label l2=new Label(“Radiobuttons”);
        CheckboxGroup cb=new CheckboxGroup();
        Checkbox b4=new Checkbox(“small”,cb,true);
        Checkbox b5=new Checkbox(“medium”,cb,false);
        Checkbox b6=new Checkbox(“large”,cb,false);

        public Checkboxtest(String title)
        {
                super(title);
                setLayout(new GridLayout(8,1));
                add(l1);
                add(b1);
                add(b2);
                add(b3);
                add(l2);
                add(b4);
                add(b5);
                add(b6);
        }
        public static void main(String args[])
        {
                Checkboxtest t=new Checkboxtest(“Checkbox and radiobutton”);
                t.setSize(300,200);
```

```
                t.show();
        }
}
```

Đầu tiên chúng ta tạo một đối tượng Frame, đối tượng này hoạt động như một container sẽ chứa thành phần checkbox mà ta đã tạo. Sau đó ta tạo 5 checkbox, không một checkbox nào được đánh dấu chọn. Để làm được điều này, ta đưa giá trị False như một tham số cho hàm contructor Checkbox, ngoài ra còn có một tham số String là nhãn của checkbox. Nếu muốn hiển thị các điều khiển này theo dạng lưới, ta phải thiết lập cách trình bày đến dạng GridLayout có 6 dòng và 1 cột. Cuối cùng, ta tạo một biểu hiện cho lớp Checkboxtest và thiết lập kích thước cho frame. Để hiển thị nó, ta cho gọi phương thức show().

Kết xuất được chỉ ra ở hình bên dưới:



Hình 5.9 Checkbox

5.3.6 Danh sách chọn lựa (Choice List)

Thỉnh thoảng, rất cần thiết để trình bày một danh sách các chọn lựa đến người dùng trên một GUI. Người dùng có thể click vào một hay nhiều item từ danh sách. Một danh sách chọn lựa được tạo bằng cách sử dụng một số các chuỗi (String) hay các giá trị văn bản.

Để tạo các danh sách chọn lựa, hãy làm theo các bước được cho sau đây:

1) Tạo danh sách các phần tử.
2) Thêm các item (có kiểu là String) vào danh sách, mỗi lần chỉ thêm được một item.
3) Bố trí danh sách trên màn hình.
4) Hiển thị danh sách trên màn hình.

Java hỗ trợ lớp Choice cho phép chúng ta tạo các danh sách chứa nhiều item. Khi danh sách vừa được tạo ra, nó sẽ rỗng.

Choice colors=new Choice();

Mỗi thời điểm chỉ thêm được một item bằng cách sử dụng phương thức addItem như được chỉ ra bên dưới:

colors.addItem("Red");
colors.addItem("Green");

Chương trình 5.8 minh họa cách tạo một danh sách chọn lựa:

Chương trình 5.8

```
import java.awt.*;
class Choicetest extends Frame
{
        Label l1=new Label("What is your favorite color");
        Choice colors=new Choice();

        public Choicetest(String title)
        {
                super(title);
                setLayout(new FlowLayout());
                add(l1);
                colors.addItem("White");
                colors.addItem("Red");
                colors.addItem("Orange");
                colors.addItem("Green");
                colors.addItem("Yellow");
                colors.addItem("Blue");
                colors.addItem("Black");
                add(colors);
        }
        public static void main(String args[])
        {
                Choicetest t=new Choicetest("Choice list");
                t.setSize(300,200);
                t.show();
        }
```

}

Kết xuất được chỉ ra ở hình bên dưới:

Hình 5.10 Danh sách chọn lựa

5.4 Trình quản lý cách trình bày (Layout manager)

Layout manager điều khiển cách trình bày vật lý của các phần tử GUI như là button, textbox, option button v.v... Một layout manager tự động bố trí các thành phần này trong container.

Các kiểu trình bày khác nhau:

- Flow layout
- Border layout
- Card layout
- Grid layout
- GridBag Layout

Tất cả các thành phần mà chúng ta vừa tạo sử dụng layout manager mặc định. Cho ví dụ, 'FlowLayout' là cách trình bày mặc định của một applet. Layout manager này sẽ tự động xắp xếp các thành phần. Tất cả các thành phần được đặt trong một container, và được xắp xếp đến layout manager tương ứng. Layout manager được thiết lập bằng phương thức được gọi là 'setLayout()'.

Bây giờ chúng ta sẽ tìm hiểu chi tiết các cách trình bày và cách bố trí các thành phần của ta vào những vị trí mong muốn.

5.4.1 FlowLayout manager

'FlowLayout' là layout manager mặc định cho các applet và các panel. Các thành phần được xắp xếp từ góc trái trên đến góc phải dưới của màn hình. Khi một số thành phần được tạo, chúng được xắp xếp theo hàng, từ trái sang phải. Các constructor của FlowLayout:

FlowLayout mylayout = new FlowLayout() // constructor

//constructor with alignment specified

FlowLayout exLayout=new FlowLayout(FlowLayout.RIGHT);

setLayout(exLayout); //setting the layout to Flowlayout

Các điều khiển có thể được canh về bên trái, bên phải hay ở giữa. Để canh các điều khiển về bên phải, bạn sử dụng cú pháp sau:

setLayout(new FlowLayout(FlowLayout.RIGHT));

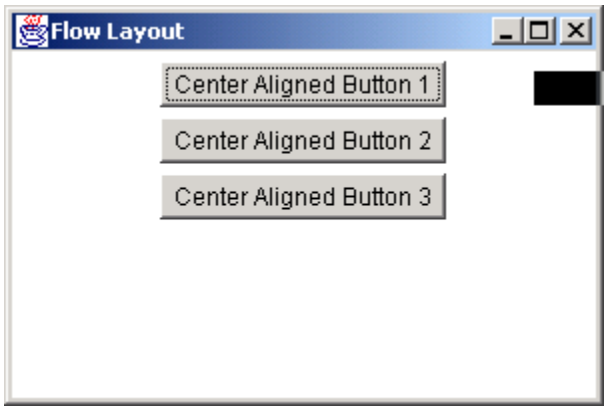
Chương trình 5.9 minh họa về FlowLayout manager. Ở đây, contructor không cần được gọi một cách tường minh, bởi vì cấu tử này được gọi mặc định cho một applet.

Chương trình 5.9

```
import java.awt.*;
class Fltest extends Frame
{
    Button b1=new Button("Center Aligned Button 1");
    Button b2=new Button("Center Aligned Button 2");
    Button b3=new Button("Center Aligned Button 3");

    public Fltest(String title)
    {
        super(title);
        setLayout(new FlowLayout(FlowLayout.CENTER));
        add(b1);
        add(b2);
        add(b3);
    }

    public static void main(String args[])
    {
```

```
        Fltest t=new Fltest("Flow Layout");
        t.setSize(300,200);
        t.show();
    }
}
```

Kết xuất của chương trình chỉ ra ở hình 5.11.



Hình 5.11 Flowlayout

5.4.2 BorderLayout Manager

'BorderLayout' là layout manager mặc định cho 'Window', 'Frame' và 'Dialog'. Layout này xắp xếp tối đa 5 thành phần trong một container. Những thành phần này có thể được đặt ở các hướng 'North', 'South', 'East', 'West' và 'Center' của container.

- **NORTH** – Đặt ở đỉnh của container.
- **EAST** – Đặt phía bên phải của container.
- **SOUTH** – Đặt ở phía dưới của container.
- **WEST** – Đặt phía bên trái của container.
- **CENTER** – Đặt ở giữa của container.

Để thêm một thành phần vào vùng 'North', bạn sử dụng cú pháp sau:

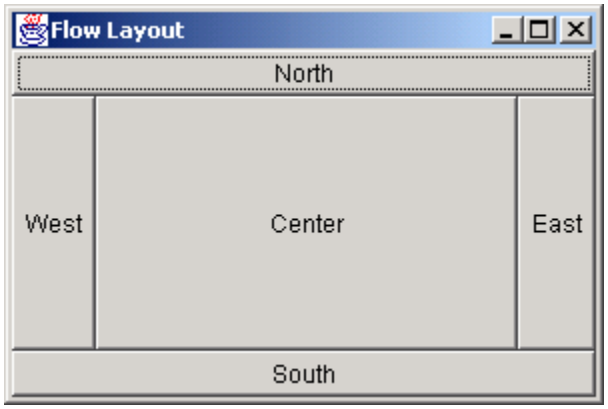
```
Button b1=new Button("North Button"); // khai báo thành phần
setLayout(new BorderLayout()); // thiết lập layout
add(b1,BorderLayout.NORTH); // thêm thành phần vào layout
```

Các thành phần vẫn giữ nguyên vị trí tương đối của chúng kể cả khi container bị thay đổi kích thước. Các thành phần được đặt trong vùng 'North', 'South' được dàn nằm ngang

trong khi đó các thành phần đặt trong vùng 'East' và 'West' lại được dàn thẳng đứng. Các thành phần được đặt trong vùng 'center' sẽ được dàn đều vào những khu vực nằm giữa của container.

add(b2,BorderLayout.CENTER); // thêm thành phần vào vùng 'center'

Khi tất cả các thành phần được đặt vào các vùng tương ứng, lúc đó Frame sẽ giống như sau:



Hình 5.12 BorderLayout

BorderLayout có thể chứa nhiều hơn 5 thành phần. Để thực hiện điều này, chúng ta có thể sử dụng các panel của các layout khác nhau để chứa các thành phần, và sau đó đặt các panel này vào trong border layout.

5.4.3 CardLayout Manager

CardLayout có thể lưu trữ một ngăn xếp (stack) các layout. Mỗi layout giống như một bảng (card). Bảng thường là đối tượng Panel. Một thành phần độc lập như button sẽ điều khiển cách trình bày các bảng ở lớp trên cùng.

Đầu tiên, chúng ta bố trí tập hợp các thành phần được yêu cầu trên các panel tương ứng. Mỗi panel sẽ được bố trí vào các layout khác nhau. Cho ví dụ:

panelTwo.setLayout(new GridLayout(2,1));

Panel chính sẽ chứa những panel này. Chúng ta thiết lập layout của panel chính là Cardlayout như sau:

CardLayout card=new CardLayout();

panelMain.setLayout(card);

Bước kế tiếp là thêm các panel khác vào panel chính:

panelMain.add("Red Panel", panelOne);

panelMain.add("Blue Panel", panelTwo);

Phương thức 'add()' sử dụng hai tham số. Tham số đầu tiên là một String làm nhãn của panel và tham số thứ hai là tên đối tượng Panel.

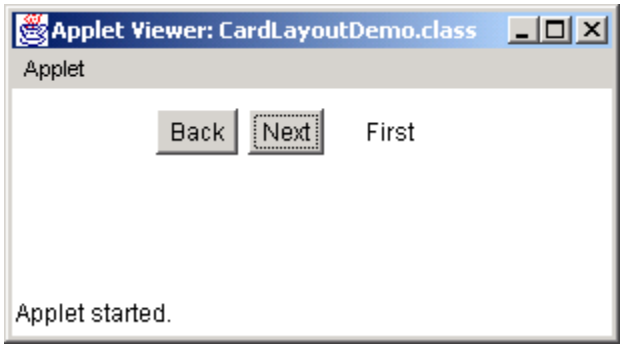
Chương trình 5.10 minh họa CardLayout:

Chương trình 5.10

```
import java.awt.*;
import java.applet.*;
/*<applet code="CardLayoutDemo.class" width="300" height="100"></applet>*/
public class CardLayoutDemo extends Applet
{
    Button back,next;
    Label lbl1,lbl2,lbl3,lbl4;
    TextField other1;
    Panel p1,first,second,third,fourth;
    CardLayout c1;
    public void init()
    {
        back=new Button("Back");
        next=new Button("Next");
        add(back);
        add(next);

        c1=new CardLayout();
        p1=new Panel();
        p1.setLayout(c1);// Set panel layout to CardLayout

        lbl1=new Label("First");
        lbl2=new Label("Second");
        lbl3=new Label("Third");
        lbl4=new Label("Fourth");
```

```
        //First panel
        first=new Panel();
        first.add(lbl1);

        //Second panel
        second=new Panel();
        second.add(lbl2);

        //Third panel
        third=new Panel();
        third.add(lbl3);

        //Fourth panel
        fourth=new Panel();
        fourth.add(lbl4);

        //Add panels to the card deck panel
        p1.add("1",first);
        p1.add("2",second);
        p1.add("3",third);
        p1.add("4",fourth);

        add(p1);
    }
}
```

Kết xuất của chương trình như sau:



Hình 5.13 CardLayout

Trong hình bên trên, các panel được thêm vào panel chính như là các thẻ riêng biệt. Vì thế chỉ có thẻ đầu tiên mới được thấy trên màn hình. Nhưng người dùng có thể điều hướng sang các panel khác sử dụng các phương thức của CardLayout.

5.4.4. GridLayout Manager

'GridLayout' trợ giúp việc chia container vào trong ô lưới. Các thành phần được đặt trong các dòng và các cột. Mỗi khung lưới nên chứa ít nhất một thành phần. Một khung lưới được sử dụng khi tất cả các thành phần có cùng kích thước.

Constructor GridLayout được tạo như sau:

Gridlayout g1=new GridLayout(4,3);

4 là số dòng và 3 là số cột.

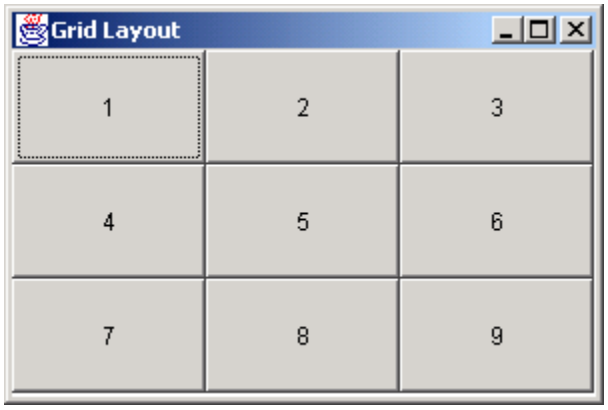
Chương trình 5.11 minh họa cách trình bày lưới:

Chương trình 5.11

```
import java.awt.*;
class Gltest extends Frame
{
    Button btn[];
    String str[]={"1", "2", "3", "4", "5", "6", "7", "8", "9"};
    public Gltest(String title)
    {
        super(title);
        setLayout(new GridLayout(3,3));
        btn=new Button[str.length];
        for (int I=0; I<str.length;I++)
        {
                btn[I]=new Button(str[I]);
                add(btn[I]);
        }
    }
    public static void main(String args[])
    {
        Gltest t=new Gltest("Grid Layout");
        t.setSize(300,200);
        t.show();
```

```
    }
}
```

Kết xuất chương trình như sau:



Hình 5.14 Grid Layout

5.4.5 GridBagLayout Manager

'GridBagLayout' hiệu quả và phức tạp hơn bất cứ layout nào khác. Layout này đặt các thành phần vào vị trí chính xác. Với layout này, các thành phần không cần có cùng kích thước. Nó tương tự như GridLayout manager, khi các thành phần được xắp xếp trong lưới theo dòng và cột. Tuy nhiên, thứ tự đặt các thành phần không theo nguyên tắc từ trái sang phải và từ trên xuống dưới.

```
GridBagLayout gb=new GridBagLayout()
ContainerName.setLayout(gb);
```

Để sử dụng layout này, bạn cần cung cấp thông tin về kích thước và layout của mỗi thành phần. Lớp 'GridBagLayoutConstraints' nắm giữ tất cả các thông tin mà lớp GridLayout cần để bố trí và định kích thước mỗi thành phần. Bảng sau liệt kê danh sách các biến thành viên của lớp GridBagConstraints:

Các biến thành viên	**Mục đích**
weightx, weighty	Chỉ ra sự phân phối của khoảng trống trong GridBagLayout. Giá trị mặc định cho các biến này là 0.
gridwidth, gridheight	Chỉ ra số lượng các ô (cell) bắt ngang hay đi xuống trong vùng hiển thị của một thành phần.
ipadx, ipady	Chỉ ra lượng làm thay đổi chiều cao và chiều

	rộng tối thiểu của thành phần. Nó sẽ thêm 2*ipadx vào chiều rộng tối thiểu và 2*ipady vào chiều cao tối thiểu của thành phần. Giá trị mặc định cho cả hai là 0.
Anchor	Chỉ ra cách xắp xếp các thành phần trong cell. Mặc định sẽ đặt vào giữa cell. Các thành viên dữ liệu tĩnh sau đây có thể được sử dụng: ➢ GridBagConstraints.NORTH ➢ GridBagConstraints.EAST ➢ GridBagConstraints.WEST ➢ GridBagConstraints.SOUTH ➢ GridBagConstraints.NORTHEAST ➢ GridBagConstraints.SOUTHEAST
gridx, gridy	Chỉ ra cell cần đặt một thành phần. Khi thiết lập giá trị của gridx là 'GridbagConstraints.RELATIVE' thì thành phần được thêm sẽ nằm ở vị trí bên phải của thành phần cuối cùng.
Fill	Chỉ ra cách mà một thành phần được bố trí vào cell thế nào nếu như cell lớn hơn thành phần. Mặc định kích thước thành phần lúc đó không thay đổi.

Bảng 5.4 Các biến thành viên của lớp GridBagConstraints

Bảng sau đây cung cấp một danh sách các biến dữ liệu tĩnh là các giá trị cho biến fill:

Giá trị	**Mô tả**
GridBagConstraints.NONE	Mặc định, không làm thay đổi kích thước của thành phần.
GridBagConstraints.	Tăng chiều rộng của thành phần theo chiều ngang (HORIZONTAL) để làm cho thành phần khớp với vùn
GridBagConstraints.	
GridBagConstraints.BOTH	
Insets	

Bảng 5.5 Các biến thành viên dữ liệu tĩnh của biến fill

Sử dụng phương thức ‘setConstraints()’ để thiết lập các hằng số cho mỗi thành phần. Cho ví dụ:

gblay.setConstraints(lb1, gbc);

‘gblay’ là đối tượng của lớp GridBagLayout, lbl là thành phần ‘Label’ và ‘gbc’ là đối tượng của lớp GridBagConstraints.

Chương trình 5.12 minh họa một ví dụ của GridBagLayout và GridBagConstraints.

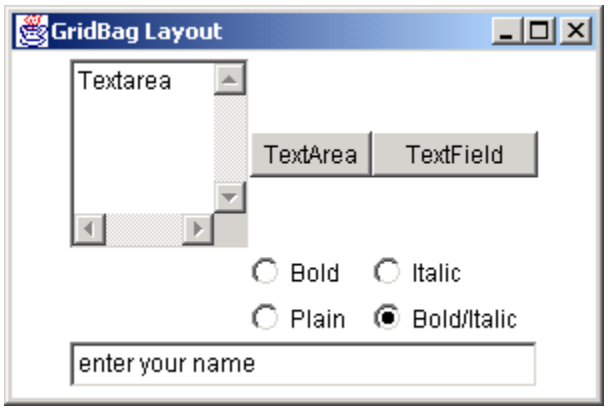
Chương trình 5.12

```
import java.awt.*;
class Gbltest extends Frame
{
    TextArea ta;
    TextField tf;
    Button b1,b2;
    CheckboxGroup cbg;
    Checkbox cb1,cb2,cb3,cb4;
    GridBagLayout gb;
    GridBagConstraints gbc;
    public GBltest(String title)
    {
        super(title);
        gb=new GridBagLayout();
        setLayout(gb);
        gbc=new GridBagConstraints();
        ta=new TextArea(“Textarea”,5,10);
        tf=new TextField(“enter your name”);
        b1=new Button(“TextArea”);
        b2=new Button(“TextField”);

        cbg=new CheckboxGroup();
        cb1=new Checkbox(“Bold”, cbg,false);
        cb2=new Checkbox(“Italic”, cbg,false);
        cb3=new Checkbox(“Plain”, cbg,false);
        cb4=new Checkbox(“Bold/Italic”, cbg,true);

        gbc.fill=GridBagConstraints.BOTH;
        addComponent(ta,0,0,4,1);
```

```
        gbc.fill=GridBagConstraints.HORIZONTAL;
        addComponent(b1,0,1,1,1);

        gbc.fill=GridBagConstraints.HORIZONTAL;
        addComponent(b2,0,2,1,1);

        gbc.fill=GridBagConstraints.HORIZONTAL;
        addComponent(cb1,2,1,1,1);

        gbc.fill=GridBagConstraints.HORIZONTAL;
        addComponent(cb2,2,2,1,1);

        gbc.fill=GridBagConstraints.HORIZONTAL;
        addComponent(cb3,3,1,1,1);

        gbc.fill=GridBagConstraints.HORIZONTAL;
        addComponent(cb4,3,2,1,1);

        gbc.fill=GridBagConstraints.HORIZONTAL;
        addComponent(tf,4,0,1,3);
    }

    public void addComponent(Component c, int row, int col, int nrow, int ncol)
    {
        gbc.gridx=col;
        gbc.gridy=row;

        gbc.gridwidth=ncol;
        gbc.gridheight=ncol;

        gb.setConstraints(c,gbc);
        add(c);
    }
    public static void main(String args[])
    {
        Gbltest t=new Gbltest("GridBag Layout");
        t.setSize(300,200);
        t.show();
    }
}
```

Khi một container bị thay đổi kích thước và khi khoảng trắng phụ tồn tại, các thành phần có chiều rộng lớn hơn sẽ chiếm giữ nhiều khoảng trống hơn là các thành phần có giá trị về chiều rộng nhỏ hơn.

Kết xuất của chương trình được chỉ ra ở hình 5.15



Hình 5.15 GridBagLayout

Giải thích đoạn mã trên:

gbc.fill=GridBagConstraints.BOTH;

Thành viên fill của lớp GridBagConstraints chỉ ra thành phần có thể được mở rộng theo hướng nằm ngang và thẳng đứng. Cú pháp sau mô tả thành phần chỉ được mở rộng theo hướng nằm ngang:

gbc.fill=GridBagConstraints.HORIZNTAL;

Cú pháp sau sẽ thêm vào thành phần TextArea với số dòng và số cột cần chiếm:

addComponent(ta,0,2,4,1);

0 – Khởi đầu từ dòng thứ 0
2 – Khởi đầu từ dòng thứ 2
4 – ta chiếm giữ 4 dòng
1 – ta chiếm 1 cột

Sử dụng cú pháp sau để bố trí các thành phần vào trong dòng và cột nào đó:

gbc.gridx=col;
gbc.gridy=row;

Ở đây (gridx,gridy) là cột và dòng nơi mà thành phần có thể được đặt vào.

Sử dụng cú pháp sau để chỉ ra số lượng các cột và dòng mà các thành phần có thể chiếm giữ:

gbc.gridwitdh=ncol;
gbc.gridheight=nrow;

Ở đây, gridwidth xác định số lượng các cột mà một thành phần chiếm giữ và gridheight xác định số lượng các dòng mà một thành phần chiếm giữ.

Khi một container bị thay đổi kích thước và khi khoảng trắng phụ tồn tại, các thành phần có chiều rộng lớn hơn sẽ chiếm giữ nhiều khoảng trống hơn là các thành phần có giá trị về chiều rộng nhỏ hơn.

5.5 Xử lý các sự kiện

Các hệ thống GUI xử lý các tương tác người dùng với sự trợ giúp của mô hình event-driven. Tương tác của người dùng có thể là di chuyển chuột, nhấn phím, nhả phím v.v...Tất cả các thao tác này thiết lập một sự kiện của một vài kiểu nào đó.

Việc xử lý những sự kiện này phụ thuộc vào ứng dụng. Abstract Windowing Toolkit (AWT) xử lý một vài sự kiện. Môi trường mà các ứng dụng này được thi hành ví dụ như trình duyệt cũng có thể xử lý các điều khiển khác. Người lập trình viên cần phải viết một hàm xử lý sự kiện.

Ứng dụng cần đăng ký một hàm xử lý sự kiện với một đối tượng. Hàm xử lý sự kiện này sẽ được gọi bất cứ khi nào sự kiện tương ứng phát sinh. JDK1.2 làm việc theo mô hình xử lý sự kiện này.

Trong tiến trình này, ứng dụng cho phép bạn đăng ký các handler, hay gọi là listener với các đối tượng. Những handler này tự động được gọi khi một sự kiện thích hợp phát sinh.

Một Event Listener lắng nghe một sự kiện nào đó mà một đối tượng thiết lập. Nghĩa là sẽ luân phiên gọi phương thức xử lý sự kiện. Mỗi event listener cung cấp các phương thức xử lý những sự kiện này. Lớp thi hành listener cần phải định nghĩa những phương thức này. Để sử dụng mô hình này, bạn làm theo các bước sau:

➢ Thực hiện giao diện listener thích hợp. Cấu trúc như sau:

public class MyApp extends Frame implements ActionListener

➢ Xác định tất cả các thành phần tạo ra sự kiện. Các thành phần có thể là các button, label, menu item, hay window.

Cho ví dụ, để đăng ký một thành phần với listener, ta có thể sử dụng:

exitbtn.addActionListener(This);

➢ Xác định tất cả các sự kiện được xử lý. Các sự kiện có thể là một 'ActionEvent' nếu một button được click hay một 'mouseEvent' nếu như chuột được kéo đi.

➢ Thi hành các phương thức của listener và viết hàm xử lý sự kiện tương ứng với các phương thức.

Bảng sau đây chỉ ra các sự kiện khác nhau và mô tả về chúng:

Lớp sự kiện	**Mô tả**
ActionEvent	Phát sinh khi một button được nhấn, một item trong danh sách chọn lựa được nhắp đôi hay một menu được chọn.
AdjustmentEvent	Phát sinh khi một thanh scrollbar được sử dụng.
ComponentEvent	Phát sinh khi một thành phần được thay đổi kích thước, được di chuyển, bị ẩn hay làm cho hoạt động được.
FocusEvent	Phát sinh khi một thành phần mất hay nhận focus từ bàn phím.
ItemEvent	Phát sinh khi một menu item được chọn hay bỏ chọn; hay khi một checkbox hay một item trong danh sách được click.
WindowEvent	Phát sinh khi một cửa sổ được kích hoạt, được đóng, được mở hay thoát.
TextEvent	Phát sinh khi giá trị trong thành phần text field hay text area bị thay đổi.
MouseEvent	Phát sinh khi chuột di chuyển, được click, được kéo hay bị thả ra.
KeyEvent	Phát sinh khi input được nhận từ bàn phím.

Các giao diện được thi hành để xử lý một trong số những sự kiện này là:

- ActionListener
- AdjustmentListener
- ComponentListener
- FocusListener
- ItemListener
- WindowListener
- TextListener
- MouseListener
- MouseMotionListener
- KeyListener

Các giao diện định nghĩa một số phương thức để xử lý mỗi sự kiện. Những phương thức này sẽ được nạp chồng trong lớp mà thi hành những giao diện này.

Chương trình sau đây sử dụng một ActionListener để xử lý các sự kiện liên quan với một button. ActionEvent có hai phương thức:

- **getSource()**: Để trả về nguồn của sự kiện.
- **toString()**: Để trả về chuỗi tương đương với sự kiện.

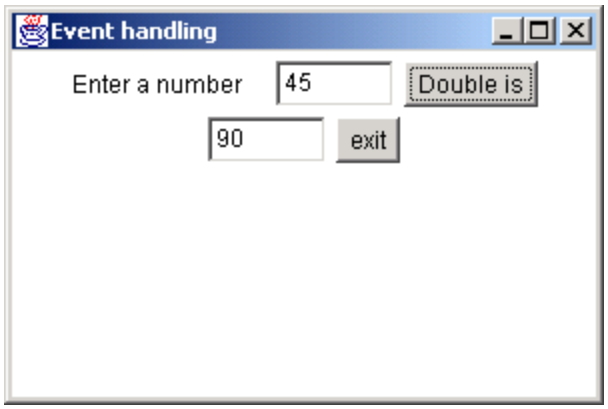
Chương trình 5.13 trình bày cách tính gấp đôi của một số được nhập vào. Chương trình này được thực hiện bằng cách kết hợp các phương thức của lớp, nghĩa là các phương thức xử lý sự kiện và giao diện. Việc click trên một button sẽ làm khởi động ActionEvent và gọi phương thức actionPerformed(). Nó sẽ kiểm tra button được click với sự trợ giúp của hàm getSource và trả về kết quả thích hợp.

Chương trình 5.13

```
import java.awt.*;
import java.awt.event.*;
class evttest extends Frame implements ActionListener
{
    Label lab=new Label("Enter a number");
    TextField tf1=new TextField(5);
    TextField tf2=new TextField(5);
    Button btnResult=new Button("Double is");
    Button ext=new Button("exit");
    public evttest(String title)
    {
```

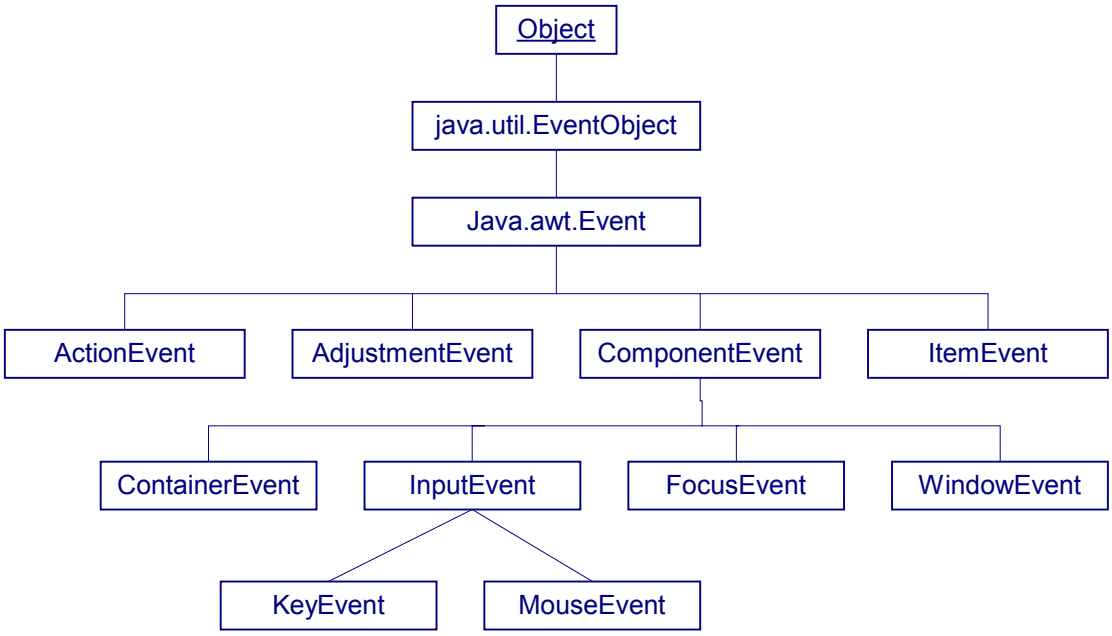
```
        super(title);
        setLayout(new FlowLayout());
        btnResult.addActionListener(this);
        ext.addActionListener(this);
        add(lab);
        add(tf1);
        add(btnResult);
        add(tf2);
        add(ext);
    }
    public void actionPerformed(ActionEvent ae)
    {
        if (ae.getSource()==btnResult)
        {
                int num=Integer.parseInt(tf1.getText())*2;
                tf2.setText(String.valueOf(num));
        }
        if (ae.getSource()==ext)
        {
                System.exit(0);
        }
    }
    public static void main(String args[])
    {
        evttest t=new evttest("Event handling");
        t.setSize(300,200);
        t.show();
    }
}
```

Kết xuất của chương trình được chỉ ra ở hình bên dưới:



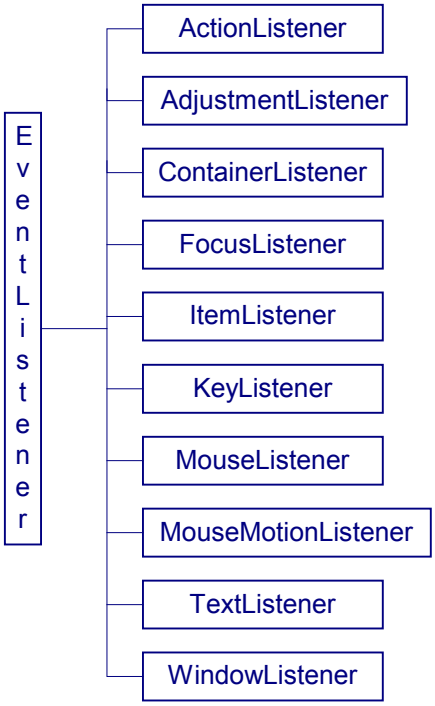
Hình 5.16 Xử lý sự kiện

Hình 5.17 chỉ ra một phần của cây phân cấp các lớp của gói event.



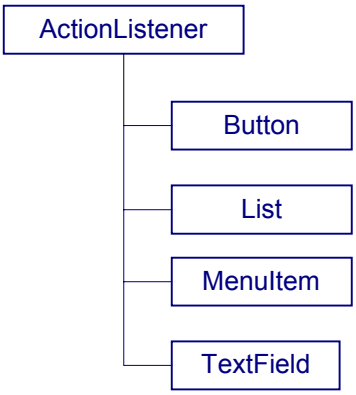
Hình 5.17 Gói Event

Hình sau chỉ ra thứ tự phân cấp các giao diện của các event listener.

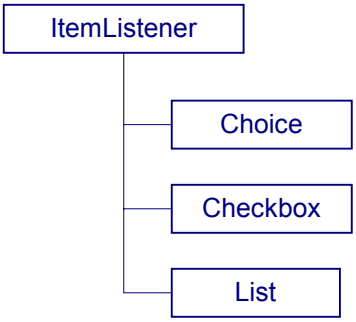


Hình 5.18 Event Listener

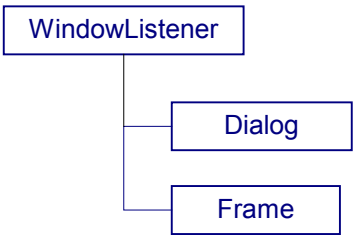
Hình sau là danh sách các listener được sử dụng cho các thành phần chỉ ra.



Hình 5.19 Action Listener

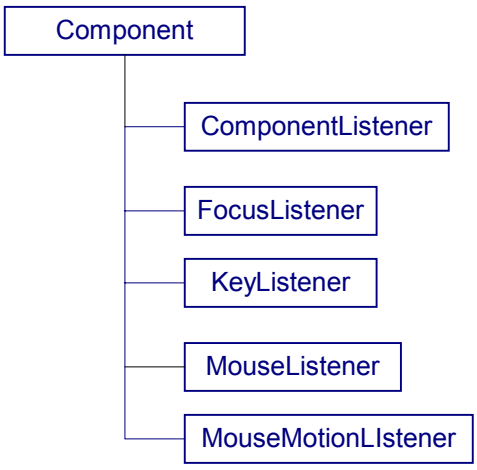


Hình 5.20 Item Listener



Hình 5.21 Window Listener

Các listener cho lớp Component được chỉ ra ở hình 5.22:



Hình 5.22 Các Component

5.6 Thực đơn (menu)

Ngôn ngữ Java có một tập hợp các lớp đối tượng để tạo các menu. Có hai loại menu – pull down và pop-up. Menu làm cho ứng dụng ta xây dựng dễ sử dụng hơn. Chỉ duy nhất một thanh menubar được đặt trong một frame. Menubar là một thanh nằm ngang được đặt tại

đỉnh của frame. Nó liệt kê các mục được chọn khác nhau hay menu. Một menu độc lập có thể chứa các mục chọn con, các mục con này được gọi là menuitem. Java cung cấp các checkbox menuitem, chúng có thể được bật hay mở, phụ thuộc vào trạng thái. Hình 5.14 minh họa cách sử dụng của menubar, menu, menuItem, và CheckboxMenuItem.

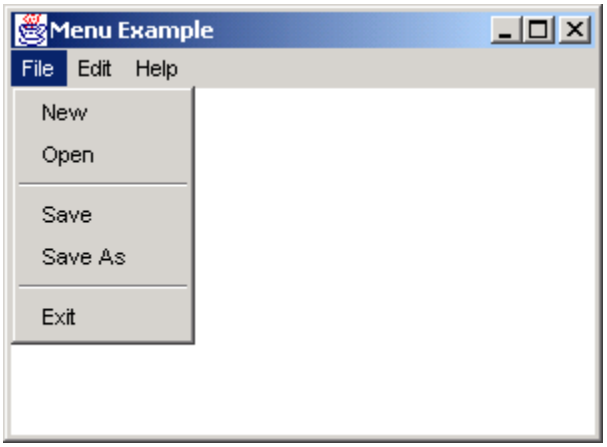
Chương trình 5.14

```
import java.awt.*;
import java.awt.event.*;
class MyFrame extends Frame implements ActionListener, MouseListener
{
    MenuItem exitItem;
    PopupMenu optionsMenu;
    Frame frame;
    public MyFrame()
    {
        setTitle("Menu Example");
        setSize(300,200);

        MenuBar mbar=new MenuBar();
        setMenuBar(mbar);

        Menu fileMenu=new Menu("File");
        mbar.add(fileMenu);
        fileMenu.addActionListener(this);
        MenuItem newItem=new MenuItem("New");
        fileMenu.add(newItem);
        MenuItem openItem=new MenuItem("Open");
        fileMenu.add(openItem);
        fileMenu.addSeparator();
        MenuItem saveItem=new MenuItem("Save");
        fileMenu.add(saveItem);
        MenuItem saveAsItem=new MenuItem("Save As");
        fileMenu.add(saveAsItem);
        fileMenu.addSeparator();
        exitItem=new MenuItem("Exit");
        fileMenu.add(exitItem);
        saveAsItem.addActionListener(this);

        Menu editMenu=new Menu("Edit");
        mbar.add(editMenu);
```

```
        editMenu.addActionListener(this);
        MenuItem cutItem=new MenuItem("Cut");
        editMenu.add(cutItem);
        MenuItem copyItem=new MenuItem("Copy");
        editMenu.add(copyItem);
        MenuItem pasteItem=new MenuItem("Paste");
        editMenu.add(pasteItem);
        editMenu.addSeparator();

        Menu helpMenu=new Menu("Help");
        mbar.add(helpMenu);
        helpMenu.addActionListener(this);
        MenuItem contentItem=new MenuItem("Content");
        helpMenu.add(contentItem);
        MenuItem indexItem=new MenuItem("Index");
        helpMenu.add(indexItem);
        Menu findMenu=new Menu("Find");
        helpMenu.add(findMenu);
        addMouseListener(this);
        MenuItem nameItem=new MenuItem("Search by Name");
        findMenu.add(nameItem);
        MenuItem cacheItem=new MenuItem("Search from cache");
        findMenu.add(cacheItem);
        optionsMenu=new PopupMenu("Options");
        editMenu.add(optionsMenu);
        optionsMenu.addActionListener(this);
        MenuItem readItem=new MenuItem("Read Only");
        optionsMenu.add(readItem);
        optionsMenu.addSeparator();
        Menu formatMenu=new Menu("Format text");
        optionsMenu.add(formatMenu);
        this.add(optionsMenu);
        formatMenu.addActionListener(this);
        CheckboxMenuItem insertItem=new CheckboxMenuItem("Insert",true);
        formatMenu.add(insertItem);
        CheckboxMenuIte overtypeItem=new CheckboxMenuItem("Overtype",false);
        formatMenu.add(overtypeItem);
    }
    public void actionPerformed(ActionEvent ae)
    {
        if (ae.getActionCommand().equals("Exit"))
```

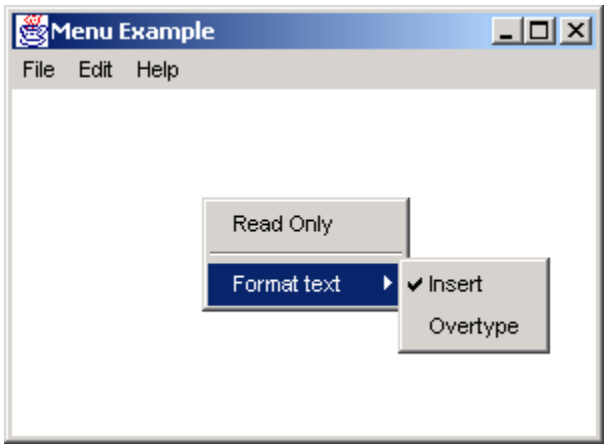
```
        {
                System.exit(0);
        }
    }
    public void mouseEntered(MouseEvent m){}
    public void mouseExited(MouseEvent m){}
    public void mouseClicked(MouseEvent m)
    {
        optionsMenu.show(this,m.getX(),m.getY());
    }
    public void mouseReleased(MouseEvent m){}
    public void mousePressed(MouseEvent m){}
    public static void main(String[] args)
    {
        MyFrame frame=new MyFrame();
        frame.show();
    }
}
```

Khi bạn thực thi chương trình trên, một màn hình với các trình đơn File, Edit và Help được hiển thị. Khi bạn click vào mục File, bạn sẽ thấy kết xuất sau đây:



Hình 5.23 Pull-down Menu

Một menu có thể chứa các menu con. Khi bạn click vào trình đơn Help, 3 mục con có tên là Content, Index và Find sẽ xuất hiện. Trong trình đơn Find, có 2 mục con là Search by name và Search from Cache. Mặt khác một pop-up menu sẽ hiện ra nếu bạn nhấn chuột phải trên màn hình:



Hình 5.24 Pop-up menu

Các mục chọn được trình bày trên pop-up menu là Read-Only và Format text. Mục 'Format text' có 2 mục con là Insert và Overtype. Những mục chọn con này thuộc kiểu Checkboxmenuitem. Khi bạn click vào mục chọn, nó sẽ được đánh dấu và bạn có thể thấy dấu chọn tương ứng trên mục được chọn đó. Ngôn ngữ Java cung cấp các lớp khác nhau. Những lớp này được sử dụng để tạo thanh Menubar, Menu, MenuItem và Checkboxmenuitem trong chương trình.

Tóm tắt

➢ GUI giúp chúng ta tạo giao diện hình ảnh cho một ứng dụng. Mặt khác nó cũng giúp ta phát triển các ứng dụng người dùng nhiều hiệu quả hơn.

➢ Thành phần GUI là một đối tượng trực quan. Người dùng có thể sử dụng chuột hay bàn phím để tương tác với đối tượng này.

➢ Các thành phần GUI như các button, label, checkbox và radio button mà được sử dụng trong ứng dụng hay applet thì có thể được thấy trên màn hình. Bất cứ thao tác nào mà liên quan tới tất cả các thành phần GUI đều được tìm thấy trong lớp Component. Ta cần sử dụng các lớp tồn tại trong gói java.awt để tạo các thành phần GUI này.

➢ Hệ thống GUI xử lý tất cả các tương tác của người dùng với sự hỗ trợ của mô hình event-driven. Một sự kiện được kích hoạt khi người sử dụng tạo một hành động như là di chuyển chuột, nhấn phím, nhả phím v.v....

➢ Các kiểu trình bày khác nhau:

- Flowlayout
- BorderLayout
- CardLayout
- GridLayout
- GridBagLayout

➢ Phương thức ‘setLayout()’ được sử dụng để tạo một layout.

➢ Flowlayout là Layout Manager mặc định cho các applet và các panel. Các thành phần được xắp xếp từ góc trái trên đến góc phải bên dưới của màn hình.

➢ Borderlayout xắp xếp các thành phần trong ‘North’, ‘South’, ‘East’, ‘West’ và ‘Center’ của một container.

➢ Gridlayout đặt các thành phần trong các dòng và các cột. Tất cả các thành phần đều có cùng kích thước.

➢ Cardlayout đặt các thành phần trên đỉnh của các thành phần khác. Nó tạo một stack của một số thành phần, thường thường là các panel.

➢ Gridlayout bố trí các thành phần một cách chính xác hơn layout manager. Nó tương tự như grid layout. Sự khác nhau duy nhất ở đây là thành phần không cần có cùng kích thước và có thể được đặt trong bất kỳ dòng hay cột nào.

➢ Trong mô hình xử lý sự kiện, ứng dụng cho phép bạn đăng ký các handler được gọi là các listener cho các đối tượng.

➢ Một Event Listener lắng nghe một sự kiện đặc biệt nào đó mà một đối tượng thiết lập. Nó sẽ gọi lần lượt các phương thức xử lý sự kiện. Lớp layout manager cung cấp một phương tiện để điều khiển cách trình bày vật lý của các thành phần GUI.

➢ Có hai kiểu menu – pull-down và pop-up.