LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

BÀI GIẢNG LƯU HÀNH NỘI BỘ

KHOA CÔNG NGHỆ ĐIỆN TỬ
TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP THÀNH PHỐ HỒ CHÍ MINH

BỘ MÔN ĐIỆN TỬ MÁY TÍNH

MỤC LỤC

BÀI 0. GIỚI THIỆU

I. LỊCH SỬ HÌNH THÀNH VÀ PHÁT TRIỂN CỦA JAVA:

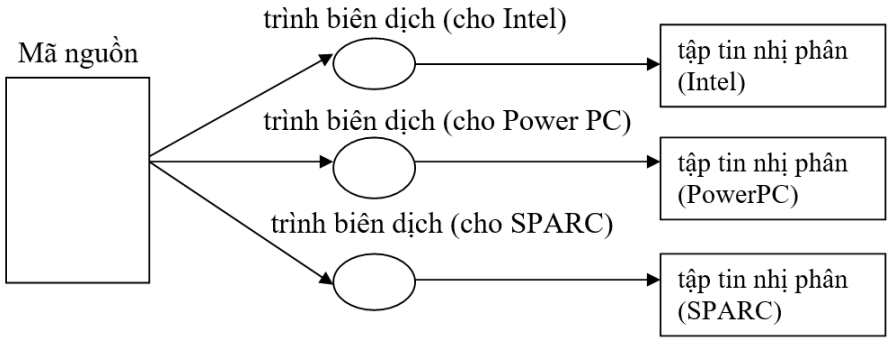
Sun Microsystem phát triển ngôn ngữ Java vào năm 1991 như một phần đề tài nghiên cứu phần mềm do nhóm Green (Green Team) đứng đầu là James Gosling, áp dụng cho việc sản xuất các thiết bị điện tử gia dụng (lập trình cho vi xử lýdùng trong các thiết bị điện tử khác nhau như TV, máy hát, lò nướng bánh, đồ chơi,…). Mục đích phải đạt được của Java khi đó là nhanh, gọn, hiệu xuất cao và dễ đem ra áp dụng cho môi trường đa dạng các sản phẩm. Những yêu cầu đó cũng giống như những yêu cầu đặt ra mà Java phải có trong việc phân phối các chương trình ứng dụng trên World Wide Web và cũng là mục đích bao trùm của ngôn ngữ lập trình cho phép phát triển các ứng dụng chạy trên các nền khác nhau một cách dễ dàng. Hiện nay các công ty phần mềm khác và Sun đang gấp rút phát triển các công cụ hỗ trợ cho việc phát triển các ứng dụng Java.

II. ĐẶC ĐIỂM:

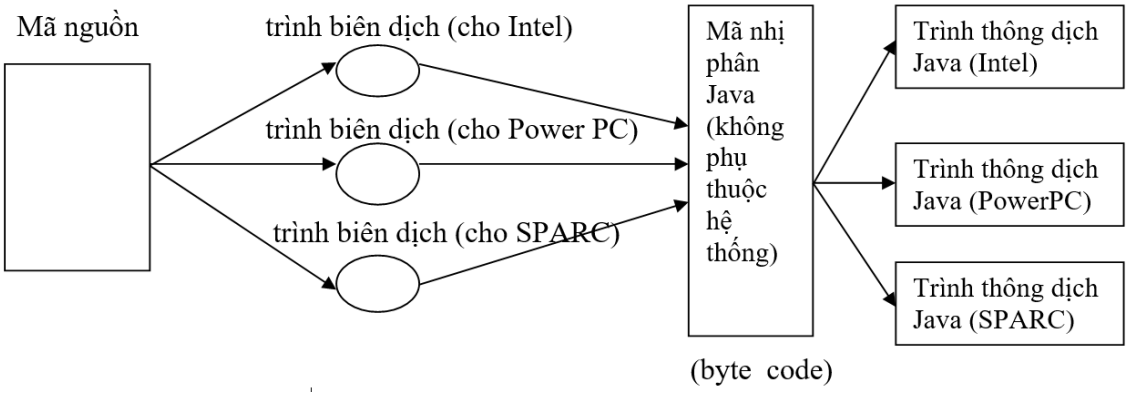
1. Máy ảo Java:

Để đạt được tính khả chuyển cao, nhóm Green đã đưa ra ý tưởng "máy ảo Java" (Java Virtual Machine). Chương trình Java được biên dịch thành ngôn ngữ máy của máy ảo Java. Chương trình như vậy có thể chạy trên mọi loại máy miễn là máy đã được thiết lập máy ảo Java. Máy ảo Java tạo thành một lớp ngăn cách giữa trình ứng dụng và hệ điều hành.

Trình ứng dụng Java chỉ chạy được sau 2 giai đoạn: biên dịch (thành ngôn ngữ máy ảo) và thông dịch (thành ngôn ngữ máy đặc thù). Trình biên dịch Java của các máy dùng các vi xử lý khác nhau đều cho ra cùng một kết quả như nhau gọi là "byte code" (ngôn ngữ máy ảo Java). Byte code chạy được trên mọi loại máy dùng vi xử lý khác nhau miễn là trong máy đó đã thiết lập máy ảo Java. Như vậy, trình ứng dụng Java chỉ cần viết một lần là có thể chạy được trên mọi loại máy, không phụ thuộc vi xử lý (Pentium, Power PC, Sparc,…) cũng như hệ điều hành (Windows, Linux, OS/2, Macintosh,…). Hình vẽ dưới đây cho thấy sự khác biệt giữa cách thức biên dịch truyền thống với 2 giai đoạn biên dịch và thông dịch của Java.



Hình 0.1. Ngôn ngữ biên dịch thông thường.



Hình 0.2. Ngôn ngữ java.

2. Các ứng dụng của Java:

Các ứng dụng của Java được chia làm 2 loại: applet và application.

a. Applet: ứng dụng "nhúng" hay ứng dụng "ký sinh", đó là chương trình hoạt động được nhờ "bám vào" trang web, được download từ Internet xuống và thi hành bởi trình duyệt web có tính năng Java trên máy người sử dụng.

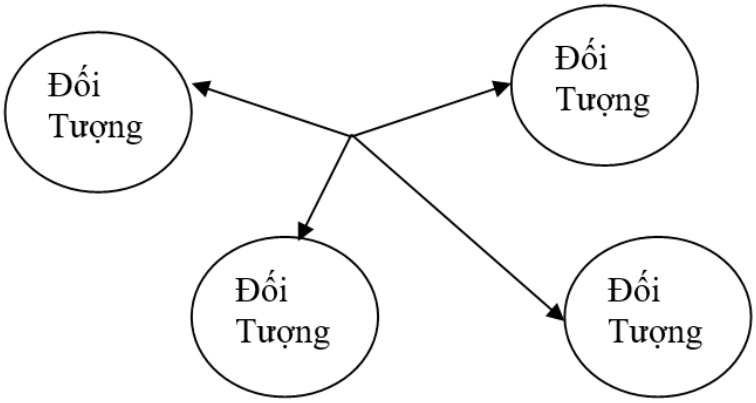
*Trình duyệt web có tính năng Java (Java-enable browser): là trình duyệt có chứa trình thông dịch Java bên trong. HotJava của Sun là trình duyệt web có tính năng Java đầu tiên. 2 trình duyệt web thông dụng nhất hiện nay là Netscape và Internet Explorer đều là trình duyệt web có tính năng Java.

b. Application: ứng dụng đơn hay ứng dụng độc lập được viết như những chương trình ứng dụng thông thường bằng ngôn ngữ Java. Những ứng dụng đơn không cần phải có trình duyệt web để chạy, Java có thể tạo ra hầu hết các loại ứng dụng mà chúng ta tạo được từ những ngôn ngữ lập trình khác. Bản thân HotJava cũng chính là một ứng dụng của Java.

3. Ngôn ngữ lập trình hướng đối tượng (Object Oriented Programming):

Đối tượng là sự kết hợp dữ liệu và thao tác trên dữ liệu đó thành một thể thống nhất. Bên trong mỗi đối tượng có chứa dữ liệu thể hiện tình trạng hay thuộc tính (property) của nó. Mỗi đối tượng được trang bị những hành vi hay phương thức (method) để thực hiện một số nhiệm vụ như thông báo hoặc thay đổi thuộc tính của chính nó. Mỗi đối tượng phát sinh từ một lớp (class). Lớp là khuôn mẫu tạo ra những đối tượng cụ thể. Trong hệ thống hướng đối tượng, các đối tượng làm việc với nhau bằng cách gởi thông điệp cho nhau. Đối tượng này có thể yêu cầu đối tượng kia làm việc gì đó. Dữ liệu luôn nằm trong các đối tượng, muốn tác động lên dữ liệu phải thông qua đối tượng chứa dữ liệu đó, vì Java là ngôn ngữ lập trình hướng đối tượng thuần túy, mọi thứ đều là đối tượng, không có biến và hàm nào nằm ngoài đối tượng.

Vị trí cụ thể của đối tượng không quan trọng. Điều này tỏ ra đặc biệt thích hợp cho môi trường mạng máy tính. Phần mềm mạng có thể gồm nhiều đối tượng nằm phân tán, có đối tượng nằm ngay trên máy của người sử dụng, có đối tượng nằm trên máy chủ ở xa. Khi chạy một ứng dụng không nhất thiết phải nạp mọi đối tượng của ứng dụng xuống máy tính đang chạy ứng dụng trong một lần. Ứng dụng chỉ gọi những đối tượng nào thự sự cần thiết cho thao tác đang thực hiện, điều này phù hợp với nguyên tắc hoạt động của mạng web.



Hình 0.3. Lập trình hướng đối tượng.

4. Tính khả chuyển (portability):

Yêu cầu về tính khả chuyển là phải làm sao cho phần mềm đáp ứng được sự đa dạng về chủng loại máy tính và hệ điều hành. Thông thường đối với một ngôn ngữ lập trình, khi chuyển qua hệ điều hành khác, mã nguồn cần được hiệu chỉnh ít nhiều trước khi biên dịch, cuối cùng sẽ có nhiều phiên bản khác nhau của cùng một sản phẩm dành cho các hệ điều hành khác nhau. Với Java, việc biên dịch mã nguồn dựa vào qui ước về máy ảo Java, mã nguồn Java được biên dịch thành một dạng đặc biệt gọi là mã byte (bytecode), được tạo thành bởi từng byte một, thuận tiện cho việc truyền qua mạng như một văn bản ASCII (nhưng người sử dụng không đọc được, chỉ có trình thông dịch mới đọc được). Nếu đặt chương trình Java dạng mã byte trên một máy chủ, nó có thể truyền qua mạng nhanh chóng và chạy trên mọi loại máy khách miễn làtại đấy có trình thông dịch Java. Điều này đặc biệt quan trọng đối với mạng web.

5. Hiệu năng cao:

Hiệu năng của chương trình Java được cải thiện đáng kể nhờ khả năng xử lý đa tuyến đoạn hay đa luồng (multithreading). Khi tương tác với người sử dụng, chương trình đơn tuyến đoạn thường lãng phí một khoảng thời gian không nhỏ để chờ đợi người sử dụng lựa chọn trước khi quyết định cần phải làm gì tiếp theo. Java cho phép tạo ra nhiều tuyến đoạn đồng hành để giải quyết nhiều việc cùng một lúc. Trong thời gian chờ đợi người sử dụng quyết định lựa chọn khả năng nào đó, bề ngoài máy tính dường như không làm việc nhưng bên trong vẫn liên tục nạp vào kết quả của những khả năng sẽ lựa chọn hoặc những khả năng tiếp theo của chương trình trên những tuyến đoạn khác, vì khâu giao tiếp với người sử dụng chỉ là một tuyến đoạn của chương trình mà thôi nhờ đó khắc phục được phần nào sự năng nề khó tránh của các ứng dụng mạng.

6. Môi trường xử lý phân tán (distributed computing):

Đặc tính này có được nhờ sự phát triển nhanh chóng của mạng máy tính. Sự phân tán phần mềm trên nhiều máy tính cho phép tạo ra khả năng xử lý mạnh mẽ nhưng phải làm sao cho đồng bộ, an toàn đồng thời giảm thiểu lưu lượng truyền thông qua mạng. Java có được nhiều yếu tố thích hợp cho việc tạo ra môi trường xử lý phân tán trên mạng máy tính, các mã thư viện liên kết với mã byte của chương trình Java được phân giải vào lúc chạy và chỉ nạp về khi cần nên chương trình Java được nạp dần một cách linh hoạt về máy khách trong thời gian chạy, mã byte của chương trình Java không chứa mã thư viện nên thường gọn nhẹ, mất ít thời gian nạp về máy khách, do đó không gây tắc nghẽn cho mạng.

BÀI 1: NGỮ PHÁP CĂN BẢN CỦA JAVA

Giới thiệu các thành phần lập trình cơ bản như: biến, hằng, loại dữ liệu, toán tử, cấu trúc điều khiển, mảng, hàm trong Java.

1. BIẾN:

Là một vùng được định danh trên bộ nhớ để chứa dữ liệu. Trước khi dùng biến ta phải định nghĩa tức là cho biết tên và kiểu của biến.

Ví dụ: int x;

String name;

Biến x thuộc kiểu nguyên (int), biến name thuộc kiểu chuỗi (String). Có thể định nghĩa biến ngay tại chỗ cần sử dụng biến ấy lần đầu hay đặt ngay đầu khối, đầu hàm hoặc đầu lớp.

Có thể định nghĩa nhiều biến thuộc cùng kiểu một lúc.

Ví dụ: int x, y, z;

String firstname, lastname;

Có thể gán giá trị ban đầu cho biến (tránh trường hợp gặp phải các giá trị "rác" ngẫu nhiên).

Ví dụ: int x;

x = 4;

hoặc int x = 4;

Được phép khởi tạo đồng loạt nhiều biến.

Ví dụ: int x=4, y=5, z=6;

Chú ý: x, y, z = 6 (chỉ có biến z được khởi tạo)

1. Tên biến: bắt đầu là một ký tự, không được bắt đầu bằng số chỉ được dùng số sau ký tự đầu tiên, không được dùng các ký tự đặc biệt như: *, @, %,…, không chứa các toán tử (Vd: +, -, *, /). Phân biệt chữ hoa và chữ thường (Vd: name, NAME, Name là những biến khác nhau).

2. Kiểu biến:

Các kiểu dữ liệu số

Tên	Khoảng biến thiên	Kích thước
Byte	-27 -> 27 – 1	8 bit
Short	-215 -> 215 - 1	16 bit
Int	-231 -> 231 - 1	32 bit
Long	-263 -> 263 - 1	64 bit
Float		32 bit (floating point)
Double		64 bit (floating point)
Boolean	True hoặc false	

2. HẰNG:

Có tên biểu thị cho dữ liệu không bao giờ thay đổi.

Cú pháp khai báo hằng: static final kiểu hằng tên hằng = giá trị gán;

Ví dụ: static final double PI=3.14159; (khai báo hằng số PI)

Hằng chuỗi: diễn đạt một chuỗi ký tự cụ thể thuộc kiểu String. Hằng chuỗi được ghi trong dấu nháy kép.

Vd: String s = "em con nho hay em da quen \n" (\n là ký tự xuống dòng)

Hằng ký tự :Gần giống như hằng chuỗi là hằng ký tự. Hằng ký tự được ghi trong dấu nháy đơn

Vd: char chu = 'a' ; char so = '3' ;

a là ký tự a, ' 3 ' là ký tự 3, không phải số 3.

** Nhập/ Xuất dữ liệu:*

Đối với Java, muốn nhập/xuất dữ liệu từ bàn phím, phải đưa thêm file MyInput.class cùng thư mục với chương trình đang viết. Tập tin MyInput.java phục vụ cho 2 kiểu dữ liệu int và float. Muốn phục vụ cho các kiểu khác, chỉ cần sửa kiểu int lại kiểu muốn sử dụng(với các kiểu nguyên: byte, short, long), kiểu float lại kiểu muốn sử dụng(với các kiểu thực: double) hoặc viết tiếp thêm đoạn phục vụ cho kiểu tương ứng. Khi sửa nhớ chú ý chữ hoa và chữ thường tại các vị trí trong tập tin này.Tập tin MyInput.java có dạng như sau:


```
import java.io.*;
import java.util.*;
public class MyInput
{
	static private StringTokenizer stok;
	static private BufferedReader br
	=new BufferedReader(new InputStreamReader(System.in),1);
	public static int readInt()
	{
		int i=0;
		try
	{
		String str=br.readLine();
		StringTokenizer stok=new StringTokenizer(str);
		i=new Integer(stok.nextToken()).intValue();
	}
	catch(IOException ex)
	{
		System.out.println(ex);
	}
	return i;
}
```

```
public static float readFloat()
{
	float d=0;
	try
	{
		String str=br.readLine();
		stok=new StringTokenizer(str);
		d=new Float(stok.nextToken()).floatValue();
	}
		catch(IOException ex)
	{
	System.out.println(ex);
	}
		return d;
	}
}
```

3. TOÁN TỬ:

Các toán tử số học

Toán tử	Ý nghĩa	Ví dụ
+	Cộng	3 + 4
-	Trừ	3 - 4
*	Nhân	3 * 4
/	Chia	10 / 2
%	Phần dư (modulo)	31 % 9

Giải thích thêm về toán tử modulo: dùng để lấy phần dư của phép chia nguyên trong trường hợp 2 giá trị thuộc kiểu nguyên chia cho nhau, kết quả cũng thuộc kiểu nguyên (chỉ lấy phần nguyên, không lấy phần thập phân).

Ví dụ: 31 / 9 kết quả là 3, dư 4. Vậy 31 % 9 là 4.

Ví dụ: Nhập vào các giá trị: x,y (int), a,b (float). In ra màn hình giá trị và kết quả của cá phép tính: x+y, x-y, x*y, x/y, x%y, a/b.

```
public class tinhtoan
{
	public static void main (String arg[])
	{
		int x;
		int y;
```

```
        float a;
        float b;

        System.out.print("nhap x:");
        x=MyInput.readInt();
        System.out.print("nhap y:");
        y=MyInput.readInt();

        System.out.print("nhap a:");
        a=MyInput.readFloat();
        System.out.print("nhap b:");
        b=MyInput.readFloat();

        System.out.println("gia tri cua x: "+x+"\n gia tri cua y: "+y);

        System.out.println("x+y = "+(x+y));
        System.out.println("x-y = "+(x-y));
        System.out.println("x * y = "+(x*y));
        System.out.println("x/y = "+(x/y));
        System.out.println("x%y = "+(x%y));
        System.out.println("a co gia tri la "+a+", b co gia tri la "+b);
        System.out.println("a/b = "+(a/b));
    }
}
```

BT. Nhập x,y với kiểu nguyên khác và a,b với kiểu thực khác.

Toán tử gán:

Vd: x=10;

x=y=z=10;

x=x+2;

Các toán tử gán rút gọn

Biểu thức	Ý nghĩa
x+=y	x=x+y
x-=y	x=x-y
x*=y	x=x*y
x/=y	x=x/y
x%=y	x=x%y

Trường hợp tăng một đơn vị:

y=x++ trị của x được gán cho y trước rồi tăng một đơn vị.

y=++x trị của x được tăng một đơn vị trước rồi mới gán cho y tăng một đơn vị.

Tương tự cho 2 trường hợp y=x-- và y=--x.

BT. Viết 1 chương trình thử nghiệm cho trường hợp tăng, giảm một đơn vị.

Các toán tử so sánh

Toán tử	Ý nghĩa	Ví dụ
==	Bằng	X==y
!=	Khác	X != y
<	Nhỏ hơn	X<3
>	Lớn hơn	x>3
<=	Nhỏ hơn hoặc bằng	X<=3
>=	Lớn hơn hoặc bằng	x>=3

Các toán tử logic

Toán tử	Ý nghĩa
&&	And
||	or

4. CÁC CẤU TRÚC ĐIỀU KHIỂN:

1. Câu lệnh if:

Dạng 1: if đơn

```
if (biểu thức)
                Khối lệnh ;
```

Dạng 2: if…else

```
if (biểu thức)
                Khối lệnh 1;
        else
                Khối lệnh 2;
```

Dạng 3: else if

```
        if (biểu thức 1)
                khối lệnh 1;
        else if (biểu thức 2)
                khối lệnh 2;

        …
        else if  (biểu thức n-1)
```

```
                khối lệnh n-1;
        else
                khối lệnh n;
```

Ngoài ra còn có thể sử dụng if lồng.

Vd:

```
        if (biểu thức 1)
          if (biểu thức 1a)
                Khối lệnh 1a;
           else
                Khối lệnh 1b;
        else
           if (biểu thức 2)
                 Khối lệnh 2;
            else
                Khối lệnh 3;
```

…

Vd: Nhập a,b. Xuất kết quả cho biết số nào lớn hơn.

Dạng 1:

```
if (a>b)
{
        System.out.println("a lon hon b");
}
System.out.println("a nho hon hoac bang b");
```

Hoặc:

Dạng 2:

```
    if (a>b)
        {
                System.out.println("a lon hon b");
        else
        System.out.println("a nho hon hoac bang b");
        }
```

Hoặc

Dạng 3:

```
        if (a>b)
    System.out.println("a lon hon b");
    else if (a==b)
                System.out.println("a bang b");
    else
                System.out.println("a nho hon b");
```

BT. Nhập 3 số a, b, c. Cho biết số nào lớn nhất, số nào nhỏ nhất.

2. Câu lệnh switch…case:

```
    Switch (biểu thức)
    {
        case n1:
            khối lệnh;
            break;
case n2:
            khối lệnh;
            break;
        …

        case nk:
            khối lệnh;
            break;
        [default:
            khối lệnh;]

}
```

n1, n2, …,nk phải có chung loại dữ liệu với (biểu thức).

Lệnh break để thoát khỏi vòng switch…case.

Phần [default:

khối lệnh;] có hay không cũng được.

Vd: Chọn thực đơn muốn dùng (1,2,3). Xuất kết quả ra màn hình ứng với từng trường hợp.

```
public class thucdon
{
    public static void main (String arg[])
    {
    int x;
```

```
        System.out.print("Ban hay cho biet thuc don muon dung(1,2,3): ");
        x=MyInput.readInt();
        switch (x)
                {
                case 1:
                    System.out.print("thuc don 1: Com \n Chuc ban ngon mieng :-)");
                    break;
                case 2:
                    System.out.print("thuc don 2: Pho \n Chuc ban ngon mieng :-)");
                    break;
                case 3:
                    System.out.print("thuc don 3: Chao \n Chuc ban ngon mieng :-)");
                    break;
                default:
                    System.out.print("Xin loi! Thuc don nay khong co ");
                }
        }
}
```

5. VÒNG LẶP:

1. Vòng lặp for:

Vd: Viết ra các số từ 1 đến 10.

```
        System.out.print("viet cac so tu 1 den 10:");
        for (int i; i<=10; i++)
        {
                System.out.print(i);
        }
```

BT:

1. Viết ra các số từ 2 đến 20 (2,4,6,…).
2. Tính tổng các số từ 1 đến 10.

2. Vòng lặp while:

```
                while (biểu thức điều kiện)
                        {
                                các câu lệnh;
                        }
```

biểu thức điều kiện cho kết quả true: còn thực hiện, kết quả false: thoát khỏi vòng lặp.

Vd và BT giống như vòng lặp for.

3. Vòng lặp do:

```
do
{
    các câu lệnh;
}
while (biểu thức điều kiện)
```

biểu thức điều kiện cho kết quả true: còn thực hiện, kết quả false: thoát khỏi vòng lặp.

Vd và BT giống như vòng lặp for.

4. Lệnh break và continue dùng trong vòng lặp:

-break: kết thúc ngay vòng lặp trong cùng chứa nó.

-continue: Kết thúc lần lặp vòng hiện hành. Điều khiển chương trình chuyển sang lần kế tiếp của vòng lặp.

Ví dụ:

```
public class brecont
{
    // Main method
    public static void main (String[]args)
    {
        int sum=0;
        int item=0;
        do
        {
            item++;
            if (item==2) break;
            sum+=item;
        }
        while(item<5);
        System.out.println("The Sum is "+sum);
        System.out.println("The item  is "+item);
    }
}
```

Ghi chú: với trường hợp break - item=2, continue - item=5

6. MẢNG (Array):

Là một tập hợp nhiều phần tử có cùng một kiểu giá trị và có chung một tên dùng để biểu diễn một dãy số hay một bảng số. Mỗi phần tử mảng biểu diễn được một giá trị. Kiểu của mảng cũng là tất cả các kiểu của biến (int, float, double,…).

1. Khai báo và khởi tạo mảng:

Ví dụ:

Khai báo mảng: int bangso[];

String bangchu[];

Hoặc: int[] bangso;

String[] bangchu;

Khởi tạo mảng: bangso = {1, 2, 12, 45};

bangchu = {"chao", "cac", "ban"};

Hoặc dùng toán tử new để khởi tạo:

bangso = new int[30];

bangchu = new String[10];

Hai câu lệnh trên trao cho mảng bangso quản lý 30 phần tử nguyên (giá trị ban đầu là 0) và trao cho mảng bangchu quản lý 10 phần tử thuộc kiểu String. (giá trị ban đầu là null). Có thể gom 2 công đoạn khai báo và khởi tạo trong một câu lệnh:

int bangso[] = new int[30];

String bangchu[] = new String[10];

Hoặc: int[] bangso = new int[30];

String[] bangchu = new String[10];

Ví dụ:

```
public class mangtho
{
    public static void main (String arg[])
    {
        int[] bangso;
        String[]  bangchu;
        bangso = new int[3];
        bangchu = new String [8];
        bangso[0] = 10;
        bangso[1] = 20;
        bangchu[0] = "Thoi gian nhu la gio";
        bangchu[1] = "Mua di cung thang nam";
        bangchu[2] = "Tuoi theo mua di mai";
        bangchu[3] = "Chi co anh va em";
        bangchu[4] = "Cung tinh yeu o lai";
        for (int i=0; i<3; i++)
           System.out.println(bangso[i]);
        for (int i=0; i<8; i++)
```

```
            System.out.println(bangchu[i]);
        }
}
```

2. Mảng nhiều chiều:

Mảng nhiều chiều là mảng của mảng. Câu lệnh sau đây định nghĩa và khởi tạo một mảng có 3 phần tử, trong đó mỗi phần tử là một mảng nguyên.

```
int j[][] = new int[3][];
```

Sau đó định nghĩa và khởi tạo riêng từng mảng phần tử.

```
j[0] = new int [4];
j[1] = new int [4];
j[2] = new int [4];
```

Kết quả là một mảng 2 chiều có kích thước 3x4 (có thể hình dung như một bảng 3 hàng, 4 cột).

Có thể gộp 2 công đoạn trên lại trong một câu lệnh như sau:

```
int j[][] = new int [3][4];
```

Gán trị cho một phần tử: `j[1][3] = 10;`

Có thể định nghĩa mảng trước, sau đó qui định kích thước thông qua các biến nguyên.

```
int j[][];
int m = 3;
int n = 4;
j = new int [m][n];
```

3. Các ví dụ khai thác ứng dụng mảng:

Ví dụ 1: Viết chương trình đọc điểm của học viên (int) từ bàn phím, tìm điểm cao nhất, rồi xếp hạng dựa trên thứ tự sau:

Hạng A nếu điểm số >= điểm cao nhất - 10.

Hạng B nếu điểm số >= điểm cao nhất - 20.

Hạng C nếu điểm số >= điểm cao nhất - 30.

Hạng D nếu điểm số >= điểm cao nhất - 40.

Các trường hợp còn lại xếp vào hạng F.

Chương trình nhắc người dùng gõ tổng số học viên và toàn bộ điểm số. Cuối cùng, chương trình hiển thị điểm số, thứ hạng của học viên.

```
public class xeploai
{
    public static void main (String arg[])
    {
        int sosinhvien;
        int best=0;
        int[] diem;
```

```
        char[] h;
        System.out.print("Cho biet tong so sinh vien: ");
        sosinhvien=MyInput.readInt();
        diem = new int[sosinhvien];
        h = new char[sosinhvien];
        for (int i=0; i<diem.length; i++)
          {
              System.out.print("Cho biet diem cua sinh vien thu " +(i+1)+" :" );
              diem[i] = MyInput.readInt();
              if (diem[i]>best)
                 best=diem[i];
          }
        //xep hang
        for (int i=0; i<diem.length; i++)
          {
              if (diem[i]>=best-10)
                 h[i]='A';
              else if (diem[i]>=best-20)
                 h[i] = 'B';
              else if (diem[i]>=best-30)
                 h[i] = 'C';
              else if (diem[i]>=best-40)
                 h[i] = 'D';
              else
                 h[i] = 'E';
           }
        System.out.println("          KET QUA        " );
        System.out.println("So thu tu     Ten       Diem    Loai " );
        for (int i=0; i<diem.length; i++)
          {
          System.out.println("    "+(i+1)+"        HV "+(i+1)+"      "+diem[i]+"      "+h[i]);
          }
    }
}
```

Ví dụ 2: Sắp Xếp.

Sắp xếp các phần tử trong mảng theo thứ tự từ nhỏ đến lớn.

```
public class xephang
{
	// ham chinh
	public static void main (String[] args)
	{
		double[] myList = {5.0, 6.4, 1.9, 2.9, 3.5, 3.6};
		System.out.println("Chuoi chua sap xep:");
		printList(myList);
		sapxep(myList);
		System.out.println("Chuoi da sap xep:");
		printList(myList);
	}
	// ham in chuoi
	static void printList(double list[])
	{
		for (int i=0; i<list.length; i++)
		  System.out.println(list[i]);
	}
	//ham sap xep
	static void sapxep(double list[])
	{
		double tam;
		int i,j;
		for (i=0; i<list.length; i++)
		{
			for (j=i+1; j<list.length; j++)
			{
			  if (list[i] > list[j])
			  {
				tam = list[i];
				list[i]= list[j];
				list[j] = tam;
			  }
		  }
	}
	}
```

```
}
```

BT: Xếp hạng từ cao xuống thấp theo điểm các học viên trong ví dụ 1. (xem sau)

Ví dụ 3: Sao chép mảng.

```
public class chepmang
{
	public static void main(String args[])
	{
		int [] list = {0, 1, 2, 3, 4, 5};
		int [] newList = new int[list.length];
		newList = list;
		System.out.println("Truoc khi thay doi:");
		printList("list is: ",list);
		printList("newList is: ",newList);
		//thay doi gia tri cua list []
		for (int i = 0; i<list.length; i++)
		  list[i] = 0;
		System.out.println("Sau khi thay doi:");
		printList("list is: ",list);
		printList("newList is: ",newList);
	}
	// ham printList
	public static void printList(String s, int[] list)
	{
		System.out.print(s+" ");
		for (int i=0; i<list.length; i++)
		  System.out.print(list[i]+" ");
		System.out.print("\n ");
	}
}
```

BÀI TẬP

Bài tập mẫu:

Viết chương trình giải phương trình bậc 2: $ax^2 + bx + c = 0$.

```
import java.util.Scanner;

/**
 * Giải phương trình bậc 2
 *
 *
 */
public class PhuongTrinhBac2 {
    private static Scanner scanner = new Scanner(System.in);
    /**
     * main
     *
     * @param args
     */
    public static void main(String[] args) {
        System.out.print("Nhập a = ");
        float a = scanner.nextFloat();
        System.out.print("Nhập b = ");
        float b = scanner.nextFloat();
        System.out.print("Nhập c = ");
        float c = scanner.nextFloat();
        giaiPTBac2(a, b, c);
    }

    public static void giaiPTBac2(float a, float b, float c) {
        // kiểm tra các hệ số
        if (a == 0) {
            if (b == 0) {
                System.out.println("Phương trình vô nghiệm!");
            } else {
                System.out.println("Phương trình có một nghiệm: "
                        + "x = " + (-c / b));
```

```
            }
            return;
        }
        // tính delta
        float delta = b*b - 4*a*c;
        float x1;
        float x2;
        // tính nghiệm
        if (delta > 0) {
            x1 = (float) ((-b + Math.sqrt(delta)) / (2*a));
            x2 = (float) ((-b - Math.sqrt(delta)) / (2*a));
            System.out.println("Phương trình có 2 nghiệm là: "
                    + "x1 = " + x1 + " và x2 = " + x2);
        } else if (delta == 0) {
            x1 = (-b / (2 * a));
            System.out.println("Phương trình có nghiệm kép: "
                    + "x1 = x2 = " + x1);
        } else {
            System.out.println("Phương trình vô nghiệm!");
        }
    }
}
```


Bài tập 1:

Viết chương trình Java mà khi chạy, màn hình console sẽ cho phép ta nhập vào một số nguyên, in ra màn hình "Đây là số nguyên dương" nếu số vừa nhập vào là một số lớn hơn hoặc bằng 0, ngược lại in ra "Đây là số nguyên âm".

Bài tập 2:

Viết chương trình cho phép nhập vào 3 số thực. Chương trình này sẽ kiểm tra 3 số này có phải là 3 cạnh của một tam giác hay không.

Bài tập 3:

Viết chương trình cho phép nhập vào một số nguyên n (n < 1000). In ra tất cả số nguyên tố trong khoảng từ 0 - n.

Bài tập 4:

Viết chương trình cho phép nhập vào một số nguyên dương n, tính tổng tất cả số chẵn trong khoảng từ 0 - n.

Bài tập 5:

Viết chương trình để nhập nhập một số nguyên, tìm kết quả phép nhân của số đó với các số từ 1 - 20 , sau đó in kết quả ra màn hình.

Bài tập 6:

Viết chương trình cho phép nhập vào n, sau đó nhập vào n phần tử số nguyên. Cuối cùng, chương trình sẽ xuất ra giá trị trung bình của mảng này.

Bài tập 7:

Viết chương trình cho phép nhập vào n, sau đó nhập vào n phần tử số nguyên dương. Cuối cùng, chương trình sẽ xuất ra phần tử có giá trị lớn nhất.

Bài tập 8:

Viết chương trình Java nhập một mảng số nguyên a0, a1, a2, …, an-1. Hãy sắp xếp mảng theo thứ tự tăng dần.

Bài tập 9:

Viết chương trình Java nhập một mảng số nguyên a0, a1, a2, …, an-1. Hãy sắp xếp mảng theo thứ tự tăng dần, sau đó chèn phần tử k vào mà vẫn đảm bảo mảng là tăng dần.

Bài tập 10:

Viết chương trình Java nhập một mảng số nguyên a0, a1, a2, …, an-1. Hãy sắp xếp mảng theo thứ tự tăng dần, sau đó chèn phần tử k vào mà vẫn đảm bảo mảng là tăng dần.

BÀI 2. LỚP VÀ ĐỐI TƯỢNG

1. Định nghĩa:

Đối tượng (Object) là thuật ngữ có định nghĩa rất rộng. Sinh viên, nhân viên, bàn ghế, hình tròn,…đều được xem là đối tượng. Đối tượng được xác định thông qua các thuộc tính và hành vi. Thuộc tính là các trường dữ liệu (data field), còn hành vi là cá phương thức (method).

Ví dụ: đối tượng Circle có trường dữ liệu là bankinh. Một hành vi của hình tròn là có thể tính diện tích.

Lớp (class) là cấu trúc định nghĩa đối tượng. Trong lớp Java, dữ liệu mô tả thuộc tính, còn phương thức định nghĩa hành vi. Lớp dành cho đối tượng sẽ chứa tập hợp định nghĩa phương thức và dữ liệu.

Ví dụ: Lớp dành cho hình tròn.

```
Class Circle
{
        double bankinh=1.0;

        double dientich()
        {
                return bankinh* bankinh*3.14159;
        }
}
```


2. Khai báo lớp và đối tượng:

Lớp là khuôn mẫu định rõ dữ liệu và phương thức của đối tượng sẽ là gì. Đối tượng là một phiên bản (instance) của lớp. Có thể tạo nhiều đối tượng. Mối quan hệ giữa lớp và đối tượng tương tự mối quan hệ giữa công thức làm món bánh bao và bánh bao. Có thể làm bao nhiêu chiếc bánh bao tùy ý từ chỉ một công thức.

Để tạo đối tượng, chúng ta phải dùng biến biểu diễn nó (tương tự như khai báo biến cho loại dữ liệu sơ cấp). Cú pháp khai báo đối tượng như sau:

ClassName objectName;

Ví dụ: Circle myCircle;

Việc khai báo một biến sơ cấp, vd: int a (câu lệnh này tạo biến a và cấp một vùng nhớ cho biến a). Tuy nhiên, với biến đối tượng, việc khai báo và tạo dựng là 2 bước riêng biệt. Khai báo đối tượng chỉ đơn giản là phối hợp đối tượng với lớp, biến nó trở thành phiên bản của lớp đó. Khai báo không tạo thành đối tượng. Muốn thực sự tạo myCircle, chúng ta phải dùng toán tử new để yêu cầu máy tính tạo đối tượng cho myCircle và phân phối không gian nhớ cho nó. Cú pháp tạo đối tượng như sau:

objectName= new ClassName();

Ví dụ: myCircle= new Circle();

Có thể kết hợp khai báo và tạo đối tượng với chỉ một câu lệnh:

ClassName objectName= new ClassName();

Ví dụ: Circle myCircle= new Circle();

Sau khi được tạo thành, đối tượng có thể truy cập dữ liệu và phương thức thông qua ký hiệu dấu chấm như sau:

ObjectName.data – tham chiếu dữ liệu của đối tượng.

ObjectName.method – tham chiếu phương thức của đối tượng.

Ví dụ: myCircle.bankinh – cho biết bán kính của myCircle.

MyCircle.tinhdientich() – diện tích của myCircle.

Ví dụ: Tạo đối tượng myCircle từ lớp Circle rồi dùng dữ liệu và phương thức của đối tượng đó tính diện tích của hình tròn bán kính bằng 1, xuất kết quả ra màn hình.

```
public class TestCircle
{
    public static void main(String[]args)
    {
        Circle myCircle = new Circle();
        System.out.println("Dien tich hinh tron ban kinh "+myCircle.bankinh+" la "+
        myCircle.tinhdientich());
    }
}
class Circle
{
    double bankinh=1.0;
    double tinhdientich()
    {
        return bankinh*bankinh*3.14159;
    }
}
```


3. Phương thức tạo dựng (constructor):

Java cho phép định nghĩa một phương thức đặc biệt trong lớp gọi là phương thức tạo dựng, dùng để khởi tạo dữ liệu của đối tượng. Trong ví dụ trên có thể phương thức tạo dựng gán bán kính ban đầu khi tạo đối tượng. Phương thức tạo dựng phải trùng tên với lớp chứa nó. Phương thức tạo dựng có thể tải chồng lên nhau, giúp dễ dàng xây dựng các đối tượng với các giá trị dữ liệu ban đầu khác nhau.

Ví dụ: Sử dụng phương thức tạo dựng trong lớp Circle1 để tạo 2 đối tượng khác nhau myCircle1 và yourCircle1.

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

```
        Circle1 myCircle1 = new Circle1(5.0);
        System.out.println("Dien tich hinh tron ban kinh "+myCircle1.bankinh+" la "+ myCircle1.tinhdientich());
        Circle1 yourCircle1 = new Circle1();
        System.out.println("Dien tich hinh tron ban kinh "+yourCircle1.bankinh+" la "+ yourCircle1.tinhdientich());
        }
}
class Circle1
{
        double bankinh;
        Circle1()
        {
                bankinh=1;
        }
        Circle1(double r)
        {
                bankinh=r;
        }
        double tinhdientich()
        {
                return bankinh*bankinh*3.14159;
        }
}
```

4. Chuyển đối tượng đến phương thức:

Cũng như việc có thể chuyển giá trị của biến đến phương thức, chúng ta được phép chuyển đối tượng đến phương thức ở dạng tham số thực.

Ví dụ: Xây dựng đối tượng myCircle2 từ lớp Circle2. Truyền đối tượng myCircle2 vào các phương thức printCircle2(), colorCircle2() để thực hiện việc in và đổi màu đối tượng myCircle2.

```
public class passobj
{
        public static void main(String[]args)
        {
                Circle2 myCircle2 = new Circle2(5.0,"white");
                printCircle2(myCircle2);
                colorCircle2(myCircle2,"black");
                printCircle2(myCircle2);
```

```
    }
    public static void colorCircle2(Circle2 c, String color)
    {
        c.color=color;
    }
    public static void printCircle2(Circle2 c)
    {
        System.out.println("Dien tich hinh tron ban kinh "+c.bankinh+" la "+
        c.tinhdientich());
        System.out.println("Mau cua hinh tron la: "+c.color);
    }
}
class Circle2
{
    double bankinh;
    String color;
    Circle2()
    {
        bankinh=1;
        color="white";
    }
    Circle2(double r, String m)
    {
        bankinh=r;
        color=m;
    }
    double tinhdientich()
    {
        return bankinh*bankinh*3.14159;
    }
}
```


5. Biến đối tượng và biến lớp:

Biến bankinh và color trong các ví dụ trên được gọi là biến đối tượng hay biến phiên bản lớp (instance variable). Biến phụ thuộc vào từng đối tượng của lớp, chúng không được chia sẻ giữa các đối tượng chung lớp. Ví dụ ta có 2 đối tượng myCircle và yourCircle được tạo bởi cùng một lớp Circle:

```
Circle  myCircle = new Circle();
```

```
Circle  yourCircle = new Circle();
```

Dữ liệu trong myCircle độc lập với dữ liệu trong yourCircle, và nằm tại 2 vị trí khác nhau trong bộ nhớ. Các thay đổi thực hiện cho dữ liệu của myCircle không ảnh hưởng tới dữ liệu của yourCircle và ngược lại. Nếu muốn các đối tượng của lớp dùng chung dữ liệu, ta có thể sử dụng biến lớp (class variable). Biến lớp lưu trữ giá trị dành cho các đối tượng tại địa điểm chung trong bộ nhớ do đó tất cả các đối tượng cùng lớp đều bị tác động đến nếu có một đối tượng thay đổi giá trị của biến lớp.

Để khai báo biến lớp, ta đặt thêm bổ từ static vào trước biến lớp. Ví dụ, muốn thêm độ đậm, nhạt (weight) vào hình tròn. Giả sử rằng tất cả các hình tròn đều có cùng độ đậm, nhạt. Ta định nghĩa biến lớp như sau:

```
static double weight;
```

Ví dụ: sử dụng biến đối tượng và biến lớp.

```
public class bienlop
{
    public static void main(String[]args)
    // tao va hien thi myCircle
    {
        Circle3 myCircle3 = new Circle3(4.0,"white",5.0);
        System.out.print("myCircle: ");
        printCircle3(myCircle3);
        // tao va hien thi yourCircle
        Circle3 yourCircle3 = new Circle3(5.0,"black",3.0);
        System.out.print("yourCircle: ");
        printCircle3(yourCircle3);
        //thay doi bien lop weight
        myCircle3.weight=15.5;
        //hien thi lai myCircle va yourCircle
        System.out.print("myCircle: ");
        printCircle3(myCircle3);
        System.out.print("yourCircle: ");
        printCircle3(yourCircle3);
    }
    //xay dung phuong thuc printCircle
    public static void printCircle3(Circle3 c)
    {
        System.out.println("ban kinh ("+c.getRadius()+"),mau( "+ c.color+") va do
        dam("+c.weight+")");
    }
}
```

```
class Circle3
{
	double bankinh;
	String color;
	static double weight; //bien lop
	//ham dung
	Circle3()
	{
		bankinh=1;
		color="white";
		weight=1.0;
	}
	Circle3(double r)
	{
		bankinh=r;
	}
	Circle3(double r, String c)
	{
		bankinh=r;
		color=c;
	}
	Circle3(double r, String c, double w)
	{
		bankinh=r;
		color=c;
		weight=w;
	}
	public double getRadius()
	{
		return bankinh;
	}
	public String getColor()
	{
		return color;
	}
}
```

6.Gói(Package):

Gói là tập hợp gồm nhiều lớp. Gói là cách thuận lợi để tổ chức lớp. Bạn có thể xếp các lớp đã thiết kế vào gói để phân phối đến người khác. Hãy tưởng tượng gói là thư viện cho nhiều người dùng. Bản thân ngôn ngữ Java cung cấp rất nhiều gói mà bạn có thể dùng để xây dựng chương trình ứng dụng. Sau đây là qui ước cách đặt tên gói: Gói có tính chất phân cấp nên đôi khi bạn sẽ thấy gói lồng trong gói.

Ví dụ, java.awt.Button cho biết Button là lớp thuộc gói awt và awt là gói chứa trong gói java. Bạn có thể dùng các cấp lồng để đảm bảo tên gói không bị trùng lặp. Bạn có thể thiết kế gói đặt trong một máy chủ cho người khác sử dụng trực tiếp qua Internet hoặc khai thác gói đã được người khác thiết kế sẵn đặt trong một máy chủ nào đó trên Internet theo qui ước đặt tên như sau: giả sử bạn muốn tạo gói mypackage.io trên máy chủ với tên vùng Internet là liangry.ipfw.indiana.edu thì phải đặt tên toàn gói là edu.indiana.ipfw.liangry.mypackage.io, bạn buộc phải tạo thư mục có đường dẫn như sau: edu\indiana\ipfw\liangry\mypackage\io. Nói cách khác, thực chất gói là thư mục chứa mã byte của lớp.

7. Đặt lớp vào gói:

Mọi lớp trong Java đều thuộc vào gói. Lớp được đưa vào gói khi cần biên dịch. Ví dụ, thư mục d:\btjava chứa chương trình nguồn (.java) thì thư mục d:\btjava\lop chứa các tập tin lớp (.class) sau khi đã biên dịch từ chương trình nguồn.

Nếu mỗi chương trình đều bắt đầu bằng câu lệnh: package lop;

-Phải định nghĩa lớp ở dạng công (vd: public class myinput) để các chương trình khác gói có thể truy cập được.

-Muốn vận dụng lớp từ một gói trong chương trình, bạn phải chèn câu lệnh import vào đầu chương trình.

Ví dụ: import lop2.myinput;

-Nếu có nhiều lớp để dùng từ một gói chung, có thể dùng dấu * để báo việc sử dụng tất cả các lớp trong gói.

Ví dụ: import lop2.*;

Ý nghĩa của các gói chính trong Java:

-java.lang: chứa các lớp Java chính như Object, string, system, Math, Number, Character, Boolean, Byte, Short, Integer, Long, Float và Double). Gói này ngầm du nhập vào mọi chương trình Java.

-java.awt: chứa lớp dùng để vẽ đối tượng hình học, quản lý bố cục và tạo các thành phần như: cửa sổ, khung, panel, menu, nút, font chữ, danh sách và nhiều thành phần khác.

-java.awt.event: chứa lớp xử lý biến cố trong lập trình đồ họa.

-javax.swing: chứa thành phần giao diện người dùng dạng đồ họa.

-java.applet: chứa lớp hỗ trợ applet.

-java.io: chứa lớp dành cho tập tin và luồng nhập/xuất.

-java.util: chứa nhiều tiện ích như ngày, tháng, lịch,…

-java.text: chứa lớp định dạng thông tin như ngày, tháng và giờ giấc, với nhiều kiểu định dạng dựa trên ngôn ngữ, quốc gia và văn hóa.

-java.net: chứa lớp hỗ trợ truyền thông mạng.

8. Lớp Math:

Chứa các phương thức cần thiết để thực hiện các hàm toán học cơ bản như: hàm lượng giác (sin, cos, tan, asin, acos, atan), hàm số mũ (exp, log, sqrt) và một số hàm tạp (min, max, abs, random). Tất cả phương thức này đều tính toán dựa trên giá trị double và trả về giá trị double; min, max, abs có thể tính toán trên giá trị int, long, float, double.

a.Phương thức lượng giác: lớp Math chứa các phương thức lượng giác sau:

public static double sin(double a)

public static double cos(double a)

public static double tan(double a)

public static double asin(double a)

public static double acos(double a)

public static double atan(double a)

Ví dụ: Math.sin(Math.PI) trả về hàm sin lượng giác của PI.

b.Phương thức số mũ:

public static double exp(double a)

public static double log(double a)

public static double sqrt(double a)

c.Phương thức min(), max(), abs() và random():

Hàm min(), max() trả về số nhỏ nhất và lớn nhất giữa 2 số (int, long, float, double). Ví dụ: max(3.4, 5.0) trả về kết quả là 5.0, và min (3, 2) trả về 2.

Hàm abs() trả về trị tuyệt đối của số (int, long, float, double). Ví dụ: abs(-3.03) trả về giá trị 3.03

Hàm random() tạo số chấm động ngẫu nhiên giữa 0 và 1.

9. Thừa kế lớp:

Với lập trình hướng đối tượng, bạn có thể nhận được lớp mới từ lớp hiện có. Đặc tính này gọi là thừa kế (inheritance). Thừa kế là khái niệm rất quan trọng trong Java. Thật ra, mọi lớp bạn định nghĩa trong Java đều là của thừa kế từ lớp hiện có, dù rõ ràng hay ngầm định. Chẳng hạn, lớp Circle được dẫn xuất ngầm từ lớp Object.

a. Lớp trên và lớp dưới:

Trong thuật ngữ Java, lớp hiện có là lớp trên (superclass). Lớp dẫn xuất từ lớp trên gọi là lớp dưới (subclass). Bạn có thể tái sử dụng hoặc thay đổi phương thức của lớp trên, bổ sung dữ liệu mới và phương thức mới vào lớp dưới. Như thế lớp dưới sẽ có nhiều tính năng hơn lớp trên.

Từ khóa super: tham chiếu đến lớp trên của lớp chứa từ khóa super. Từ khóa này có thể áp dụng theo 2 cách:

-Gọi phương thức tạo dựng thuộc lớp trên.

-Gọi phương thức thuộc lớp trên.

Ví dụ: Xây dựng đối tượng đối tượng myCylinder1 từ lớp Cylinder1. Tìm chiều cao, bán kính, thể tích, diện tích đáy hình trụ.

```
//Circle1.java
 public class Circle1
```

```
{
	double bankinh;
	Circle1()
	{
		bankinh=1;
	}
	Circle1(double r)
	{
		bankinh=r;
	}
	double tinhdientich()
	{
		return bankinh*bankinh*3.14159;
	}
}
//Cylinder1.java
public class Cylinder1 extends Circle1
{
	double length;
	public Cylinder1()
	{
		super();
		length=1.0;
	}
	public Cylinder1(double r, double l)
	{
		super(r);
		length=l;
	}
	public double getLength()
	{
		return length;
	}
	public double tinhthetich()
	{
		return tinhdientich()*length;
```

```
	}
}
// TestCyl.java
public class TestCyl
{
	public static void main(String[]args)
	{
		Cylinder1 myCylinder1 = new Cylinder1(5.0,2.0);
		System.out.println("chieu cao hinh tru: "+myCylinder1.getLength());
		System.out.println("ban kinh hinh tru: "+myCylinder1.bankinh);
		System.out.println("the tich hinh tru: "+myCylinder1.tinhthetich());
		System.out.println("Dien tich day hinh tru: "+myCylinder1.tinhdientich());
	}
}
```

Ghi chú: dùng JDK biên dịch 3 file này. Không được để các file class này trong cùng một thư mục (package). Vd: Circle1.class (package chapter7), Cylinder1.class (package chapter5), TestCyl.class (không cần câu lệnh package, nhưng phải thêm câu lệnh import chapter5.Cylinder1). Với JDK, quá trình biên dịch sẽ tạo file .class ngay tại thư mục hiện hành, nếu trong các file liên quan dùng lệnh import, package thì phải copy các file.class đúng vào thư mục có tên tương ứng với lệnh package, thư mục này tạo ra ngay tại thư mục đang chứa các file.java (vd: thư mục chứa các file .java là d:\bt thì các thư mục chapter5 và chapter7 phải nằm ngay tại d:\bt\chapter5, d:\bt\chapter7) .

```
//Circle1.java
 package chapter7;
public class Circle1
{
	public double bankinh;
	public Circle1()
	{
		bankinh=1;
	}
	public Circle1(double r)
	{
		bankinh=r;
	}
	public double tinhdientich()
	{
		return bankinh*bankinh*3.14159;
```

```
        }
}
//Cylinder1.java
package chapter5;
public class Cylinder1 extends chapter7.Circle1
{
        double length;
        public Cylinder1()
        {
                super();
                length=1.0;
        }
        public Cylinder1(double r, double l)
        {
                super(r);
                length=l;
        }
        public double getLength()
        {
                return length;
        }
        public double tinhthetich()
        {
                return tinhdientich()*length;
        }
}
// TestCyl.java
// package chapter5;
import chapter5.Cylinder1;
public class TestCyl
{
        public static void main(String[]args)
        {
                Cylinder1 myCylinder1 = new Cylinder1(5.0,2.0);
                System.out.println("chieu cao hinh tru: "+myCylinder1.getLength());
                System.out.println("ban kinh hinh tru: "+myCylinder1.bankinh);
```

```
            System.out.println("the tich hinh tru: "+myCylinder1.tinhthetich());
            System.out.println("Dien tich day hinh tru: "+myCylinder1.tinhdientich());
        }
}
```

b. Giành quyền phương thức:

Lớp dưới thừa kế từ lớp trên. Đôi khi, lớp dưới cần chỉnh sửa phương thức định nghĩa ở lớp trên. Điều này được gọi là giành quyền phương thức (method overriding).

Ví dụ: lớp Cylinder2 định nghĩa ở ví dụ trên sẽ được chỉnh sửa để giành quyền phương thức tinhdientich() trong lớp Circle1. Phương thức tinhdientich() trong lớp Circle1 tính diện tích hình tròn, còn tinhdientich() thuộc lớp Cylinder2 tính diện tích bề mặt của hình trụ.

```
//TestOver.java
import chapter5.Cylinder2;
public class TestOver
{
        public static void main(String[] args)
        {
            Cylinder2 myCylinder2 = new Cylinder2(5.0, 2.0);
            System.out.println("Dien tich be mat hinh tru: "+myCylinder2.tinhdientich());
            System.out.println("The tich hinh tru: "+myCylinder2.tinhthetich());
        }
}
//Cylinder2.java
package chapter5;
public class Cylinder2 extends chapter7.Circle1
{
        double length;
        public Cylinder2()
        {
            super();
            length=1.0;
        }
        public Cylinder2(double r, double l)
        {
            super(r);
            length=l;
        }
```

```
    public double getLength()
    {
        return length;
    }
    public double tinhdientich()
    {
        return 2*super.tinhdientich()+(2*bankinh*Math.PI)*length;
    }
    public double tinhthetich()
    {
        return super.tinhdientich()*length;
    }
}
```

Phương thức tinhdientich() được định nghĩa trong lớp Circle1 được chỉnh sửa tại lớp Cylinder2. Đối tượng Cylinder2 có thể tận dụng cả 2 phương thức. Muốn sử dụng phương thức tinhdientich() ở lớp Circle1, đối tượng Cylinder2 phải gọi super. tinhdientich(). Lớp dưới của Cylinder2 không còn có thể truy cập phương thức tinhdientich()định nghĩa trong lớp Circle1 do phương thức tinhdientich() đã được định nghĩa lại tại lớp Cylinder2.

10.Các từ khóa:

-Từ khóa static:

Từ khóa static trong Java được sử dụng chính để quản lý bộ nhớ. Chúng ta có thể áp dụng từ khóa static với các biến, các phương thức.

Trong java, static có thể là:

Biến static: Khi khai báo một biến là static, thì biến đó được gọi là biến tĩnh, hay biến static.

Phương thức static: Khi khai báo một phương thức là static, thì phương thức đó gọi là phương thức static.

-Từ khóa final:

Từ khóa final trong Java được sử dụng để hạn chế người dùng. Từ khóa final có thể được sử dụng trong nhiều ngữ cảnh:

Biến final: không thể thay đổi giá trị của biến final (nó sẽ là hằng số). Phương thức final: không thể ghi đè phương thức final.

Lớp final: không thể kế thừa lớp final.

-Từ khóa super:

Từ khóa super trong java là một biến tham chiếu được sử dụng để tham chiếu trực tiếp đến đối tượng của lớp cha gần nhất.

Bất cứ khi nào bạn tạo ra instance(thể hiển) của lớp con, một instance của lớp cha được tạo ra ngầm định, nghĩa là được tham chiếu bởi biến super.

Trong java, từ khóa super có 3 cách sử dụng sau: Từ khóa super được sử dụng để tham chiếu trực tiếp đến biến instance của lớp cha gần nhất.; Từ khóa super() được sử dụng để gọi trực tiếp Constructor của lớp cha.; Từ khóa super được sử dụng để gọi trực tiếp phương thức của lớp cha.

-Access Modifier trong Java:

Access Modifer trong Java xác định phạm vi có thể truy cập của biến, phương thức, constructor hoặc lớp. Trong java, có 4 phạm vi truy cập của Access Modifier như sau: private, default, protected, public.

- Phạm vi truy cập private: private chỉ được truy cập trong phạm vi lớp.
- Phạm vi truy cập default: nếu không khai báo modifier nào, thì nó chính là trường hợp mặc định. default chỉ được phép truy cập trong cùng package
- Phạm vi truy cập protected: protected được truy cập bên trong package và bên ngoài package nhưng phải kế thừa. Protected có thể được áp dụng cho biến, phương thức, constructor. Nó không thể áp dụng cho lớp.
- Phạm vi truy cập public: public được truy cập ở mọi nơi.

BÀI TẬP

Bài tập 1:

Để quản lý các hộ dân trong một khu phố, người ta quản lý các thông tin như sau:
- Với mỗi hộ dân, có các thuộc tính:
+ Số thành viên trong hộ (số người)
+ Số nhà của hộ dân đó. (Số nhà được gắn cho mỗi hộ dân)
+ Thông tin về mỗi cá nhân trong hộ gia đình.
- Với mỗi cá nhân, người ta quản lý các thông tin như: họ và tên, ngày sinh, nghề nghiệp.
1. Hãy xây dựng lớp Nguoi để quản lý thông tin về mỗi cá nhân.
2. Xây dựng lớp KhuPho để quản lý thông tin về các hộ gia đình.
3. Viết các phương thức nhập, hiển thị thông tin cho mỗi cá nhân.
4. Cài đặt chương trình thực hiện các công việc sau:
- Nhập vào một dãy gồm n hộ dân (n - nhập từ bàn phím).
- Hiển thị ra màn hình thông tin về các hộ trong khu phố năm nay có người mừng thượng thọ (80 tuổi)

Hướng dẫn:

```
//Lớp Người
import java.util.Scanner;
import java.util.Date;
import java.text.SimpleDateFormat;
public class Nguoi
{
   private String hoTen;
   private Date ngaySinh;
   private String ngNghiep;

   public Nguoi()
   {
   }
   public Nguoi(String hoTen,Date ngaySinh,String ngNghiep)
   {
      this.hoTen=hoTen;
      this.ngaySinh=ngaySinh;
      this.ngNghiep=ngNghiep;
   }

   public void nhapThongTin(Scanner sc)
   {
      System.out.print("Nhap ho ten: ");
      this.hoTen=sc.nextLine();
      System.out.print("Nhap ngay sinh theo dinh dang dd-MM-yyyy: ");
      String ns=sc.nextLine();
      this.ngaySinh=chuyenStringDate(ns);
      System.out.print("Nhap nghe nghiep: ");
      this.ngNghiep=sc.nextLine();
   }
   public Date chuyenStringDate(String ngay)
   {
```

```
        SimpleDateFormat sdf=new SimpleDateFormat("dd-MM-yyyy");
        Date ns=null;
        try
        {
           ns=sdf.parse(ngay);
        }catch(Exception e)
        {
           System.out.println("Loi dinh dang thoi gian.!");
        }
        return ns;
    }
    public void hienThongTin()
    {
        System.out.println("Ho va ten: "+this.hoTen);
        System.out.println("Ngay sinh: "+this.ngaySinh);
        System.out.println("Nghe nghiep: "+this.ngNghiep);
    }
    public Date getNgaySinh()
    {
        return this.ngaySinh;
    }

}
//Lớp Hộ Dân
import java.util.Scanner;
import java.util.Date;
public class HoDan
{
   private int soTVien;
   private int soNha;
   private Nguoi[] thanhVien;

   public HoDan()
   {
      this.thanhVien=new Nguoi[10];
   }
   public HoDan(int soTVien,int soNha,Nguoi[] thanhVien)
   {
      this.soTVien=soTVien;
      this.soNha=soNha;
      this.thanhVien=thanhVien;
   }

   public void nhapThongTin(Scanner sc)
   {
      System.out.print("Nhap so thanh vien: ");
      this.soTVien=sc.nextInt();
      System.out.print("Nhap so nha: ");
      this.soNha=sc.nextInt();sc.nextLine();
      for(int i=0;i<n;i++)
      {
         System.out.println("Nhap thong tin thanh vien thu "+(i+1)+": ");
```

```
            thanhVien[i]=new Nguoi();
            thanhVien[i].nhapThongTin(sc);
        }
    }
    public void hienThongTin()
    {
        System.out.println("So nha: "+this.soNha);
        System.out.println("So thanh vien: "+this.soTVien);
        for(int i=0;i
        {
            System.out.println("Thanh vien thu "+(i+1)+": ");
            thanhVien[i].hienThongTin();
        }
    }
    public Nguoi[] getThanhVien()
    {
        return this.thanhVien;
    }


//Lớp khu phố
import java.util.Scanner;
import java.util.ArrayList;
public class KhuPho
{
    private HoDan[] dsHoDan;
    private int n;

    public void nhapHoDan(Scanner sc)
    {
        System.out.print("Nhap so ho dan can nhap: ");
        this.n=sc.nextInt();sc.nextLine();
        this.dsHoDan=new HoDan[n];
        for(int i=0;i<n;i++)
        {
            System.out.println("Nhap thong tin dan thu "+(i+1)+": ");
            this.dsHoDan[i]=new HoDan();
            this.dsHoDan[i].nhapThongTin(sc);
        }
    }

    public static void main(String[] args)
    {
        KhuPho ql=new KhuPho();
        Scanner sc=new Scanner(System.in);
        ql.nhapHoDan(sc);
        System.out.println("Nhap nam mung tho: ");
        int nam=sc.nextInt();
        ql.mungTho(nam);
    }
    public void mungTho(int nam)
    {
```

```
        Nguoi thanhVien[]=null;
        boolean a=false;
        int soTvien;
        int b;
        for(int i=0; i<n;i++)
        {
           b=0;
           thanhVien=dsHoDan[i].getThanhVien();
           soTvien=thanhVien.length;
              while(soTvien!=0)
              {
                 if((nam-thanhVien[b].getNgaySinh().getYear()-1900)>=80)
                 {
                     a=true;
                     break;
                 }
                 soTvien--;
                 b++;
              }
            if(a==true)
              dsHoDan[i].hienThongTin();
        }
    }

}
```


Bài tập 2:

Để quản lý khách hàng đến thuê phòng trọ của một khách sạn, người ta cần quản lý những thông tin sau:
- Số ngày trọ, loại phòng trọ, giá phòng, và các thông tin cá nhân về mỗi khách trọ.
- Với mỗi cá nhân, người ta cần quản lý các thông tin : Họ và tên, ngày sinh, số chứng minh thư nhân dân.

1. Hãy xây dựng lớp Nguoi để quản lý thông tin cá nhân về mỗi cá nhân
2. Xây dựng lớp KhachSan để quản lý các thông tin về khách trọ.
3. Viết các phương thức : nhập, hiển thị, xóa các thông tin về một khách trọ,
4. Cài đặt chương trình thực hiện các công việc sau:
- Nhập vào một dãy gồm n khách trọ (n - nhập từ bàn phím)
- Hiển thị ra màn hình thông tin về các cá nhân hiện đang trọ ở khách sạn đó.
- Tính số tiền cần phải trả nếu một khách hàng trả phòng trọ (căn cứ vào số CMND để tìm kiếm trong mảng)

Bài tập 3:

Để quản lý hồ sơ học sinh của trường THPT, người ta cần quản lý những thông tin như sau:
- Các thông tin về : lớp, khoá học, kỳ học, và các thông tin cá nhân của mỗi học sinh.
- Với mỗi học sinh, các thông tin cá nhân cần quản lý gồm có: Họ và tên, ngày sinh, quê quán.

1. Hãy xây dựng lớp Nguoi để quản lý các thông tin cá nhân của mỗi học sinh.
2. Xây dựng lớp HSHocSinh (hồ sơ học sinh) để lý các thông tin về mỗi học sinh.
3. Xây dựng các phương thức : nhập, hiển thị các thông tin về mỗi cá nhân.
4. Cài đặt chương trình thực hiện các công việc sau:

- Nhập vào một danh sách gồm n học sinh (n- nhập từ bàn phím)
- Hiển thị ra màn hình tất cả những học sinh sinh năm 1985 và quê ở Thái Nguyên
- Hiển thị ra màn hình tất cả những học sinh của lớp 10A1

Bài tập 4:

Để quản lý các biên lai thu tiền điện, người ta cần các thông tin như sau:
- Với mỗi biên lai, có các thông tin sau: thông tin về hộ sử dụng điện, chỉ số cũ, chỉ số mới, số tiền phải
trả của mỗi hộ sử dụng điện
- Các thông tin riêng của mỗi hộ sử dụng điện gồm: Họ tên chủ hộ, số nhà, mã số công tơ của hộ dân sử dụng điện.
1. Hãy xây dựng lớp KhachHang để lưu trữ các thông tin riêng của mỗi hộ sử dụng điện.
2. Xây dựng lớp BienLai để quản lý việc sử dụng và thanh toán tiền điện của các hộ dân.
3. Xây dựng các phương thức nhập, và hiển thị một thông tin riêng của mỗI hộ sử dụng điện.
4. Cài đặt chương trình thực hiện các công việc sau:
+ Nhập vào các thông tin cho n hộ sử dụng điện
+ Hiển thị thông tin về các biên lai đã nhập
+ Tính tiền điện phải trả cho mỗi hộ dân, nếu giả sử rằng tiền phải trả được tính theo công thức sau:
số tiền phải trả=(Số mới - số cũ) * 750.

Bài tập 5:

Thư viện của trường đại học có nhu cầu cần quản lý việc mượn sách. Sinh viên đăng ký và tham gia mượn sách thông qua các thẻ mượn mà thư viện đã thiết kế.
- Với mỗi thẻ mượn, có các thông tin sau: số phiếu mượn , ngày mượn, hạn trả , số hiệu sách, và các thông tin riêng về mỗi sinh viên đó.
- Các thông tin riêng về mỗi sinh viên đó bao gồm: Họ tên, MSV, ngày sinh, lớp.
1. Hãy xây dựng lớp SinhVien để quản lý các thông tin riêng về mỗi sinh viên.
2. Xây dựng lớp TheMuon để quản lý việc mượn sách của mỗi đọc giả.
3. Xây dựng các phương thức để nhập và hiện thị các thông tin riêng cho mỗi sinh viên
4. In ra danh sách sinh viên, tên sách mượn cần trả vào ngày cuối tháng

BÀI 3. TÍNH ĐA HÌNH (POLYMORPHISM) TRONG JAVA

1/ Khái niệm:

Tính đa hình là khả năng một đối tượng có thể thực hiện một tác vụ theo nhiều cách khác nhau.

Đối với tính chất này, nó được thể hiện rõ nhất qua việc gọi phương thức của đối tượng. Các phương thức hoàn toàn có thể giống nhau, nhưng việc xử lý luồng có thể khác nhau. Nói cách khác: Tính đa hình cung cấp khả năng cho phép người lập trình gọi trước một phương thức của đối tượng, tuy chưa xác định đối tượng có phương thức muốn gọi hay không. Đến khi thực hiện (**run-time**), chương trình mới xác định được đối tượng và gọi phương thức tương ứng của đối tượng đó. Kết nối trễ giúp chương trình được uyển chuyển hơn, chỉ yêu cầu đối tượng cung cấp đúng phương thức cần thiết là đủ.

Trong Java, chúng ta sử dụng nạp chồng phương thức (method **overloading**) và ghi đè phương thức (method **overriding**) để có tính đa hình.

Nạp chồng (Overloading): Đây là khả năng cho phép một lớp có nhiều thuộc tính, phương thức cùng tên nhưng với các tham số khác nhau về loại cũng như về số lượng. Khi được gọi, dựa vào tham số truyền vào, phương thức tương ứng sẽ được thực hiện.

Ghi đè (Overriding): là hai phương thức cùng tên, cùng tham số, cùng kiểu trả về nhưng thằng con viết lại và dùng theo cách của nó, và xuất hiện ở lớp cha và tiếp tục xuất hiện ở lớp con. Khi dùng override, lúc thực thi, nếu lớp Con không có phương thức riêng, phương thức của lớp Cha sẽ được gọi, ngược lại nếu có, phương thức của lớp Con được gọi.

2/ Đa hình lúc runtime trong java:

Đa hình lúc runtime là quá trình gọi phương thức đã được ghi đè trong thời gian thực thi chương trình. Trong quá trình này, một phương thức được ghi đè được gọi thông qua biến tham chiếu của một lớp cha.

Trước khi tìm hiểu về đa hình tại runtime, chúng ta cùng tìm hiểu về Upcasting.

Upcasting là gì?

Khi biến tham chiếu của lớp cha tham chiếu tới đối tượng của lớp con, thì đó là Upcasting.

Ví dụ:

```
class A {
}
 class B extends A {
}
A a = new B(); // upcasting
```

Ví dụ về đa hình tại runtime trong Java:

Ví dụ 1: chúng ta tạo hai lớp Bike và Splendar. Lớp Splendar kế thừa lớp Bike và ghi đè phương thức run() của nó. Chúng ta gọi phương thức run bởi biến tham chiếu của lớp cha. Khi nó tham chiếu tới đối tượng của lớp con và phương thức lớp con ghi đè phương thức của lớp cha, phương thức lớp con được triệu hồi tại runtime. Khi việc gọi phương thức được quyết định bởi JVM chứ không phải Compiler, vì thế đó là đa hình tại runtime.

```
public class Bike {
   public void run() {
```

```
        System.out.println("running");
    }
}
public class Splender extends Bike {
    public void run() {
        System.out.println("running safely with 60km");
    }
public static void main(String args[]) {
        Bike b = new Splender(); // upcasting
        b.run();
    }
}
```

Kết quả:

running safely with 60km

Ví dụ 2: Giả sử Bank là một lớp cung cấp phương thức để lấy lãi suất. Nhưng lãi suất lại khác nhau giữa từng ngân hàng. Ví dụ, các ngân hàng VCB, AGR và CTG có thể cung cấp các lãi suất lần lượt là 8%, 7% và 9%. (Ví dụ này cũng có trong chương ghi đè phương thức nhưng không có Upcasting).

```
class Bank {
    int getRateOfInterest() {
        return 0;
    }
}
class VCB extends Bank {
    int getRateOfInterest() {
        return 8;
    }
}
class AGR extends Bank {
    int getRateOfInterest() {
        return 7;
    }
}
class CTG extends Bank {
    int getRateOfInterest() {
        return 9;
```

```
    }
}
class Test3 {
    public static void main(String args[]) {
        Bank b1 = new VCB(); // upcasting
        Bank b2 = new AGR(); // upcasting
        Bank b3 = new CTG(); // upcasting
        System.out.println("VCB lai suat la: " + b1.getRateOfInterest());
        System.out.println("AGR lai suat la: " + b2.getRateOfInterest());
        System.out.println("CTG lai suat la: " + b3.getRateOfInterest());
    }
}
```

Kết quả:

VCB lai suat la: 8

VCB lai suat la: 7

VCB lai suat la: 9

Ví dụ 3: Shape

```
class Shape {
    void draw() {
        System.out.println("drawing...");
    }
}
class Rectangle extends Shape {
    void draw() {
        System.out.println("drawing rectangle...");
    }
}
class Circle extends Shape {
    void draw() {
        System.out.println("drawing circle...");
    }
}
class Triangle extends Shape {
    void draw() {
        System.out.println("drawing triangle...");
```

```
    }
}
class TestPolymorphism2 {
    public static void main(String args[]) {
        Shape s;
        s = new Rectangle();
        s.draw();
        s = new Circle();
        s.draw();
        s = new Triangle();
        s.draw();
    }
}
```

Kết quả:

drawing rectangle...

drawing circle...

drawing triangle...

3/ Đa hình tại runtime trong Java với thành viên dữ liệu

Phương thức bị ghi đè không là thành viên dữ liệu, vì thế đa hình tại runtime không thể có được bởi thành viên dữ liệu. Trong ví dụ sau đây, cả hai lớp có một thành viên dữ liệu là speedlimit, chúng ta truy cập thành viên dữ liệu bởi biến tham chiếu của lớp cha mà tham chiếu tới đối tượng lớp con. Khi chúng ta truy cập thành viên dữ liệu mà không bị ghi đè, thì nó sẽ luôn luôn truy cập thành viên dữ liệu của lớp cha.

Qui tắc: Đa hình tại runtime không thể có được bởi thành viên dữ liệu.

Ví dụ:

```
class Bike {
   int speedlimit = 90;
}
class Honda3 extends Bike {
   int speedlimit = 150;

   public static void main(String args[]){
     Bike obj=new Honda3();
     System.out.println(obj.speedlimit); // 90
   }
}
```

4/ Đa hình lúc runtime trong Java với kế thừa nhiều tầng:

Ví dụ 1:

```
class Animal {
   void eat() {
      System.out.println("eating");
   }
}
class Dog extends Animal {
   void eat() {
      System.out.println("eating fruits");
   }
}
class BabyDog extends Dog {
   void eat() {
      System.out.println("drinking milk");
   }
public static void main(String args[]) {
      Animal a1, a2, a3;
      a1 = new Animal();
      a2 = new Dog();
      a3 = new BabyDog();
      a1.eat();
      a2.eat();
      a3.eat();
   }
}
```


Kết quả:

eating

eating fruits

drinking Milk

Ví dụ 2:

```
class Animal {
   void eat() {
      System.out.println("animal is eating...");
   }
```

```
}
class Dog extends Animal {
  void eat() {
    System.out.println("dog is eating...");
  }
}
class BabyDog1 extends Dog {
  public static void main(String args[]) {
    Animal a = new BabyDog1();
    a.eat();
  }
}
```

Kết quả:

Dog is eating

Vì BabyDog1 không ghi đè phương thức eat(), nên phương thức eat() của lớp Dog được gọi.

5/ Nạp chồng phương thức (method overloading):

Nếu một lớp có nhiều phương thức cùng tên nhưng khác nhau về kiểu dữ liệu hoặc số lượng các tham số, thì đó là nạp chồng phương thức (Method Overloading).

Sử dụng nạp chồng phương thức giúp tăng khả năng đọc hiểu chương trình.

Nạp chồng phương thức được sử dụng để thu được tính đa hình lúc biên dịch (compile).

Có 2 cách nạp chồng phương thức trong java

•Thay đổi số lượng các tham số

•Thay đổi kiểu dữ liệu của các tham số

5.1/ Nạp chồng phương thức: thay đổi số lượng các tham số:

Ví dụ: tạo 2 phương thức có cùng kiểu dữ liệu: phương thức add() đầu tiên thực hiện việc tính tổng của 2 số, phương thức thứ hai thực hiện việc tính tổng của 3 số.

```
class Adder {
  static int add(int a, int b) {
    return a + b;
  }

  static int add(int a, int b, int c) {
    return a + b + c;
  }
```

```
}
 class TestOverloading1 {
   public static void main(String[] args) {
      System.out.println(Adder.add(5, 5));
      System.out.println(Adder.add(5, 5, 5));
   }
}
```

Kết quả:

10

15

5.2 / Nạp chồng phương thức: thay đổi kiểu dữ liệu của các tham số:

Ví dụ: tạo 2 phương thức có kiểu dữ liệu khác nhau: phương thức add() đầu tiên nhận 2 đối số có kiểu giá trị là integer, phương thức thứ hai nhận 2 đối số có kiểu giá trị là double.

```
class Adder {
        static int add(int a, int b) {
                return a + b;
        }
        static double add(double a, double b) {
                return a + b;
        }
}
class TestOverloading2 {
        public static void main(String[] args) {
                System.out.println(Adder.add(5, 5));
                System.out.println(Adder.add(4.3, 5.6));
        }
}
```

Kết quả chạy chương trình trên:

10

9.9

6/ Ghi đè phương thức (method overriding):

Ghi đè phương thức trong java xảy ra nếu lớp con có phương thức giống lớp cha.

Nói cách khác, nếu lớp con cung cấp sự cài đặt cụ thể cho phương thức đã được cung cấp bởi một lớp cha của nó được gọi là ghi đè phương thức (method overriding) trong java.

Ghi đè phương thức được sử dụng để thu được tính đa hình tại runtime.

Nguyên tắc ghi đè phương thức:

•Phương thức phải có tên giống với lớp cha.

•Phương thức phải có tham số giống với lớp cha.

•Lớp con và lớp cha có mối quan hệ kế thừa.

Ví dụ về ghi đè phương thức (method overriding)

Ví dụ 1: chúng ta định nghĩa phương thức run() trong lớp con giống như đã được định nghĩa trong lớp cha, nhưng được cài đặt rõ ràng trong lớp con. Tên và tham số của phương thức là giống nhau, 2 lớp cha và con có quan hệ kế thừa

```
. class Vehicle {
   void run() {
      System.out.println("Vehicle is running");
   }
}
 class Bike extends Vehicle {
   void run() {
      System.out.println("Bike is running safely");
   }
    public static void main(String args[]) {
      Bike obj = new Bike();
      obj.run();
   }
 }
```

Kết quả:

Bike is running safely

Ví dụ 2: Giả sử Bank là một đối tượng cung cấp lãi suất. Nhưng lãi suất lại khác nhau giữa từng ngân hàng. Ví dụ, các ngân hàng VCB, AGR và CTG có thể cung cấp các lãi suất lần lượt là 8%, 7% và 9%.

```
class Bank {
   int getRateOfInterest() {
      return 0;
   }
}
 class VCB extends Bank {
   int getRateOfInterest() {
      return 8;
   }
```

```
}
 class AGR extends Bank {
   int getRateOfInterest() {
      return 7;
   }
}
 class CTG extends Bank {
   int getRateOfInterest() {
      return 9;
   }
}
 class BankApp {
   public static void main(String args[]) {
      VCB s = new VCB();
      AGR i = new AGR();
      CTG a = new CTG();
      System.out.println("VCB Rate of Interest: " + s.getRateOfInterest());
      System.out.println("AGR Rate of Interest: " + i.getRateOfInterest());
      System.out.println("CTG Rate of Interest: " + a.getRateOfInterest());
   }
}
```

Kết quả:

VCB Rate of Interest: 8

AGR Rate of Interest: 7

CTG Rate of Interest: 9

BÀI 4. TÍNH KẾ THỪA (INHERITANCE) TRONG JAVA

1/ Tính kế thừa là gì?

Tính kế thừa (Inheritance) là một trong bốn tính chất cơ bản của lập trình hướng đối tượng trong Java.

Kế thừa là sự liên quan giữa hai class với nhau, trong đó có class cha (superclass) và class con (subclass). Khi kế thừa class con được hưởng tất cả các phương thức và thuộc tính của class cha. Tuy nhiên, nó chỉ được truy cập các thành viên public và protected của class cha. Nó không được phép truy cập đến thành viên private của class cha.

Tư tưởng của kế thừa trong java là có thể tạo ra một class mới được xây dựng trên các lớp đang tồn tại. Khi kế thừa từ một lớp đang tồn tại bạn có sử dụng lại các phương thức và thuộc tính của lớp cha, đồng thời có thể khai báo thêm các phương thức và thuộc tính khác.

Từ khóa extends được sử dụng để thể hiện sự kế thừa của một lớp.

Cú pháp:

```
class Super {

}

class Sub extends Super {

}
```

Ví Dụ:

```
public class Employee {
    protected float salary = 40000;
 }

public class Programmer extends Employee {
    int bonus = 10000;
    public static void main(String args[]) {
      Programmer p = new Programmer();
      System.out.println("Programmer salary is:" + p.salary);
      System.out.println("Bonus of Programmer is:" + p.bonus);
    }
 }
```

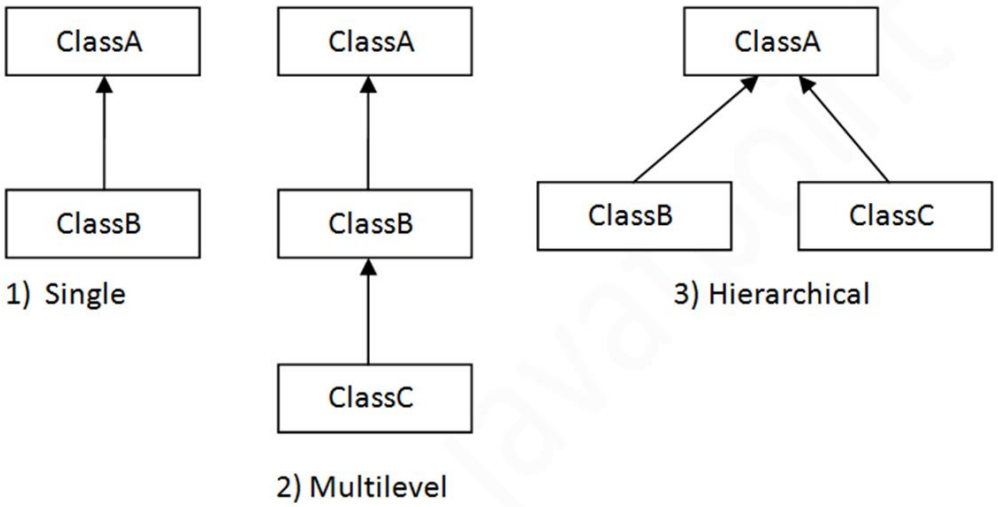
Trong ví dụ trên, class Programmer là con của class Employee, nên nó được phép truy cập đến field salary của class Programmer.

2/ Các kiểu kế thừa trong java:

Có 3 kiểu kế thừa trong java: đơn kế thừa (single), kế thừa nhiều cấp (multilevel), kế thừa thứ bậc (hierarchical).

Khi một class được kế thừa từ nhiều class đươc gọi là đa kế thừa.

Trong java, một class không được phép thừa kế từ nhiều class, có thể cài đặt (implement) nhiều interface. Tuy nhiên, một interface có thể thừa kế nhiều interface.



2.1/ Thừa kế đơn (Single Inheritance):

Ví dụ:

```
public class Animal {
   public void eat() {
      System.out.println("eating...");
   }
}
 public class Dog extends Animal {
   public void bark() {
      System.out.println("barking...");
   }
}

public class TestInheritance {
   public static void main(String args[]) {
      Dog d = new Dog();
      d.bark();
      d.eat();
   }
}
```

Kết quả:

barking...

eating...

2.2/ Thừa kế nhiều cấp (Multilevel Inheritance):

Ví dụ:

```
public class Animal {
    public void eat() {
        System.out.println("eating...");
    }
}
 public class Dog extends Animal {
    public void bark() {
        System.out.println("barking...");
    }
}
 public class BabyDog extends Dog {
    public void weep() {
        System.out.println("weeping...");
    }
}
 public class TestInheritance2 {
    public static void main(String args[]) {
        BabyDog d = new BabyDog();
        d.weep();
        d.bark();
        d.eat();
    }
}
```

Kết quả:

weeping...

barking...

eating...

2.3/ Thừa kế thứ bậc (Hierarchical Inheritance):

Ví dụ:

```
class Animal {
```

```
    public void eat() {
        System.out.println("eating...");
    }
}
class Dog extends Animal {
    public void bark() {
        System.out.println("barking...");
    }
}
 public class Cat extends Animal {
    public void meow() {
        System.out.println("meowing...");
    }
}
 public class TestInheritance3 {
    public static void main(String args[]) {
        Cat c = new Cat();
        c.meow();
        c.eat();
    }
}
```

Kết quả:

meowing...

eating...

BÀI TẬP

1/ Viết một class point có các thuộc tính vị trí (x,y) và các phương thức set,get để làm việc với hai thuộc tính đó.

2/ Viết một class circle kế thừa từ class point. Có thêm thuộc tính r (bán kính) và các phương thức tính chu vi, diện tích của hình tròn đó.

3/

Viết các lớp Account, NormalAccount, NickelNDime, Gambler về các loại tài khoản ngân hàng theo mô tả sau: Thông tin về mỗi tài khoản ngân hàng gồm có số dư hiện tại (int balance), số giao dịch đã thực hiện kể từ đầu tháng (int transactions). Mỗi tài khoản cần đáp ứng các thao tác sau:

a) Một hàm khởi tạo cho phép mở một tài khoản mới với một số dư ban đầu cho trước;

b) Các phương thức boolean deposit(int) cho phép gửi tiền vào tài khoản, boolean withdraw(int) cho phép rút tiền từ tài khoản. Các phương thức này trả về true nếu giao dịch thành công, nếu không thì trả về false, tương tự cập nhật số đếm giao dịch.

c) Phương thức void endMonth() thực hiện tất toán, sẽ được mô đun quản lí tài khoản (nằm ngoài phạm vi bài này) gọi định kì vào các thời điểm cuối tháng. Phương thức này tính phí hàng tháng nếu có bằng cách gọi phương thức int endMonthCharge(), trừ phí, in thông tin tài khoản (số dư, số giao dịch, phí), và đặt lại số giao dịch về 0 để sẵn sàng cho tháng sau.

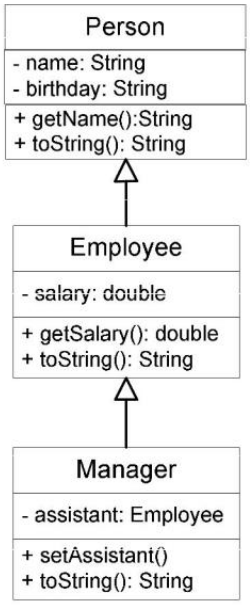
d) phương thức endMonthCharge() trả về phí tài khoản trong tháng vừa qua.

Phí tài khoản được tính tùy theo từng loại tài khoản. Loại NormalAccount tính phí hàng tháng là 10.000 đồng. Loại NickelNDime tính phí theo số lần rút tiền, phí cho mỗi lần rút là 2000 đồng, cuối tháng mới thu. Loại Gambler không tính phí cuối tháng nhưng thu phí tại từng lần rút tiền theo xác suất như sau: Với xác suất 49%, tài khoản không bị hụt đi đồng nào và giao dịch thành công miễn phí. Với xác suất 51%, phí rút tiền bằng đúng số tiền rút được.

Account là lớp cha của NormalAccount, NickelNDime, và Gambler. Cần thiết kế sao cho tái sử dụng và tránh lặp code được càng nhiều càng tốt.

4/

Viết các lớp Person, Employee, Manager như thiết kế trong sơ đồ sau. Bổ sung các phương thức thích hợp nếu thấy cần. Định nghĩa lại các phương thức toString() cho phù hợp với dữ liệu tại mỗi lớp.



Viết lớp PeopleTest để chạy thử các lớp trên: tạo một vài đối tượng và in thông tin của chúng ra màn hình. Trong hàm main của lớp PeopleTest, tạo một mảng kiểu Person, gắn ba đối tượng ở trên vào mảng, rồi dùng vòng lặp để in ra thông tin về các đối tượng trong mảng.

Đọc Phụ lục B. Tách các lớp Person, Employee vào trong gói peoples. Đặt Manager và PeopleTest ở gói mặc định (nằm ngoài gói peoples). Chỉnh lại các khai báo quyền truy nhập tại các lớp để chương trình viết ở trên lại chạy được.

BÀI 5: THỰC HÀNH VỀ COLLECTIONS

Mục đích:

Sinh viên hiểu được : ArrayList, LinkedList, Vector, Hash table, HashSet, LinkedHashSet, PriorityQueue.

Ví dụ 1: Viết chương trình quản lý sinh viên dưới dạng console, yêu cầu sử dụng **ArrayList** hoặc **LinkedList** hoặc **Vector** và thực hiện các chức năng sau (thông tin sinh viên bao gồm: mã số sinh viên, họ tên, năm sinh, địa chỉ, lớp học):

a) Cho phép thêm, sửa, xóa danh sách sinh viên

b) Xuất ra số lượng sinh viên

c) Xuất ra danh sách các sinh viên thuộc một lớp học bất kỳ nhập vào từ bàn phím

d) Cho phép lưu/mở danh sách sinh viên trên ổ cứng

Coding mẫu câu a, các câu khác sinh viên tự làm:

Class **Sinhvien**

```
package chap2.quanlysinhvien;
import java.util.Date;
public class Sinhvien {
	private String Masv;
	private String Tensv;
	private Date Namsinh;
	private String Diachi;
	private String Lop;
	public String getMasv() {
		return Masv;
	}
	public void setMasv(String masv) {
		Masv = masv;
	}
	public String getTensv() {
		return Tensv;
	}
	public void setTensv(String tensv) {
		Tensv = tensv;
	}
	public Date getNamsinh() {
		return Namsinh;
```

```
	}
	public void setNamsinh(Date namsinh) {
		Namsinh = namsinh;
	}
	public String getDiachi() {
		return Diachi;
	}
	public void setDiachi(String diachi) {
		Diachi = diachi;
	}
	public String getLop() {
		return Lop;
	}
	public void setLop(String lop) {
		Lop = lop;
	}
	public String toString() {
		return "Sinhvien [Masv=" + Masv + ", Tensv=" + Tensv + ", Namsinh="
				+ Namsinh + ", Diachi=" + Diachi + ", Lop=" + Lop + "]";
	}

}
```


Class **DanhSachSinhvien**

```
package chap2.quanlysinhvien;
import java.util.ArrayList;
public class DanhSachSinhvien {
	private ArrayList<Sinhvien> dsSv=new ArrayList<Sinhvien>();
	public boolean ktTrungma(String masv)
	{
		for(Sinhvien sv : dsSv)
		{
			if(sv.getMasv().equalsIgnoreCase(masv))
				return true;
		}
		return false;
```

```
}
public boolean addSinhvien(Sinhvien sv)
{
	if(ktTrungma(sv.getMasv()))
		return false;
	return dsSv.add(sv);
}
public Sinhvien findSinhvien1(String masv)
{
	for(Sinhvien sv : dsSv)
	{
		if(sv.getMasv().equalsIgnoreCase(masv))
			return sv;
	}
	return null;
}
public int findSinhvien2(String masv)
{
	for(int i=0;i<dsSv.size();i++)
	{
		if(dsSv.get(i).getMasv().equalsIgnoreCase(masv))
			return i;
	}
	return -1;
}
public Sinhvien updateSinhvien(int index,Sinhvien sv)
{
	return dsSv.set(index, sv);
}
public void removeSinhvien(String masv)
{
	Sinhvien sv=findSinhvien1(masv);
	dsSv.remove(sv);
	//int pos=findSinhvien2(masv);
	//dsSv.remove(pos);
}
```

```
	public String toString() {
		return dsSv.toString();
	}
}
```

Class TestSinhvien

```
package chap2.quanlysinhvien;
public class TestSinhvien {
public static void main(String[] args) {
		DanhSachSinhvien qlsv=new DanhSachSinhvien();
		Sinhvien teo=new Sinhvien();
		teo.setMasv("113");
		teo.setTensv("Nguyễn Văn Tèo");
		qlsv.addSinhvien(teo);
		Sinhvien ty=new Sinhvien();
		ty.setMasv("114");
		ty.setTensv("Nguyễn Thị tý");
		qlsv.addSinhvien(ty);
		System.out.println(qlsv);
	}

}
```


Ví dụ 2: Thực hành với **Hashtable**.

Hashtable đi theo Key và Value. Nếu Key trùng nhau thì nó sẽ lấy theo Key thứ 2.

Hashtable cung cấp nhiều hàm hữu ích: put, get, remove, elements… tùy vào từng trường hợp mà chúng ta ứng dụng cho phù hợp, dưới đây cung cấp một số ví dụ về **Hashtable**, yêu cầu sinh viên thực hiện và cho nhận xét.

```
package chap1.myHashTable;
public class Person {
	private String Id;
	private String Name;
	public String getId() {
		return Id;
	}
	public void setId(String id) {
```

```
		Id = id;
	}
	public String getName() {
		return Name;
	}
	public void setName(String name) {
		Name = name;
	}
	public Person(String id, String name) {
		super();
		Id = id;
		Name = name;
	}
	public String toString() {
		return "Person [Id=" + Id + ", Name=" + Name + "]";
	}
}
```

```
package chap1.myHashTable;
import java.util.Enumeration;
import java.util.Hashtable;
public class TestHashTable {
	public static void main(String[] args) {
		Hashtable<String, Person> hashTbl=new Hashtable<String, Person>();
		hashTbl.put("1", new Person("1", "teo"));
		Person p=new Person("2", "ty");

		if(hashTbl.containsKey(p.getId()))
		{
			System.out.println("Trung");
		}
		else
			hashTbl.put(p.getId(), p);

		Person p2=new Person("3", "Nuong");
```

```
            hashTbl.put(p2.getId(), p2);
            hashTbl.remove("1");
            System.out.println(hashTbl);
            Person px= hashTbl.get("3");

            Enumeration<Person> list=hashTbl.elements();
            while(list.hasMoreElements())
            {
                  Person x=list.nextElement();
                  System.out.println(x);
            }
      }
}
```

Sinh viên có thể dựa vào ví dụ này để đọc cơ sở dữ liệu và đổ vào Hashtable, với Key chính là khóa chính, còn Value chính là đối tượng trong từng dòng dữ liệu.

Ví dụ 3: Thực hành với HashSet , hãy so sánh sự khác biệt giữa HashSet , Hashtable và ArrayList.

Sinh viên thực hiện ví dụ mẫu bên dưới, từ đó ứng dụng để viết chương trình quản lý sản phẩm.

```
package chap2.myHashSet;

public class Product {
      private String Id;
      private String Name;
      private int Quantity;
      private double UnitPrice;
      public String getId() {
            return Id;
      }
      public void setId(String id) {
            Id = id;
      }
      public String getName() {
            return Name;
      }
```

```
	public void setName(String name) {
		Name = name;
	}
	public int getQuantity() {
		return Quantity;
	}
	public void setQuantity(int quantity) {
		Quantity = quantity;
	}
	public double getUnitPrice() {
		return UnitPrice;
	}
	public void setUnitPrice(double unitPrice) {
		UnitPrice = unitPrice;
	}
	public Product(String id, String name, int quantity, double unitPrice) {
		super();
		Id = id;
		Name = name;
		Quantity = quantity;
		UnitPrice = unitPrice;
	}
	public String toString() {
		return "Product [Id=" + Id + ", Name=" + Name + ", Quantity="
				+ Quantity + ", UnitPrice=" + UnitPrice + "]";
	}

}
```



```
package chap2.myHashSet;

import java.util.HashSet;
import java.util.Iterator;

public class TestHashSet {
```

```
	public static void main(String[] args) {
		HashSet<Product> myhs=new HashSet<Product>();
		Product p1=new Product("p1", "Iphone 5", 2, 15);
		myhs.add(p1);
		Product p2=new Product("p2", "Samsung sII",3, 11);
		myhs.add(p2);
		System.out.println(myhs);

		Iterator<Product> list= myhs.iterator();
		while(list.hasNext())
		{
			Product x= list.next();
			System.out.println(x);
		}

	}
}
```

Ví dụ 4: Thực hành về PriorityQueue

Hãy sử dụng PriorityQueue để quản lý độ ưu tiên công việc cần thực hiện. Những công việc nào có độ ưu tiên cao nhất thì được thực hiện trước.

```
public class Task implements Comparable<Task> {
	private String description;
	private int priority;
	public String getDescription() {
		return description;
	}
	public void setDescription(String description) {
		this.description = description;
	}
	public int getPriority() {
		return priority;
	}

	public void setPriority(int priority) {
```

```
            this.priority = priority;
        }
        public Task(String description, int priority)
        {
            this.description=description;
            this.priority=priority;
        }
        public Task()
        {
            super();
        }
        public String toString()
        {
            return "Task [description="+description+",priority"+"]";
        }
        @Override
        public int compareTo(Task arg0) {
            // TODO Auto-generated method stub
            if (this.priority < arg0.priority)
                return -1;
            else if (this.priority> arg0.priority)
                return 1;
            return 0;
        }


}
```



```
import java.util.Iterator;
import java.util.PriorityQueue;

public class TestPriority {

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

```
		// TODO Auto-generated constructor stub
		PriorityQueue<Task> queue=new PriorityQueue<Task>();
		Task t1=new Task("Di Hoc",10);
		Task t2=new Task("Di Lam",2);
		Task  t3=new Task("Di Choi",5);
		queue.add(t1);
		queue.add(t2);
		queue.add(t3);

		Task t=queue.poll();
		System.out.println(t);
		System.out.println(queue.poll());
		//hoa dung vong lap ben duoi de duyet cac task
		Iterator<Task> itor=queue.iterator();
		while(itor.hasNext())
		{
			Task x=itor.next();
			System.out.println(x);
		}

	}

}
```

BÀI TẬP

Bài tập 1: Thêm chức năng sắp xếp danh sách sinh viên theo mã số ở ví dụ 1.

Bài tập này giúp sinh viên hiểu được chức năng sort của **Collections** và interface **Comparable.**

Bài tập 2: Hiệu chỉnh ví dụ 3, yêu cầu sử dụng **LinkedHashSet** để quản lý sản phẩm. Sinh viên hãy cho biết sự khác biệt giữa **HashSet** và **LinkedHashSet**.

Bài tập 3: Cải tiến ví dụ 4

Hãy sử dụng PriorityQueue để quản lý độ ưu tiên và độ phức tạp của công việc.

- Nếu công việc nào có độ ưu tiên lớn hơn thì được thực hiện trước
- Nếu công việc nào có cùng độ ưu tiên thì tính theo độ phức tạp, công việc nào có độ phức tạp nhiều nhất thì được thực hiện trước.