



Chương 5. Gói Java I/O, Swing & Layouts



Mục tiêu

- Gợi Java.IO
- Các thành phần Swing và quản lý Layout

5.1. Gói **java.io**

- ❑ Cấu trúc vào ra của Java về thực chất dựa trên khái niệm dòng chảy (stream), đó là các lớp trợ giúp việc đọc và ghi tệp.
- ❑ Gói **java.io** chứa gần như tất cả các lớp cần thiết để thực hiện input và output (I/O) trong Java. Tất cả những stream này biểu diễn một nguồn input và một đích đến output. Stream trong gói java.io hỗ trợ nhiều kiểu dữ liệu như các kiểu dữ liệu nguyên thủy, kiểu Object,...



Input và Output với file

- Lớp **java.io.File** của Java cung cấp các hàm để điều hướng các tập tin cục bộ, mô tả các tập tin và thư mục.



Lớp *File*

- Lớp ***File*** đại diện cho tên của một tập tin hoặc thư mục đã tồn tại.
- Phương thức khởi tạo:
File(String pathname);



Các phương thức lớp File

- **boolean exists().**
- **String getAbsolutePath()**
- **String getName()**
- **String getParent()**
- **boolean isDirectory()**
- **boolean isFile()**
- **String[] list()**
- **boolean canRead()**
- **boolean canWrite()**
- **boolean createNewFile()**
- **boolean delete()**
- **long length()**
- **boolean mkdir()**
- **boolean renameTo(File *newname*)**

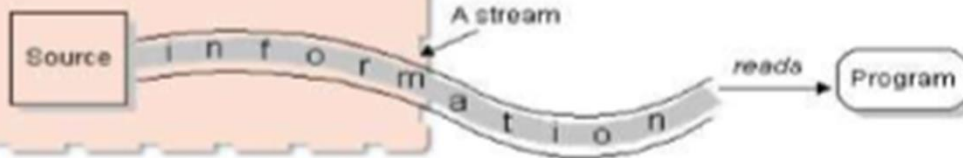


Streams, Readers, and Writers

- Các lớp luồng đọc, ghi của Java xem đầu vào và ra như là một trình tự sắp xếp của các byte.
- **Đọc và ghi file trong java** là các hoạt động **nhập/xuất** dữ liệu (nhập dữ liệu từ bàn phím, đọc dữ liệu từ file, ghi dữ liệu lên màn hình, ghi ra file, ghi ra đĩa, ghi ra máy in...) đều được gọi là stream.
- Một stream có thể được định nghĩa như là một dãy liên tục dữ liệu. **InputStream** được sử dụng để đọc dữ liệu từ một nguồn và **OutputStream** được sử dụng để ghi dữ liệu tới một đích đến.

Các loại luồng, kiểu luồng

Input Streams – lấy dữ liệu từ các nguồn: Files, Buffers và Sockets



Output Streams – ghi dữ liệu vào Files, Buffers in Memory, và Sockets



Luồng byte

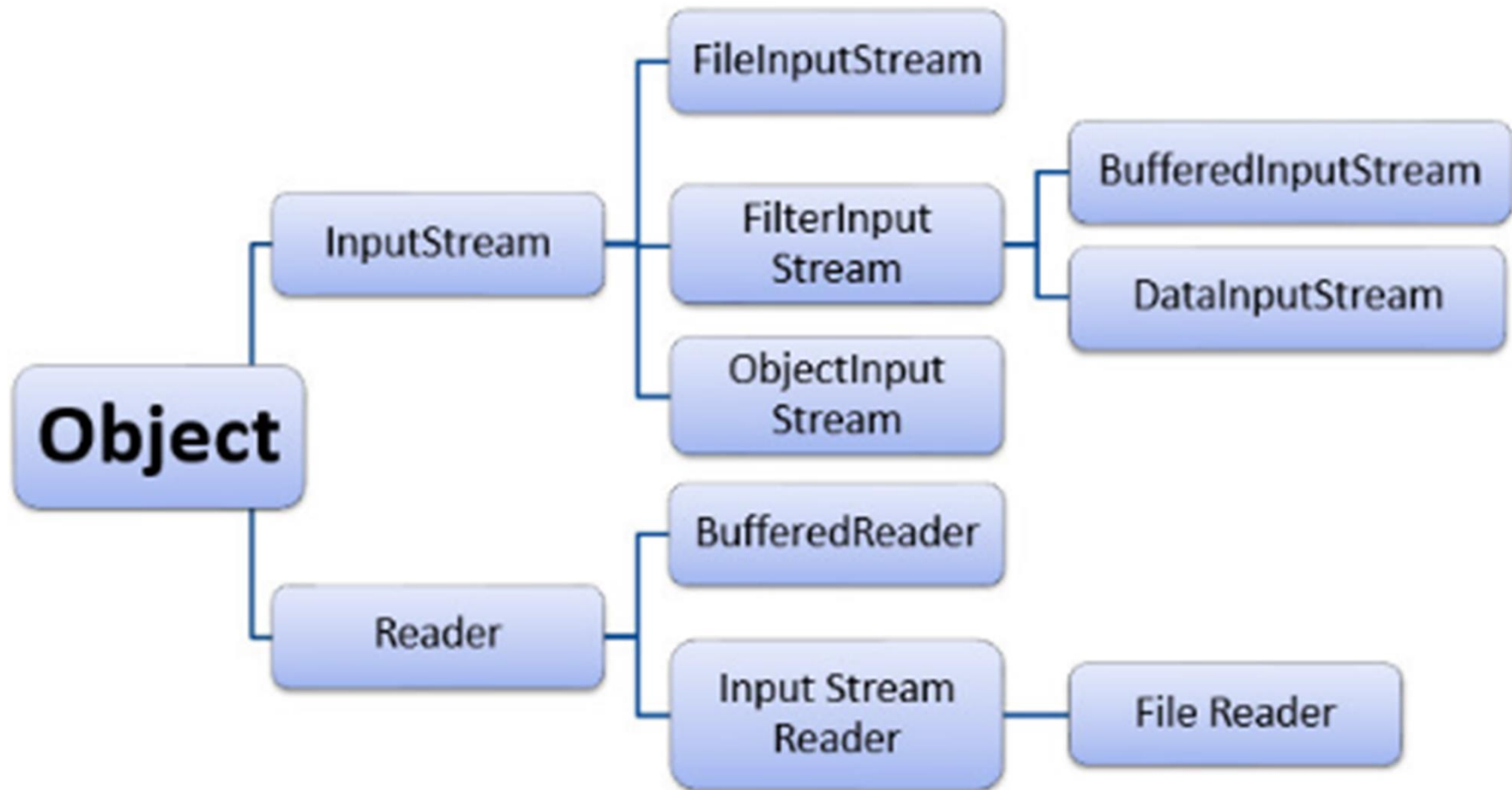
- Hỗ trợ việc xuất nhập dữ liệu trên byte,
- Thường được dùng khi đọc ghi dữ liệu nhị phân.

Luồng character

- Luồng character (ký tự) được thiết kế hỗ trợ việc xuất nhập dữ liệu kiểu ký tự (Unicode)

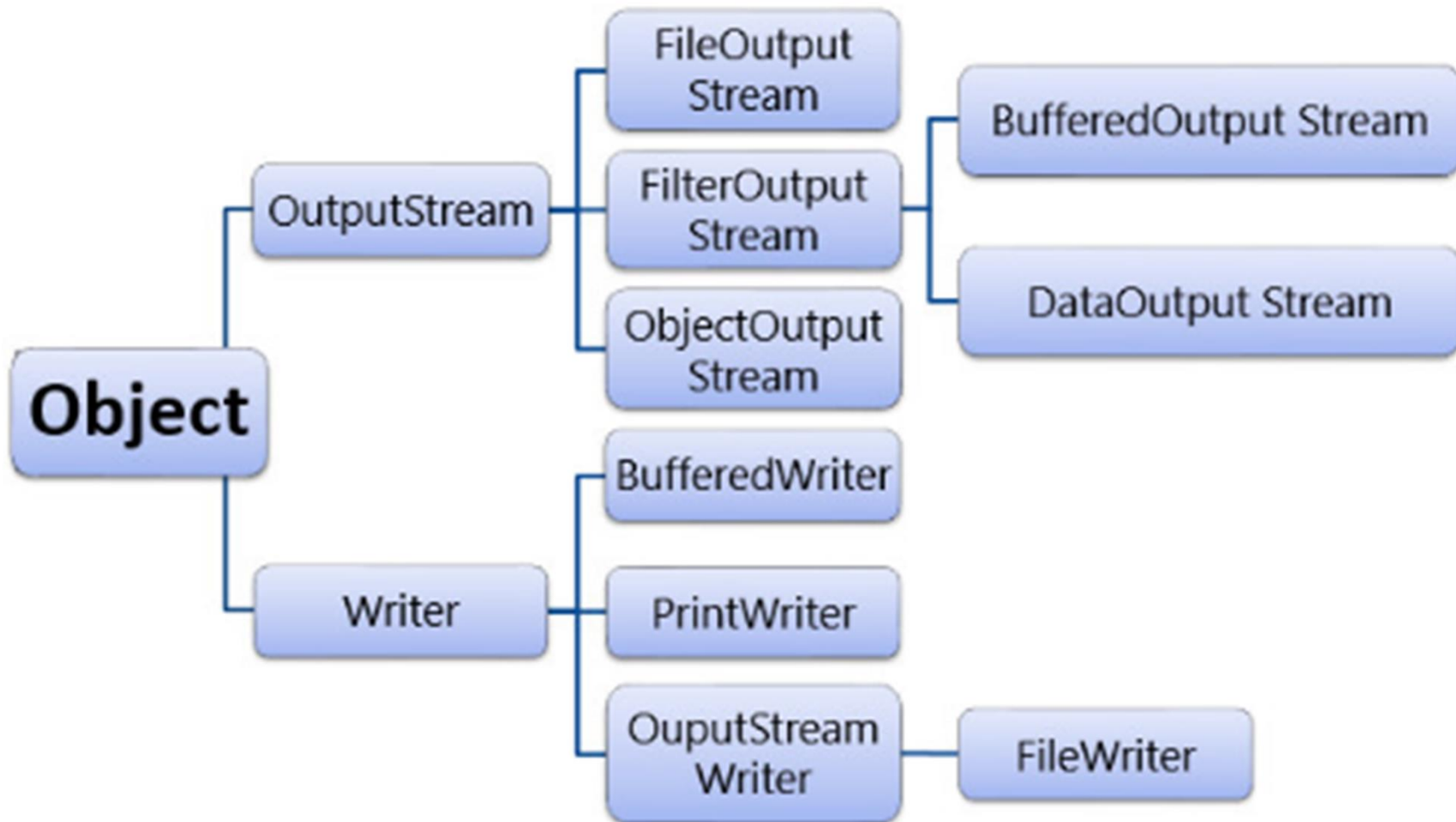
Input Stream

Đọc và ghi file trong java – Kiến trúc Input Stream (Luồng nhập dữ liệu)



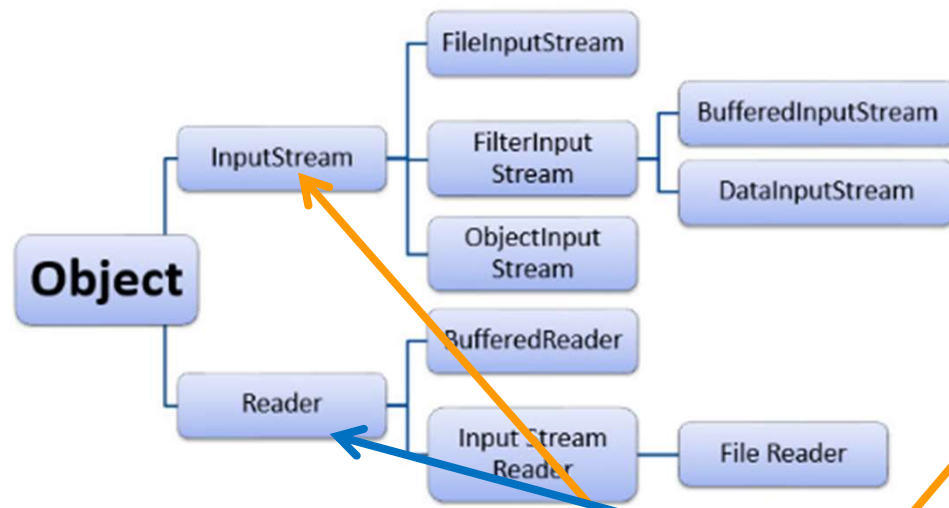
Output Stream

Đọc và ghi file trong java – Kiến trúc Output Stream (Luồng xuất dữ liệu)

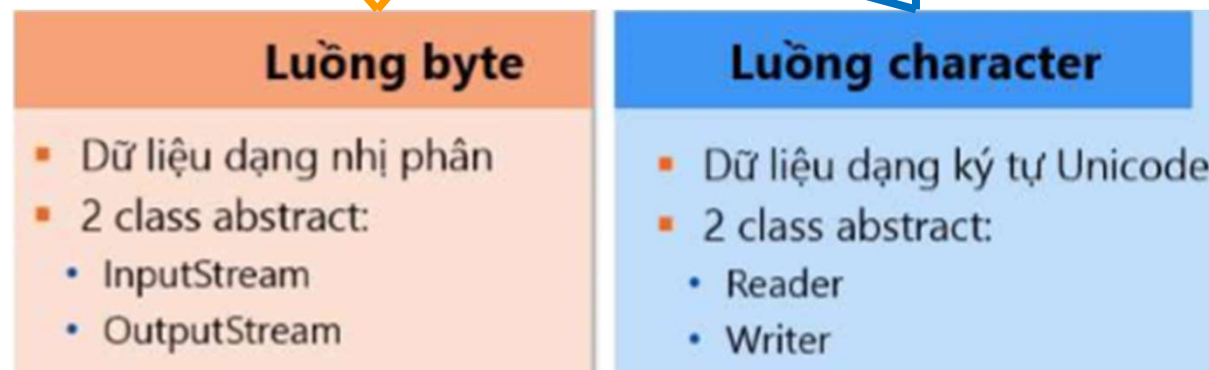
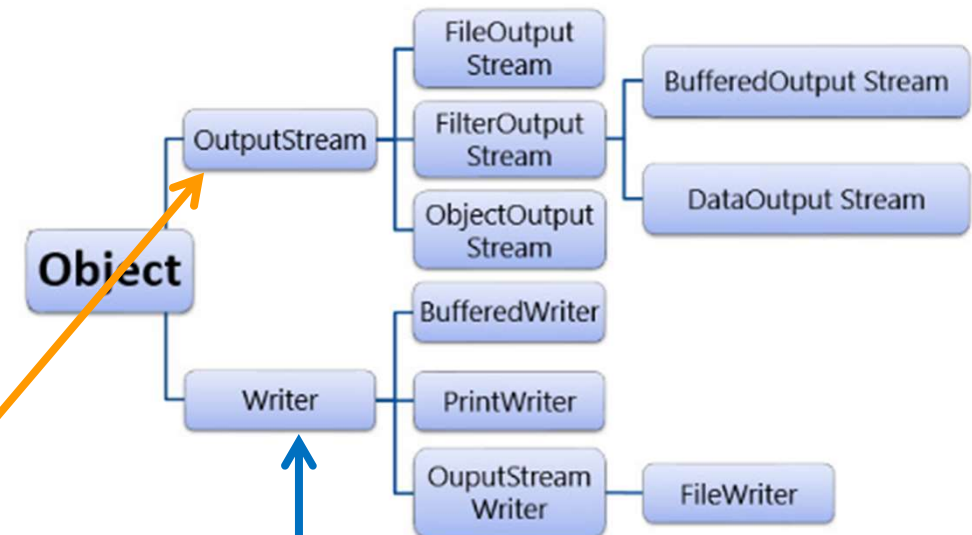


Các loại luồng, kiểu luồng

Đọc và ghi file trong java – Kiến trúc Input Stream (Luồng nhập dữ liệu)



Đọc và ghi file trong java – Kiến trúc Output Stream (Luồng xuất dữ liệu)





Các bước Đọc và Ghi file trong Java

- ☺ **Bước 1**: Tạo đối tượng luồng và liên kết với nguồn dữ liệu.
- ☺ **Bước 2**: Thao tác với dữ liệu (đọc hoặc ghi hoặc cả hai)
- ☺ **Bước 3**: Đóng luồng.



Xử lý nhập xuất dữ liệu sử dụng luồng byte

Sử dụng luồng byte trong các trường hợp như nhập xuất dữ liệu nguyên thủy, nhập xuất dữ liệu kiểu đối tượng (object)

Khi muốn tạo file chứa các kiểu dữ liệu như short, int, long, float, double, String, boolean... thì sử dụng 2 class

DataInputStream

xử lý việc nhập dữ liệu (đọc)

DataOutputStream

xử lý việc xuất dữ liệu (ghi)



DataInputStream trong Java

DataInputStream được sử dụng để đọc các dữ liệu kiểu nguyên thủy trong Java.

Một số phương thức:

- `boolean readBoolean()`
- `byte readByte()`
- `char readChar()`
- `double readDouble()`
- `float readFloat()`
- `int readInt()`
- `long readLong()`
- `short readShort()`



DataOutputStream trong Java

Một số phương thức:

- void writeBoolean(boolean var)
- void writeByte(int var)
- void writeChar(char var)
- void writeDouble(double var)
- void writeFloat(float var)
- void writeInt(int var)
- void writeIntLong(long var)
- void writeIntShort(short var)



Lớp FileReader trong Java

Lớp FileReader trong Java kế thừa từ lớp InputStreamReader được sử dụng để đọc các luồng ký tự.

Một số phương thức:

- **public int read() throws IOException**

Đọc một ký tự đơn. Trả về một int, biểu diễn ký tự đã đọc

- **public int read(char [] c, int offset, int len)**

Đọc các ký tự vào trong một mảng. Trả về số ký tự đã đọc



Lớp FileWriter trong Java

Lớp FileWriter trong Java kế thừa từ lớp OutputStreamWriter. Lớp này được sử dụng để ghi các luồng ký tự.

Một số phương thức:

- **public void write(int c) throws IOException**

Ghi một ký tự đơn

- **public void write(char [] c, int offset, int len)**

*Ghi một phần của mảng các ký tự bắt đầu từ **offset** với độ dài **len***

- **public void write(String s, int offset, int len)**

*Ghi một phần của chuỗi **s** bắt đầu từ **offset** với độ dài **len***



Đọc và Ghi dữ liệu kiểu Object

Sử dụng các lớp **ObjectInputStream** và **ObjectOutputStream** để đọc và ghi dữ liệu kiểu đối tượng.

Lưu ý: khi định nghĩa lớp đối tượng phải thừa kế giao diện ***Serializable*** của gói **java.io.Serializable**.

Một số phương thức:

- void writeObject(Object var)
- Object readObject()



5.2. Các thành phần của Swing và Quản lý Layout



Giới thiệu chung về AWT & Swing

❑ AWT: Abstract Windows Toolkit

- Thư viện API cung cấp các đối tượng GUI (Graphics User Interface)
- Tạo liên kết giao diện giữa ứng dụng Java và OS
- Gói java.awt

❑ Swing: là sự mở rộng của AWT

- Tạo giao diện nhất quán độc lập với môi trường
- Được nâng cấp nhiều thành phần
- Không sử dụng trộn lẫn AWT và Swing GUI
- Gói javax.swing
- Look and feel
- Thiết kế theo ý tưởng mô hình MVC (Model – View – Controller) và mô hình thành phần (component model)



5.2.1. Giới thiệu chung về AWT & Swing

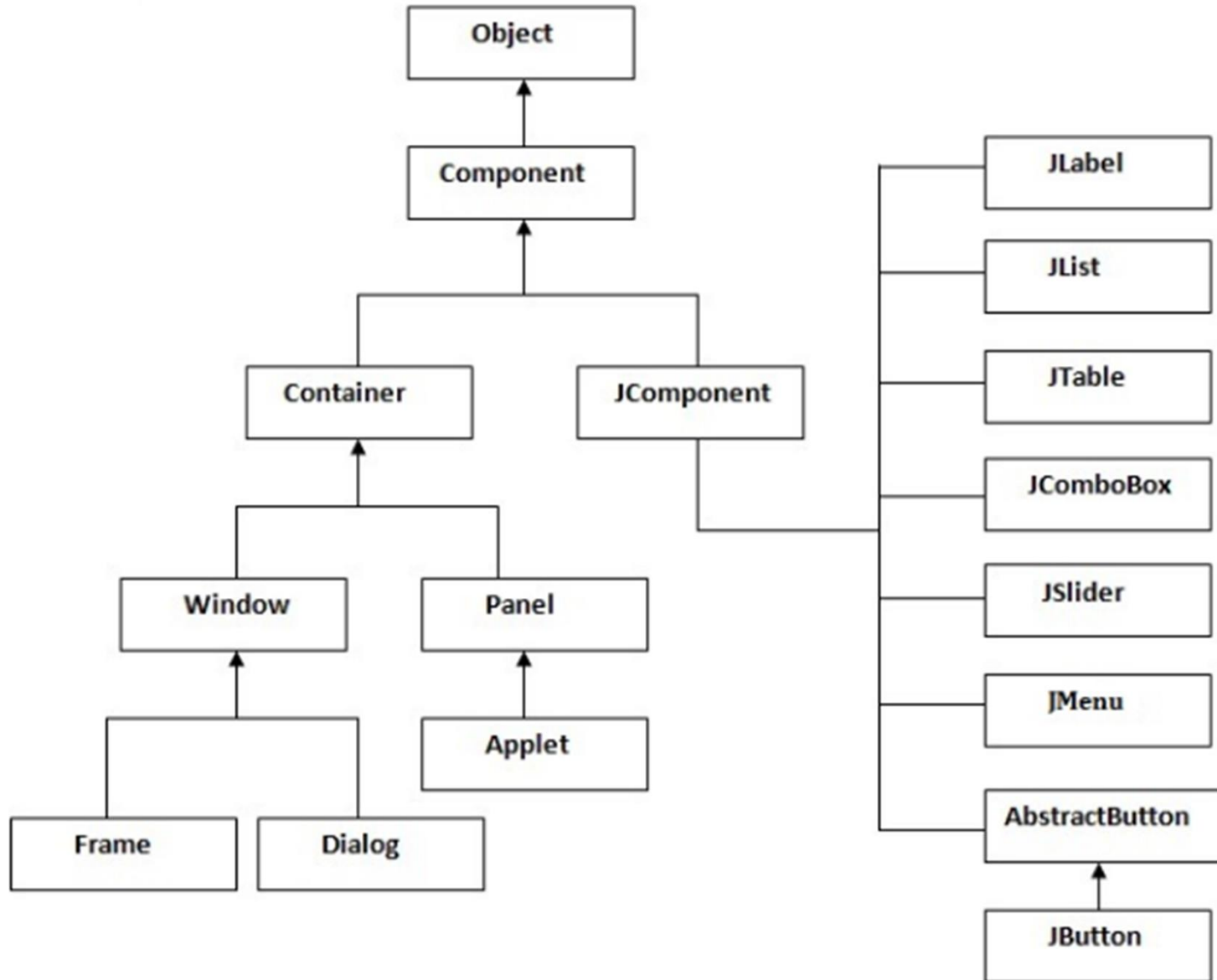
- Tương tác giữa người dùng và giao diện GUI điều khiển bởi controller.
- Standard Widget Toolkit (SWT) là sản phẩm công nghệ tương ứng của IBM – tích hợp trong Eclipse IDE.
- ❑ Cấu trúc giao diện GUI: tổ chức các đối tượng GUI trên 1 vật chứa (container) quản lý bởi trình quản lý bố trí (layout manager)

4 bước tiếp cận để tổ chức giao diện:

- ✓ Xác định các thành phần cần thiết.
- ✓ Tách các khu vực dựa theo hành vi.
- ✓ Phác họa GUI.
- ✓ Lựa chọn, bố trí các Layout.



Cấu trúc thứ bậc các lớp trong Java Swing



Một số phương thức phổ biến

Phương thức	Miêu tả
<code>public void add(Component c)</code>	Thêm một thành phần trên một thành phần khác
<code>public void setSize(int width,int height)</code>	Thiết lập kích cỡ của thành phần
<code>public void setLayout(LayoutManager m)</code>	Thiết lập Layout Manager cho thành phần
<code>public void setVisible(boolean b)</code>	Thiết lập tính nhìn thấy (visible) của thành phần. Theo mặc định là false



Các thành phần Swing Cơ bản

- Các lớp thành phần của Swing nằm trong gói javax.swing. Tên của tất cả các lớp được bắt đầu bằng chữ J.
- Có 3 nhóm chính:
 - ✓ Nhóm vật chứa (chứa đựng – container)
 - ✓ Nhóm thông dụng
 - ✓ Nhóm Menu

Vật chứa (Container)

- ❑ *Container* là thành phần mà nó chứa đựng các thành phần khác.
- ❑ Vật chứa (container) sử dụng các trình quản lý layout để xác định kích thước và vị trí của các thành phần trên nó.
- ❑ Khi xem xét về Container, cần chú ý các điểm sau:
 - Các lớp con của Container được gọi là Container. Một số ví dụ về các lớp con của Container là JPanel, JFrame và JWindow.
 - Container chỉ có thể thêm Component vào chính nó.
 - Một layout mặc định có mặt trong mỗi container. Layout này có thể bị ghi đè bởi sử dụng phương thức **setLayout()**.



Container: JFrame

- ***JFrame*** là một cửa sổ độc lập mà nó có thể di chuyển được trên màn hình.
- Bất kỳ một ứng dụng nào sử dụng GUI đều có ít nhất 1 frame để chứa các thành phần khác.



Container: JPanel

- ***Jpanel*** là một hình chữ nhật rỗng, nó có thể chứa các thành phần khác.
- Mỗi một panel sử dụng một trình quản lý layout để xác định vị trí và kích thước các thành phần con của nó.

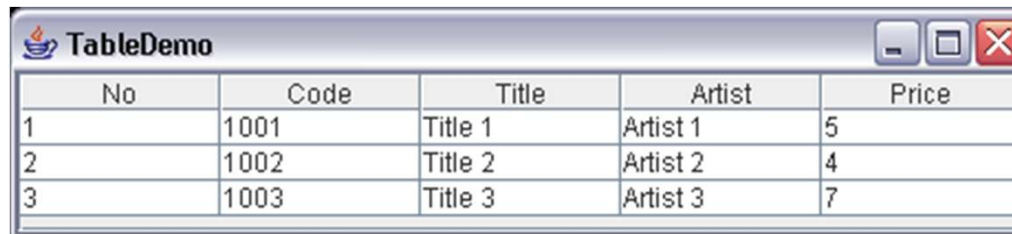


Các thành phần thông dụng

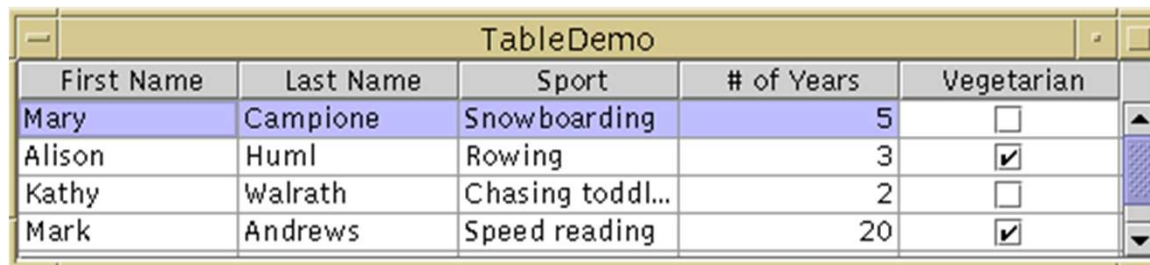
- JLabel
- JButton
- JCheckBox
- JRadioButton
- JScrollBar
- JTextField
- JTextArea
- JComboBox
- ...

Với lớp **JTable** ta có thể hiển thị bảng dữ liệu, cho phép người sử dụng có thể chỉnh sửa dữ liệu.

JTable không lưu trữ dữ liệu, nó chỉ đơn giản là hiển thị dữ liệu mà thôi

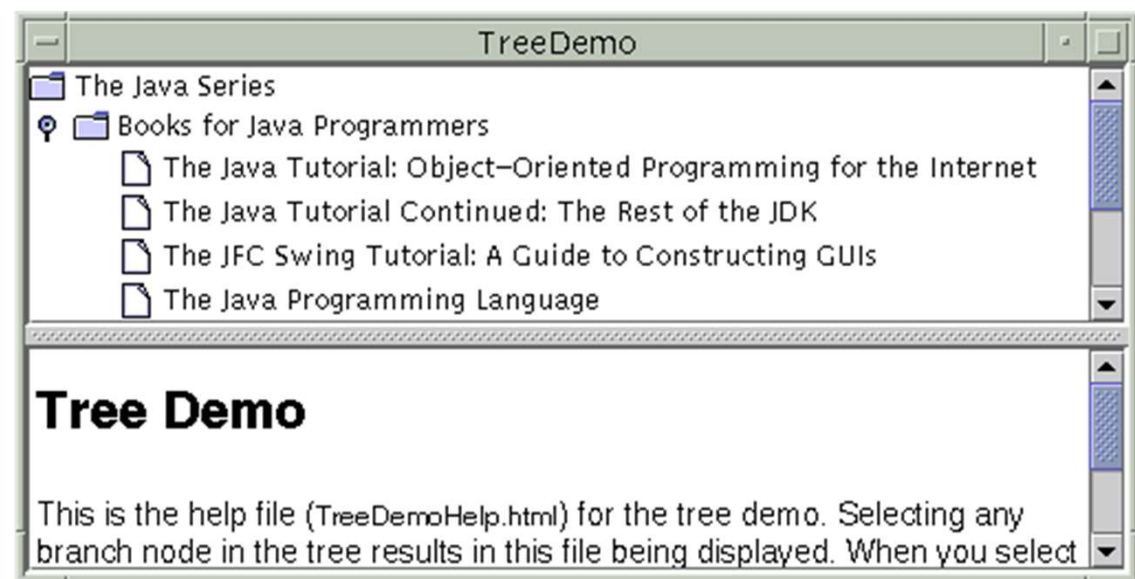
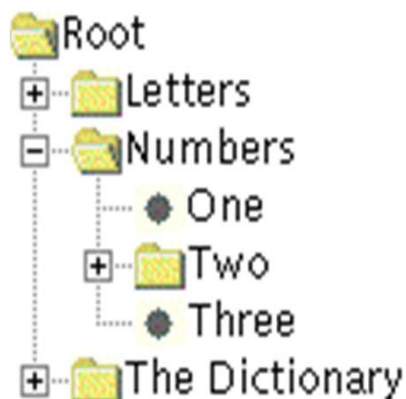
A screenshot of a Java Swing window titled "TableDemo". It contains a JTable with 5 columns: "No", "Code", "Title", "Artist", and "Price". The table has 3 rows of data.

No	Code	Title	Artist	Price
1	1001	Title 1	Artist 1	5
2	1002	Title 2	Artist 2	4
3	1003	Title 3	Artist 3	7

A screenshot of a Java Swing window titled "TableDemo". It contains a JTable with 5 columns: "First Name", "Last Name", "Sport", "# of Years", and "Vegetarian". The table has 4 rows of data. The first row is selected (highlighted in blue). The "Vegetarian" column contains checkboxes.

First Name	Last Name	Sport	# of Years	Vegetarian
Mary	Campione	Snowboarding	5	<input type="checkbox"/>
Alison	Huml	Rowing	3	<input checked="" type="checkbox"/>
Kathy	Walrath	Chasing toddl...	2	<input type="checkbox"/>
Mark	Andrews	Speed reading	20	<input checked="" type="checkbox"/>

- Với lớp **JTree** ta có thể hiển thị thứ bậc của dữ liệu. Một đối tượng JTree không chứa dữ liệu, nó đơn giản cung cấp một cách hiển thị dữ liệu.





Xử lý các sự kiện (Handling Events)

- Mỗi khi người sử dụng gõ 1 ký tự hoặc bấm chuột là nó xảy ra một sự kiện.
- Bất kỳ một đối tượng nào cũng có thể cho ra một sự kiện.



Handling Events...

public class SwingApplication implements **ActionListener** {

...

JButton button = new JButton("I'm a Swing button!");

button.addActionListener(this);

....

public void **actionPerformed**(ActionEvent e) {

numClicks++;

label.setText(labelPrefix + numClicks);

}

}



Handling Events...

Mỗi một sự kiện cần xử lý đều yêu cầu 3 việc sau:

- Khai báo đối với lớp là có xử lý đến sự kiện

public class Handler implements ActionListener {...}

- Đăng ký sẽ xử lý sự kiện đối với đối tượng

component.addActionListener(instanceOf Handler);

- Triển khai các hành động tương ứng sự kiện của đối tượng

public void actionPerformed(ActionEvent e) {

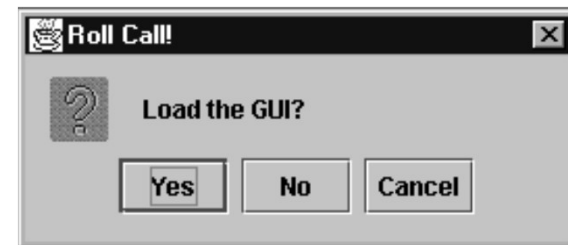
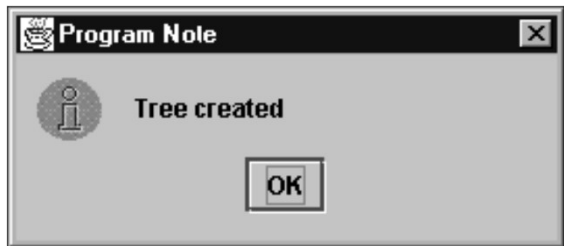
...//code that reacts to the action...

}

Some Events and Their Associated Event Listeners	
Act that Results in the Event	Listener Type
User clicks a button, presses Enter while typing in a text field, or chooses a menu item	ActionListener
User closes a frame (main window)	WindowListener
User presses a mouse button while the cursor is over a component	MouseListener
User moves the mouse over a component	MouseMotionListener
Component becomes visible	ComponentListener
Component gets the keyboard focus	FocusListener
Table or list selection changes	ListSelectionListener
Any property in a component changes such as the text on a label	PropertyChangeListener

❑ *JOptionPane*

- *showMessageDialog*
- *showInputDialog*



Các trình quản lý Layout

Mỗi một container đều có Layout ngầm định của nó.

Đơn giản: **1** FlowLayout GridLayout **2**

Mục đích đặc biệt: **3** BorderLayout CardLayout **4**

Linh hoạt: **5** GridBagLayout

Trình quản lý FlowLayout

Đơn giản:



```
Container contentPane = getContentPane();  
contentPane.setLayout(new FlowLayout());  
contentPane.add(new JButton("Button 1"));
```

- FlowLayout là layout ngầm định của **JPanel**.
- FlowLayout được sử dụng để sắp xếp các thành phần trong một line từ trái qua phải, line sau nối tiếp line trước.



Trình quản lý **FlowLayout**

Lớp **FlowLayout** bao gồm các trường:

- Static int **CENTER**: căn chỉnh vào giữa
- Static int **LEFT**: căn chỉnh trái
- Static int **RIGHT**: căn chỉnh phải
- Static int **LEADING**: căn đầu dòng theo hướng
- Static int **TRAILING**: căn cuối dòng theo hướng

Các hàm khởi tạo:

- **FlowLayout()**: căn chỉnh trung tâm, kẽ hở 5 đơn vị
- **FlowLayout(int align)**: căn chỉnh align, kẽ hở 5 đơn vị
- **FlowLayout(int align, int hgap, int vgap)**: căn chỉnh align và kẽ hở hgap, vgap.



Trình quản lý GridLayout

- ❑ GridLayout chia nhỏ vùng của nó thành ma trận các hàng và các cột (lưới các ô chữ nhật). Mỗi thành phần được hiển thị trong một ô.
- ❑ Lớp GridLayout gồm các constructor sau
 - **GridLayout()**: Tạo một grid layout với mặc định là một cột mỗi thành phần, trong một hàng đơn.
 - **GridLayout(int rows, int columns)**: Tạo một grid layout với số hàng và cột đã cho, và không có khoảng cách giữa các thành phần.
 - **GridLayout(int rows, int columns, int hgap, int vgap)**: Tạo một grid layout với các hàng và cột đã cho cùng với các khoảng cách theo chiều dọc và ngang đã xác định.

Trình quản lý GridLayout

Một số phương thức của lớp GridLayout

- ❑ void **addLayoutComponent**(String name, Component comp): Thêm thành phần comp đã cho với tên đã xác định tới layout.
- ❑ void **layoutContainer**(Container parent): Bố trí container đã cho sử dụng layout này.
- ❑ void **removeLayoutComponent**(Component comp): Xóa thành phần đã cho từ layout.
- ❑ void **setColumns**(int cols): Thiết lập số cột trong layout.
- ❑ void **setRows**(int rows): Thiết lập số hàng trong layout.
- ❑ void **setHgap**(int hgap): Thiết lập khoảng cách theo chiều ngang giữa các thành phần.
- ❑ void **setVgap**(int vgap): Thiết lập khoảng cách theo chiều dọc giữa các thành phần.



Trình quản lý BorderLayout

- BorderLayout manager chia vùng của nó thành 5 miền:
 - ✓ North
 - ✓ South
 - ✓ East
 - ✓ West
 - ✓ Center
- Mỗi một miền có thể rỗng hoặc chứa 1 thành phần.
- Bổ sung 1 thành phần vào BorderLayout

container.add(component, BorderLayout.NORTH);

Trình quản lý CardLayout

- ❑ Lớp CardLayout quản lý các thành phần theo một phương thức mà chỉ có một thành phần là nhìn thấy (visible) tại một thời điểm.

Hàm khởi tạo:

- **CardLayout()**: tạo một Card Layout với các khoảng cách kẽ hở theo chiều dọc và ngang là 0.
- **CardLayout(int hgap, int vgap)**: tạo một Card Layout với các khoảng cách với biên theo chiều dọc và ngang đã cho.

Các phương thức

- ❑ **public void next(Container parent):** được sử dụng để lật tới card tiếp theo của container đã cho
- ❑ **public void previous(Container parent):** được sử dụng để lật tới card trước đó của container đã cho
- ❑ **public void first(Container parent):** được sử dụng để lật tới card đầu tiên của container đã cho
- ❑ **public void last(Container parent):** được sử dụng để lật tới card cuối cùng của container đã cho
- ❑ **public void show(Container parent, String name):** được sử dụng để lật tới card đã được xác định bởi tên name đã cho.



Bổ sung một thành phần vào CardLayout

```
JPanel p = new JPanel();  
p.setLayout(new CardLayout());  
JButton b = new JButton("A Component");  
p.add(b, "Button-B");
```



Component's Name

Trình quản lý GridBagLayout

- ❑ **GridBagLayout** là một lớp quản lý layout linh động. Đối tượng của GridBagLayout căn chỉnh các thành phần theo chiều dọc, ngang hoặc theo đường gốc của chúng mà không yêu cầu các thành phần phải có cùng kích cỡ.
- ❑ **GridBagLayout** là một lớp bố cục khá hữu dụng, nếu được sử dụng một cách hiệu quả thì nó sẽ là sự thay thế hoàn hảo cho công việc của FlowLayout, GridLayout và BorderLayout. GridBagLayout chia một container thành một mảng của các ô (cell), nhưng không giống như các ô trong GridLayout, mỗi ô trong GridBagLayout có thể có chiều cao và chiều rộng khác nhau. Một component có thể chiếm giữ một phần hoặc toàn bộ không gian của một ô, hoặc nó có thể chiếm giữ nhiều ô cùng một lúc.



Trình quản lý GridBagLayout

- ❑ Để đặt một component đúng vị trí GridBagLayout cần rất nhiều thông tin, những thông tin này nằm trong lớp GridBagConstraints như gridx, gridy, gridwidth, gridheight, anchor, fill ...
- ❑ Khi đưa một component vào container ta sử dụng phương thức **add(Component, Object)**, ở đây **Object** chính là đối tượng **GridBagConstraints**.
- ❑ Để sử dụng GridBagLayout, có 3 vấn đề cần xem xét:
 - Kích thước của các ô (cell) khác nhau, chúng co giãn như thế nào khi kích thước của container thay đổi
 - Vị trí để đặt component phải được xác định rõ ràng
 - Component được đặt như thế nào trong vùng không gian mà nó chiếm giữ



Trình quản lý GridBagLayout

Các giá trị của GridBagConstraints

1. **gridx** và **gridy**: Vị trí ô trên khung lưới của đối tượng mà ta đưa vào: ví dụ gridx=1, gridy=2 tức là đặt đối tượng tại ô cột 1, hàng 2
2. **gridwidth**, **gridheight**: kích thước của đối tượng đưa vào, tức là đối tượng chứa mấy ô theo chiều ngang (width) và chiều dọc (height)
3. **fill**: giả sử một đối tượng đưa vào không khít hết các ô nó chiếm thì ta có thể dùng các giá trị sau:
 - Giá trị **GridBagConstraints.NONE** tức là giữ nguyên không đổi kích thước đối tượng.
 - Giá trị **GridBagConstraints.VERTICAL** giãn chiều cao của đối tượng cho khít
 - Giá trị **GridBagConstraints.HORIZONTAL** giãn chiều ngang của đối tượng cho khít
 - Giá trị **GridBagConstraints.BOTH** giãn cả 2 chiều của đối tượng cho khít



Các giá trị của GridBagConstraints

1. **ipadx, ipady**: tăng kích thước hai bên trái phải (hoặc trên dưới) đối tượng thêm ipadx (ipady) pixel khi kích thước khung chứa thay đổi
2. **insets**: đối tượng thuộc lớp Insets chỉ ra khoảng cách 4 phía so với đối tượng khác.
Ví dụ: `constraint.insets = new Insets(5, 5, 5, 5)` thì các đối tượng sẽ cách nhau 5 pixel các chiều
3. **anchor**: neo đối tượng theo vị trí nào đấy so với ô đặt nó. Có các giá trị: NORTH, NORTHEAST, NORTHWEST, WEST, EAST, SOUTH, SOUTHEAST, SOUTHWEST.
4. **weightx, weighty**: khoảng cách lớn ra tương đối giữa các đối tượng