

**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG**



# **LẬP TRÌNH MẠNG**

*(Dùng cho sinh viên hệ đào tạo đại học từ xa)*

**Lưu hành nội bộ**

**2010**

# LẬP TRÌNH MẠNG

Biên soạn : NINH XUÂN HẢI

# CHƯƠNG I

## NGÔN NGỮ JAVA

### I. LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

#### 1. Các phương pháp lập trình

##### *a) Lập trình tuyến tính*

Toàn bộ chương trình chỉ là một đơn thể duy nhất, các lệnh được thực hiện tuần tự theo thứ tự xuất hiện trong chương trình. Lập trình tuyến tính đơn giản nhưng khó sửa lỗi, khó mở rộng.

##### *b) Lập trình hướng thủ tục*

Chương trình được tách thành nhiều phần gọi là hàm hay thủ tục. Mỗi hàm sẽ thực hiện một chức năng của chương trình. Trong chương trình thường có một hàm chính (main), khi chương trình thực thi sẽ gọi hàm main, hàm main có thể gọi các hàm khác, các hàm khác lại có thể gọi lẫn nhau. Lập trình hướng thủ tục dễ sửa lỗi, dễ mở rộng, nhưng vì dữ liệu và hàm tách biệt nên khó bảo vệ dữ liệu và hàm, để không bị truy xuất bởi các hàm không mong đợi. Khi sửa đổi dữ liệu các hàm truy xuất phải thay đổi theo, ngoài ra khó sử dụng lại các hàm đã viết sẵn.

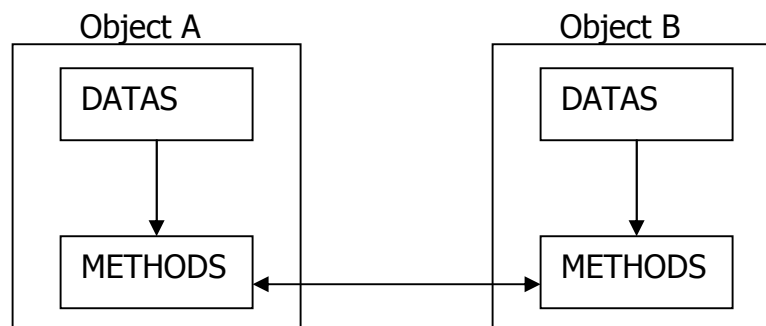
##### *c) Lập trình hướng đối tượng*

Chương trình sẽ được tách thành nhiều lớp, mỗi lớp gồm có dữ liệu (biến) và phương thức (hàm) xử lý dữ liệu. Do dữ liệu và hàm được đóng gói thành lớp nên LTHĐT sẽ có 3 đặc tính sau:

- Tính đóng gói (Encapsulation): Việc tổ chức dữ liệu và phương thức trong một lớp gọi là tính đóng gói, tính đóng gói cho phép bảo vệ dữ liệu, che dấu chi tiết cài đặt.

- Tính thừa kế (Inheritance): Sử dụng lớp có trước (lớp cha) để xây dựng lớp mới (lớp con) gọi là tính thừa kế. Lớp con được thừa hưởng những thuộc tính, phương thức của lớp cha và có thể có thêm những thuộc tính, phương thức riêng. Tính thừa kế giúp người lập trình dễ dàng sử dụng lại mã chương trình đã viết trước đó.

- Tính đa hình (Polymorphism): Một phương thức có thể thực hiện theo nhiều cách khác nhau trên các lớp khác nhau gọi là tính đa hình. Tính đa hình giúp cho việc lập trình trở nên đơn giản, dễ mở rộng.



**MÔ HÌNH CỦA LTHĐT**

## 2. Các khái niệm về LTHĐT

### 2.1 Đối tượng (object)

Đối tượng dùng để biểu diễn một thực thể của thế giới thực. Mỗi đối tượng được xác định bởi thuộc tính (dữ liệu, biến) và hành vi (phương thức, hàm). Thuộc tính để xác định tính chất riêng của đối tượng, hành vi là hành động tác động lên đối tượng.

Object = Variables + Methods
------------------------------

**Ví dụ :** Một đối tượng sinh viên có thể có thuộc tính là: họ tên, đlt, đtb và hành vi tác động lên đối tượng sinh viên là tính đtb của sv đó

* Đối tượng sinh viên thứ 1	* Đối tượng sinh viên thứ 2
- Thuộc tính: họ tên: Trần văn An đlt= 1 đth= 2 - Hành vi: tính đtb của sv: $dtb = (dlt + dth) / 2 = 1.5$	- Thuộc tính: họ tên: Đoàn Dự đlt= 2 đth= 3 - Hành vi: tính đtb của sv: $dtb = (dlt + dth) / 2 = 2.5$

**Ví dụ:** Một đối tượng hcn có thể có thuộc tính là chiều dài, chiều rộng và hành vi tác động lên đối tượng hcn là tính diện tích của hcn đó

Đối tượng hcn thứ 1	Đối tượng hcn thứ 2
- Thuộc tính: chiều dài=3 chiều rộng=4 - Hành vi: Tính dt: $dt = cd * cr = 12$	- Thuộc tính: chiều dài=5 chiều rộng=6 - Hành vi: Tính dt: $dt = cd * cr = 30$

### 2.2 Lớp (class)

Là cấu trúc mô tả các đối tượng có cùng thuộc tính và hành vi. Mỗi lớp sẽ khai báo các thuộc tính, hành vi của các đối tượng thuộc lớp. Các đối tượng thuộc lớp sẽ có cùng tên thuộc tính nhưng có giá trị thuộc tính khác nhau. Thuộc tính còn gọi là dữ liệu hay là biến, hành vi còn gọi là hàm hay phương thức.

## II. Ngôn ngữ lập trình Java

### 1. Giới thiệu:

NNLT Java do hãng Sun Microsystem thiết kế năm 1991 tên là Oak, mục đích lập trình cho các thiết bị điện tử, 1995 được mở rộng để viết ứng dụng trên Internet và lấy tên là Java.

Ưu điểm của ngôn ngữ Java là ngôn ngữ lập trình hướng đối tượng, không phụ thuộc phần cứng và hệ điều hành, hỗ trợ lập trình mạng rất mạnh và đơn giản và dễ học hơn C++ nhiều.

Java có thể soạn bằng bất cứ trình soạn thảo văn bản nào (notepad, wordpad, Jbuilder,...), sau đó được dịch sang file .class dạng bytecode. Mã bytecode không phụ thuộc phần cứng và hệ điều hành nhưng muốn thực thi trên một máy có hệ điều hành cụ thể thì máy đó cần cài đặt

JVM (Java Virtual Machine) tương ứng, JVM là môi trường để thực thi file dạng bytecode trên một máy cụ thể.

Để biên dịch, thực thi chương trình Java ta có thể cài đặt chương trình JDK (Java Development Toolkit) gồm có trình biên dịch (javac.exe) để dịch file .java thành file .class, trình thông dịch (java.exe) để thực thi file .class, thư viện chứa các hàm chuẩn (APIs) và một số tiện ích khác nhưng do JDK không có trình soạn thảo Java và không có giao tiếp đồ họa với người dùng nên khi viết Java thường ta sử dụng phần mềm thân thiện hơn như là Jbuilder, Visual J++,...

### Sơ đồ hoạt động của một ct ứng dụng Java:



## 2. Sử dụng ngôn ngữ

### \* Các kiểu cơ bản

Từ khóa	kích thước
Số nguyên có dấu	
byte	1 byte
short	2 bytes
int	4 bytes
Số thực	
float	8 bytes
double	8 byte
Các kiểu khác	
char	kiểu kí tự 2 byte
boolean	kiểu luận lý (true, false)

### \* Hằng số:

123 (int), 123L (long), 123.45F (float), 123.45 (double), 'c' (char)

### \* Kiểm soát việc truy xuất biến, phương thức của lớp:

Specifier	class	subclass	package	world
private	X			
protected	X	X	X	
public	X	X	X	X

public: có thể truy xuất ở gói khác

protected: chỉ truy xuất ở cùng gói (mặc định là protected)

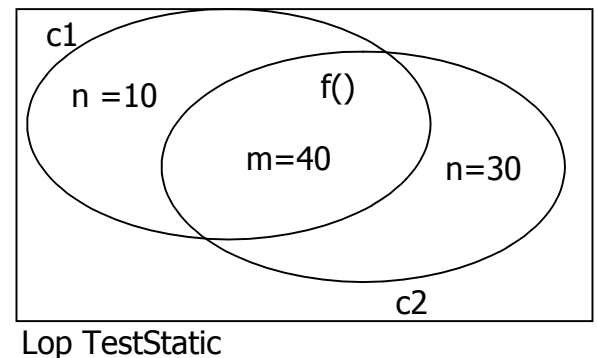
private: chỉ truy xuất trong lớp

### \* **Biến tĩnh, phương thức tĩnh**

- Biến tĩnh: khai báo static, dùng chung cho mọi đt thuộc lớp. Không có static gọi là biến thể hiện, mỗi đối tượng có biến thể hiện riêng.
- Phương thức tĩnh: khai báo static, được gọi bằng tên lớp hoặc tên đối tượng, trong pt tĩnh chỉ được truy xuất biến tĩnh.

#### **Ví dụ:**

```
public class TestStatic
{
    int n; //biến thể hiện
    static int m; //biến tĩnh
    void f()
    {
        int x=5; //biến cục bộ
    }
    public static void main(String[] args)
    {
        TestStatic c1=new TestStatic(); c1.n=10; c1.m=20;
        TestStatic c2=new TestStatic(); c2.n=30; c2.m=40;
        System.out.println("kq: "+c1.n+","+c1.m+","+c2.n+","+c2.m+","+TestStatic.m);
    }
}
```



#### **Kết quả:**

kq: 10,40,30,40,40

### \* **Lớp trừu tượng, pt trừu tượng**

Lớp trừu tượng và phương thức trừu tượng được khai báo abstract. Pt trừu tượng chỉ có phần khai báo chưa có cài đặt. Lớp trừu tượng có những pt trừu tượng và các pt khác. Không thể tạo đt từ pt trừu tượng, lớp thừa kế lớp trừu tượng cần phải cài đặt các pt trừu tượng nếu không sẽ trở thành lớp trừu tượng. Pt trừu tượng sẽ có tính đa hình.

#### **Ví dụ:**

```
public abstract class Nguoi //lớp trừu tượng
{
    String hoten;
    public void NhapHoTen() throws IOException
    {
        BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in));
        System.out.print("Ho Ten:"); hoten=stdIn.readLine();
    }
    public void XuatHoTen()
    {

```

```

        System.out.println("Ten: "+ten);
    }
    public abstract boolean Thuong(); //pt xet dieu kien duoc thuong la phương thức trừu tượng.
}

```

### **\* Giao diện:**

Dùng từ khoá interface. Trong giao diện chỉ có các biến hằng số (final) và các pt trong giao diện chỉ có phần khai báo, chưa có cài đặt. Một lớp có thể sử dụng nhiều giao diện, giao diện dùng để giải quyết vấn đề đa thừa kế. Một lớp có thể sử dụng (thừa kế) nhiều giao diện và phải cài đặt tất cả các pt trong các giao diện nếu không lớp phải khai báo trừu tượng.

### **Ví dụ:**

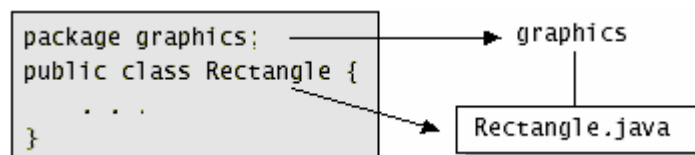
```

public interface SoSanh //giao diện
{
    public static final int LESS = -1;
    public static final int EQUAL = 0;
    public static final int GREATER = 1;
    public int sosanh(SoSanh obj);
}

```

**\* Gói:** là tập hợp các lớp và các giao diện có liên quan, mục đích của gói là bảo vệ sự truy cập và giải quyết vấn đề trùng tên. Khai báo lớp thuộc gói dùng lệnh package, sử dụng lớp của gói dùng lệnh import. Java cung cấp sẵn rất nhiều gói, khi sử dụng lớp thuộc gói nào thì cần import gói tương ứng, các lớp thuộc gói java.lang mặc định đã import.

### **Ví dụ:**



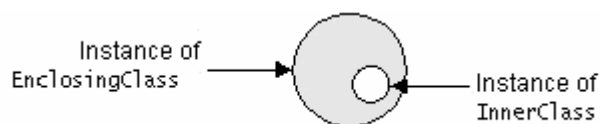
### **\* Lớp lồng nhau:**

Đối tượng thuộc lớp trong có thể truy xuất biến và phương thức không tĩnh của lớp ngoài.

```

class EnclosingClass{
    . . .
    class ANestedClass {
        . . .
    }
}

```



## **BÀI TẬP CHƯƠNG I:** **GIỚI THIỆU NGÔN NGỮ JAVA**

\* **Bài 1:** Viết chương trình giải pt bậc nhất, tham số đưa vào từ dòng lệnh, nếu thiếu tham số phải thông báo cách sử dụng, nếu tham số a, b không phải là số phải báo lỗi. Dùng trình biên dịch, thông dịch của JDK để biên dịch và thực thi chương trình.

\* **Bài 2 :** Xây dựng lớp hình chữ nhật, có thuộc tính là chiều dài, chiều rộng và có phương thức tính diện tích, chu vi hcn, phương thức khởi tạo; viết hàm main để thử lớp này. Soạn và thực thi chương trình bằng Jbuilder.

\* **Bài 3:** Xây dựng lớp sinh viên, có thuộc tính là họ tên, điểm lý thuyết, điểm thực hành, điểm trung bình= (đlt+đth)/2, và có phương thức nhập thông tin sv, tính đtb, xuất thông tin sv. Viết hàm main để thử lớp này.

\* **Bài 4:** Xây dựng lớp hình vuông thừa kế lớp hình chữ nhật. Viết ct tính dt, cv hình vuông.

\* **Bài 5:** Xây dựng lớp điểm có thuộc tính x1,y1 và pt hiện tọa độ điểm. Xây dựng lớp đường thẳng, thừa kế lớp điểm có pt tính chiều dài đoạn thẳng. Xây dựng lớp tam giác thừa kế lớp đoạn thẳng, có pt tính diện tích, chu vi tam giác.

\* **Bài 6:** Nhập một danh sách gồm giảng viên và sinh viên, in ra danh sách những người được thưởng. Biết rằng điều kiện được thưởng là giảng viên có số bài báo >3, sinh viên có điểm thi tốt nghiệp >8.

\* **Bài 7:** Thiết kế các lớp cho các danh mục cần quản lý trong thư viện bao gồm bài báo, sách, luận văn. Mỗi danh mục có nhan đề, tác giả, với mỗi loại danh mục ta có thêm các thông tin khác nhau:

- a. Bài báo phải có tên tạp chí đăng bài báo và số phát hành của tạp chí.
- b. Sách thì phải có nhà xuất bản.
- c. Luận văn phải có tên trường.

Viết chương trình cho phép nhập, xuất thông tin về các danh mục này.

\* **Bài 8:** Xây dựng lớp ma trận có phương thức cộng hai ma trận sao cho có thể sử dụng phương thức này cộng các ma trận có dữ liệu khác nhau như là ma trận số nguyên, ma trận phân số.

\* **Bài 9:** Xây dựng lớp sắp xếp có phương thức sắp xếp sao cho có thể sử dụng phương thức này sắp xếp các dãy khác nhau như là dãy hình tròn sắp theo bán kính, dãy hình hình vuông sắp theo cạnh (biết rằng lớp hình vuông đã được thiết kế là thừa kế lớp hình chữ nhật).

- Hết -



## Chương II

# SỬ DỤNG GIAO DIỆN VÀ TRUY XUẤT CSDL

### (GRAPHICAL USER INTERFACES – DATABASE)

## I. GRAPHICAL USER INTERFACES (GUI)

### 1. Giới thiệu:

- Để người dùng có thể giao tiếp với chương trình ở dạng trực quan (dạng đồ họa), Java cung cấp nhiều thành phần giao tiếp như là: button, text box, menu,...
- Các gói java.awt (JDK 1.0) (AWT: Abstract Window Toolkit ), javax.swing (JDK 2.0) chứa các lớp để tạo và quản lý các thành phần giao tiếp.
- Các thành phần trong gói swing bắt đầu bằng chữ J, và các thành phần này có nhiều chức năng hơn các thành phần trong gói awt và độc lập hoàn toàn với phần cứng.

### 2. Tạo khung cho cửa sổ ứng dụng

- Khi viết ứng dụng có dùng giao diện, ta cần tạo khung cửa sổ cho ứng dụng để chứa các thành phần giao tiếp.
- Trong gói swing có ba lớp dùng để tạo khung là: JFrame (khung chính), JDialog (khung phụ, phụ thuộc vào khung khác) và JApplet (khung chứa applet). Khi viết applet không cần tạo khung vì applet thừa kế Frame.
- Thường các thành phần giao diện được ghép thành nhóm trong panel, trong panel lại có thể chứa panel con và frame có thể chứa nhiều panel.

### 3. Sắp xếp các thành phần giao tiếp

- Muốn đặt các thành phần giao tiếp ở vị trí theo ý muốn mà không phụ thuộc vào độ phân giải của màn hình, ta dùng lớp quản lý bố cục (Layout Manager).
- Có 5 lớp quản lý bố cục là: FlowLayout, GridLayout, BorderLayout, CardLayout, GridBagLayout
  - + FlowLayout: bố trí các thành phần từ trái sang phải, hết dòng sẽ xuống dòng.
  - + GridLayout: bố trí các thành phần theo dạng bảng, theo thứ tự từ trái sang phải, từ trên xuống dưới.
  - + BorderLayout: chia khung cửa sổ thành 5 phần: East, West, South, North, Center. North, South kéo dài theo phương ngang, West, East kéo dài theo phương dọc, Center kéo dài theo hai phương.
- Khi cần bố trí phức tạp, ta dùng các panel, mỗi panel có thể dùng trình quản lý bố cục riêng để bố trí những thành phần trong panel. Đối tượng Jpanel mặc định sử dụng FlowLayout, containers trong đối tượng JApplet, JDialog, JFrame mặc định dùng BorderLayout, nhưng có thể dùng phương thức setLayout() để thay đổi lớp quản lý bố cục.

### 4. Xử lý sự kiện (Event)

- Khi người dùng tác động lên thành phần giao tiếp (ví dụ nhấn chuột, nhấn phím,...) sẽ sinh ra một sự kiện, Java sẽ gọi sự kiện này cho đối tượng lắng nghe sự kiện, đối tượng lắng nghe sự kiện sẽ gọi một phương thức đặc biệt để xử lý sự kiện.
- Mỗi thành phần giao tiếp (đối tượng nguồn) cần phải đăng ký các đối tượng lắng nghe sự kiện (đối tượng đích) và trong mỗi đối tượng đích cần cài đặt phương thức xử lý sự kiện (đối tượng nguồn và đối tượng đích có khi chỉ là một)

<b>HÀNH ĐỘNG</b>	<b>ĐỐI TƯỢNG NGUỒN</b>	<b>SỰ KIỆN ĐƯỢC TẠO RA</b>
Nhấn nút button	JButton	ActionEvent
Thay đổi văn bản của textfield	JTextComponent	TextEvent
Nhấn enter trên textfield	JTextFeild	ActionEvent
Chọn mục của combobox	JComboBox	ItemEvent, ActionEvent
Chọn các mục của list	JList	ListSelectionEvent
Chọn mục của menu	JMenuItem	ActionEvent
Đời focus đến hoặc đi khỏi thành phần	Component	FocusEvent
Nhấn hoặc thả phím trên thành phần	Component	KeyEvent
Di chuyển chuột trên thành phần	Component	MouseEvent
Mở, đóng, phóng to, thu nhỏ cửa sổ	Window	WindowEvent

Đối tượng lắng nghe sự kiện (đối tượng đích) cần dùng giao diện phù hợp để cung cấp phương thức xử lý sự kiện tương ứng. Thông thường sự kiện tên là XEvent thì đối tượng đích phải dùng giao diện XListener và phương thức đăng ký là addXListener.

**Ví dụ:** Khi nhấn nút button (đối tượng nguồn) sẽ sinh ra sự kiện ActionEvent, sự kiện ActionEvent được gửi cho đối tượng lắng nghe sự kiện (đối tượng đích). Đối tượng nguồn dùng phương thức addActionEvent để đăng ký đối tượng đích, đối tượng đích dùng giao diện ActionListener cung cấp phương thức actionPerformed(ActionEvent e) để xử lý sự kiện ActionEvent.

<b>Sự kiện</b>	<b>Dùng giao diện</b>	<b>Phương thức xử lý sự kiện</b>
ActionEvent	ActionListener	actionPerformed(ActionEvent e)
TextEvent	TextListener	textValueChanged(TextEvent e)
ItemEvent	ItemListener	itemStateChanged(ItemEvent e)
ListSelectionEvent	ItemListener	itemStateChanged(ItemEvent e)
FocusEvent	FocusListener	focusGained(FocusEvent e) focusLost(FocusEvent e)
KeyEvent	KeyListener	keyPressed(KeyEvent e) keyReleased(KeyEvent e) keyTyped(KeyEvent e)
MouseEvent	MouseListener  MouseMotionListener	mousePressed(MouseEvent e) mouseReleased(MouseEvent e) mouseClicked(MouseEvent e) mouseMoved(MouseEvent e) mouseDragged(MouseEvent e)
WindowEvent	WindowListener	windowClosed(WindowEvent e) windowOpened(WindowEvent e)

**\* Cài đặt đối tượng lắng nghe sự kiện**

**Ví dụ:** xử lý sự kiện nhấn nút button

**- xây dựng lớp dùng để tạo đối tượng lắng nghe sự kiện**

```
public class MyClass implements ActionListener{  
    public void actionPerformed(ActionEvent e) {  
        //các lệnh xử lý sự kiện  
    }  
}
```

**- Đăng ký đối tượng lắng nghe sự kiện sinh ra từ thành phần giao tiếp**  
button.addActionListener(instanceOfMyClass);

- Nếu phương thức xử lý sự kiện đơn giản ta có thể khai báo là lớp vô danh bên trong ( anonymous inner class), không cần xây dựng lớp lắng nghe sự kiện

**Ví dụ:**

```
button.addActionListener(new ActionListener()  
{  
    public void actionPerformed(ActionEvent e)  
    {  
        ...  
    }  
});
```

**Ghi chú:** mã xử lý sự kiện được thực thi trong một tiểu trình riêng

## 5. Chương trình mẫu GUI

### **Dạng 1: không xử lý sự kiện**

```
import javax.swing.*;
public class SwingApplication
{
    public SwingApplication() //phương thức khởi tạo có nhiệm vụ tạo GUI
    {
        JFrame frame = new JFrame("Tiêu đề của sổ");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().add(<thành phần giao tiếp>);
        frame.setSize(200,100);//hoặc frame.pack(); vừa đủ chứa các thành phần
        giao tiếp
        frame.setVisible(true); //Hiện cửa sổ
    }

    public static void main(String[] args)
    {
        new SwingApplication(); //tạo GUI
    }
}
/*HIDE_ON_CLOSE:che, DO_NOTHING_ON_CLOSE:không làm gì
EXIT_ON_CLOSE:đóng ứng dụng,DISPOSE_ON_CLOSE:hủy,đóng cửa sổ
*/
```

### **Dạng 2: có xử lý sự kiện**

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

public class SwingApplication implements ActionListener
{
```

```

JFrame Frm;

public SwingApplication() //phương thức khai tạo
{
    Frm = new JFrame("Tieu de cua so");
        Frm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        //Frm.setSize(new Dimension (120, 40));

    ...
    Frm.pack();
        Frm.setVisible(true);
}

public void actionPerformed(ActionEvent event) //phương thức xử lý ActionEvent
{
    //cac lenh xu ly su kien action
}

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

```

## II. DATABASE

### 1. Truy xuất CSDL: gồm các bước sau

STT	Thao tác	Lệnh (ví dụ)
1	Nạp driver truy xuất CSDL	Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
2	Kết nối CSDL	Connection con = DriverManager.getConnection("jdbc:odbc:student","sa","");
3	Tạo đối tượng câu lệnh	Statement stmt=con.createStatement();
4	Thực thi câu sql	ResultSet rs=stmt.executeQuery(sql);//select int n= stmt. executeUpdate (sql);// create, alter,drop, update, delete, insert
5	Lấy thông tin bảng kết quả rs	ResultSetMetaData rsmd=rs.getMetaData();
6	Lấy số cột trong bảng	int col=rsmd.getColumnCount();
7	Lấy tên cột thứ i (i=1,...)	String label=rsmd.getColumnLabel(i)
8	Lấy giá trị ở cột i hàng hiện tại	String value=rs.getString(i)//rs.getFloat(i),.. Tham số có thể là tên cột
9	Dời con trỏ mẫu tin tới hàng kế	rs.next(): false là cuối bảng, con trỏ bắt đầu trên dòng 1, lệnh next() đầu tiên sẽ dời con trỏ tới dòng 1

**Ví dụ:** in bangdiem

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection con = DriverManager.getConnection("jdbc:odbc:qlid", "sa", "");
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT masv,diemthi FROM bangdiem");
while (rs.next()) {
    String masv = srs.getString("masv");//srs.getString(1);
    int diemthi = srs.getFloat("diemthi");//srs.getFloat(2);
    System.out.println(masv + "    " + diemthi);
}
```

**\* Cách truy xuất CSDL nhanh hơn**

Nếu muốn thực thi câu lệnh sql nhiều lần với các giá trị tham số khác nhau nên dùng đối tượng PreparedStatement thay cho đối tượng Statement thì việc thực hiện sẽ nhanh hơn nhiều. Với đối tượng Statement mỗi khi gọi stmt.executeQuery(sql); hoặc stmt.executeUpdate(sql); thì câu lệnh sql được gửi lên cho hệ quản trị CSDL, hệ quản trị sẽ biên dịch và thực thi câu sql, nếu việc biên dịch thực hiện nhiều lần sẽ làm chậm việc thực thi. Với đối tượng PreparedStatement thì việc biên dịch câu lệnh sql chỉ thực hiện một lần do đó sẽ nhanh hơn nhiều.

**Ví dụ:**

```
Statement stmt=con.createStatement();
for(int i = 0; i < 10; i++)
    stmt. executeUpdate ( "insert into bangdiem values( `dh01`"+i+"", "+i+"") );
```

Có thể thay bằng đoạn mã sau nhanh hơn

```
PreparedStatement ps = con.prepareStatement( "insert into bangdiem values( ?, ?)");
for(int i = 0; i < 10; i++)
{
    ps.setString(1, "dh01"+i);
    ps.setInt(2, i);
    ps.executeUpdate();
}
```

## 2. Sử dụng Stored Procedure của SQL server

### - Tạo Stored Procedure:

Stored Procedure có thể tạo sẵn trong SQL server hoặc dùng đoạn mã Java sau:

```
String CreateProc = "create procedure ProcName (d/s ts nếu có) as ..."
```

```
Statement stmt = con.createStatement();
```

```
stmt.executeUpdate(CreateProc);
```

Khi thực thi, đoạn mã này sẽ tạo một Stored Procedure lưu vào CSDL.

### - Gọi Stored Procedure:

#### **//tạo đt chứa lời gọi SP**

```
CallableStatement cs = con.prepareCall("{call ProcName (?,?,...)}");
```

#### **//gửi tham số cho SP**

```
cs.setString(int parameterIndex, String x);// parameterIndex=1,2... là số thứ tự của tham số
```

hoặc

```
cs.setInt(int parameterIndex, int x); cs.setFloat(int parameterIndex, float x); ...
```

#### **//đăng kí tham số là tham số trả về**

```
cs.registerOutParameter(int parameterIndex, int sqlType)
```

sqlType có thể là các hằng sau: java.sql.Types.INTEGER, java.sql.Types.FLOAT, java.sql.Types.VARCHAR

#### **//thực thi SP**

```
ResultSet rs = cs.executeQuery(); //SP la select, rs là bảng kết quả
```

hoặc

```
int rec = cs.executeUpdate(); //SP la insert, delete, update,...rec là số mẫu tin được xử lý
```

hoặc

```
boolean flag = cs.execute(); //SP phối hợp nhiều select, insert,...flag=true là thành công
```

#### **//lấy kết quả do SP trả về**

```
ResultSet rs=cs.getResultSet();//lấy kq ResultSet
```

```
int rec=cs.getUpdateCount();//lấy kq là số mẫu tin được xử lý
```

```
boolean flag= cs.getMoreResult();// (cs.getMoreResults() == false) &&  
(cs.getUpdateCount()==-1): là hết kết quả.
```

### **3. Các chức năng khác**

#### **3.1 Di chuyển cursor**

##### **\* Tạo đối tượng ResultSet có thể cuộn**

Statement stmt = con.createStatement(int resultSetType, int resultSetConcurrency);

resultSetType có thể là: ResultSet.TYPE\_FORWARD\_ONLY, ResultSet.TYPE\_SCROLL\_INSENSITIVE, hoặc ResultSet.TYPE\_SCROLL\_SENSITIVE (không hay có phản ánh những thay đổi khi ResultSet đang mở)

resultSetConcurrency có thể là: ResultSet.CONCUR\_READ\_ONLY hay ResultSet.CONCUR\_UPDATABLE (ResultSet chỉ đọc hay có thể cập nhật)

Mặc định resultSetType=ResultSet.TYPE\_FORWARD\_ONLY, resultSetConcurrency = ResultSet.CONCUR\_READ\_ONLY

ResultSet srs = stmt.executeQuery("SELECT COF\_NAME, PRICE FROM COFFEES");

**Ví dụ:** in bangdiem theo thứ tự ngược

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

```
Connection con = DriverManager.getConnection("jdbc:odbc:qlid", "sa", "");
```

```
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,  
                                     ResultSet.CONCUR_READ_ONLY);
```

```
ResultSet rs = stmt.executeQuery("SELECT masv,diemthi FROM bangdiem");
```

```
rs.afterLast();
```

```
while (rs.previous()) {
```

```
    String masv = srs.getString("masv");
```

```
    int diemthi = srs.getFloat("diemthi");
```

```
    System.out.println(masv + "    " + diemthi);
```

```
}
```

##### **\* Di chuyển cursor**

```
boolean rs.first();
```

```
boolean rs.last();
```

```
void rs.beforeFirst();
```

```
void rs.afterLast();
```

int absolute(int) :âm di chuyển ngược tính từ cuối: -1:cuối, -2 kể cuối,...

boolean rs.relative(int): di chuyển tính từ hàng hiện hành.

int getRow(): trả về vị trí cursor

vv...



**Ví dụ:**

```
rs.absolute(4); //tới hàng 4
int rowNum = rs.getRow(); // rowNum = 4
rs.relative(-3);
int rowNum = rs.getRow(); // rowNum = 1
```

```
rs.relative(2);
int rowNum = rs.getRow(); // rowNum = 3
```

isFirst, isLast, isBeforeFirst, isAfterLast: kiểm tra vị trí cursor

ví dụ:

```
if (rs.isAfterLast() == false) {
    rs.afterLast();
}
while (rs.previous()) {
    String masv = rs.getString("masv");
    int diemthi = rs.getFloat("diemthi");
    System.out.println(masv + "    " + diemthi);
}
```

**3.2 Sử dụng ResultSet cập nhật CSDL****Ví dụ:**

```
Connection con = DriverManager.getConnection("jdbc:odbc:qlđ", "sa", "");
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                                     ResultSet.CONCUR_UPDATABLE);
ResultSet rs = stmt.executeQuery("SELECT masv,diemthi FROM bangdiem");
rs.last();
rs.updateInt("diemthi", 9); //cập nhật trong ResultSet
rs.cancelRowUpdates();      //bỏ cập nhật trong ResultSet
rs.updateFloat("diemthi", 8);
rs.updateRow(); //cập nhật trong CSDL
```

```
rs.moveToInsertRow(); //InsertRow là buffer chứa hàng cần thêm
rs.updateString("masv", "dh0101");
rs.updateInt("diemthi", 5);
rs.insertRow(); //thêm vào rs và CSDL
rs.updateString("masv", "dh0102");
rs.updateInt("diemthi", 7);
rs.insertRow(); //thêm vào rs và CSDL
rs.moveToInsertRow();//di chuyển tới hàng mới thêm
rs.deleteRow(); //xóa hàng mới thêm khỏi rs và CSDL
rs.first();
rs.refreshRow();//làm mới hàng hiện tại
```

**Chú ý:** mặc dù rs là SENSITIVE nhưng có khi rs vẫn chưa cập nhật record mới do phụ thuộc vào Driver và DBMS nên để chắc chắn có rs mới, thực hiện lại lệnh : rs.close();

```
rs = stmt.executeQuery("SELECT masv,diemthi FROM bangdiem");
```

## **BÀI TẬP CHƯƠNG II:** **GUI – DATABASE**

### **\* Bài 1:**

Viết chương trình tính diện tích, chu vi hcn, hv có giao diện như sau:

TÍNH DT,CV HCN, HV	
<input type="button" value="HÌNH CHU NHAT"/>	<input type="button" value="HÌNH VUONG"/>

Khi click nút "HÌNH CHU NHAT" sẽ hiện:

CHIEU DAI	<input type="text"/>
CHIEU RONG	<input type="text"/>
DIEN TICH	<input type="text"/>
CHU VI	<input type="text"/>
<input type="button" value="OK"/>	<input type="button" value="CLEAR"/>

Khi click nút "HÌNH VUONG" sẽ hiện:

CANH	<input type="text"/>
DIEN TICH	<input type="text"/>
CHU VI	<input type="text"/>
<input type="button" value="OK"/>	<input type="button" value="CLEAR"/>

Nhập cd, cr, click OK, hiện dt, cv. Click CLEAR xóa dữ liệu

### **\* Bài 2:**

Viết chương trình xử lý file văn bản có các chức năng sau: tạo file, xem file, sao chép file. Chương trình viết bằng hai cách: không có giao diện và có giao diện.

### **\* Bài 3:**

Viết chương trình quản lý sinh viên có các chức năng sau: nhập mã sv, họ tên, đlt, đth, tính đtb và ghi thông tin này vào file truy xuất ngẫu nhiên; tìm/xóa/sửa thông tin sinh viên khi biết masv; xem danh sách sinh viên. Chương trình viết bằng hai cách: không có giao diện và có giao diện

### **\* Bài 4:**

Tương tự bài 3 nhưng thông tin lưu trong CSDL SQL server.

- Hết -

## CHƯƠNG III

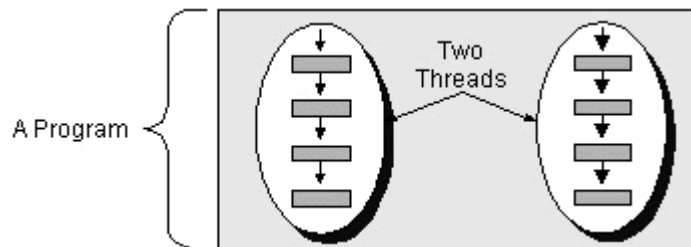
### MULTI THREADS & APPLET & URL

#### I. LẬP TRÌNH MULTI THREADS

##### 1. Khái niệm tiểu trình (thread):

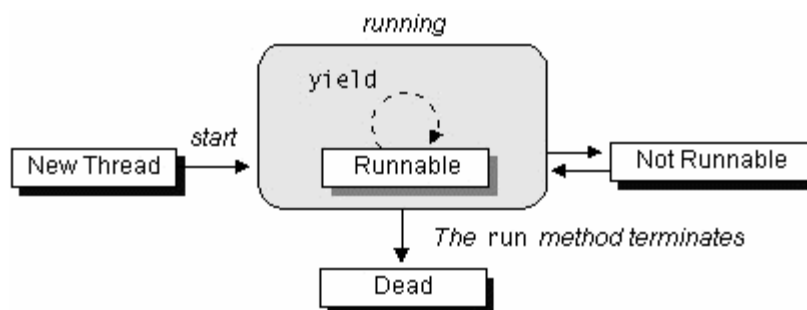
- Tiểu trình là một dòng điều khiển tuần tự bên trong một tiến trình.
- Một tiến trình có thể tạo nhiều tiểu trình, mỗi tiểu trình thực hiện một công việc nào đó và thực thi độc lập, đồng thời bằng cách chia sẻ CPU.
- Các tiểu trình trong cùng một tiến trình dùng chung không gian địa chỉ tiến trình nhưng có con trỏ lệnh, tập các thanh ghi và ngăn xếp riêng.
- Các tiến trình chỉ có thể liên lạc với nhau thông qua các cơ chế do hệ điều hành cung cấp. Các tiểu trình liên lạc với nhau dễ dàng thông qua các biến toàn cục của tiến trình.

**Ví dụ:** Trình duyệt là một chương trình có nhiều tiểu trình: tiểu trình tải hình, download file, tiểu trình thực hiện hoạt hình, ...



\* Khi nào cần viết chương trình có sử dụng tiểu trình?

##### 2. Chu kỳ sống của tiểu trình:



Các trạng thái mà tiểu trình có thể trải qua trong quá trình sống là:

Khi thread được tạo bởi lệnh new, thread chưa được cấp phát tài nguyên và ở trạng thái "New Thread". Pt start() sẽ cấp tài nguyên cho thread, sắp lịch thực thi cho thread và gọi pt run(), lúc này thread ở trạng thái "Runnable" (đợi CPU). Khi CPU rỗi hoặc đến lượt thread, JVM sẽ chuyển thread sang trạng thái "running". Thread sẽ chuyển sang trạng thái "Not Runnable" khi gọi pt sleep() hoặc wait() hoặc thread đang chờ nhập/xuất. Khi pt run() thực thi xong, thread sẽ chuyển sang trạng thái "Dead" và kết thúc, có thể dùng pt isAlive() để kiểm tra thread còn sống hay chết.

### **3. Độ ưu tiên của tiểu trình**

Do có một CPU, nên tại một thời điểm chỉ có thể có một thread thực thi. Java dùng giải thuật "độ ưu tiên cố định" để điều phối hoạt động của các thread. Thread được chọn sẽ thực hiện cho tới khi một trong các trường hợp sau xảy ra:

- Thread có độ ưu tiên cao hơn ở trạng thái Runnable
- Thread đang thực thi nhường quyền thực thi hay pt run() của nó đã kết thúc
- Hết thời lượng quantum dành cho nó.

Vì để tránh trường hợp thread đói CPU, bộ lập lịch của Java không đảm bảo thread có độ ưu tiên cao nhất đang thực thi, nên khi lập trình không nên dựa vào độ ưu tiên. Ngoài ra cũng không nên dựa vào quantum vì có thể hệ thống thread đang thực thi không hỗ trợ quantum.

#### **\* Cú pháp lệnh:**

set Priority (n): dùng để thay đổi độ ưu tiên, n là độ ưu tiên trong khoảng (MIN\_PRIORITY, MAX\_PRIORITY), MIN\_PRIORITY, MAX\_PRIORITY là hằng số định nghĩa trong lớp Thread, n lớn là độ ưu tiên cao. Khi một thread được tạo ra sẽ có cùng độ ưu tiên với thread tạo ra nó

### **4. Viết chương trình sử dụng tiểu trình**

#### **4.1 Sử dụng lớp Timer và TimerTask**

Java cung cấp các lớp sau: java.util.Timer, java.util.TimerTask, javax.swing.Timer, javax.swing.SwingWorker để viết ứng dụng có nhiều công việc thực hiện đồng thời và mỗi công việc được thực hiện lặp lại sau một khoảng thời gian nào đó.

#### **\* Cú pháp lệnh**

- **public void schedule(TimerTask task, long delay):**  
task sẽ được thực hiện sau khoảng thời gian delay (milliseconds)
- **public void schedule(TimerTask task, Date time)**  
task sẽ được thực hiện vào thời điểm xác định

**Ví dụ:** thực hiện task lúc 11:01 p.m

```
Calendar calendar = Calendar.getInstance();
calendar.set(Calendar.HOUR_OF_DAY, 23);
calendar.set(Calendar.MINUTE, 1);
calendar.set(Calendar.SECOND, 0);
Date time = calendar.getTime();
timer = new Timer();
timer.schedule(new Task(), time);
```

- **public void schedule(TimerTask task, long delay, long period)**  
**public void schedule(TimerTask task, Date time, long period)**

Lặp lại task sau khoảng thời gian period (milliseconds)

- **public void cancel()**

Kết thúc timer, loại bỏ các công việc do timer sắp đặt thực hiện. Nếu timer chưa kết thúc, chương trình vẫn thực thi, nếu muốn kết thúc ct có thể sử dụng lệnh System.exit(0)

#### **4.2 Sử dụng lớp Thread**

Java cung cấp lớp java.lang.Thread để viết tiểu trình dạng tổng quát hơn. Pt start() sẽ gọi pt run() thực thi công việc của thread.

#### **4.3 Sử dụng giao diện Runnable**

Nếu lớp đã thừa kế một lớp khác, vì do java không cho phép đa thừa kế, ta cần dùng giao diện Runnable

### **5. Đồng bộ các tiểu trình**

Thông thường các tiểu trình thực thi độc lập, không cần quan tâm tới hoạt động của tiểu trình khác nhưng nếu các tiểu trình dùng chung tài nguyên, hoặc cần phải phối hợp để hoàn thành công việc thì ta cần phải đồng bộ việc thực thi của các tiểu trình.

Đoạn mã truy xuất tài nguyên dùng chung mà có khả năng xảy ra lỗi gọi là miền găng(Critical Section). Có thể giải quyết lỗi nếu bảo đảm tại một thời điểm chỉ có một tiểu trình truy xuất tài nguyên dùng chung. Trong Java miền găng là một phương thức được

khai báo bằng từ khoá synchronized, một đối tượng có miền găng sẽ là độc quyền truy xuất (tại một thời điểm chỉ có một tiểu trình truy xuất miền găng của đối tượng)

## II. VIẾT ỨNG DỤNG APPLET

### 1. Giới thiệu

Applet là một ứng dụng java, thừa kế lớp java.applet.Applet hoặc lớp javax.swing.JApplet. Trong Applet không có hàm main nên không thể tự thực thi, muốn thực thi Applet cần “nhúng” Applet vào trang web và trình duyệt web sẽ thực thi Applet.

Applet có thể để ở máy Web Server, client dùng browser yêu cầu Web Server gửi cho client trang web có nhúng Applet, trang web cùng với Applet được download về máy client, trình duyệt ở máy client chịu trách nhiệm thông dịch trang web và thực thi Applet.

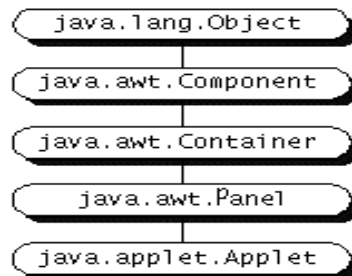
Lập trình Applet chính là một dạng lập trình mạng dạng đơn giản: viết applet để ở máy server, client dùng browser lấy applet về máy client và cho thực thi. Applet thường chậm vì phải download về máy client, ngoài ra Applet còn tiềm ẩn sự không an toàn do Applet thực thi trên máy client.

#### \* Tính an toàn

Vì lý do an toàn nên Applet bị hạn chế sau:

- Không thể đọc/ghi file hoặc thực thi ct trên máy mà Applet đang thực thi
- Không thể kết nối với máy khác ngoại trừ máy cung cấp Applet.
- Không thể thực thi ct trên

#### \* Sơ đồ thừa kế của lớp Applet:



Tất cả các lớp đều thừa kế lớp java.lang.Object và lớp này tự động import vào tất cả các lớp khác.

### 2. Các pt đặc biệt:

```
public void init() { . . . }  
public void start() { . . . }  
public void stop() { . . . }  
public void destroy() { . . . }  
public void paint(Graphics g) {...}
```

- Khi Applet được download về máy client thì pt init() và start(), paint() được thực thi. Pt init(), start() thường có các lệnh thực hiện các công việc khởi tạo ban đầu như kết nối CSDL, mở socket,...Pt paint() vẽ Applet.
- Khi user chuyển qua trang khác thì pt stop() được gọi và khi user trở về trang chứa Applet thì pt start(), paint() lại được gọi. Pt stop() thường có các lệnh để ngừng thực hiện Applet.
- Khi Applet kết thúc việc thực hiện hoặc khi đóng trình duyệt thì pt destroy() được gọi. Pt destroy() thường có các lệnh thực hiện các công việc dọn dẹp như đóng kết nối CSDL, đóng socket,...

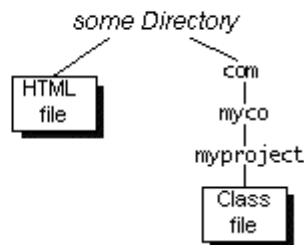
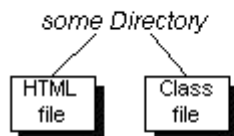
### 3. Thực thi Applet

- **Tạo trang Web**
- **Nhúng Applet vào trang web**

```
<APPLET CODE=AppletSubclass.class WIDTH=anInt HEIGHT=anInt>
</APPLET>
```

No Package Statements in Applet Code

```
package com.myco.myproject;
```



- **Dùng trình duyệt, xem trang web có nhúng applet**

### 4. Sử dụng thẻ <APPLET>

- **Trình duyệt gửi tham số cho Applet:**

```
<APPLET CODE=AppletSubclass.class WIDTH=anInt HEIGHT=anInt>
<PARAM NAME=parameter1Name VALUE=aValue>
<PARAM NAME=parameter2Name VALUE=anotherValue>
</APPLET>
```

- **Applet lấy tham số do trình duyệt gửi**

```
public String getParameter(String name)
```



## - Xác định thư mục chứa Applet

Mặc định browser tìm Applet trong thư mục chứa file html, nếu Applet trong gói, browser dùng tên gói để xây dựng đường dẫn bên dưới tm chứa file html. Nếu applet để ở nơi khác, dùng thuộc tính CODEBASE để xác định vị trí applet.

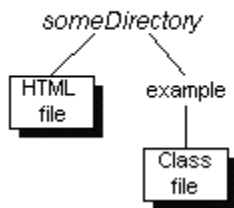
```
<APPLET CODE=AppletSubclass.class CODEBASE=aURL  
    WIDTH=anInt HEIGHT=anInt>  
</APPLET>
```

Nếu aURL là một URL tương đối thì xét từ vị trí của file html.

Ví dụ:

```
<APPLET CODE=Simple.class CODEBASE="example/"  
    WIDTH=500 HEIGHT=20>
```

CODEBASE="example/"



CODEBASE="http://someServer/...someOtherDirectory/"



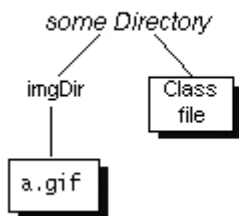
</APPLET>

## 5. Một số phương thức thông dụng

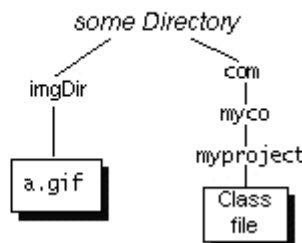
### - Xác định thư mục chứa file dữ liệu

Ví dụ:

No Package Statements in Applet Code



package com.myco.myproject;



```
Image image = getImage(getCodeBase(), "imgDir/a.gif");
```

- **Hiện một thông báo ở thanh trạng thái (status bar)**

**Ví dụ:**

```
showStatus("MyApplet: Loading image file " + file);
```

- Hiện tài liệu trong browser

```
public void showDocument(java.net.URL url)
```

```
public void showDocument(java.net.URL url, String targetWindow)
```

targetWindow có thể có các giá trị sau:

"\_blank" : hiện tài liệu trong một cửa sổ mới, không có tên

"<windowName>" : hiện tài liệu trong cửa sổ tên là <windowName>

"\_self" : hiện tài liệu trong cửa sổ và trong cùng khung chứa applet

"\_parent" : hiện tài liệu trong cửa sổ chứa applet nhưng trong khung cha của khung chứa applet . Nếu không có khung cha thì hiện giống "\_self"

"\_top" : hiện tài liệu trong cửa sổ chứa applet nhưng trong khung mức cao nhất

Ví dụ:

```
AppletContext appletContext=getAppletContext();
```

```
appletContext.showDocument(url);
```

- **Gởi thông điệp tới applet khác**

Vì lý do an toàn nên một số browser yêu cầu các applet phải cùng một tm trên cùng một server và nhúng trong cùng trang. Mỗi applet phải được đặt tên dùng thuộc tính NAME:

```
<APPLET CODEBASE=example/ CODE=Sender.class
```

```
WIDTH=450
```

```
HEIGHT=200
```

```
NAME="buddy">
```

```
</applet>
```

hoặc gởi tên như là tham số:

```
<APPLET CODEBASE=example/ CODE=Receiver.class
```

```
WIDTH=450
```

```
HEIGHT=35>  
<PARAM NAME="name" value="old pal">  
</applet>
```

### + Tìm 1 applet

```
String receiverName = ...;  
Applet receiver = getAppletContext().getApplet(receiverName);  
((Receiver)receiver).processRequestFrom(myName); //gọi pt của receiver
```

### + Tìm tất cả applet

```
Enumeration e = getAppletContext().getApplets();
```

### - Thực thi file âm thanh \*.au

getAudioClip(URL), getAudioClip(URL, String) : Trả về đối tượng sử dụng giao diện AudioClip, giao diện AudioClip có các pt sau: loop, play, stop.

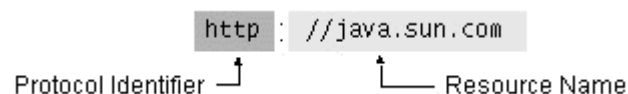
play(URL), play(URL, String) : thực thi AudioClip xác định bởi URL

## III. SỬ DỤNG URL(Uniform Resource Locator)

### 1. Khái niệm:

- URL Là một tham chiếu đến một tài nguyên trên mạng (địa chỉ tài nguyên trên mạng), Web Browser hoặc các ứng dụng khác dùng URL để tìm tài nguyên trên mạng. Gói java.net có lớp URL dùng để biểu diễn địa chỉ URL
- URL có dạng chuỗi gồm: giao thức dùng để truy xuất tài nguyên và tên tài nguyên.

Ví dụ:

  
Protocol Identifier      Resource Name

Có thể dùng các giao thức khác như: File Transfer Protocol (FTP), Gopher, File, News...

- Tên tài nguyên là thông tin đầy đủ để xác định tài nguyên, thông tin này phụ thuộc vào giao thức sử dụng, nhưng thường có các thông tin sau:

<b>Host Name</b>	Tên máy chứa tài nguyên.
<b>Filename</b>	Đường dẫn tới tài nguyên
<b>Port Number</b>	Số hiệu cổng dùng khi kết nối

Đối với http, filename mặc định là index.html

## **2. Tạo và sử dụng đối tượng URL**

<b>Phương thức</b>	<b>Ý nghĩa</b>
URL(String spec)	Tạo đối tượng URL từ một chuỗi URL
URL(String protocol, String host, int port, String file)	Tạo đối tượng URL từ protocol, host, port và tên file
URL(String protocol, String host, String file)	Tạo đối tượng URL từ protocol, host, port và tên file, dùng port mặc định
Object getContent()	Lấy nội dung của URL
String getFile()	Lấy tên file cùng đường dẫn của URL
String getHost()	Lấy tên máy của URL
int getPort()	Lấy số hiệu cổng của URL
String getProtocol()	Lấy tên protocol của URL
InputStream openStream()	Kết nối tới url và mở luồng nhập để đọc thông tin từ url
URLConnection.openConnection()	Mở kết nối tới URL

- Hết -

### **BÀI TẬP CHƯƠNG III**

#### **LẬP TRÌNH MULTITHREAD , APPLET , URL**

##### **\* Bài 1:**

Viết chương trình thực hiện n bài toán cộng ngẫu nhiên. Khi gần hết giờ sẽ kêu ba tiếng, mỗi tiếng cách nhau 2 giây, sau đó thông báo hết giờ, và đóng chương trình. Thời gian làm bài qui định là n giây (không tính thời gian thông báo hết giờ).

##### **\* Bài 2:**

- a) Viết chương trình gồm hai tiểu trình thực thi đồng thời, một tiểu trình in các kí tự từ 'a' đến 'z', một tiểu trình in số từ 1 đến 26.
- b) Viết lại chương trình trên sao cho hai tiểu trình in luân phiên, kết quả có dạng sau: 1 a 2 b 3 c 4 d ...

##### **\* Bài 3:**

Viết chương trình mô phỏng bài toán "nhà sản xuất - người tiêu thụ". Các Nsx sản xuất ra các số nguyên, đặt vào kho, các Ntt lấy các số này ra sử dụng. Yêu cầu đặt ra:

- Nsx phải tạm ngừng sx khi kho đầy
- Ntt phải tạm ngừng tt khi kho rỗng

Khi Nsx và Ntt không được truy xuất kho đồng thời.

##### **\* Bài 4:**

Viết Applet giải phương trình bậc hai, tham số a,b,c gởi cho Applet bằng hai cách: dùng thẻ PARAM hoặc cung cấp giao diện nhập.

##### **\* Bài 5:**

Viết Applet hoạt hình "người đánh trống", khi qua trang khác "người đánh trống" phải ngừng đánh, khi trở lại trang có applet thì "người đánh trống" lại đánh tiếp.

##### **\* Bài 6:**

Viết Applet đồng hồ đa năng có các chức năng sau: hiện đồng hồ điện tử, hiện đồng hồ số có kim quay, hiện quả lắc đang lắc, hiện dòng chữ quảng cáo đang chạy.

##### **\* Bài 7:**

Viết chương trình xem file văn bản từ xa thông qua web server

##### **\* Bài 8:**

Viết chương trình download file thông qua web server

- Hết -

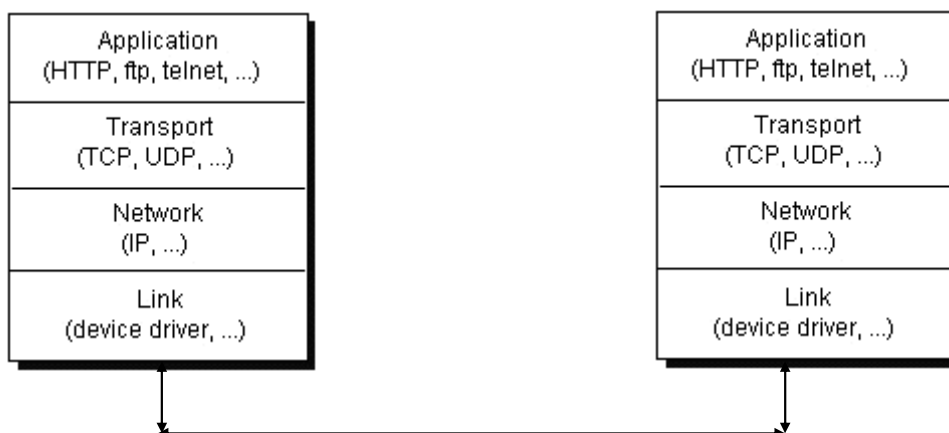
## CHƯƠNG IV

# LẬP TRÌNH CLIENT\SERVER

### I. Tổng quan về lập trình mạng:

#### 1. Giới thiệu

- Máy tính trên Internet liên lạc với nhau dùng giao thức TCP (Transmission Control Protocol) hoặc UDP (User Datagram Protocol).
- Khi viết chương trình Java liên lạc trên mạng là viết ở lớp ứng dụng (lớp application) và sử dụng những lớp trong gói java.net để truy xuất lớp TCP/UDP (lớp transport).



- **Lập trình client/server:** Là viết ứng dụng trên mạng gồm hai chương trình: chương trình client và chương trình server. Chương trình client gửi yêu cầu tới chương trình server, ct server xử lý yêu cầu và trả kết quả về cho ct client. Ct server có thể phục vụ đồng thời nhiều yêu cầu của các ct client.
- **Lập trình WEB:** là trường hợp đặc biệt của lập trình client/server. Ct client là ct Browser (trình duyệt web), ct server là Web Server nhận yêu cầu trang web từ Browser, Web Server tìm trang web gửi về cho Browser, browser thực thi trang web hiện kết quả trên màn hình client. Browser và web server liên lạc qua giao thức HTTP thông qua cổng mặc định là 80.

Trang Web là file .html viết bằng ngôn ngữ HTML (HyperText Markup Language), Browser sẽ thông dịch trang web. Khi client muốn yêu cầu trang web, dùng browser gõ vào chuỗi có dạng sau:

**`http://NameServer:port/path/file.html`**

**http là giao thức liên lạc giữa Browser và Web server, NameServer là tên máy web server đang thực thi, port là số hiệu cổng web server sử dụng, path/file.html là trang web được yêu cầu.**

ví dụ: `http://www.microsoft.com/index.html` (không có port thì mặc định là 80)

Chuỗi này gọi là chuỗi định vị tài nguyên URL (Uniform Resource Locator) dùng để xác định tài nguyên trên mạng Internet. Ngoài giao thức HTTP còn có thể sử dụng các giao thức khác như là FTP, Gopher, File, và News. Hiện có các Browser thông dụng như: Internet Explorer, Netscape Navigator, và các Web Server như: IIS (Internet Information Server), PWS (Personal Web Server), JRUN, Tomcat,...

## **2. Giao thức TCP/UDP**

### **a) Giao thức TCP:**

- Thiết lập kết nối và duy trì kết nối.
- Đảm bảo dữ liệu gửi, được nhận chính xác và đúng thứ tự, ngược lại sẽ báo lỗi.

Các giao thức Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), và Telnet là những ứng dụng dùng kết nối TCP. Máy tính liên lạc dùng giao thức TCP giống như con người liên lạc bằng điện thoại.

### **b) Giao thức UDP:**

- Không duy trì kết nối
- Không đảm bảo dữ liệu gửi, được nhận chính xác và đúng thứ tự,
- Gửi/nhận dữ liệu dạng gói (datagram), các gói gửi/nhận độc lập với nhau.

UDP nhanh hơn TCP vì không kiểm tra dữ liệu, không cần duy trì kết nối. Những ứng dụng như hỏi giờ, nhắn tin, lệnh ping nên dùng UDP. Máy tính liên lạc dùng giao thức UDP giống như con người liên lạc bằng thư tín.

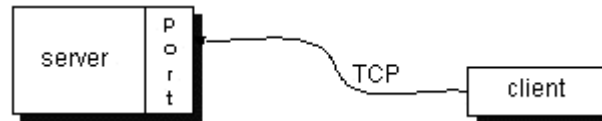
Lưu ý: có những firewalls và routers không cho phép gửi/nhận gói UDP do admin đã đặt cấu hình cấm gói UDP.

## **3. Địa chỉ IP, cổng (Port), socket:**

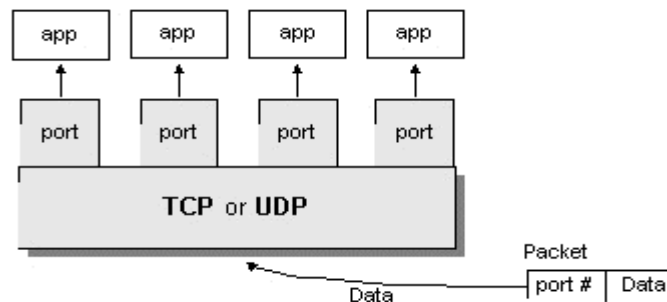
- **Địa chỉ IP:** là số 32 bit mà IP dùng để xác định máy tính.
- **Cổng:** là số 16 bit mà TCP/UDP dùng để xác định ứng dụng trên máy tính sẽ nhận dữ liệu. Dữ liệu khi gửi đi, được gửi kèm theo địa chỉ IP của máy nhận và cổng mà ứng dụng trên máy nhận sử dụng. Số hiệu cổng trong phạm vi từ 0 ... 65,535, những số hiệu cổng từ 0 đến 1023 nên hạn chế sử dụng vì chúng đã được dùng cho những dịch vụ thông dụng như HTTP, FTP.

<b>Dịch vụ</b>	<b>Cổng</b>
FTP (truyền /nhận file)	21
HTTP (web)	80
TELNET (truy xuất máy tính từ xa)	23
SMTP (gửi mail)	25
POP3 (lấy mail)	110

- **Socket:** là cấu trúc dữ liệu lưu trữ các thông tin dùng để kết nối, dữ liệu được gửi/nhận thông qua socket. Trong liên lạc TCP, ứng dụng server kết buộc một socket với một cổng cụ thể, nghĩa là ứng dụng server đăng ký nhận tất cả dữ liệu gửi cho cổng đó.



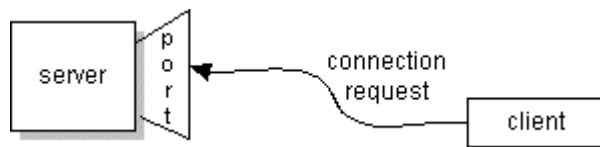
Trong liên lạc UDP, dữ liệu gửi/nhận dạng gói. Gói chứa số hiệu cổng, UDP sẽ gửi gói cho ứng dụng tương ứng.



## II. Lập trình TCP - Sử dụng socket

### 1. Khái niệm:

- Socket là một đầu trong kết nối liên lạc hai chiều giữa hai chương trình trên mạng. Gói java.net cung cấp lớp Socket để cài đặt kết nối phía client và ServerSocket để cài đặt kết nối phía server.
- SocketServer đợi, lắng nghe yêu cầu kết nối từ SocketClient



Khi chấp nhận kết nối, SocketServer tạo một socket mới kết buộc với một cổng khác để phục vụ cho client đã kết nối, trong khi SocketServer vẫn tiếp tục lắng nghe yêu cầu kết nối từ các client khác.





Sau đó server và client sẽ liên lạc với nhau thông qua socket của chúng.

Lưu ý:

- + Client dùng cổng cục bộ trên máy của client.
- + Nếu kết nối tới Web Server thì lớp URL thích hợp hơn lớp Socket, thực ra lớp URL cũng sẽ sử dụng lớp socket

## **2. Cấu trúc chương trình client**

STT	Thao tác	Sử dụng lệnh
1	Mở socket (dùng để kết nối với server)	<code>Socket cs = new Socket("ServerName", port);</code>
2	Mở luồng đọc (dùng để đọc dl do server gửi)	<code>BufferedReader in = new BufferedReader(new InputStreamReader(cs.getInputStream() ));</code>
3	Mở luồng ghi (dùng để gửi dl cho server)	<code>PrintWriter out = new PrintWriter(cs.getOutputStream(), true);</code>
4	Mở luồng đọc dl từ bàn phím (nếu cần)	<code>BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in) );</code>
5	Gửi dl cho server	<code>out.println(String str);</code>
6	Lấy dl do server gửi	<code>String str=in.readLine();</code>
7	Đóng các luồng đọc, ghi, đóng socket	<code>in.close();out.close(); sdtIn.close(); cs.close();</code>

**Ghi chú:**

- Reader, writer: có thể đọc ghi kí tự Unicode qua socket
- Đóng luồng đọc/ghi, sau đó mới đóng socket

### \* Chương trình client mẫu

```
import java.io.*;
import java.net.*;
public class Client
{
    public static void main(String[] args) throws IOException
    {
        Socket cs = new Socket("ServerName", 1234);
        PrintWriter out = new PrintWriter(cs.getOutputStream(), true);
        BufferedReader in = new BufferedReader(new InputStreamReader(
            cs.getInputStream() ));
        // các lệnh gọi, nhận dữ liệu với server
        out.close();    in.close();    cs.close();
    }
}
```

### 3. Cấu trúc chương trình server

STT	Thao tác	Sử dụng lệnh
1	Mở socket (dùng để kết nối với client)	ServerSocket ss = new ServerSocket(port);
2	Chờ client kết nối và chấp nhận kết nối	Socket cs= ss.accept();
3	Mở luồng đọc (dùng để đọc dl do server gửi)	BufferedReader in = new BufferedReader (new InputStreamReader( cs.getInputStream() ));
4	Mở luồng ghi (dùng để gửi dl cho client)	PrintWrite out = new PrintWriter (cs.getOutputStream(), true);
5	Tạo luồng đọc dl từ bàn phím (nếu cần)	BufferedReader stdIn = new BufferedReader (new InputStreamReader(System.in) );
6	Gửi dl cho client	out.println(String str);
7	Lấy dl do client gửi	String str=in.readLine();
8	Đóng các luồng đọc/ghi, đóng các socket	in.close();out.close(); sdtIn.close(); cs.close(); ss.close();

### \* Chương trình server mẫu (1 client)

```
import java.net.*;
import java.io.*;
public class Server {
    public static void main(String[] args) throws IOException
    {
        ServerSocket ss = new ServerSocket(1234);
        Socket cs = ss.accept();
        PrintWriter out = new PrintWriter(cs.getOutputStream(), true);
        BufferedReader in = new BufferedReader(new InputStreamReader(
            cs.getInputStream()));
        //các lệnh gửi/nhận dữ liệu với một client
        out.close();    in.close();    cs.close();    ss.close();
    }
}
```

**Để phục vụ nhiều client kết nối đồng thời thì server cần tạo ra các tiểu trình, mỗi tiểu trình sẽ gửi/nhận dữ liệu với một client.**

### \* Chương trình server mẫu (nhiều client)

```
import java.net.*;
import java.io.*;
public class Server
{
    public static void main(String[] args) throws IOException
    {
        ServerSocket ss = new ServerSocket(1234);    //có thể thay port khác
        boolean listening = true;
        while (listening)
        {
            Socket cs=ss.accept();
            new ServiceThread(cs).start();
        }
        ss.close();
    }
}
```

```
}  
}
```

**//lớp ServiceThread : tiểu trình phục vụ cho một client**

```
import java.net.*;
```

```
import java.io.*;
```

```
public class ServiceThread extends Thread
```

```
{
```

```
    private Socket socket = null;
```

```
    public ServiceThread(Socket socket)
```

```
{
```

```
        super("ServiceThread");
```

```
        this.socket = socket;
```

```
}
```

```
    public void run()
```

```
{
```

```
        try
```

```
        {
```

```
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
```

```
            BufferedReader in = new BufferedReader( new InputStreamReader(  
socket.getInputStream()));
```

```
    //các lệnh gửi, nhận dữ liệu với một client
```

```
        out.close(); in.close(); socket.close();
```

```
    } catch (IOException e) { e.printStackTrace(); }
```

```
}
```

```
}
```

**\* Chương trình server mẫu (nhiều client) (cách khác)**

```
import java.net.*;
```

```
import java.io.*;
```

```
public class Server extends Thread
```

```
{
```

```
    private Socket socket = null;
```

```
    public Server(Socket socket)
```

```

{
    super("ServiceThread");
    this.socket = socket;
}

public static void main(String[] args) throws IOException
{
    ServerSocket ss = new ServerSocket(1234);    //có thể thay port khác
    boolean listening = true;
    while (listening)
    {
        Socket cs=ss.accept();
        new Server(cs).start();
    }
    ss.close();
}

public void run()
{
    try
    {
        PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
        BufferedReader in = new BufferedReader( new InputStreamReader(
        socket.getInputStream()));
        //các lệnh gửi, nhận dữ liệu với một client
        out.close(); in.close(); socket.close();
    } catch (IOException e) { e.printStackTrace(); }
}
}

```

### **III. Lập trình UDP- sử dụng datagram**

#### **1. Khái niệm**

Datagram là gói chứa dữ liệu được UDP sử dụng để gửi dữ liệu qua mạng. Thứ tự nhận, nội dung các gói không đảm bảo giống như khi gửi. Gói java.net có các lớp DatagramSocket và DatagramPacket, MulticastSocket dùng để gửi/nhận gói.

## 2. Cấu trúc chương trình client

STT	Thao tác	Sử dụng lệnh
1	Mở một datagram socket dùng để liên lạc với máy server	DatagramSocket socket = new DatagramSocket();
2	Tạo gói gửi và gói chứa dữ liệu	byte[] buf = new byte[256]; //mảng dùng để chứa dữ liệu String str=...; //dữ liệu cần gửi buf=str.getBytes(); //cất dữ liệu vào mảng InetAddress address = InetAddress.getByName(NameServer); //lấy IP của máy server DatagramPacket packet = new DatagramPacket(buf, buf.length, address,1234); //tạo gói gửi socket.send(packet); //gửi gói
3	Tạo gói nhận và nhận gói trả lời	packet = new DatagramPacket(buf, buf.length); socket.receive(packet);
4	Lấy dữ liệu trong gói	String received = new String(packet.getData()).trim();
5	Đóng socket	socket.close();

### \* Chương trình client mẫu

```
import java.io.*;
```

```
import java.net.*;
```

```
public class Client {
```

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

```
        DatagramSocket socket = new DatagramSocket();
```

```
        byte[] buf = new byte[256];
```

```
        InetAddress address = InetAddress.getByName(NameServer);
```

```
        DatagramPacket packet = new DatagramPacket(buf, buf.length, address,1234);
```

```
        socket.send(packet);
```

```

    packet = new DatagramPacket(buf, buf.length);
    socket.receive(packet);
    String received = new String(packet.getData()).trim();
    //các lệnh xử lý dữ liệu
    socket.close();
}
}

```

### 3. Cấu trúc chương trình server

STT	Thao tác	Sử dụng lệnh
1	Tạo một datagram socket dùng để liên lạc với máy client	DatagramSocket socket = new DatagramSocket(1234);
2	Tạo gói nhận và nhận gói do client gửi	byte[] buf = new byte[256]; packet = new DatagramPacket(buf, buf.length); socket.receive(packet);
3	Lấy dữ liệu trong gói	String received = new String(packet.getData()).trim();
4	Tạo gói gửi và gửi gói trả lời cho client	String sendStr = "chuoi dl goi"; buf = sendStr.getBytes(); InetAddress address = packet.getAddress(); int port = packet.getPort(); packet = new DatagramPacket(buf, buf.length, address, port); socket.send(packet);
5	Đóng socket	socket.close();

Ct server không cần tạo tiểu trình vì không có kết nối nào cần duy trì giữa client và server. Để phục vụ nhiều client, chương trình server chỉ cần dùng một vòng lặp lần lượt nhận các gói của các client và trả lời. Nếu công việc trả lời thực hiện tốn nhiều thời gian thì khi đó nên tạo tiểu trình thực hiện công việc trả lời.

#### \* Chương trình server mẫu

```

import java.io.*;
import java.net.*;
public class Server {
    public static void main(String[] args) throws IOException {
        DatagramSocket socket = new DatagramSocket(1234);
        while (true) {
            byte[] buf = new byte[256];
            DatagramPacket packet = new DatagramPacket(buf, buf.length);
            socket.receive(packet);
            String received = new String(packet.getData()).trim();
            //xử lý dl nhận
            String sendStr = "chuoi dl goi";
            buf = sendStr.getBytes();
            InetAddress address = packet.getAddress();
            int port = packet.getPort();
            packet = new DatagramPacket(buf, buf.length, address, port);
            socket.send(packet);
        }
        socket.close();
    }
}

```

- Hết -



## **BÀI TẬP CHƯƠNG IV**

### **LẬP TRÌNH CLIENT-SERVER**

#### **\* Bài 1:**

Viết chương trình giải phương thức bậc nhất, dạng client-server dùng giao thức TCP, đáp ứng yêu cầu một client. Cải tiến chương trình để chương trình có thể đáp ứng yêu cầu của nhiều client đồng thời và client có thể yêu cầu server nhiều lần.

#### **\* Bài 2:**

Tương tự bài 1 nhưng client viết ở dạng Applet.

#### **\* Bài 3:**

Viết chương trình download, upload dạng client-server dùng giao thức TCP. Server đáp ứng yêu cầu download hay upload của nhiều client đồng thời.

#### **\* Bài 4:**

Viết chương trình quản lý sinh viên có các chức năng sau: nhập mã sv, họ tên, đlt, đth, tính đtb và ghi thông tin này vào CSDL SQL SERVER; tìm/xóa/sửa thông tin sinh viên khi biết masv; xem danh sách sinh viên. Chương trình viết dạng client-server, dùng giao thức TCP, server có thể đáp ứng yêu cầu của nhiều client đồng thời và client có thể yêu cầu server nhiều lần. Hãy viết chương trình bằng hai cách: không có giao diện và có giao diện

#### **\* Bài 5:**

Viết chương trình chat dạng client-server, dùng giao thức TCP. Server nhận một thông điệp từ một client, server sẽ gửi thông điệp này cho các client khác.

#### **\* Bài 6:**

Viết chương trình hỏi thời tiết dạng client-server, dùng giao thức UDP. Biết rằng thông tin về thời tiết lưu trong tập tin thoitiet.txt dạng sau:

Tỉnh/tp	nhietdo	doam
Hanoi	20-30	10%
Hcm	30-40	5%
Vv...		

#### **\* Bài 7:**

Viết chương trình hỏi tỉ giá, biết rằng thông tin tỉ giá lưu trong CSDL SQL SERVER.

#### **\* Bài 8:**

Viết hàm ping dùng giao thức UDP.

- Hết -

## CHƯƠNG V

# LẬP TRÌNH WEB

### I. Khái niệm

#### 1.1 Lập trình Web

- Khi lập trình mạng dạng client/server, nếu không có nhu cầu đặc biệt, thường ta không phải viết ct client và ct server mà sử dụng ct client có sẵn là Browser (web client), server có sẵn là Web Server, kỹ thuật này gọi là lập trình web.
- Người lập trình web viết những trang web bằng ngôn ngữ HTML, các trang web này để ở Web server. Web server và Web client liên lạc với nhau theo giao thức HTTP. Khi Web client gửi yêu cầu trang web tới Web Server tại cổng 80 (là cổng mặc định của Web Server), Web Server tìm trang web và trả về trang web cho Web Client. Web client sẽ thông dịch trang Web và hiện trên màn hình client.
- Các Web Server thông dụng như IIS (Internet Information Service), PWS (PersonalWeb Server), JSP (Java Web Server), Jrun, Tomcat,... Các Web Client thông dụng như là Netscape Navigator, Internet Explore,...
- Client được server trả về cho những trang web đã có sẵn trên máy server, trang web này có nội dung không thay đổi, không thể tương tác với người dùng, không có chức năng xử lý, không thể truy vấn CSDL,... gọi là trang web tĩnh (đây là hạn chế của ngôn ngữ HTML). Trang web động là trang web chưa có sẵn trên server mà do một chương trình khác chạy trên server sinh ra khi nhận được yêu cầu từ client, trang web động sẽ được gửi về cho client. Trang web động có thể thay đổi nội dung tùy theo các tham số do client gửi hoặc tùy theo nội dung CSDL,...
- Các kỹ thuật lập trình web động thông dụng là CGI, ASP, APPLET, SERVLET, JSP, J2EE, EJB,...

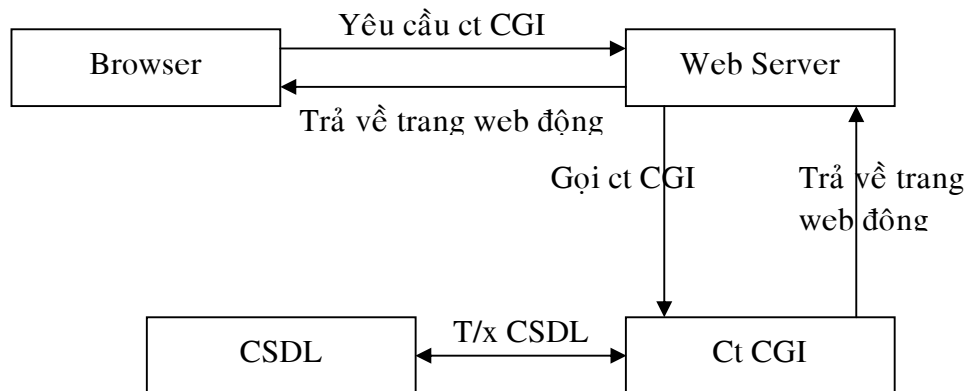
#### 1.2 Giao thức HTTP

- Giao thức mà Web Client dùng để yêu cầu trang Web từ Web Server. Mỗi khi yêu cầu được đáp ứng thì kết nối sẽ kết thúc (kết nối phi trạng thái).
- Giao thức có hai lệnh cơ bản là GET, POST. Khi NSD nhập URL vào hộp address của Browser thì Browser dùng lệnh GET để liên lạc với web server. Nếu trang web dùng thẻ lệnh <FORM METHOD=POST/GET> thì web client dùng lệnh POST/GET để gửi dữ liệu trong form cho web server. Method có thể là GET khi đó dữ liệu trong form sẽ hiện trên URL của client.

#### 1.3 Lập trình CGI (Common Gateway Interface)

- Khi nhận yêu cầu từ client, server gọi một chương trình CGI xử lý yêu cầu này. Ct CGI sinh ra trang web và gửi về cho client. Ct CGI là một ct thực thi được (.exe, .com) có thể viết bằng C, Pascal,.. dùng để sinh ra trang web động.

- **Cơ chế hoạt động của ct CGI:** Mỗi khi client yêu cầu ct CGI, server sẽ thực hiện các bước sau:
  - . Nạp ct CGI vào bộ nhớ
  - . Thực thi ct CGI, ct CGI tạo ra trang web động.
  - . Trả kết quả là trang web động về cho client
  - . Thu hồi bộ nhớ đã cấp phát cho ct CGI.



**Nhận xét:** kỹ thuật CGI thường chậm, tốn tài nguyên vì mỗi lần nhận được yêu cầu, server phải nạp ct CGI, sau đó phải giải phóng.

**Ví dụ:**

-viết file hello.cpp

```

#include <stdio.h>

void main()
{
    printf("<html>");
    printf("<h1> Hello World!</h1>");
    printf("</html>");
}
  
```

dịch ra hello.exe, chép vào web root của web server

- ở client gõ: <http://localhost/hello.exe>

## II. Lập trình web bằng SERVLET

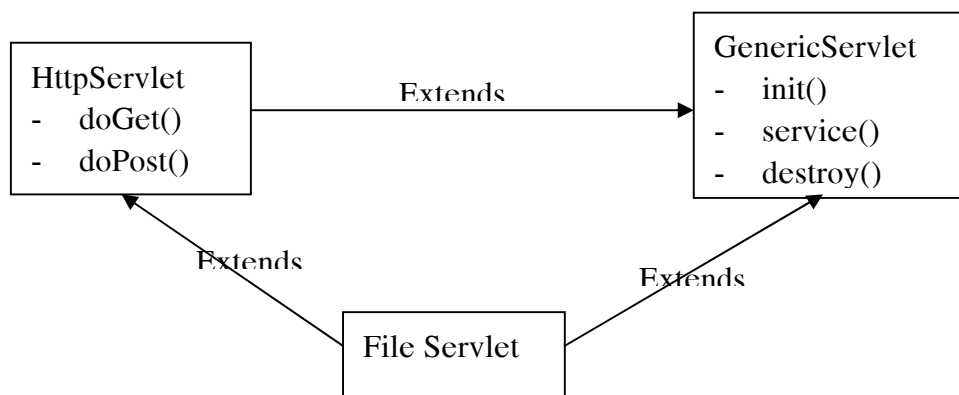
### 2.1 Khái niệm

#### - Servlet là gì?

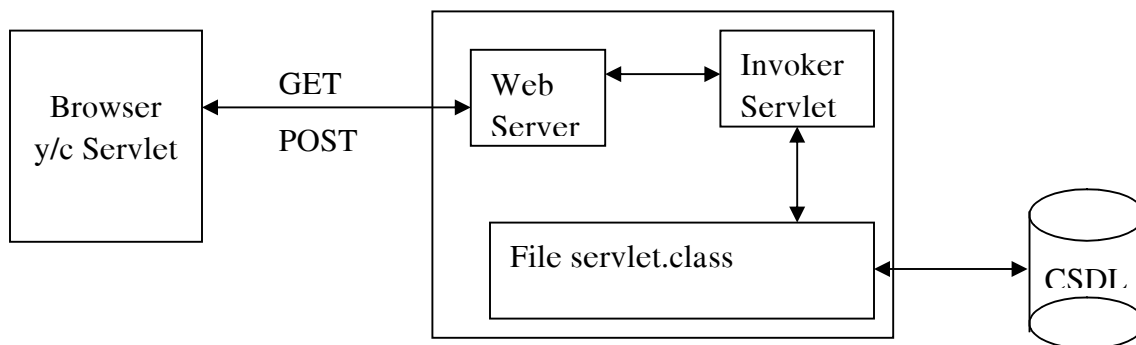
Servlet là file Java, thừa kế lớp *javax.servlet.http.HttpServlet* hoặc lớp *javax.servlet.GenericServlet* dùng để sinh ra trang web động. Khi client yêu cầu một servlet, bằng lệnh GET hoặc POST, web server nhận yêu cầu và gọi ct Invoker Servlet thực thi file sevlet và sẽ sinh ra một trang web động gửi về cho client.

Invoker Servlet là chương trình được nhúng sẵn vào các web server hiểu Java (JRUN, Tomcat, ...) dùng để thực thi file sevlet. Một số web server như là IIS, PWS không có trình Invoker Servlet nên không thể thực thi file servlet.

#### - Sơ đồ thừa kế



#### - Mô hình hoạt động của servlet



- So sánh CGI, Servlet
- Chu kỳ sống của servlet: Nạp servlet , khởi tạo servlet , thực thi servlet, dọn dẹp servlet
- So sánh kỹ thuật servlet và applet.

KỸ THUẬT APPLET	KỸ THUẬT SERVLET
Là lớp java thừa kế lớp Applet hoặc Japplet, có các phương thức đặc biệt như là: init, start, stop, run, destroy,... các phương thức này được thực thi bởi browser	Là lớp java thừa kế lớp HttpServlet hoặc GenericServlet, có các phương thức đặc biệt như là init, service, doGet, doPost,..., các phương thức này được Invoker Servlet thực thi.
Applet được download về máy client và thực thi nhờ trình duyệt. Lập trình Applet còn gọi là lập trình phía client)	File servlet thực thi trên máy server, tạo trang web động, trả về cho client (Lập trình servlet còn gọi là lập trình phía server)
Applet thường chậm, máy client phải có cấu hình cao vì phải download, và việc thực thi ở phía client	Servlet thực thi nhanh hơn, máy client không cần có cấu hình cao vì việc thực thi ở phía server.
Phụ thuộc vào Browser có hỗ trợ Java	Phụ thuộc vào Web Server có hỗ trợ Java, nhưng không phụ thuộc vào Browser.

## 2.2 Các phương thức đặc biệt

- ***public void init (ServletConfig config) throws ServletException***

```
{
    /*các lệnh khởi tạo, lệnh chỉ thực hiện một lần như là: gán trị ban đầu cho biến, kết nối
    server, kết nối CSDL, ...*/
}
```

Phương thức này được Invoker Servlet gọi khi thực thi servlet lần đầu tiên.

- ***public void destroy (ServletConfig config) throws ServletException***

```
{
    /*các lệnh dọn dẹp như là: đóng kết nối server, đóng kết nối CSDL,... */
}
```

Phương thức này được Invoker Servlet gọi khi servlet hết thời gian qui định lưu trong bộ nhớ. Thông thường servlet khi được nạp vào bộ nhớ để thực thi, thực thi xong vẫn lưu trong bộ nhớ trong một khoảng thời gian do web server qui định, nếu các client yêu cầu servlet lần nữa thì Invoker Servlet không phải nạp servlet vào bộ nhớ nữa mà chỉ việc thực thi servlet, tốc độ sẽ nhanh hơn nhiều và đây cũng là ưu điểm của servlet so với CGI.

- ***public void service (ServletRequest req, ServletResponse res) throws ServletException, IOException***

```
{
    /*Xử lý yêu cầu GET và yêu cầu POST: Lấy dl do client gửi, xử lý dl, truy xuất CSDL, gửi
    trang web động về cho client */
}
```

Phương thức này được Invoker Servlet gọi khi client gửi yêu cầu bằng lệnh POST hoặc GET. req là đối tượng tiếp nhận dl do client gửi, res là đối tượng chứa dl trả về cho client.

- ***public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException***

```
{
    /*Xử lý yêu cầu GET: lấy dl do client gửi, xử lý dl, truy xuất CSDL, gửi trang web động về
    cho client */
}
```

Phương thức này được Invoker Servlet gọi khi client gửi yêu cầu bằng lệnh GET. req là đối tượng tiếp nhận dl do client gửi, res là đối tượng chứa dl trả về cho client.

- *public void doPost(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException*

```
{
    /*Xử lý yêu cầu POST: lấy dl do client gửi, xử lý dl, truy xuất CSDL, gửi trang web động về
    cho client */
}
```

Phương thức này được Invoker Servlet gọi khi client gửi yêu cầu bằng lệnh POST. req là đối tượng tiếp nhận dl do client gửi, res là đối tượng chứa dl trả về cho client.

#### **Lưu ý:**

- Do doPost(), doGet() là các phương thức trong lớp HttpServlet nên muốn sử dụng các phương thức này, servlet phải thừa kế lớp HttpServlet.
- Nếu client gửi yêu cầu bằng lệnh GET hoặc POST thì servlet phải có phương thức tương ứng là doGet hoặc doPost hoặc service để xử lý yêu cầu, nếu không sẽ báo lỗi.

## **2.3 Dạng file servlet**

- **Dạng 1: thừa kế lớp GenericServlet**

```
import javax.servlet.*;
import java.io.*;

public class File_Name_Servlet extends GenericServlet
{
    public void service (ServletRequest req, ServletResponse res) throws ServletException,
    IOException
    {
        res.setContentType("text/html");// đặt kiểu dl trả về là text hoặc html

        PrintWriter out=res.getWriter();// tạo luồng xuất, dùng luồng này để xuất dl về cho
        client

        //xử lý lệnh GET, POST
    }
}
```

- **Dạng 2: thừa kế lớp HttpServlet**

```
import javax.servlet.*;
```

```
import javax.servlet.http.*;
```

```
import java.io.*;
```

```
public class File_Name_Servlet extends HttpServlet
```

```
{
```

```
    public void doPost (ServletRequest req, ServletResponse res) throws ServletException, IOException
```

```
    {
```

```
        res.setContentType("text/html");// đặt kiểu dl trả về là text hoặc html
```

```
        PrintWriter out=res.getWriter();// tạo luồng xuất, dùng luồng này để xuất dl về cho client
```

```
        //xử lý lệnh POST hoặc gọi doGet(req,res)
```

```
    }
```

```
    public void doGet (ServletRequest req, ServletResponse res) throws ServletException, IOException
```

```
    {
```

```
        res.setContentType("text/html");// đặt kiểu dl trả về là text hoặc html
```

```
        PrintWriter out=res.getWriter();// tạo luồng xuất, dùng luồng này để xuất dl về cho client
```

```
        //xử lý lệnh GET hoặc gọi doPost(req,res)
```

```
    }
```

```
}
```



## 2.4 Sử dụng servlet trong Jrun: Tạo file servlet, dịch ra .class

- **Cách 1:**

- Chép file servlet.class vào thư mục [JRUN\_HOME]\servers\default\default-app\web-inf\classes.

(JRUN\_HOME là thư mục cài Jrun)

- Ở client để gọi file servlet.class gõ URL

http://< ServerName>:8100/<ServletName> (không có .class)

Hoặc gọi servlet.class thông qua một trang web khác: tạo trang web, trong trang web dùng thẻ

**<form action = ServletName method=post/get>**

...

**<input type=submit value=submit>**

**</form>**

rồi ở client thay vì gọi trực tiếp file servlet, ta gọi trang web có thẻ lệnh <form action = ServletName>, khi click nút submit sẽ gọi file servlet

- **Cách 2:**

- Chép file servlet.class vào thư mục [JRUN\_HOME]\servlets.

- Ở client để gọi file servlet.class gõ

http://< ServerName>:8100/servlet/<ServletName> (không có .class)

Hoặc dùng trang web trung gian có thẻ lệnh form

**<form action = servlet/ServletName method=post/get>**

- **Chú ý:** các file .class thông thường (không phải là servlet) chép vào tm [JRUN\_HOME]\servers\default\default-app\web-inf\classes

## 2.5 Gửi/lấy tham số

- **Gửi tham số cho servlet: có hai cách**

- Gửi trực tiếp qua URL:

URL?< tênthamsố1>=<giá trị1>&< tênthamsố2>=<giá trị2> [...]

- Gửi qua trung gian trang web có sử dụng thẻ form và nút submit

- **Lấy giá trị tham số:** nếu giá trị là null thì tham số không được gửi hoặc lấy sai tên ts

- **Lấy tham số có một giá trị**

```
String value= req.getParameter("tênthamsố");
```

- **Lấy tham số có nhiều giá trị**

```
String values[]= req.getParameterValues("tênthamsố");
```

- **Lấy tất cả các tham số**

```
Enumeration names[]= req.getParameterNames(); //lấy tất cả các tên tham số
```

```
While (names.hasMoreElements()) // trong khi còn phần tử (còn tên)
```

```
{
```

```
String name= (String) names.nextElement();// lấy tên một tham số
```

```
String value= req.getParameter(name);// lấy giá trị của tham số đó
```

```
// xử lý giá trị này
```

```
}
```

## 2.6 Các ví dụ

- VD1: Xem giờ trên web
- VD2: Tính diện tích hình tròn
- VD3: Thi trắc nghiệm có nhiều dạng

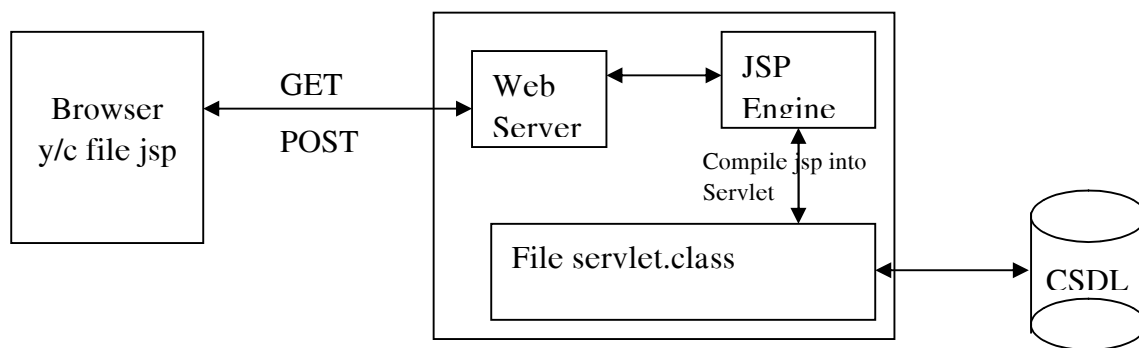
### III. Lập trình web bằng JSP

#### 3.1 Khái niệm

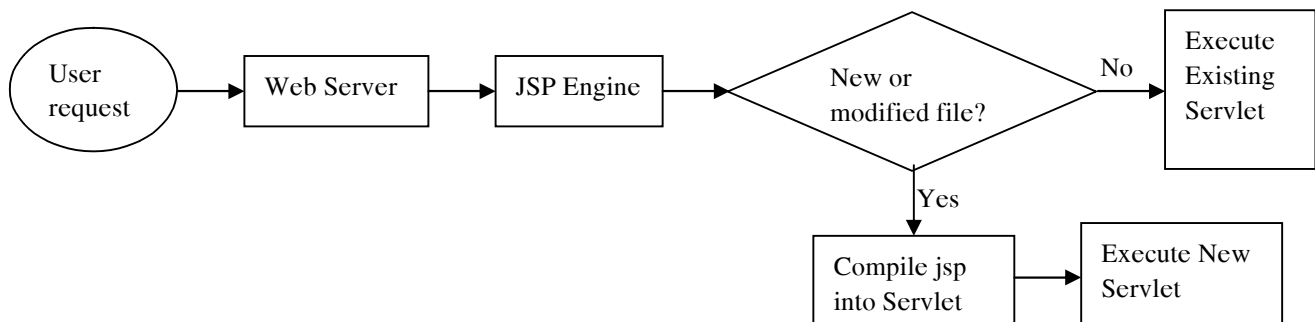
##### - File jsp là gì?

File Jsp là file văn bản mà tên có phần mở rộng là jsp. Trong file trộn lẫn các lệnh java và lệnh html. Khi client yêu cầu trang jsp, web server gọi JSP Engine, JSP Engine biên dịch file jsp thành file servlet và nạp servlet vào bộ nhớ cho thực thi, sinh ra trang web động và trả về cho client. Nếu client yêu cầu trang jsp đã được biên dịch thành servlet.class thì JSP Engine sẽ không biên dịch nữa nên file jsp chỉ thực hiện lần đầu tiên là chậm, những lần sau sẽ nhanh hơn.

##### - Mô hình hoạt động của công nghệ JSP



##### - Lưu đồ hoạt động của mô hình JSP



### 3.2 Các đối tượng do web server tạo sẵn

- out: là đối tượng dùng để gửi dữ liệu cho client (browser)
- request: là đối tượng dùng để lấy dữ liệu do client gửi
- response: là đối tượng dùng để gửi dữ liệu cho client
- session: là đối tượng dùng để lưu các biến session. Biến session có hiệu lực trong một phiên làm việc.
- application: là đối tượng dùng để lưu các biến application. Biến application có hiệu lực trong tất cả các phiên làm việc.

### 3.3 Các thẻ lệnh của JSP

- **Chú thích:**

`<%-- chú thích các lệnh html, java --%>`

Chú thích java dùng `//` hoặc `/* */`

- **Thông báo khối lệnh Java**

`<% các lệnh Java %>`

- **Hiện giá trị biến (không có dấu chấm phẩy sau tên biến)**

`<%=tên biến%>`

giống lệnh `out.print(tên_biến);`

- **Định hàm hoặc khai báo biến**

`<%! Định hàm hoặc khai báo biến có phạm vi ứng dụng %>`

Trong các hàm muốn sử dụng đối tượng out, cần phải gọi như tham số.

- **Nhúng file (file nhúng thường là html hoặc jsp hoặc class)**

- a) **Nhúng nội dung**

`<%@ include file="tenfile"%>`

### b) Nhúng kết quả

```
<jsp:include page="tenfile" />
```

hoặc

```
<jsp:include page="tenfile" >
```

```
<jsp:param name="tenbien" value="giatri"/>
```

```
</jsp:include>
```

### c) Nhúng lớp

```
<%@ page import="tengoi.tenlop"%>
```

- **Chuyển trang**

- a) **Tự động chuyển tham số**

```
<jsp:forward page="tenfile"/>
```

- b) **Không tự chuyển tham số, để chuyển tham số dùng URL**

```
response.sendRedirect("tenfile?tents=gt&tents=gt");
```

## 3.4 Biến session, cookies, application

Ngay sau khi yêu cầu trang web của client được đáp ứng, kết nối http giữa client và server sẽ bị đóng, nên giao thức http còn gọi là giao thức kết nối phi trạng thái. Do kết nối bị đóng nên lần kết nối sau để yêu cầu một trang web khác thì thông tin liên quan đến kết nối trước, kết nối sau sẽ không thể biết được. Để khắc phục khuyết điểm này Java cung cấp ba loại biến là: session, cookies, application.

## 3.5 Session

- **Phiên làm việc (session):** là khoảng thời gian được tính từ thời điểm mở trình duyệt liên lạc với web server đến khi đóng trình duyệt. Nếu mở trình duyệt mới là qua phiên làm việc khác
- **Đối tượng session:** Mỗi phiên làm việc, web server tạo ra một đối tượng session dùng để lưu những biến session. Đối tượng session chỉ tồn tại ở máy server trong một phiên làm việc hoặc trong khoảng thời gian qui định bởi web server , thường web server có option để thay đổi thời gian này.

- **Biến session:** là biến lưu trong đối tượng session nên biến session cũng tồn tại trong một phiên làm việc, mọi trang cùng phiên làm việc đều truy cập được biến session.
- **Các phương thức:** Đối tượng session có các phương thức thông dụng sau:

**a) public void setAttribute(String name, Object value)**

Đặt biến session tên là name có giá trị là value vào trong đối tượng session

**b) public Object getAttribute(String name)**

Lấy giá trị biến session trong đối tượng session có tên là name. Trả về null nếu không tìm thấy biến này. Các giá trị lưu trữ hay lấy ra có kiểu là Object.

**c) public void removeAttribute(String name)**

Hủy biến session tên là name trong đối tượng session

**d) public void invalidate()**

Hủy đối tượng session, khi đó tất cả các biến session trong đối tượng session cũng bị hủy.

### 3.6 Application

- **Đối tượng application :** web server tạo ra một đối tượng application dùng để lưu những biến application. Đối tượng application tồn tại ở máy server trong khoảng thời gian qui định bởi web server , thường web server có option để thay đổi thời gian này, nếu hết thời gian qui định sẽ bị hủy và tạo mới. Nếu khởi động lại web server thì đối tượng application cũng sẽ bị hủy và sẽ tạo mới.
- **Biến application:** là biến trong đối tượng application, mọi trang cùng phiên làm việc hoặc khác phiên đều truy cập được biến application.
- **Các phương thức:** Đối tượng application cũng có các phương thức giống như đối tượng session.

### 3.7 Cookies

- **Biến cookies**

Là biến do người dùng tạo ra và lưu ở máy client trong khi đó biến session, application lưu ở server. Các biến cookie mặc định được browser gửi cho web server khi gửi yêu cầu trang web. Biến cookie sẽ bị hủy khi đóng trình duyệt hoặc hết thời gian sống được qui định vào lúc tạo biến cookie.

- **Các phương thức:**

- a) **Tạo biến cookie**

```
Cookie c= new Cookie("tenbien",giatri);
```

- b) **Đặt thời gian sống cho biến cookie**

```
c.setMaxAge(time); time tính bằng giây, time=0 là hủy cookie
```

- c) **Ghi biến cookie vào client**

```
response.addCookie(c)
```

- d) **Lấy tất cả các biến cookie**

```
Cookie[] cs=request.getCookies();
```

- e) **Lấy tên biến cookie thứ i**

```
cs[i].getName();
```

- f) **Lấy giá trị biến cookie thứ i**

```
cs[i].getValue();
```

### 3.8 Truyền dữ liệu qua nhiều trang

- Dùng biến ẩn
- Dùng URL
- Dùng session, application, cookie

#### IV. Java bean

- **Java bean**

Là tập hợp của một hay nhiều lớp java dùng giao diện Serializable , bean độc lập nền và có thể dùng lại được. Trong file bean phải có các thuộc tính, phương thức set thuộc tính, get thuộc tính.. Bean có thể dùng trong application, applet, jsp,servlet. Trong file jsp, Bean sẽ che dấu và để truy xuất đối tượng bean dùng các thẻ lệnh của JSP.

- **Các thẻ lệnh jsp để truy xuất bean**

**a) Khai báo**

```
<jsp:useBean id="tênđtbean" scope="page/session/application" class="tênlớpbean"/>
```

**b) Gán thuộc tính cho bean**

```
<jsp:setProperty name="tênđtbean" property="tênthuộctính" value="giá trị"/>
```

**c) Trả về giá trị thuộc tính của bean dạng chuỗi**

```
<jsp:getProperty name="tênđtbean" property="tênthuộctính"/>
```

**d) Đóng khai báo bean: nếu không dùng các thẻ setProperty, getProperty thì không cần đóng**

```
</jsp:useBean>
```

- **Nhận xét**

Để đặt thuộc tính, lấy thuộc tính có thể dùng thẻ hoặc tạo đối tượng bean và gọi trực tiếp.

#### **BÀI TẬP**

**\* SERVLET**

1. Viết servlet xem cửu chương, nhập số hiệu cửu chương và hiện cửu chương tương ứng.
2. Viết servlet giải pt bn.
3. Viết ct thi trắc nghiệm có nhiều loại chọn

- Hết -



# CHƯƠNG 1

## LẬP TRÌNH ĐỐI TƯỢNG PHÂN TÁN

(DISTRIBUTED OBJECT PROGRAMMING)

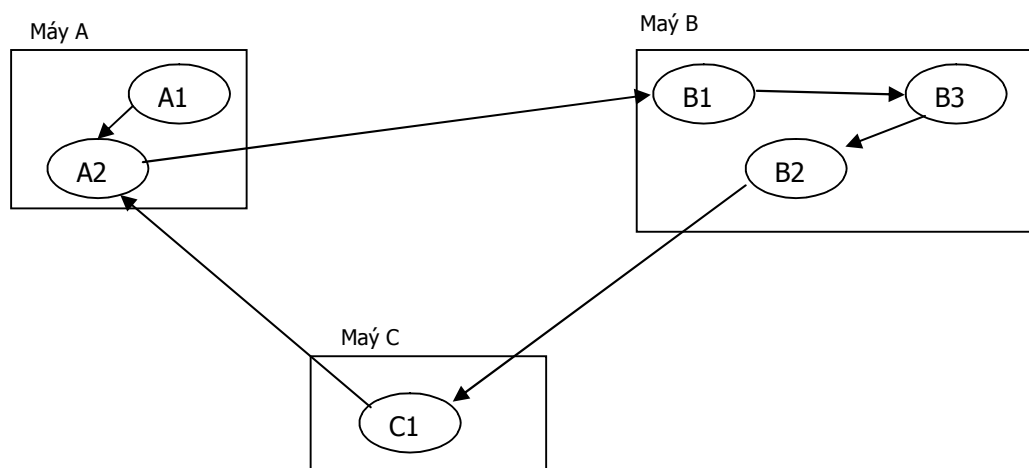
### I. TỔNG QUAN

Công nghệ lập trình đối tượng phân tán là công nghệ cho phép các đối tượng viết bằng những ngôn ngữ khác nhau, chạy trên các máy khác nhau, có thể gọi các phương thức của nhau. Thông thường ứng dụng phân tán có dạng client-server, chương trình server tạo những đối tượng có thể được gọi từ xa (remote objects), chương trình client sẽ triệu gọi phương thức của đối tượng trên máy server.

#### \* Ưu điểm của lập trình phân tán

- Các chương trình đã viết sẵn bằng những ngôn ngữ khác nhau không cần phải viết lại mà vẫn giao tiếp được với nhau.
- Các đối tượng trên các máy khác nhau, do các chương trình khác nhau sinh ra, giao tiếp được với nhau như trên cùng một máy.
- Một ứng dụng lớn có thể được xử lý phân tán ở nhiều máy khác nhau trên mạng nên công nghệ đối tượng phân tán còn gọi là công nghệ xử lý phân tán (Distributed Computing).
- Công nghệ này đặc biệt hữu ích khi một công việc chỉ có thể thực thi trên một máy chuyên dụng. Ví dụ client có một công việc mà công việc này chỉ có thể thực hiện ở một máy server chuyên dụng, client có thể gửi công việc (gửi đối tượng) cho một phương thức của đối tượng trên server thực hiện sau đó trả kết quả về cho client.

#### \* Mô hình lập trình phân tán



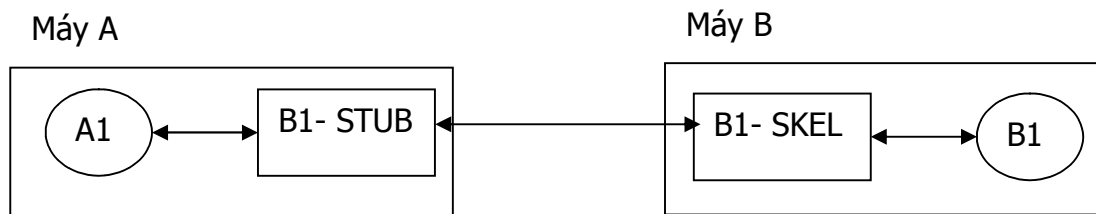
### \* Các vấn đề phát sinh khi gọi đối tượng từ xa

- Các đối tượng trên hai máy khác nhau hoạt động trên hai không gian địa chỉ khác nhau nên việc tham chiếu biến, địa chỉ của đối tượng trên mỗi máy là khác nhau. Ví dụ khi gửi một biến con trỏ (biến chứa địa chỉ) cho phương thức của đối tượng ở xa thì có thể địa chỉ này không tồn tại ở máy chứa đối tượng.
- Lời gọi phương thức ở xa phải thông qua kết nối mạng nên có thể không thực hiện hoặc nhận được kết quả sai khi mạng hay server có vấn đề.

### \* Các lớp trung gian

Để giải quyết hai vấn đề trên các đối tượng trên hai máy khác nhau không gọi trực tiếp lẫn nhau mà thông qua hai lớp trung gian là lớp STUB (lớp móc) và lớp SKEL (lớp nối)

### \* Mô hình lập trình phân tán thông qua lớp trung gian Stub, Skel



Để A1 có thể gọi phương thức của B1, B1 cần cung cấp hai lớp B1-STUB và B1-SKEL. B1-STUB cài đặt ở máy A (máy client), B1-SKEL cài đặt ở máy B (máy server).

Khi A1 gọi B1, lời gọi sẽ chuyển tới B1-STUB. B1-Stub đóng gói lời gọi cùng tham số, chuyển gói đến B1-SKEL. B1-SKEL lấy tham số trong gói lưu vào vùng địa chỉ phù hợp của máy B, sau đó gọi phương thức của B1 cùng với tham số đã nhận. Kết quả nếu có sẽ được B1-SKEL đóng gói và trả về cho B1-STUB, B1-STUB lấy kết quả trong gói và chuyển kết quả cho A1. Nếu mạng gặp sự cố thì B1-STUB và B1-SKEL sẽ gửi lại gói hoặc báo lỗi.

## II. CÔNG NGHỆ RMI (Remote Method Invocation )

### \* Giới thiệu

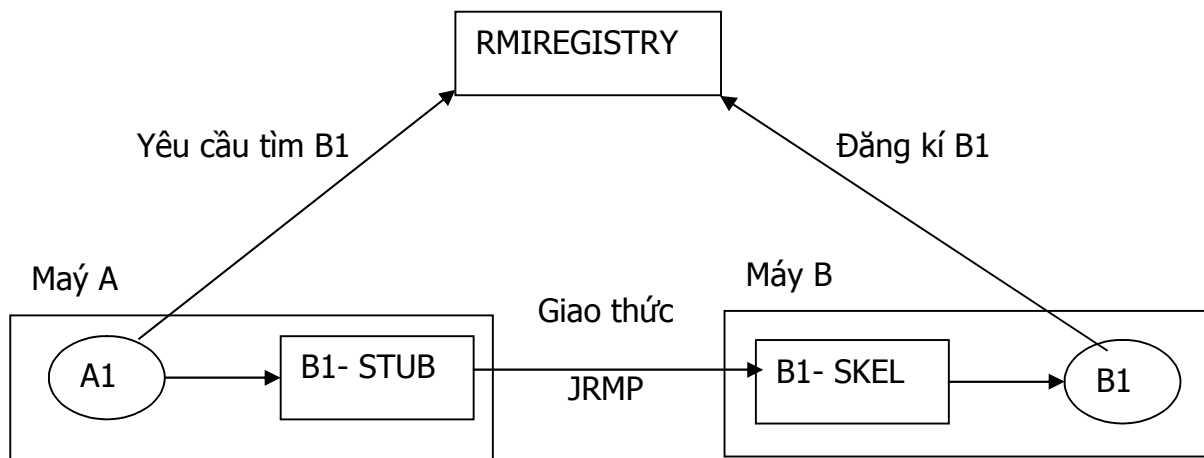
Công nghệ RMI là công nghệ lập trình đối tượng phân tán, trong đó các đối tượng viết bằng cùng ngôn ngữ Java, liên lạc với nhau theo giao thức Java Remote Method Protocol (JRMP).

- Ct sever tạo các đối tượng có thể được gọi từ xa (đối tượng remote), đăng kí các đối tượng này với trình quản lý đối tượng (rmiregistry), sau đó đợi client gọi những pt trên các đối tượng remote.
- Ct client yêu cầu rmiregistry tìm đối tượng remote, rmiregistry trả về tham chiếu tới đối tượng remote, client dùng tham chiếu này gọi pt của đt remote.

RMI cung cấp môi trường để client và server liên lạc với nhau, ngoài ra RMI còn cho phép client gửi một đt cho đt remote, điều này làm mở rộng khả năng của ứng dụng ở xa.

### \* **Mô hình RMI:**

Server đăng kí đt remote với rmiregistry. Client yêu cầu rmiregistry tìm đt remote, nếu tìm thấy sẽ gọi pt của đt remote.



### \* **Đối tượng remote**

Một đt trở thành đt remote bằng cách dùng giao diện remote. Giao diện remote có những đặc tính sau:

- . Khai báo public
- . Thừa kế giao diện java.rmi.Remote
- . Mỗi pt của giao diện khai báo throws java.rmi.RemoteException

### \* **Cài đặt ứng dụng RMI**

- . Định nghĩa giao diện remote: giao diện remote khai báo các pt mà có thể gọi từ xa
- . Xây dựng lớp cài đặt đối tượng remote, lớp này sử dụng giao diện remote
- . Xây dựng lớp server
- . Xây dựng lớp client
- . Biên dịch mã nguồn và sinh ra lớp stub, skel : dùng javac và rmic
- . Khởi động ứng dụng: chạy rmiregistry, server, client

### **Lưu ý:**

- Máy server cần có 5 lớp sau: giao diện remote, lớp cài đặt, lớp Stub, lớp Skel, lớp server. Nếu dùng JDK 1.3 trở lên thì lớp Skel được tích hợp trong lớp cài đặt.
- Máy client cần có 3 lớp sau: giao diện remote, lớp Stub, lớp client. Lớp giao tiếp rất nhỏ nên hầu như khi sử dụng sẽ không tốn tài nguyên của client. Ngoài ra Java cũng cung cấp cơ chế để tự động nạp Lớp Stub xuống máy client.

### \* Truyền tham số

Khi gọi phương thức ở xa, biến kiểu cơ bản như int, float ,... được truyền theo tham trị. Biến kiểu đối tượng muốn truyền qua mạng phải dùng giao diện Remote (truyền theo tham chiếu) hay Serializable (truyền theo tham trị).

## III. CÔNG NGHỆ CORBA

### 1. Khái niệm:

- CORBA (Common Object Request Brokerage Architecture): Công nghệ lập trình đối tượng phân tán do tổ chức OMG (Object Management Group) đề xuất.
- IDL(Interface Definition Language), là ngôn ngữ đặc tả giao diện, dùng IDL để có thể cài đặt client/server bằng các ngôn ngữ khác nhau có hỗ trợ CORBA như là C, C++, Smalltalk, COBOL, Ada, Java, mỗi ngôn ngữ hỗ trợ CORBA đều có trình chuyển đổi IDL sang ngôn ngữ đó. IDL không có lệnh, chỉ có các mô tả hàm, kiểu dữ liệu, các khai báo để đặc tả đối tượng. Các giao diện đầu tiên được đặc tả bằng ngôn ngữ IDL, sau đó dùng trình biên dịch tương ứng để dịch sang ngôn ngữ cụ thể, đồng thời tạo lớp stub, skel và một số lớp hỗ trợ CORBA.

#### ví dụ:

idlj <tengiaodien.idl>: dịch sang Java

idl2cpp <tengiaodien.idl>: dịch sang C++

idl2pas <tengiaodien.idl>: dịch sang Pascal

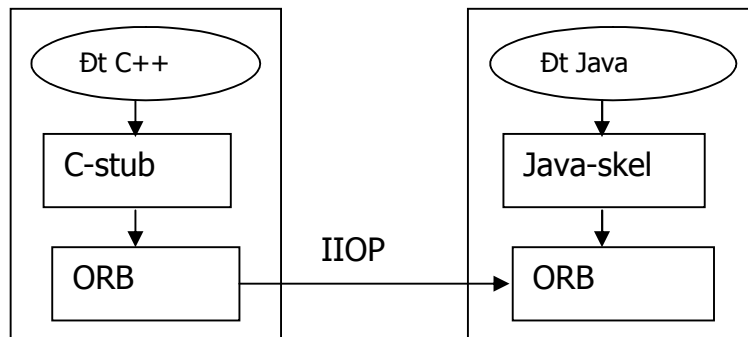
- ORB(Object Request Broker): Các đối tượng viết bằng những ngôn ngữ khác nhau muốn gọi lẫn nhau phải thông qua chương trình môi giới ORB. Trong Jdk 1.3 có sẵn ORB, trong Jbuilder có ORB tên là VisiBroker và OrbixWeb. IIOP(the Internet Inter-ORB Protocol) là giao thức liên lạc giữa các ORB, ORB có thể cung cấp một số dịch vụ khác như là dịch vụ tìm kiếm đối tượng theo tên, dịch vụ duy trì đối tượng lâu dài,...

### 2. Cơ chế hoạt động của CORBA

Khi client gọi phương thức của đối tượng trên server, lời gọi được chuyển qua lớp Stub, rồi qua ORB trên client. ORB ở client kết nối với ORB ở server theo giao thức IIOP. ORB trên server chuyển lời gọi cho Skel, rồi phương thức của đối tượng trên server được gọi. Kết quả nếu có sẽ được đối tượng trên server trả lại qua Skel, ORB trên server, ORB trên client, rồi đến đối tượng trên client

#### Ví dụ:

Giả sử muốn cài đặt đối tượng ở client bằng C++, đối tượng ở server là Java: ở client dùng idl2cpp dịch giao diện sang C++, đồng thời tạo lớp Stub, sau đó viết đt client bằng C++. Ở server dùng idlj dịch giao diện sang Java, đồng thời tạo lớp Skeleton, sau đó viết đt server bằng Java.



### 3. So sánh RMI, CORBA

Công nghệ RMI	Công nghệ CORBA
Các đt viết bằng Java	Các đt có thể viết bằng ngôn ngữ khác nhau
Không phụ thuộc hđh, phần cứng nơi đt thực thi	Phụ thuộc hđh, phần cứng nơi đt thực thi
Không cần biết nn IDL, không cần trình môi giới ORB	Cần biết nn IDL, cần có trình môi giới ORB
Cho phép gọi đt theo tham chiếu hoặc tham trị	Chỉ cho phép gọi đt theo tham chiếu

### 4. Viết ứng dụng CORBA

- Định nghĩa giao diện remote: dùng ngôn ngữ IDL viết giao diện remote.
- Biên dịch giao diện remote sang ngôn ngữ mong muốn: ví dụ dùng trình biên dịch idlj.exe để biên dịch giao diện remote từ IDL sang Java và sinh ra stub, skel cùng với mã dùng để kết nối với ORB
- Cài đặt Server
- Cài đặt Client
- Thực thi ứng dụng

**Ví dụ:** Viết ứng dụng hiện câu chào "Hello World" dùng ORB của Jdk 1.3

**B1:** Viết lớp giao diện bằng IDL (Hello.idl). file này để ở trong thư mục src

```
module HelloCorba
{
    interface Hello
    {
        string sayHello();
    };
};
```

**B2: Trên client dịch Hello.idl sang Java:**

**idlj -fclient Hello.idl**

Sẽ tạo ra 5 file sau: *Hello.java*, *HelloHelper.java*, *HelloHolder.java*, *\_HelloStub.java*, *HelloOperations.java*. Có thể

**B3: Trên server dịch Hello.idl sang Java:**

**idlj -fserver Hello.idl**

Sẽ tạo ra 3 file sau: *HelloOperations.java* (Hello.idl dạng Java), *Hello.java*, *\_HelloImpBase.java*.

Trong vd này B2, B3 có thể thay bằng một lệnh: `idlj -fall Hello.idl` và sẽ sinh ra 6 file: *Hello.java*, *HelloHelper.java*, *HelloHolder.java*, *\_HelloStub.java*, *HelloOperations.java*, *\_HelloImpBase.java*.

- *HelloOperations.java*: chính là bản dịch sang Java của giao diện *Hello.idl*
- *Hello.java*: thừa kế *HelloOperations*, *org.omg.CORBA.Object*, *org.omg.CORBA.portable.IDLEntity* để trở thành đt CORBA
- *\_HelloStub.java* (lớp stub): cung cấp chức năng CORBA cho client, dùng giao diện *Hello.java*
- *\_HelloImpBase.java*: dùng để tạo lớp cài đặt đối tượng trên server
- *HelloHelper.java*, *HelloHolder.java*: cung cấp các pt truy xuất đt CORBA

#### **B4: Xây dựng lớp dùng để tạo đối tượng CORBA (file HelloServant.java)**

```
package HelloCorba;

class HelloServant extends _HelloImplBase
{
    public String sayHello()
    {
        return "\nHello world !\n";
    }
}
```

#### **B5: Viết ct server dùng để tạo đối tượng và đăng ký đối tượng với trình quản lý đối tượng**

```
package HelloCorba;

import org.omg.CosNaming.*; // HelloServer sẽ dùng dịch vụ ql ten
import org.omg.CORBA.*;    // tất cả ứng dụng CORBA cần lớp này
import org.omg.CosNaming.NamingContextPackage.*;

public class HelloServer
{
    public static void main(String args[])
    {
        try{

            //khởi động trình ORB
            ORB orb = ORB.init(args, null);

            // tạo dt corba
            HelloServant helloRef = new HelloServant();

            // kết nối ORB và thông báo dt với ORB, giống exportObject ở RMI
            orb.connect(helloRef);

            // lấy tham chiếu đến dịch vụ quản lý tên tnameserv, giống rmiregistry
            org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
```

```

//chuyen dt objRef la dt corba tong quat sang dt kieu NamingContext (ep kieu)
//dt nay dung de goi dich vu ql ten
NamingContext ncRef = NamingContextHelper.narrow(objRef);
// tao ten dt
NameComponent nc = new NameComponent("Hello", "");
//tao duong dan de luu ten dt
NameComponent path[] = {nc};
//dang ky dt helloRef ten la "Hello"
ncRef.rebind(path, helloRef);
// tao dt Object
java.lang.Object sync = new java.lang.Object();
//vong lap vo han cho nhan yeu cau tu may khach
synchronized(sync){
    sync.wait();
}
} catch(Exception e) {
    System.err.println("ERROR: " + e);
    e.printStackTrace(System.out);
}
}
}

```

## **B6: Viết ct client triệu gọi phương thức của đối tượng CORBA**

```

package HelloCorba;
import org.omg.CosNaming.*; // HelloClient se dung dich vu ql ten
import org.omg.CORBA.*;    // tat ca ung dung CORBA can lop nay
public class HelloClient
{
    public static void main(String args[])
    {
        try{
            // khoi dong trinh moi gioi ORB

```



```

ORB orb = ORB.init(args, null);
// lay tham chieu den dich vu quan ly ten (tnameserv.exe)
org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
//chuyen dt objRef la dt corba tong quat sang dt kieu NamingContext (ep kieu)
//dt nay dung de goi dich vu ql ten
NamingContext ncRef = NamingContextHelper.narrow(objRef);
// tao ten dt
NameComponent nc = new NameComponent("Hello", "");
//tao duong dan de luu ten dt
NameComponent path[] = {nc};
//dung duong dan de lay tham chieu den dt tren server
Hello helloRef = HelloHelper.narrow(ncRef.resolve(path));
String hello = helloRef.sayHello();
System.out.println(hello);
} catch(Exception e) {
    System.out.println("ERROR : " + e);
    e.printStackTrace(System.out);
}
}
}

```

### **B7: Thử nghiệm ct:**

- Dịch tat ca \*.java sang \*.class : click chột phải trên tên gói corbaHello, chọn make
- Nạp trình quản lý tên đối tượng (tnameserv.exe): có thể thực thi ct này ở bất cứ vị trí nào.
- Chạy ct server/ct client

## 1. Viết ứng dụng CORBA dùng ORB VisiBroker (tích hợp trong Jbuilder)

- Thiết lập các tùy chọn cho Jbuilder để viết các ứng dụng CORBA (chỉ cần làm 1 lần)

Tools/Enterprise Setup/Corba, chọn "Configuration" là VisiBroker, chọn ba check box. Chọn Edit, nhập vào hộp "Path for ORB tool": D:/Borland/AppServer/bin (đường dẫn tới osagent.exe). Thêm thư viện Borland/AppServer/lib/vbjorb.jar

- Chọn Project/ Project Properties/Build/IDL, chọn "IDL Compiler" là VisiBroker

- *B1: Tạo project, nhớ chọn thư viện VisiBroker*
- *B2: Định nghĩa giao tiếp bằng IDL: File/New/Sample IDL*
- *B3: Sinh ra file Stub và Servant (Skeleton): click chuột phải trên file idl, chọn make*
- *B4: Cài đặt client:*
  - *Tạo ứng dụng: file/new/application*
  - *Tạo giao diện giao tiếp với server: Wizards/Use CORBA Interface.*
  - *Đăng ký giao diện với ORB: Thêm dt OrbConnect vào ứng dụng, ràng buộc giao diện với ORB*
- *B6: Cài đặt server: File/New/corba/ CORBA Server Application*
- *B7: Viết cài đặt cho dt server*
- *B8: Chạy thử nghiệm:*
  - *Thực thi Smart Agent*
  - *Thực thi server*
  - *Thực thi client*

## BÀI TẬP

**Dùng công nghệ RMI và CORBA làm các bài tập sau:**

1. Giải phương thức bậc nhất, cho client hỏi nhiều lần
2. Viết ct hỏi thời tiết, cho client hỏi nhiều lần. Biết rằng thông tin thời tiết lưu trong tập tin thoitiet.txt có dạng sau  

TP	Nhiệt Độ
HCM	25-30
HN	20-25
vv...	
3. Tương tự bài 2 nhưng thông tin thời tiết lưu trong table thoitiet, dùng hệ quản trị CSDL SQL
4. Viết ct hỏi điểm, cho client hỏi nhiều lần, thông tin về điểm lưu trong table bangdiem, dùng hệ quản trị CSDL SQL. Biết rằng  $dtb = (dlt + dth) / 2$

masv	dlt	dth	dtb
1	10	6	8
2	6	8	7
vv...			

- Hết -

## **ON TAP THI**

### **I. APPLET**

- Goi tham so cho applet
- su dung thread trong applet
- Su dung giao dien trong applet

### **II. LAP TRINH CLIENT/SERVER**

Dang 1: dung giao thuc TCP

- Nhieu client y/c dong thoi (dung thread)
- 1 client co the y/c nhieu lan
- du lieu truy xuat o dang file van ban hoac csdl
- Co su dung giao dien

Dang 2: dung giao thuc UDP

- Cac y/c tuong tu dang 1

---

### **BAI TAP ON**

#### **I. Applet**

1. viet applet cong hai so, co su dung giao dien, co goi tham so a, b.

a=	2X
b=	3
a+b=	5
OK	CLEAR
hien thong bao	Du lieu sai

(xmoi,ymoi),(xmoi+cd/4,ymoi+cr/4)  
(xmoi+3cd/4,ymoi+cr/4,xmoi+cd,ymoi)

2. Viet applet cho mot dia bay, bay ngau nhien, khi qua trang khac phai ngung bay, kich thuoc dia bay duoc goi nhu tham so.

#### **II. CLIENT/SERVER**

1. Quan ly diem, dung TCP, co giao dien, co CSDL SQL.

CSDL QLD.MDF, TABLE BANGDIEM(masv varchar(10), diemthi int)

- ct chinh

CHUONG TRINH QUAN LY DIEM				
NHAP DIEM	XEM DIEM	SUA DIEM	XOA SV	THONG KE

- ct nhap

MASV	1
DIEM THI	9
OK	CLEAR
THONG BAO	DU LIEU SAI/TRUNG KHOA/DA NHAP

2) Lam lai bai 1 bang giao thuc UDP