

Chương 3. Nền Tảng Của Ngôn Ngữ Java

Mục tiêu của bài:

Kết thúc chương này bạn có thể :

- Đọc hiểu một chương trình viết bằng Java
- Nắm bắt những khái niệm cơ bản về ngôn ngữ Java
- Nhận dạng các kiểu dữ liệu
- Nhận dạng các toán tử
- Định dạng kết quả xuất liệu (output) sử dụng các chuỗi thoát (escape sequence)
- Nhận biết các cấu trúc lập trình cơ bản

3.1 Cấu trúc một chương trình Java

Phần đầu của một chương trình Java xác định thông tin môi trường. Để làm được việc này, chương trình được chia thành các lớp hoặc các gói riêng biệt. Những gói này sẽ được chỉ dẫn trong chương trình. Thông tin này được chỉ ra với sự trợ giúp của phát biểu nhập “import”. Mỗi chương trình có thể có nhiều hơn một phát biểu nhập. Dưới đây là một ví dụ về phát biểu nhập:

import java. awt.*;

Phát biểu này nhập gói ‘awt’. Gói này dùng để tạo một đối tượng GUI. Ở đây java là tên của thư mục chứa tất cả các gói ‘awt’. Ký hiệu “*” chỉ tất cả các lớp thuộc gói này.

Trong java, tất cả các mã, bao gồm các biến và cách khai báo nên được thực hiện trong phạm vi một lớp. Bởi vậy, từng khai báo lớp được tiến hành sau một phát biểu nhập. Một chương trình đơn giản có thể chỉ có một vài lớp. Những lớp này có thể mở rộng thành các lớp khác. Mỗi phát biểu đều được kết thúc bởi dấu chấm phẩy “;”. Chương trình còn có thể bao gồm các ghi chú, chỉ dẫn. Khi dịch, chương trình dịch sẽ tự loại bỏ các ghi chú này.

Dạng cơ bản của một lớp được xác định như sau :

Class classname

{

/ Đây là dòng ghi chú*/*

int num1,num2; // Khai báo biến với các dấu phẩy giữa các biến

Show()

{

// Method body

statement (s); // Kết thúc bởi dấu chấm phẩy

}

}

“Token” là đơn vị riêng lẻ, nhỏ nhất, có ý nghĩa đối với trình biên dịch của một chương trình Java. Một chương trình java là tập hợp của các “token”

Các “token” được chia thành năm loại:

- Định danh (identifiers): Dùng để thể hiện tên biến, phương thức, hoặc các lớp. Chương trình biên dịch sẽ xác định các tên này là duy nhất trong chương trình. Khi khai báo định danh cần lưu ý các điểm sau đây :
 - Mỗi định danh được bắt đầu bằng một chữ cái, một ký tự gạch dưới hay dấu đôla (\$). Các ký tự tiếp theo có thể là chữ cái, chữ số, dấu \$ hoặc một ký tự được gạch dưới.
 - Mỗi định danh chỉ được chứa hai ký tự đặc biệt, tức là chỉ được chứa một ký tự gạch dưới và một ký tự dấu \$. Ngoài ra không được phép sử dụng bất kỳ ký tự đặc biệt nào khác.
 - Các định danh không được sử dụng dấu cách “ ” (space).
- Từ khoá/từ dự phòng (Keyword/Reserve Words): Một số định danh đã được Java xác định trước. Người lập trình không được phép sử dụng chúng như một định danh. Ví dụ ‘class’, ‘import’ là những từ khoá.
- Ký tự phân cách (separator): Thông báo cho trình biên dịch việc phân nhóm các phần tử của chương trình. Một vài ký tự phân cách của java được chỉ ra dưới đây:

{ } ; ,
- Nguyên dạng (literals): Là các giá trị không đổi trong chương trình. Nguyên dạng có thể là các số, chuỗi, các ký tự hoặc các giá trị Boolean. Ví dụ 21, ‘A’, 31.2, “This is a sentence” là những nguyên dạng.
- Các toán tử: Các quá trình xác định, tính toán được hình thành bởi dữ liệu và các đối tượng. Java có một tập lớn các toán tử. Chúng ta sẽ thảo luận chi tiết ở chương này.

3.2 Chương trình JAVA đầu tiên

Chúng ta hãy bắt đầu từ chương trình Java cổ điển nhất với một ứng dụng đơn giản. Chương trình sau đây cho phép hiển thị một thông điệp:

Chương trình 3.1

// This is a simple program called “First.java”

```
class First
```

```
{
```

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

```
    {
```

```
        System.out.println(“My first program in Java”);
```

```
    }
```

```
}
```

Tên file đóng vai trò rất quan trọng trong Java. Chương trình biên dịch Java chấp nhận phần mở rộng **.java**. Trong Java các mã cần phải gom thành các lớp. Bởi vậy tên lớp và tên file có thể trùng nhau. Do đó Java phân biệt rạch ròi chữ in hoa và chữ in thường (case-sensitive). Nói chung tên lớp và tên file nên khác nhau. Ví dụ tên file ‘First’ và ‘first’ là hai file khác nhau.

Để biên dịch mã nguồn, ta sử dụng trình biên dịch **java**. Trình biên dịch xác định tên của file nguồn tại dòng lệnh như mô tả dưới đây:

C:\jdk1,2,1\bin>javac First.Java

Trình dịch java tạo ra file First.class chứa các mã “bytecodes”. Những mã này chưa thể thực thi được. Để chương trình thực thi được ta cần dùng trình thông dịch “**java interpreter**”

Lệnh được thực hiện như sau:

C:\jdk1,1,1\bin>java First

Kết quả sẽ hiển thị trên màn hình như sau:

My first program in Java

3.2.1 Phân tích chương trình đầu tiên

// This is a simple program called “First.java”

Ký hiệu “//” dùng để thuyết minh dòng lệnh. Trình biên dịch sẽ bỏ qua dòng thuyết minh này. Java còn hỗ trợ thuyết minh nhiều dòng. Loại thuyết minh này có thể bắt đầu với /* và kết thúc với */

**/*This is a comment that
extends to two lines*/
/*This is
a multi line
comment */**

Dòng kế tiếp khai báo lớp có tên ‘First’. Để tạo một lớp thêm ta bắt đầu với từ khóa ‘class’, kế đến là tên lớp (và cũng chính là tên file).

class First

Tên lớp nói chung nên bắt đầu bằng chữ in hoa.

Từ khóa ‘class’ khai báo định nghĩa lớp. ‘First’ là định danh cho tên của lớp. Một định nghĩa lớp trọn vẹn không nằm giữa hai ngoặc móc (curly braces) đóng và mở. Các ngoặc này đánh dấu bắt đầu và kết thúc một khối các lớp được định nghĩa.

public static void main(String args[])

Đây là phương thức chính, từ đây chương trình bắt đầu việc thực thi của mình. Tất cả các ứng dụng java đều sử dụng một phương pháp “**main**” này. Chúng ta sẽ tìm hiểu từng từ trong phát biểu này.

Từ khóa ‘public’ là một chỉ định truy xuất. Nó cho biết thành viên của lớp có thể được truy xuất từ bất cứ đâu trong chương trình. Trong trường hợp này, phương thức “**main**” được khai báo ‘public’, bởi vậy JVM có thể truy xuất phương thức này.

Từ khóa ‘**static**’ cho phép **main** được gọi tới mà không cần tạo ra một thể hiện (instance) của lớp. Nhưng trong trường hợp này, bản copy của phương thức **main** được phép tồn tại trên bộ nhớ, thậm chí nếu không có một thể hiện của lớp đó được tạo ra. Điều này rất quan trọng vì JVM trước tiên gọi phương thức **main** để thực thi chương trình. Vì lý do này phương thức **main** cần phải là tĩnh (static). Nó không phụ thuộc vào các thể hiện của lớp được tạo ra.

Từ khóa ‘**void**’ thông báo cho máy tính biết rằng phương thức sẽ không trả lại bất cứ giá trị nào khi thực thi chương trình.

Phương thức '**main()**' sẽ thực hiện một số tác vụ nào đó, nó là điểm mốc mà từ đó tất cả các ứng dụng Java được khởi động.

'**String args[]**' là tham số dùng trong phương thức '**main**'. Các biến số trong dấu ngoặc đơn nhận từng thông tin được chuyển vào '**main**'. Những biến này là các tham số của phương thức. Thậm chí ngay khi không có một thông tin nào được chuyển vào '**main**', phương thức vẫn được thực hiện với các dữ liệu rỗng – không có gì trong dấu ngoặc đơn.

'**args[]**' là một mảng kiểu "String". Các đối số (arguments) từ các dòng lệnh được lưu vào mảng. Mã nằm giữa dấu ngoặc móc của '**main**' được gọi là '**method block**'. Các phát biểu được thực thi trong '**main**' cần được chỉ rõ trong khối này.

System.out.println("My first program in Java");

Dòng lệnh này hiển thị chuỗi "My first program in Java" trên màn hình. Phát biểu '**println()**' tạo ra một công xuất (output). Phương thức này cho phép hiển thị một chuỗi nếu chuỗi đó được đưa vào với sự trợ giúp của '**System.out**'. Ở đây '**System**' là một lớp đã định trước, nó cho phép truy nhập vào hệ thống và '**out**' là một chuỗi xuất được kết nối với đầu nhắc (console).

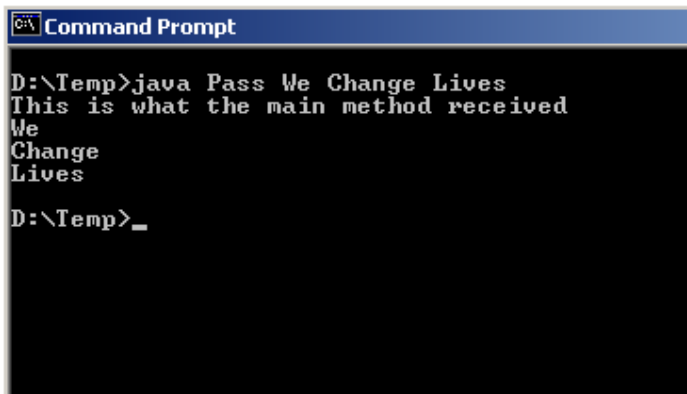
3.2.2 Truyền đối số trong dòng lệnh

Các mã sau đây cho ta thấy các tham số (argument) của các dòng lệnh được tiếp nhận như thế nào trong phương thức '**main**'.

Program 3.2

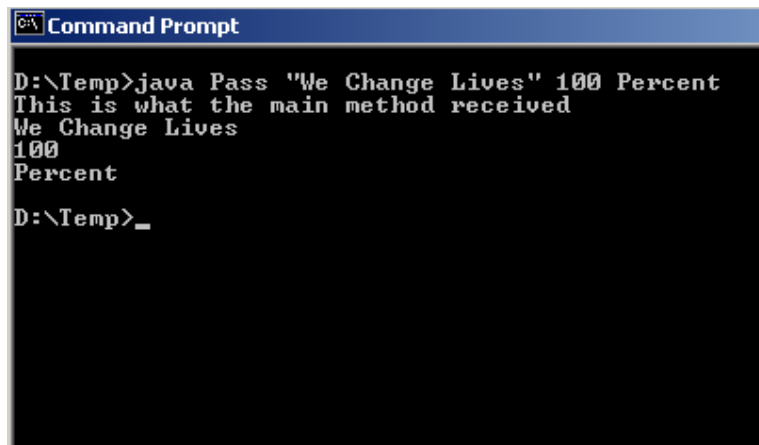
```
class Pass{  
public static void main(String parameters[])  
{  
System.out.println("This is what the main method received");  
System.out.println(parameters [0 ]);  
System.out.println(parameters [1 ]);  
System.out.println(parameters [2 ]);  
}  
}
```

Hình vẽ sau đây mô tả các đối tượng được thực hiện tại các dòng lệnh như thế nào



```
Command Prompt  
D:\Temp>java Pass We Change Lives  
This is what the main method received  
We  
Change  
Lives  
D:\Temp>_
```

Khi gặp một dấu trắng (space), có thể hiểu một chuỗi được kết thúc. Nhưng thông thường một chuỗi được kết thúc khi gặp dấu nhảy kíp. Hình vẽ dưới đây sẽ mô tả điều này.



```
C:\> Command Prompt

D:\Temp>java Pass 'We Change Lives' 100 Percent
This is what the main method received
We Change Lives
100
Percent
D:\Temp>_
```

3.3 Cơ bản về ngôn ngữ Java

Chương trình là tập hợp những hành động được sắp xếp theo một trật tự nhất định để máy tính có thể thực hiện được. Chương trình có thể được coi như một tài liệu hướng dẫn có chứa các thành phần được gọi là các biến và danh sách các hướng dẫn được gọi là phát biểu. Các phát biểu nói cho máy tính biết cần phải làm gì với các biến.

Biến là các giá trị có thể được thay đổi phụ thuộc vào điều kiện hoặc thông tin được nhập vào máy tính. Các biến được xác định nhờ các kiểu dữ liệu. Kiểu dữ liệu là một tập các dữ liệu với các giá trị có các đặc tính đã được xác định trước.

Các phát biểu dạng điều khiển quyết định việc thực thi từng phần trong chương trình. Chúng còn quyết định trật tự việc thực thi chương trình và số lần chương trình cần thực hiện. Giá trị nạp vào biến có thể định hướng cho chương trình hoạt động.

Chúng ta hãy bắt đầu với những khái niệm nền tảng của ngôn ngữ Java như lớp và phương thức, kiểu dữ liệu, biến, toán tử và cấu trúc điều khiển.

3.4 Các lớp đối tượng trong Java

Trong ngôn ngữ Java, lớp là một đơn vị mẫu có chứa các số liệu và các mã liên quan đến một thực thể nào đó. Chúng hình thành nền tảng của toàn bộ ngôn ngữ Java. Dữ liệu hoặc mã nguồn được viết ra luôn đặt bên trong một lớp. Khi xác định một lớp, bạn thực chất xác định một kiểu dữ liệu. Loại dữ liệu mới này được sử dụng để xác định các biến mà ta thường gọi là “đối tượng”. Đối tượng là các thể hiện (instance) của lớp. Tất cả các đối tượng đều thuộc về một lớp có chung đặc tính và hành vi. Mỗi lớp xác định một thực thể, trong khi đó mỗi đối tượng là một thể hiện thực sự.

Bạn còn có thể định nghĩa một lớp bên trong. Đây là một lớp kiểu xếp lồng vào nhau, các thể hiện (instance) của lớp này tồn tại bên trong thể hiện của một lớp che phủ chúng. Nó chi phối việc truy nhập đến các thể hiện thành phần của thể hiện bao phủ chúng.

3.4.1 Khai báo lớp

Khi bạn khai báo một lớp, bạn cần xác định dữ liệu và các phương thức xây dựng nên lớp đó.

Cú pháp:

```
class name
{ var_datatype variablename;
:
met_datatype methodname(parameter_list)
:
}
```

Trong đó:

class - Từ khoá xác định lớp

classname - Tên của lớp

var_datatype - kiểu dữ liệu của biến

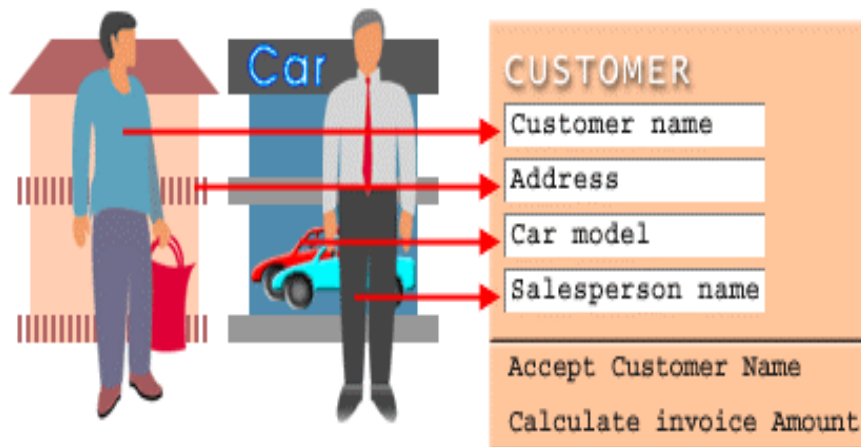
variablename - Tên của biến

met_datatype - Kiểu dữ liệu trả về của phương thức

methodname - Tên của phương thức

parameter_lits – Các tham số được dùng trong phương thức

Hình 3.3 mô tả bằng hình ảnh lớp “Khách hàng”. Những đặc điểm của lớp xác định các khoản mục dữ liệu được lưu cất, và các hành vi xác định các phương thức được tính đến. Đối tượng của lớp này sẽ lưu lại các chi tiết cá nhân của khách hàng.



Hình 3.3

Trong lớp “Khách hàng”, các khoản mục dữ liệu bao gồm:

- Tên khách hàng
- Địa chỉ
- Kiểu xe
- Tên người bán hàng

Các phương thức gồm:

- Chấp thuận các chi tiết của khách hàng

- In các hoá đơn

3.4.2 Các lớp xếp lồng vào nhau (nested classes)

Việc định nghĩa một lớp bên trong một lớp khác được gọi là lớp lồng (nesting). Lớp lồng chỉ nằm trong phạm vi lớp bao quanh nó. Có hai loại lớp lồng:

- Lớp kiểu tĩnh (static)

Lớp kiểu tĩnh được định nghĩa với từ khoá **static**. Lớp tĩnh có thể truy nhập vào các thành viên của lớp phủ nó thông qua một đối tượng. Do vậy lớp tĩnh thường ít được sử dụng.

- Lớp kiểu động (non static)

Lớp bên trong (inner) thuộc loại quan trọng nhất của các lớp kiểu lồng. Đó là các lớp non-static. Định nghĩa lớp bên trong chỉ có thể xác định được trong phạm vi lớp ngoài cùng. Lớp bên trong có thể truy nhập tất cả các thành viên của lớp bao nó, song không thể ngược lại. Đoạn chương trình sau mô tả lớp được tạo lập ra sao và sử dụng như thế nào:

```
class Outer
{
    //Outer class constructor

    class Inner
    {
        //Inner class constructor
    }
}
```

Cú pháp sau đây cho phép truy nhập vào lớp bên trong

Outer.Inner obj=new Outer().new Inner();

3.5 Kiểu dữ liệu

Các ứng dụng luôn yêu cầu một cổng xuất (output). Cổng nhập, cổng xuất, và kết quả của các quá trình tính toán tạo ra các dữ liệu. Trong môi trường tính toán, dữ liệu được phân lớp theo các tiêu chí khác nhau phụ thuộc vào bản chất của nó. Ở mỗi tiêu chí, dữ liệu có một tính chất xác định và có một kiểu thể hiện riêng biệt.

Java cung cấp một vài kiểu dữ liệu. Chúng được hỗ trợ trên tất cả các nền. Ví dụ, dữ liệu loại int (integer) của Java được thể hiện bằng 4 bytes trong bộ nhớ của tất cả các loại máy bất luận ở đâu chạy chương trình Java. Bởi vậy các chương trình Java không cần phải thay đổi khi chạy trên các nền khác nhau.

Trong Java kiểu dữ liệu được chia thành hai loại:

- Các kiểu dữ liệu nguyên thủy (primitive)
- Các kiểu dữ liệu tham chiếu (reference)

3.5.1 Dữ liệu kiểu nguyên thủy

Java cung cấp tám kiểu dữ liệu nguyên thủy

Chương trình đào tạo kỹ thuật viên quốc tế
Core Java

Kiểu dữ liệu	Độ dài theo số bit	Phạm vi	Mô tả
byte	8	-128 đến 127	Số liệu kiểu byte là một loại điển hình dùng để lưu trữ một giá trị bằng một byte. Chúng được sử dụng rộng rãi khi xử lý một file văn bản
char	16	'\u0000' to '\uffff'	Kiểu Char sử dụng để lưu tên hoặc các dữ liệu ký tự. Ví dụ tên người lao động
Boolean	1	"True" hoặc "False"	Dữ liệu boolean dùng để lưu các giá trị "Đúng" hoặc "sai" Ví dụ : Người lao động có đáp ứng được yêu cầu của công ty hay không ?
short	16	-32768 đến 32767	Kiểu short dùng để lưu các số có giá trị nhỏ dưới 32767. Ví dụ số lượng người lao động.
Int	32	-2,147,483,648 đến +2,147,483,648	Kiểu int dùng để lưu một số có giá trị lớn đến 2,147,483,648. Ví dụ tổng lương mà công ty phải trả cho nhân viên.
long	64	-9,223,372,036'854,775,808 đến +9,223,372,036'854,775,808	Kiểu long được sử dụng để lưu một số có giá trị rất lớn đến 9,223,372,036'854,775,808. Ví dụ dân số của một nước
Float	32	-3.40292347E+38 đến +3.40292347E+38	Kiểu float dùng để lưu các số thập phân đến 3.40292347E+38 Ví dụ : giá thành sản phẩm
double	64	-1,79769313486231570E+308 đến +1,79769313486231570E+308	Kiểu double dùng để lưu các số thập phân có giá trị lớn đến 1,79769313486231570E+308 Ví dụ giá trị tín dụng của ngân hàng nhà nước.

Bảng 3.1 Dữ liệu kiểu nguyên thủy

3.5.2 Kiểu dữ liệu tham chiếu (reference)

Trong Java có 3 kiểu dữ liệu tham chiếu

Kiểu dữ liệu	Mô tả
Mảng (Array)	Tập hợp các dữ liệu cùng loại. Ví dụ : tên sinh viên

Lớp (Class)	Tập hợp các biến và các phương thức. Ví dụ : lớp “Sinh viên” chứa toàn bộ các chi tiết của một sinh viên và các phương thức thực thi trên các chi tiết đó.
Giao diện (Interface)	Là một lớp trừu tượng được tạo ra để bổ sung cho các kế thừa đa lớp trong Java.

Bảng 3.2 Dữ liệu kiểu tham chiếu

3.5.3 Ép kiểu (Type casting)

Có thể bạn sẽ gặp tình huống khi cộng một biến có dạng **integer** với một biến có dạng **float**. Để xử lý tình huống này, Java sử dụng tính năng ép kiểu (type casting) của các phần mềm trước đó C, C++. Lúc này một kiểu dữ liệu sẽ chuyển đổi sang kiểu khác. Khi sử dụng tính chất này, bạn cần thận trọng vì khi điều chỉnh dữ liệu có thể bị mất.

Đoạn mã sau đây thực hiện phép cộng một giá trị dấu phẩy động (float) với một giá trị nguyên (integer).

```
Float c=34.896751F;
```

```
Int b = (int)c +10;
```

Đầu tiên giá trị dấu phẩy động **c** được đổi thành giá trị nguyên 34. Sau đó nó được cộng với 10 và kết quả là giá trị 44 được lưu vào **b**.

Sự nói rộng (widening) – quá trình làm tròn số theo hướng nói rộng không làm mất thông tin về độ lớn của mỗi giá trị. Biến đổi theo hướng nói rộng chuyển một giá trị sang một dạng khác có độ rộng phù hợp hơn so với nguyên bản. Biến đổi theo hướng lại thu nhỏ lại (narrowwing) làm mất thông tin về độ lớn của giá trị được chuyển đổi. Chúng không được thực hiện khi thực hiện phép gán. Ở ví dụ trên giá trị thập phân sau dấu phẩy sẽ bị mất.

3.6 Các biến

Các ứng dụng sử dụng các biến để lưu trữ các dữ liệu cần thiết hoặc các dữ liệu được tạo ra trong quá trình thực thi chương trình. Các biến được xác định bởi một tên biến và có một phạm vi tác động. Phạm vi tác động của biến được xác định một cách rõ ràng trong chương trình. Mỗi biến được khai báo trong một khối chương trình chỉ có tác động trong phạm vi khối đó, không có ý nghĩa và không được phép truy nhập từ bên ngoài khối.

Việc khai báo một biến bao gồm 3 thành phần: kiểu biến, tên của nó và giá trị ban đầu được gán cho biến (không bắt buộc). Để khai báo nhiều biến ta sử dụng dấu phẩy để phân cách các biến, Khi khai báo biến, luôn nhớ rằng Java phân biệt chữ thường và chữ in hoa (case - sensitive).

Cú pháp:

```
Datatype identifier [=value] [, identifier [=value]... ];
```

Để khai báo một biến nguyên (int) có tên là **counter** dùng để lưu giá trị ban đầu là 1, ta có thể thực hiện phát biểu sau đây:

```
int counter = 1;
```

Java có những yêu cầu hạn chế đặt tên biến mà bạn có thể gán giá trị vào. Những hạn chế này cũng giống các hạn chế khi đặt tên cho các định danh mà ta đã thảo luận ở các phần trước của chương này.

3.6.1 Khai báo mảng

Mảng được dùng để lưu trữ các khoản mục (items) của cùng một kiểu dữ liệu trên những vùng nhớ liên tục. Mỗi lần ta khai báo kích thước của một mảng, nó sẽ không bị thay đổi. Dữ liệu trên mảng có thể là kiểu dữ liệu nguyên thủy hoặc đối tượng. Cũng như các biến, ta có thể gán các giá trị vào mảng tại các phần tử được tạo ra trong mảng. Nếu những giá trị này không tồn tại, Java sẽ gán giá trị mặc định vào tất cả các phần tử của mảng phụ thuộc vào kiểu dữ liệu. Ví dụ: nếu kiểu dữ liệu là nguyên (int) thì giá trị mặc định ban đầu sẽ là “zero”.

Mảng có thể được khai báo bằng ba cách:

Cách khai báo	Mô tả	Cú pháp	Ví dụ
Chỉ đơn thuần khai báo	Chỉ đơn thuần khai báo mảng	Datatype identifier[]	char ch[] ; khai báo mảng ký tự có tên ch
Khai báo và tạo mảng	Khai báo và cấp phát bộ nhớ cho các phần tử mảng sử dụng từ “new”	Datatype identifier[] = new datatype [size]	char ch[] = new char [10] ; Khai báo một mảng ch và lưu trữ 10 ký tự
Khai báo, kiến tạo và khởi tạo	Khai báo mảng, cấp phát bộ nhớ cho nó và gán các giá trị ban đầu cho các phần tử của mảng	Datatype identifier[] = {value1,value2...valueN };	char ch [] = { 'A', 'B', 'C', 'D' }; khai báo mảng ch và lưu 4 chữ cái kiểu ký tự

Bảng 3.3 Khai báo mảng

Để xác định tên và số phần tử của mảng ta cần xem xét các phần tử mảng. Số phần tử bắt đầu với 0 cho phần tử đầu, 1 cho phần tử thứ hai và cứ tiếp như vậy.

3.7 Phương thức trong một lớp (method)

Phương thức xác định giao diện cho phần lớn các lớp. Trong khi đó Java cho phép bạn định nghĩa các lớp mà không cần phương thức. Bạn cần định nghĩa phương thức truy cập dữ liệu mà bạn đã lưu trong một lớp.

Phương thức được định nghĩa như một hành động hoặc một tác vụ thật sự của đối tượng. Nó còn được định nghĩa như một hành vi mà trên đó các thao tác cần thiết được thực thi.

Cú pháp

access_specifier modifier datatype method_name(parameter_list)

{ //body of method

}

Trong đó:

access_specifier: Chỉ định truy cập vào phương thức.

modifier: Cho phép bạn gán các thuộc tính cho phương thức.

datatype: Kiểu dữ liệu mà giá trị của nó được phương thức trả về. Nếu không có một giá trị

nào được trả về, kiểu dữ liệu có thể là **void**.

method_name: Tên của phương thức

parameter_list: Chứa tên của tham số được sử dụng trong phương thức và kiểu dữ liệu. Dấu phẩy được dùng để phân cách các tham số.

Ví dụ khai báo phương thức trong một lớp

Đoạn mã sau đây định nghĩa lớp **Temp** chứa một giá trị nguyên (int). Lớp này chứa hai phương thức là: **show()** và **main()**. Cả hai phương thức đều có khả năng truy cập bên ngoài lớp khi chúng được khai báo như **public**. Nếu chúng không trả về một giá trị nào, kiểu dữ liệu trả về được định nghĩa như kiểu **void**.

Phương thức **show()** hiển thị một giá trị của biến **x**. Ở phương thức **main()**, hai thí dụ của đối tượng thuộc lớp **Temp** được khai báo. Đối tượng thứ nhất gồm giá trị mặc định của biến **x**. Nó được hiển thị ngay khi gọi phương thức **show()** lần đầu tiên. Giá trị của **x** được thay đổi dùng cho đối tượng thứ hai. Nó được hiển thị khi ta gọi phương thức **show()** lần thứ hai.

Chương trình 3.3

Class Temp

```
{ static int x=10;//variable
public static void show();//method
{ System.out.println(x);
}
public static void main(String args[])
{ Temp t = new Temp();// object 1
t.show();//method call
Temp t1=new Temp();// object 2
t1x=20;
t1.show();
}
}
```

3.7.1 Các chỉ định truy xuất của phương thức

Các chỉ định truy xuất dùng để giới hạn khả năng truy nhập vào một phương thức. Java cung cấp các chỉ định truy xuất sau đây:

- **Công cộng (Public):** Phương thức có chỉ định truy xuất public có thể được nhìn thấy từ mọi gói hoặc mọi lớp.
- **Bảo vệ (Protected):** Các lớp mở rộng từ lớp hiện hành trong cùng một gói, hoặc tại các gói khác nhau có thể truy cập các phương thức sử dụng chỉ định truy xuất này.
- **Riêng tư (Private):** Phương thức riêng tư có thể được truy cập nhờ phương thức công cộng trên cùng một lớp.

3.7.2 Các bổ nghĩa loại phương thức

Các bổ nghĩa loại phương thức cho phép ta thiết lập các thuộc tính của phương thức. Java cung cấp các bổ nghĩa sau:

- **Tĩnh (static):** Các trạng thái mà phương thức có thể được thay đổi mà không cần đến đối tượng. Nó chỉ được sử dụng đối với các dữ liệu và các phương thức tĩnh.
- **Trừu tượng (abstract):** Ngụ ý rằng phương thức không có một mã cụ thể (code) và nó sẽ được bổ sung ở các lớp con (subclass). Loại phương thức này được sử dụng trong các lớp kế thừa.
- **Kết thúc (final):** Phương thức không thể được thừa kế hoặc ghi đè (Overridden).
- **Tự nhiên (native):** Chỉ ra rằng phần thân của phương thức được viết trên các ngôn ngữ khác Java ví dụ C, hoặc C++.
- **Đồng bộ (synchronized):** Sử dụng với phương thức trong quá trình thực thi threads. Nó cho phép chỉ một thread được truy cập vào khối mã vào một thời điểm.
- **Linh hoạt (volatile):** Được sử dụng với các biến để thông báo rằng giá trị của biến có thể được thay đổi vài lần khi thực thi chương trình và giá trị của nó không được ghi vào thanh ghi.

Bảng dưới đây chỉ ra nơi mà các bổ nghĩa được sử dụng:

Bổ nghĩa	Phương thức	Biến	Lớp
public	Yes	Yes	Yes
private	Yes	Yes	Yes (Nested class)
protected	Yes	Yes	Yes (Nested class)
abstrac	Yes	No	Yes
final	Yes	Yes	Yes
native	Yes	No	No
volatile	No	Yes	No

Bảng 3.4 Sử dụng các bổ nghĩa

3.7.3 Nạp chồng (overloading) và Ghi đè (overriding) phương thức

Những phương thức được nạp chồng (**overload**) là những phương thức trong cùng một lớp, có cùng một tên song có danh sách các tham số khác nhau. Sử dụng việc nạp chồng phương thức để thực thi các phương thức giống nhau đối với các kiểu dữ liệu khác nhau. Ví dụ phương thức **swap()** có thể bị nạp chồng (overload) bởi các tham số của kiểu dữ liệu khác như **integer**, **double** và **float**

Phương thức được ghi đè (**overridden**) là phương thức có mặt ở lớp cha (superclasses) cũng như ở các lớp kế thừa. Phương thức này cho phép một lớp tổng quát chỉ định các phương thức sẽ là phương thức chung trong các lớp con. Ví dụ lớp xác định phương thức tổng quát 'area()'. Phương thức này có thể được hiện thực trong một lớp con để tìm diện tích một hình cụ thể như hình chữ nhật, hình vuông ...

Phương thức nạp chồng là một hình thức đa hình (polymorphism) trong quá trình biên dịch (compile). Còn phương thức ghi đè là một hình thức đa hình trong quá trình thực thi (runtime).

Đoạn chương trình sau mô tả nạp chồng phương thức được thực hiện như thế nào

```
//defined once
```

```
protected void performTask(double salary){
.....
System.out.println("Salary is : " + salary);
....
}
//overloaded –defined the second time with different parameters
protected void performTask(double salary,int bonus){
.....
System.out.println("Total Salary is: " + salary+bonus);
....
}
```

Phương thức khởi tạo (Constructor) của lớp có thể bị nạp chồng (overload)

Phương thức ghi đè (Overriden) được định nghĩa lại ở các lớp con. Đoạn mã sau đây mô tả phương thức ghi đè.

Ở đây ta dùng từ khóa “this” biểu thị đối tượng hiện hành, trong khi đó ‘super’ được sử dụng để chỉ đối tượng lớp cha.

Phương thức ghi đè không phải là phương thức tĩnh (static). Nó là loại động (non-static).

Các đoạn mã sau đây mô tả việc thực thi ghi đè phương thức trong Java.

```
class SupperClass // Tạo lớp cơ bản
{
int a;
Super(Class() // constuctor
{
}
SuperClass(int b) //overloaded constructor
{
a=b;
}
class Subclass Extends SupperClass { // derriving a class
int a;
SubClass(int a) { //subclass constructor
Thí.a;
}
public void message(){ // overriding the base class message()
Sýtem.out.println("In the sub class");
```

```
}  
}
```

Bây giờ chúng ta sẽ tạo ra một đối tượng lớp cha và gán một lớp nhỏ tham chiếu đến nó như sau:

SuperClass spObj=new Subclass(22);

Câu lệnh ‘spObj.message’ thuộc phương thức nhóm con. Ở đây kiểu đối tượng được gán cho ‘spObj’ sẽ chỉ được xác định khi chương trình thực thi. Điều này được biết dưới khái niệm ‘liên kết động’ (dynamic binding).

3.7.4 Phương thức khởi tạo lớp

Phương thức khởi tạo lớp là một loại phương thức đặc biệt rất khác với các kiểu khởi tạo cơ bản. Nó không có kiểu trả về. Nó có tên trùng với tên của lớp. Hàm khởi tạo lớp thực thi như một phương thức hoặc một chức năng bình thường song nó không trả về bất cứ một giá trị nào. Nói chung chúng được dùng để khởi tạo các biến thành viên của một lớp và nó được gọi bất cứ lúc nào bạn tạo ra đối tượng của lớp đó.

Phương thức khởi tạo lớp có hai loại:

- **Tường minh (explicit):** Bạn có thể lập trình những phương thức khởi tạo lớp khi định nghĩa lớp. Khi tạo một đối tượng của một lớp, những giá trị mà bạn truyền vào phải khớp với những tham số của phương thức khởi tạo (số lượng, thứ tự và kiểu dữ liệu của các tham số)
- **Ngầm định (Implicit):** Khi bạn không định nghĩa một hàm khởi tạo cho một lớp, JVM cung cấp một giá trị mặc định hay một phương thức khởi tạo ngầm định.

Bạn có thể định nghĩa nhiều phương thức khởi tạo cho một lớp. Giống như các phương thức khác, phương thức khởi tạo lớp có thể bị nạp chồng (overload)

Ví dụ một phương thức khởi tạo:

Đoạn mã sau đây định nghĩa một phương thức khởi tạo tường minh (explicit) cho một lớp Employee. Phương thức khởi tạo bao gồm tên và tuổi. Chúng được coi như các tham số và gán các giá trị của chúng vào các biến của lớp. Chú ý rằng từ khoá ‘this’ được sử dụng để tham chiếu đến đối tượng hiện hành của lớp.

Chương trình 3.4

Class Employee

```
{ String name;  
int age;  
Employee (String var name,int varage)  
{ this.name = varname;  
this.age = varage;  
}  
public static void main (String arg[])  
{  
Employee e = new Employee ("Allen".30);
```

```
}  
}
```

3.7.5 Phương thức khởi tạo của lớp dẫn xuất

Phương thức khởi tạo của một lớp dẫn xuất có tên trùng với tên của lớp dẫn xuất đó. Câu lệnh dùng để gọi phương thức khởi tạo của một lớp dẫn xuất phải là câu lệnh đầu tiên trên phương thức khởi tạo của lớp con đó. Lý do là lớp cha hình thành trước khi có các lớp dẫn xuất.

3.8 Các toán tử

Một chương trình thực tế bao hàm việc tạo ra các biến. Các toán tử kết hợp các giá trị đơn giản hoặc các biểu thức con thành những biểu thức mới, phức tạp hơn và có thể trả về các giá trị. Điều này có hàm ý tạo ra các toán tử luận lý, số học, quan hệ và so sánh trên các biểu thức.

Java cung cấp nhiều dạng toán tử. Chúng bao gồm:

- Toán tử số học
- Toán tử dạng bit
- Toán tử quan hệ
- Toán tử luận lý
- Toán tử điều kiện
- Toán tử gán

3.8.1 Các toán tử số học

Các toán hạng của các toán tử số học phải ở dạng số. Các toán hạng kiểu Boolean không sử dụng được, song các toán hạng ký tự cho phép sử dụng loại toán tử này. Một vài kiểu toán tử được liệt kê trong bảng dưới đây.

Toán tử	Mô tả
+	Cộng. Trả về giá trị tổng hai toán hạng Ví dụ 5+3 trả về kết quả là 8
-	Trừ Trả về giá trị khác nhau giữa hai toán hạng hoặc giá trị phủ định của toán hạng. Ví dụ 5-3 kết quả là 2 và -10 trả về giá trị âm của 10
*	Nhân Trả về giá trị là tích hai toán hạng. Ví dụ 5*3 kết quả là 15
/	Chia Trả về giá trị là thương của phép chia Ví dụ 6/3 kết quả là 2
%	Phép lấy modulo Giá trị trả về là phần dư của toán tử chia Ví dụ 10%3 giá trị trả về là 1
++	Tăng dần Tăng giá trị của biến lên 1. Ví dụ a++ tương đương với a=a+1
--	Giảm dần Giảm giá trị của biến 1 đơn vị. Ví dụ a-- tương đương với a=a-1
+=	Cộng và gán giá trị

	Cộng các giá trị của toán hạng bên trái vào toán hạng bên phải và gán giá trị trả về vào toán hạng bên trái. Ví dụ $c+=a$ tương đương $c=c+a$
$-=$	Trừ và gán giá trị Trừ các giá trị của toán hạng bên trái vào toán hạng bên phải và gán giá trị trả về vào toán hạng bên trái. Ví dụ $c-=a$ tương đương với $c=c-a$
$*=$	Nhân và gán Nhân các giá trị của toán hạng bên trái với toán hạng bên phải và gán giá trị trả về vào toán hạng bên trái. Ví dụ $c*=a$ tương đương với $c=c*a$
$/=$	Chia và gán Chia giá trị của toán hạng bên trái cho toán hạng bên phải và gán giá trị trả về vào toán hạng bên trái. Ví dụ $c/=a$ tương đương với $c=c/a$
$\%=$	Lấy số dư và gán Chia giá trị của toán hạng bên trái cho toán hạng bên phải và gán giá trị số dư vào toán hạng bên trái. Ví dụ $c\%=a$ tương đương với $c=c\%a$

Bảng 3.5 Các toán tử số học

Chương trình sau mô tả việc sử dụng toán tử số học

```
class ArithmeticOp {
public static void main(String args[]){
int p=5,q=12,r=20,s;
s=p+q;
System.out.println("p+q is "+s);
s=p%q;
System.out.println("p%q is "+s);
s*=r;
System.out.println("s*=r is "+s);
System.out.println("Value of p before operation is "+p);
p++;
System.out.println("Value of p after operation is "+p);
double x=25.75,y=14.25,z;
z=x-y;
System.out.println("x-y is "+z);
z-=2.50;
System.out.println("z-=2.50 is "+z);
System.out.println("Value of z before operation is "+z);
z--;
```



```
System.out.println("Value of z after operation is "+z);
```

```
Z=x/y;
```

```
System.out.println("x/y is" +z);
```

```
}
```

```
}
```

Đầu ra của chương trình là

p+q is 17

p%q is 5

s=r is 100*

Value of p before operation is 9.0

Value of z after operation is 8.0

x/y is 1.8070175438596429

3.8.2 Toán tử Bit

Các toán tử dạng Bit cho phép ta tạo những Bit riêng biệt trong các kiểu dữ liệu nguyên thủy. Toán tử Bit dựa trên cơ sở đại số Boolean. Nó thực hiện phép tính trên hai đối số là các bit để tạo ra một kết quả mới. Một vài dạng toán tử kiểu này được liệt kê dưới đây

Toán tử	Mô tả
~	Phủ định (NOT) Trả về giá trị phủ định của một số .Ví dụ a=10 thì ~a=-10
&	Toán tử AND Trả về giá trị là 1 nếu các toán hạng là 1 và 0 trong các trường hợp khác.Ví dụ nếu a=1 và b=0 thì a&b trả về giá trị 0
	Toán tử OR Trả về giá trị là 1 nếu một trong các toán hạng là 1 và 0 trong các trường hợp khác.Ví dụ nếu a=1 và b=0 thì a b trả về giá trị 1
^	Exclusive OR Trả về giá trị là 1 nếu chỉ một trong các toán hạng là 1 và trả về 0 trong các trường hợp khác. Ví dụ nếu a=1 và b=1 thì a^b trả về giá trị 0
>>	Dịch sang phải Chuyển toàn bộ các bit của một số sang phải một vị trí , giữ nguyên dấu của số âm.Toán hạng b bên trái 1 à số bị dịch còn số bên phải chỉ số vị trí mà các bit cần dịch. Ví dụ x=37 tức là 00011111 vậy x>>2 sẽ là 00000111.
<<	Dịch sang trái Chuyển toàn bộ các bit của một số sang trái một vị trí , giữ nguyên dấu của số âm.Toán hạng bên trái là số bị dịch còn số bên phải chỉ số vị trí mà các bit cần dịch. Ví dụ x=37 tức là 00011111 vậy x<<2 sẽ là 00000111

Bảng 3.6 Các toán tử Bit

3.8.3 Các toán tử quan hệ

Các toán tử quan hệ kiểm tra mối quan hệ giữa hai toán hạng. Kết quả của một biểu thức có dùng các toán tử quan hệ là những giá trị Boolean (logic “đúng” hoặc “sai”). Các toán tử quan hệ được sử dụng trong các cấu trúc điều khiển.

Toán tử	Mô tả
==	So sánh bằng Toán tử này kiểm tra sự tương đương của hai toán hạng Ví dụ if (a==b) trả về giá trị “True” nếu giá trị của a và b như nhau
!=	So sánh khác Kiểm tra sự khác nhau của hai toán hạng Ví dụ if(a!=b) Trả về giá trị “true” nếu a khác b
>	Lớn hơn Kiểm tra giá trị của toán hạng bên phải lớn hơn toán hạng bên trái hay không Ví dụ if(a>b) . Trả về giá trị “true” nếu a lớn hơn b, ngược lại (nhỏ hơn hoặc bằng), trả về ‘False’
<	Nhỏ hơn Kiểm tra giá trị của toán hạng bên phải có nhỏ hơn toán hạng bên trái hay không Ví dụ if(a<b) . Trả về giá trị “true” nếu a nhỏ hơn b , ngược lại (lớn hơn hoặc bằng trả về ‘False’
>=	Lớn hơn hoặc bằng Kiểm tra giá trị của toán hạng bên phải có lớn hơn hoặc bằng toán hạng bên trái hay không Ví dụ if(a>=b) . Trả về giá trị “true” nếu a lớn hơn hoặc bằng b , ngược lại (nhỏ hơn trả về ‘False’
<=	Nhỏ hơn hoặc bằng Kiểm tra giá trị của toán hạng bên phải có nhỏ hơn hoặc bằng toán hạng bên trái hay không Ví dụ if(a<=b) . Trả về giá trị “true” nếu a nhỏ hơn hoặc bằng b , ngược lại (lớn hơn trả về ‘False’

Bảng 3.6 Các toán tử quan hệ

Đoạn chương trình sau đây mô tả việc sử dụng các toán tử quan hệ

Chương trình 3.6

```
class RelationalOp {
    public static void main (String args[]){
        float a= 10.0F;
        double b=10.0;
        if (a== b)
            System.out.println(a and b are equal");
        else
            System.out.println("a and b are not equal");
    }
}
```

}

Kết quả chương trình sẽ hiển thị

a and b are not equal

Trong chương trình trên cả a và b là những số có dấu phẩy động, dạng dữ liệu có khác nhau, a là kiểu float còn b là kiểu double. Tuy vậy chúng không phải là cùng một kiểu. Bởi vậy khi kiểm tra giá trị của các toán hạng, kiểu dữ liệu cần phải được kiểm tra.

3.8.4 Các toán tử logic

Các toán tử logic làm việc với các toán hạng Boolean. Một vài toán tử kiểu này được chỉ ra dưới đây

Toán tử	Mô tả
&	Và (AND) Trả về một giá trị “Đúng” (True) nếu chỉ khi cả hai toán tử có giá trị “True” Ví dụ : if(science>90) AND (math>75) thì gán “Y” - có đủ tư cách ra nhập nhóm học tập.
I	Hoặc (OR) Trả về giá trị “True” nếu một giá trị là True hoặc cả hai đều là True Ví dụ Nếu age_category is ‘Senior_citizen’ OR special_category is ‘handicapped’ hạ giá tua lữ hành hoặc cả hai điều kiện đều được thực hiện
^	XOR Trả về giá trị True nếu chỉ một trong các giá trị là True .các trường hợp còn lại cho giá trị False (sai)
!	Toán hạng đơn từ NOT. Chuyển giá trị từ True sang False và ngược lại. Ví dụ : Quá trình thực thi các dòng lệnh tiếp tục cho đến khi kết thúc chương trình.

Bảng 3.8 Các toán tử logic

3.8.5 Các toán tử điều kiện

Toán tử điều kiện là một loại toán tử đặc biệt vì nó gồm ba thành phần cấu thành biểu thức điều kiện

Cú pháp :

biểu thức 1?biểu thức 2: biểu thức 3;

biểu thức 1

Điều kiện luận lý (Boolean) mà nó trả về giá trị True hoặc False

biểu thức 2

Giá trị trả về nếu biểu thức 1 xác định là True

biểu thức 3

Giá trị trả về nếu biểu thức 1 xác định là False

Câu lệnh sau đây kiểm tra có những người đi làm bằng vé tháng có tuổi lớn hơn 65 không và gán một tiêu chuẩn cho họ. Nếu những người này có tuổi là 55, tiêu chuẩn gán là “Regular”

CommuterCategory = (CommuterAge>65)? “Senior Citizen”: “Regular”

3.8.6 Toán tử gán

Toán tử gán (=) dùng để gán một giá trị vào một biến. Bạn nên gán nhiều giá trị đến nhiều biến cùng một lúc.

Ví dụ đoạn lệnh sau gán một giá trị cho biến **num**. Thì giá trị trong biến **num** được gán cho nhiều biến trên một dòng lệnh đơn.

```
int num = 20000;
```

```
int p,q,r,s;
```

```
p=q=r=s=num;
```

Dòng lệnh cuối cùng được thực hiện từ phải qua trái. Đầu tiên giá trị ở biến num được gán cho ‘s’, sau đó giá trị của ‘s’ được gán cho ‘r’ và cứ tiếp như vậy.

3.8.7 Thứ tự ưu tiên của các toán tử

Các biểu thức được viết ra nói chung gồm nhiều toán tử. Thứ tự ưu tiên quyết định trật tự thực hiện các toán tử trên các biểu thức. Bảng dưới đây liệt kê thứ tự thực hiện các toán tử trong Java

Thứ tự	Toán tử
1.	Các toán tử đơn như +, -, ++, --
2.	Các toán tử số học và các toán tử dịch như *, /, +, -, <<, >>
3.	Các toán tử quan hệ như >, <, >=, <=, =, !=
4.	Các toán tử logic và Bit như &&, , &, , ^
5.	Các toán tử gán như =, *=, /=, +=, -=

Bảng 3.9 Trật tự ưu tiên

3.8.8 Thay đổi thứ tự ưu tiên

Để thay đổi thứ tự ưu tiên trên một biểu thức, bạn có thể sử dụng dấu ngoặc đơn (). Từng phần của biểu thức được giới hạn trong ngoặc đơn được thực hiện trước tiên. Nếu bạn sử dụng nhiều ngoặc đơn lồng nhau thì toán tử nằm trong ngoặc đơn phía trong sẽ thực thi trước, sau đó đến các vòng phía ngoài. Nhưng trong phạm vi một ngoặc đơn thì quy tắc thứ tự ưu tiên vẫn giữ nguyên tác dụng.

3.9 Định dạng dữ liệu xuất dùng chuỗi thoát (Escape sequence)

Nhiều khi dữ liệu xuất được hiển thị trên màn hình, chúng cần phải được định dạng. Việc định dạng này cần sự trợ giúp của chuỗi thoát (Escape sequences) do Java cung cấp

Chúng ta hãy xem ví dụ dưới đây

```
System.out.println(“Happy\tBirthday”);
```

Cho ta dữ liệu xuất như sau : **Happy Birthday**

Bảng dưới đây liệt kê một số chuỗi thoát và công dụng của chúng

Chuỗi thoát	Mô tả
\n	Đưa con trỏ đến dòng kế tiếp (Bắt đầu một dòng mới)
\r	Đưa con trỏ về đầu dòng (Giống ký tự carriage return)

\t	Đưa con trỏ đến vị trí Tab-Stop (Như vị trí Tab của ký tự)
\\	In vạch chéo ngược (backslash)
\'	In dấu nháy đơn (')
\"	In dấu nháy kép (")

Bảng 3.10 Các chuỗi thoát

3.10 Điều khiển luồng

Tất cả các môi trường phát triển ứng dụng đều cung cấp một quy trình ra quyết định (decision-making) được gọi là điều khiển luồng, nó trực tiếp thực thi các ứng dụng. Điều khiển luồng cho phép người phát triển phần mềm tạo một ứng dụng dùng để kiểm tra sự tồn tại của một điều kiện nào đó và ra quyết định phù hợp với điều kiện đó.

Vòng lặp là một cấu trúc chương trình giúp bạn có thể dùng để thực hiện việc lặp lại các hành động khi thực thi chương trình mà không cần viết lại các đoạn chương trình nhiều lần.

Điều khiển rẽ nhánh

- Mệnh đề if-else
- Mệnh đề switch-case

Vòng lặp (Loops)

- Vòng lặp while
- Vòng lặp do-while
- Vòng lặp for

3.10.1 Câu lệnh if-else

Câu lệnh if-else kiểm tra kết quả của một điều kiện và thực thi một thao tác phù hợp trên cơ sở kết quả đó. Dạng của câu lệnh if-else rất đơn giản

Cú pháp

If (condition)

{ action 1 statements; }

else

{ action 2 statements; }

Condition: Biểu thức Boolean như toán tử so sánh. Biểu thức này trả về giá trị True hoặc False

action 1: Các dòng lệnh được thực thi khi giá trị trả về là True

else: Từ khóa xác định các câu lệnh tiếp sau được thực hiện nếu điều kiện trả về giá trị False

action 2: Các câu lệnh được thực thi nếu điều kiện trả về giá trị False

Đoạn chương trình sau kiểm tra xem các số là chẵn hay lẻ và hiển thị thông báo phù hợp

Chương trình 3.7

Class CheckNumber

{

public static void main(String args[])

```
{
int num =10;
if(num %2 == 0
System.out.println (num+ "is an even number");
else
System.out.println (num + "is an odd number");
}}
```

Ở đoạn chương trình trên num được gán giá trị nguyên là 10. Trong câu lệnh **if-else** điều kiện **num %2** trả về giá trị 0 và điều kiện thực hiện là True. Thông báo "10 is an even number" được in ra. Lưu ý rằng cho đến giờ chỉ có một câu lệnh tác động được viết trong đoạn "if" và "else", bởi vậy không cần phải được đưa vào dấu ngoặc móc.

Hình vẽ dưới đây mô tả cách dùng **if-else**

Tên		Tom	Else-if	John	else	Henry
Điều kiện	if	Giám đốc				
Tăng lương	✓		✓		✗	

Hình 3.4 If-else

3.10.2 Câu lệnh switch-case

Phát biểu switch-case có thể được sử dụng tại câu lệnh if-else. Nó được sử dụng trong tình huống một biểu thức cho ra nhiều kết quả. Việc sử dụng câu lệnh switch-case cho phép việc lập trình dễ dàng và đơn giản hơn.

Cú pháp

switch (expression)

```
{
case 'value':action 1 statement;
break;
case 'value':action 2 statement;
break;
:
:
case 'valueN': actionN statement (s);
}
```

expression - Biến chứa một giá trị xác định

value1,value 2,...valueN: Các giá trị hằng số phù hợp với giá trị trên biến **expression** .

action1,action2...actionN: Các phát biểu được thực thi khi một trường hợp tương ứng có giá trị True

break: Từ khoá được sử dụng để bỏ qua tất cả các câu lệnh sau đó và giành quyền điều khiển

cho cấu trúc bên ngoài **switch**

default: Từ khóa tùy chọn được sử dụng để chỉ rõ các câu lệnh nào được thực hiện chỉ khi tất cả các trường hợp nhận giá trị False

default - action: Các câu lệnh được thực hiện chỉ khi tất cả các trường hợp nhận giá trị False

Đoạn chương trình sau xác định giá trị trong một biến nguyên và hiển thị ngày trong tuần được thể hiện dưới dạng chuỗi. Để kiểm tra các giá trị nằm trong khoảng 0 đến 6

chương trình sẽ thông báo lỗi nếu nằm ngoài phạm vi trên.

Chương trình 3.8

Class SwitchDemo

```
{  
public static void main(String agrs[])  
{  
int day =4;  
switch(day)  
{  
case 0 : system.out.println("Sunday");  
break;  
case 1 : System.out.println("Monday");  
break;  
case 2 : System.out.println("Tuesday");  
break;  
case 3 : System.out.println("Wednesday");  
break;  
case 4 : System.out.println("Thursday");  
break;  
case 5 :System.out.println("Friday");  
break;  
case 6 :System.out.println("Satuday");  
break;  
case 7 :System.out.println("Saturday");  
break;  
default :System.out.println("Invalid day of week");  
}  
}
```

```
}
```

Nếu giá trị của biến day là 4 ,chương trình sẽ hiển thị **Thursday**,và cứ tiếp như vậy .

3.10.3 Vòng lặp While

Vòng lặp **while** được sử dụng khi vòng lặp được thực hiện mãi cho đến khi điều kiện thực thi vẫn là True. Số lượng vòng lặp không được xác định trước song nó sẽ phụ thuộc vào từng điều kiện.

Cú pháp

while(condition)

```
{
```

action statement;

```
:
```

```
:
```

```
}
```

condition: Biểu thức Boolean, nó trả về giá trị True hoặc False. Vòng lặp sẽ tiếp tục cho đến khi nào giá trị True được trả về.

action statement: Các câu lệnh được thực hiện nếu **condition** nhận giá trị True

Đoạn chương trình sau tính giai thừa của số 5.Giai thừa được tính như tích $5*4*3*2*1$.

Chương trình 3.9

Class WhileDemo

```
{
```

```
    Public static void main(String args[])
```

```
{
```

```
int a = 5,fact = 1;
```

```
while (a.>= 1)
```

```
{
```

```
    fact *=a;
```

```
a--;
```

```
}
```

```
System.out.println(The Factorial of 5 is “+fact);
```

```
}
```

```
}
```

Ở ví dụ trên, vòng lặp được thực thi cho đến khi điều kiện $a \geq 1$ là **True**. Biến **a** được khai báo bên ngoài vòng lặp và được gán giá trị là 5. Cuối mỗi vòng lặp, giá trị của **a** giảm đi 1. Sau năm vòng giá trị của **a** bằng 0. Điều kiện trả về giá trị False và vòng lặp kết thúc. Kết quả sẽ được hiển thị “ **The factorial of 5 is 120**”

Đoạn chương trình sau hiển thị tổng của 5 số chẵn đầu tiên

Chương trình 3.11

Class ForDemo

```
{  
public static void main(String args[])  
{  
    int i=1,sum=0;  
    for (i=1;i<=10;i+=2)  
        sum+=i;  
    System.out.println ("sum of first five odd numbers is "+sum);  
}  
}
```

Ở ví dụ trên, i và sum là hai biến được gán các giá trị đầu là 1 và 0 tương ứng. Điều kiện được kiểm tra và khi nó còn nhận giá trị True, câu lệnh tác động trong vòng lặp được thực hiện. Tiếp theo giá trị của i được tăng lên 2 để tạo ra số chẵn tiếp theo. Một lần nữa, điều kiện lại được kiểm tra và câu lệnh tác động lại được thực hiện. Sau năm vòng, i tăng lên 11, điều kiện trả về giá trị False và vòng lặp kết thúc. Thông báo: **Sum of first five odd numbers is 25** được hiển thị.

Tóm tắt bài học

- Phát biểu **import** được sử dụng trong chương trình để truy cập các gói Java.
- Chương trình Java chứa một tập hợp các gói. Chương trình có thể chứa các dòng giải thích. Trình biên dịch sẽ bỏ qua các dòng giải thích này.
- “Token” là thành phần nhỏ nhất của chương trình. Có năm loại “token”
 - ✓ Định danh (identifiers)
 - ✓ Từ khóa (keywords)
 - ✓ Ký tự phân cách (separators)
 - ✓ Nguyên dạng (Literals)
 - ✓ Các toán tử
- Java có các kiểu cấu trúc dữ liệu như kiểu nguyên thủy mà ta đã biết. Các biến được khai báo cho từng kiểu dữ liệu xác định. Hãy thận trọng khi khai báo tên biến để loại trừ khả năng hỗn loạn.
- Java cung cấp các chỉ định truy xuất sau đây :
 - ✓ Công cộng (public)
 - ✓ Bảo vệ (protected)
 - ✓ Riêng tư (private)
- Java cung cấp các bổ nghĩa (modifiers) sau đây:
 - ✓ Tĩnh (static)
 - ✓ Trừu tượng (abstract)
 - ✓ Final

- Khởi tạo hàm có hai kiểu :
 - ✓ Tường minh (Explicit)
 - ✓ Ngầm định (Implicit)
- Java cung cấp nhiều dạng toán tử, chúng gồm :
 - ✓ Các toán tử số học
 - ✓ Các toán tử dạng bit
 - ✓ Các toán tử quan hệ
 - ✓ Các toán tử logic
 - ✓ Toán tử điều kiện
 - ✓ Toán tử gán
- Ứng dụng Java có một lớp chứa phương thức main. Các tham số có thể được truyền vào phương thức main nhờ các tham số lệnh (command line parameters) .Trong Java cung cấp những cấu trúc chương trình sau đây
 - ✓ if-else
 - ✓ switch
 - ✓ for
 - ✓ while
 - ✓ do while

Kiểm tra kiến thức của bạn

1. Trong Java, kiểu dữ liệu dạng byte nằm trong giới hạn từ.....đến.....
2. Hãy chỉ các danh định hợp lệ trong :
 - a. Tel_num
 - b. Empl
 - c. 8678
 - d. batch.no
3. Cái gì là output của đoạn chương trình sau?
class me
{
public static void main(String srgs[])
{
int sales=820;
int profit=200;
System.out.println(((sale +profit)/10*5);
}
}
4.là sự bổ sung (implementation) của các phép toán trên các đối tượng
5. Phương thức **public** có thể truy cập phương thức **private** trong cùng một lớp.
True/False
6. '**Static**' hàm ý rằng phương thức không có mã (code) và được bổ xung trong các lớp con **True/False**
7. Khi bạn không định nghĩa một hàm khởi tạo cho một lớp, JVM sẽ cung cấp một hàm mặc định hoặc một hàm khởi tạo ẩn (implicit). **True/False**
8. Vòng lặp while thực thi ít nhất một lần thậm chí nếu điều kiện được xác định là False **True/False**

Bài tập

1. Hãy viết một đoạn chương trình để xuất dòng chữ :” Welcome to the world of Java”
2. Hãy viết hai hàm kiến tạo mở (explicit) cho một lớp dùng để tính diện tích hình chữ nhật.Khi một giá trị đơn được truyền vào hàm khởi tạo,nó có thể bị ngộ nhận rằng độ dài và chiều cao giống như biến được truyền vào.Lúc đó, nó sẽ tính một diện tích phù hợp.Khi hai giá trị được truyền vào hàm,nó sẽ tính diện tích hình chữ nhật.
3. Viết một chương trình thực hiện những việc sau đây:
 - a. Khai báo và gán giá trị đầu cho các biến m và n là 100 và 200 tương ứng.
 - b. Theo các điều kiện : nếu m bằng 0 , hiển thị kết quả tương ứng.
 - c. Nếu m lớn hơn n , hiển thị kết quả tương ứng.
 - d. Kiểm tra các giá trị n là chẵn hay lẻ.
4. Viết một chương trình hiển thị tổng các bội số của 7 nằm giữa 1 và 100.
5. Viết chương trình để cộng bảy số hạng của dãy sau :
1!+2!+3!.....

Chương 4. Các Gói & Giao Diện

4.1 Mục tiêu bài học

Kết thúc chương này, các bạn học viên có thể:

- Định nghĩa một giao diện
- Hiện thực một giao diện
- Sử dụng giao diện như là một kiểu dữ liệu
- Định nghĩa gói
- Tạo và sử dụng các gói
- Vai trò của các gói trong việc điều khiển truy cập
- Những đặc trưng của gói **java.lang**
- Những đặc trưng của gói **java.util**

4.2 Giới thiệu

Gói và giao diện là hai thành phần chính của chương trình Java. Các gói được lưu trữ theo kiểu phân cấp, và được nhập (import) một cách tường minh vào những lớp mới được định nghĩa. Các giao diện có thể được sử dụng để chỉ định một tập các phương thức. Các phương thức này có thể được hiện thực bởi một hay nhiều lớp.

Một tập tin nguồn Java có thể chứa một hoặc tất cả bốn phần nội tại sau đây:

- Một câu lệnh khai báo gói. (package)
- Những câu lệnh nhập thêm các gói hoặc các lớp khác vào chương trình (import)
- Một khai báo lớp công cộng (public) đơn
- Một số các lớp dạng riêng tư (private) của gói.

Một tập tin nguồn Java sẽ có khai báo lớp public đơn. Tất cả những phát biểu khác tùy chọn. Chương trình có thể được viết trong một dòng các gói với các lệnh nhập (import), và lớp (class).

4.3 Các giao diện

Giao diện là một trong những khái niệm quan trọng nhất của ngôn ngữ Java. Nó cho phép một lớp có nhiều lớp cha (superclass). Các chương trình Java có thể thừa kế chỉ một lớp tại một thời điểm, nhưng có thể hiện thực hàng loạt giao diện. Giao diện được sử dụng để thay thế một lớp trừu tượng, nơi mà không có một sự thực thi nào được kế thừa. Giao diện tương tự như các lớp trừu tượng. Sự khác nhau ở chỗ một lớp trừu tượng có thể có những hành vi cụ thể, nhưng một giao diện thì không thể có một phương thức cụ thể có hành vi của riêng mình. Các giao diện cần được hiện thực. Một lớp trừu tượng có thể được mở rộng, nhưng không thể được mô tả bằng một ví dụ minh họa cụ thể.

Các bước để tạo một giao diện được liệt kê ở dưới đây:

- Định nghĩa giao diện: Một giao diện được định nghĩa như sau:

Chương trình 4.1

```
//Giao diện với các phương thức
public interface myinterface
{
    public void add(int x,int y);
    public void volume(int x,int y,int z);
}

//Giao diện để định nghĩa các hằng
public interface myconstants
{
    public static final double price=1450.00;
    public static final int counter=5;
}
```

- Chương trình trên được dịch như sau:

javac myinterface.java

- Một giao diện được hiện thực với từ khoá “implement”. Trong trường hợp trên, giao diện cho phép ứng dụng mối quan hệ “is a”. Ví dụ:

class demo implements myinterface

- Nếu nhiều hơn một giao diện được thực thi, các tên sẽ được ngăn cách với nhau bởi một dấu phẩy. Điều này được trình bày như sau:

class Demo implements MyCalc, Mycount

Hãy ghi nhớ các lưu ý sau trong khi tạo một giao diện:

- Tất cả các phương thức trong các giao diện này phải là kiểu public.
- Các phương thức được định nghĩa trong một lớp mà lớp này hiện thực giao diện.

4.3.1 Hiện thực giao diện

Các giao diện không thể mở rộng (extend) các lớp, nhưng chúng có thể mở rộng các giao diện khác. Nếu khi bạn hiện thực một giao diện mà làm mở rộng nó, bạn cần ghi đè (**override**) các phương thức trong giao diện mới này một cách hợp lý như trong giao diện cũ. Trong ví dụ trên, các phương thức chỉ được khai báo, mà không được định nghĩa. Các phương thức phải được định nghĩa trong một lớp mà lớp đó hiện thực giao diện này. Nói một cách khác, bạn cần chỉ ra hành vi của phương thức. Tất cả các phương thức trong các giao diện phải là kiểu **public**. Bạn không được sử dụng các bộ ngữ (modifiers) chuẩn khác như protected, private..., khi khai báo các phương thức trong một giao diện.

Đoạn mã Chương trình 4.2 biểu diễn một giao diện được thực thi như thế nào:

Chương trình 4.2

```
import java.io.*;

class Demo implements myinterface
{
```

```
public void add(int x,int y)
{
    System.out.println(" "+(x+y));
    //Giả sử phương thức add được khai báo trong giao diện
}
public void volume(int x,int y,int z)
{
    System.out.println(" "+(x*y*z));
    //Giả sử phương thức volume được khai báo trong giao diện
}
public static void main(String args[])
{
    Demo d=new Demo();
    d.add(10,20);
    d.volume(10,10,10);
}
}
```

Khi bạn định nghĩa một giao diện mới, có nghĩa là bạn đang định nghĩa một kiểu tham chiếu dữ liệu mới. Bạn có thể sử dụng các tên giao diện ở bất cứ nơi đâu như bất kỳ tên kiểu dữ liệu khác. Chỉ có một thể hiện (instance) của lớp mà lớp đó thực thi giao diện có thể được gán đến một biến tham chiếu. Kiểu của biến tham chiếu đó là tên của giao diện.

4.4 Các gói

Gói được coi như các thư mục, đó là nơi bạn tổ chức các lớp và các giao diện của bạn. Các chương trình Java được tổ chức như những tập của các gói. Mỗi gói gồm có nhiều lớp, và/hoặc các giao diện được coi như là các thành viên của nó. Đó là một phương án thuận lợi để lưu trữ các nhóm của những lớp có liên quan với nhau dưới một cái tên đặc biệt. Khi bạn đang làm việc với một chương trình ứng dụng, bạn tạo ra một số lớp. Các lớp đó cần được tổ chức một cách hợp lý. Điều đó sẽ dễ dàng để tổ chức các tập tin lớp thành các gói khác nhau. Hãy tưởng tượng rằng mỗi gói giống như một thư mục con. Tất cả các điều mà bạn cần làm là đặt các lớp và các giao diện có liên quan với nhau vào các thư mục riêng, với một cái tên phản ánh được mục đích của các lớp.

Nói tóm lại, các gói có ích cho các mục đích sau:

- Chúng cho phép bạn tổ chức các lớp thành các đơn vị nhỏ hơn (như là các thư mục), và làm cho việc xác định vị trí trở nên dễ dàng và sử dụng các tập tin của lớp một cách phù hợp.
- Giúp đỡ để tránh cho việc đặt tên bị xung đột (trùng lặp định danh). Khi bạn làm việc với một số các lớp bạn sẽ cảm thấy khó để quyết định đặt tên cho các lớp và các phương thức. Đôi lúc bạn muốn sử dụng tên giống nhau mà tên đó liên quan đến lớp khác. Các gói giấu các lớp để tránh việc đặt tên bị xung đột.

- Các gói cho phép bạn bảo vệ các lớp, dữ liệu và phương thức ở mức rộng hơn trên một nền tảng class-to-class.
- Các tên của gói có thể được sử dụng để nhận dạng các lớp.

Các gói cũng có thể chứa các gói khác.

Để tạo ra một lớp là thành viên của gói, bạn cần bắt đầu mã nguồn của bạn với một khai báo gói, như sau:

package mypackage;

Hãy ghi nhớ các điểm sau trong khi tạo gói:

- Đoạn mã phải bắt đầu với một phát biểu “package”. Điều này nói lên rằng lớp được định nghĩa trong tập tin là một phần của gói xác định.
- Mã nguồn phải nằm trong cùng một thư mục, mà thư mục đó lại là tên gói của bạn.
- Quy ước rằng, các tên gói sẽ bắt đầu bằng một chữ thường để phân biệt giữa lớp và gói.
- Các phát biểu khác có thể xuất hiện sau khai báo gói là các câu lệnh nhập, sau chúng bạn có thể bắt đầu định nghĩa lớp của bạn.
- Tương tự tất cả các tập tin khác, mỗi lớp trong một gói cần được biên dịch.
- Để cho chương trình Java của bạn có khả năng sử dụng các gói đó, hãy nhập (import) chúng vào mã nguồn của bạn.
- Sự khai báo sau đây là hợp lệ và không hợp lệ :

Hợp lệ

package mypackage;

import java.io.;*

Không hợp lệ

import java.io.;*

package mypackage;

Bạn có các tùy chọn sau trong khi nhập vào một gói:

- Bạn có thể nhập vào một tập tin cụ thể từ gói:

import java.mypackage.calculate

- Bạn có thể nhập (import) toàn bộ gói:

import java.mypackage.*;

Máy ảo Java (JVM) phải giữ lại một track (rãnh ghi) của tất cả các phần tử hiện hữu trong gói mà được khai báo.

Bạn đã sẵn sàng làm việc với một phát biểu nhập import – java.io.*. Bản thân Java đã được cài đặt sẵn một tập các gói, bảng dưới đây đề cập đến một vài gói có sẵn của Java:

Gói	Mô tả
java.lang	Không cần phải khai báo một cách rõ ràng. Gói này luôn được nhập cho bạn.

java.io	Bao gồm các lớp để trợ giúp cho bạn tất cả các thao tác nhập và xuất.
java.applet	Bao gồm các lớp để bạn cần thực thi một applet trong trình duyệt.
java.awt	Hữu dụng để tạo nên các ứng dụng giao diện đồ họa (GUI).
java.util	Cung cấp nhiều lớp và nhiều giao diện khác nhau để tạo nên các ứng dụng, các applet, như là các cấu trúc dữ liệu, các lịch biểu, ngày tháng, v.v..
java.net	Cung cấp các lớp và các giao diện cho việc lập trình mạng TCP/IP.

Bảng 4.1 Các gói trong Java.

Bên cạnh đó, Java còn cung cấp thêm nhiều gói để phát triển các ứng dụng và các applet của bạn. Nếu bạn không khai báo các gói trong đoạn mã của bạn, thì các lớp và các giao diện của bạn sau khi kết thúc sẽ nằm trong một gói mặc định mà không có tên. Thông thường, gói mặc định này chỉ có ý nghĩa cho các ứng dụng nhỏ hoặc các ứng dụng tạm thời, như là các ứng dụng mà bạn vừa mới bắt đầu để phát triển sau này. Khi bạn bắt đầu việc phát triển cho một ứng dụng lớn, bạn có khuynh hướng phát triển một số các lớp. Bạn cần tổ chức các lớp đó trong các thư mục khác nhau để dễ dàng truy cập và vận dụng. Để làm được điều này, bạn phải đặt chúng vào các gói đã đặt tên.

Phần lớn về việc làm với các gói là bạn có đặc quyền để sử dụng các tên lớp giống nhau, nhưng bạn phải đặt chúng vào các gói khác nhau.

4.4.1 Tạo một gói

Gói là một phương thức hữu dụng để nhóm các lớp mà tránh được các tên trùng nhau. Các lớp với những tên giống nhau có thể đặt vào các gói khác nhau. Các lớp được định nghĩa bởi người sử dụng cũng có thể được nhúng lại trong các gói.

Các bước sau đây cho phép tạo nên một gói do người dùng định nghĩa:

- Khai báo gói bằng cách sử dụng cú pháp thích hợp. Đoạn mã phải bắt đầu với khai báo gói. Điều này chỉ ra rằng lớp được định nghĩa trong tập tin là một phần của gói xác định.

package mypackage;

- Sử dụng phát biểu import để nhập các gói chuẩn theo yêu cầu.

import java.util.*;

- Khai báo và định nghĩa các lớp sẽ nằm trong gói đó. Tất cả các thành phần của gói sẽ là public, để có thể được truy cập từ bên ngoài. Máy ảo Java (JVM) giữ lại track (rãnh ghi) của tất cả các phần tử nằm trong gói đó.

package mypackage; //khai báo gói

import java.util.;*

public class Calculate //định nghĩa một lớp

{

int var;

Calculate(int n)


```
{  
...  
var = n;  
//các phương thức  
//...  
public class Display //định nghĩa một lớp  
{  
    ...//Các phương thức  
}  
}  
}
```

- Lưu các định nghĩa trên trong một tập tin với phần mở rộng .java, và dịch các lớp được định nghĩa trong gói. Việc dịch có thể thực hiện với chức năng “-d”. Chức năng này tạo một thư mục trùng với tên gói, và đặt tập tin .class vào thư mục được chỉ rõ.

javac -d d:\temp Calculate.java

Nếu khai báo gói không có trong chương trình, lớp hoặc giao diện đó sẽ kết thúc trong một gói mặc định mà không có tên. Nói chung, gói mặc định này thì chỉ có nghĩa cho các ứng dụng nhỏ hoặc tạm thời.

Hãy ghi nhớ các điểm sau đây khi bạn khai thác các gói do người dùng định nghĩa trong các chương trình khác:

- Mã nguồn của các chương trình đó phải tồn tại trong cùng một thư mục với gói được định nghĩa bởi người sử dụng.
- Để cho các chương trình Java khác sử dụng được các gói đó, hãy khai báo chúng vào đoạn mã nguồn.
- Để nhập một lớp ta dùng:

import java.mypackage.Calculate;

- Để nhập toàn bộ một gói, ta làm như sau:

import java.mypackage.*;

- Tạo một tham chiếu đến các thành phần của gói. Ta dùng đoạn mã đơn giản sau:

```
import java.io.*;  
import mypackage.Calculate;  
class PackageDemo{  
    public static void main(String args[]){  
        Calculate calc = new Calculate();  
    }  
}
```

Nếu phát biểu import cho gói đó không được sử dụng, thì tên lớp phải được sử dụng với tên gói của nó sao cho phù hợp với phương thức trong lớp đó. Cú pháp như sau:

```
mypackage.Calculate calc = new mypackage.Calculate();
```

4.4.2 Thiết lập đường dẫn cho lớp (classpath)

Chương trình dịch và chương trình thông dịch tìm kiếm các lớp trong thư mục hiện hành, và tập tin nén (zip) chứa các tập tin class JDK. Điều này có nghĩa các tập tin class JDK và thư mục nguồn tự động thiết lập **classpath** cho bạn. Tuy nhiên, trong một vài trường hợp, bạn cần phải tự thiết lập classpath cho bạn.

Classpath là một danh sách các thư mục, danh sách này trợ giúp để tìm kiếm các tập tin class tương ứng. Thông thường, ta không nên thiết lập môi trường classpath một thời gian dài. Nó chỉ thích hợp khi thiết lập CLASSPATH để chạy chương trình, như khi ta thiết lập đường dẫn cho việc thực thi hiện thời.

```
javac -classpath c:\temp Packagedemo.java
```

Thứ tự của các mục trong classpath thì rất quan trọng. Khi bạn thực thi đoạn mã của bạn, máy ảo Java sẽ tìm kiếm các mục trong classpath của bạn giống như thứ tự đã đề cập, cho đến khi nó tìm thấy lớp cần tìm.

Ví dụ của một gói

Chương trình 4.3

Package mypackage;

Public class calculate

```
{  
    public double volume(double height, double width, double depth)  
    {  
        return (height*width*depth);  
    }  
    public int add(int x, int y)  
    {  
        return (x+y);  
    }  
    public int divide(int x, int y)  
    {  
        return (x/y);  
    }  
}
```

Để sử dụng gói này, bạn cần phải:

- Khai báo lớp được sử dụng.
- Khai báo toàn bộ gói.
- Đề cập đến các thành phần của gói.

Bạn cần dịch tập tin này. Nó có thể được dịch với tùy chọn `-d`, nhờ đó, nó tạo một thư mục với tên của gói và đặt tập tin `.class` vào thư mục này.

javac -d c:\temp calculate.java

Chương trình biên dịch tạo một thư mục được gọi là “mypackage” trong thư mục temp, và lưu trữ tập tin `calculate.class` vào thư mục này.

Ví dụ sau biểu diễn cách sử dụng một gói:

Chương trình 4.4

```
import java.io.*;
import mypackage.calculate;
Class PackageDemo{
public static void main(String args[]){
Calculate calc = new calculate();
int sum = calc.add(10,20);
double vol = calc.volume(10.3f,13.2f,32.32f);
int div = calc.divide(20,4);
System.out.println("The addition is: "+sum);
System.out.println("The Volume is: "+vol);
System.out.println("The division is: "+sum);
}
}
```

Nếu bạn sử dụng một lớp từ một gói khác, mà không sử dụng khai báo import cho gói đó, thì khi đó, bạn cần phải sử dụng tên lớp với tên gói.

Mypackage.calculate calc = new mypackage.calculate();

4.5 Gói và điều khiển truy xuất

Các gói chứa các lớp và các gói con. Các lớp chứa dữ liệu và đoạn mã. Java cung cấp nhiều mức độ truy cập thông qua các lớp, các gói và các chỉ định truy cập. Bảng sau đây sẽ tóm tắt quyền truy cập các thành phần của lớp:

	public	protected	No modifier	private
Same class	Yes	Yes	Yes	Yes
Same packages subclass	Yes	Yes	Yes	No
Same package non-subclass	Yes	Yes	Yes	No
Different package	Yes	Yes	No	No

subclass				
Different package non-subclass	Yes	No	No	No

Bảng 4.2: Truy cập đến các thành phần của lớp.

4.6 Gói java.lang

Theo mặc định, mỗi chương trình java đều nhập gói java.lang. Vì thế, không cần phải khai báo một cách rõ ràng gói java.lang này trong chương trình.

Lớp trình bao bọc (wrapper class)

Các kiểu dữ liệu nguyên thủy thì không phải là các đối tượng. Vì thế, chúng không thể tạo hay truy cập các phương thức. Để tạo hay vận dụng kiểu dữ liệu nguyên thủy, ta sử dụng “wrap” tương ứng với “wrapper class”. Bảng sau liệt kê các lớp trình bao bọc (wrapper). Các phương thức của mỗi lớp này có trong phần phụ lục.

Kiểu dữ liệu	Lớp trình bao bọc
boolean	Boolean
byte	Byte
char	Character
double	Double
float	Float
int	Integer
long	Long
short	Short

Bảng 4.3: Các lớp trình bao bọc cho các kiểu dữ liệu nguyên thủy.

Ví dụ một vài phương thức của lớp wrapper:

```
Boolean wrapBool = new Boolean("false");
```

```
Integer num1 = new Integer("31");
```

```
Integer num2 = new Integer("3");
```

```
Int sum = num1.intValue()*num2.intValue();
```

//intValue() là một hàm của lớp trình bao bọc Integer.

Chương trình sau đây minh họa cách sử dụng lớp wrapper cho kiểu dữ liệu int

Chương trình 4.5

```
Class CmdArg
```

```
{
```

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

```
{
```

```
int sum = 0;
```

```
for(int i = 0; i < args.length; i++)  
    sum += Integer.parseInt(args[i]);  
System.out.println("Tổng là: " + sum);  
}  
}
```

Vòng lặp for được sử dụng để tìm tổng của các số thỏa mãn điều kiện (hợp quy cách) tại dòng lệnh. Các số đó được lưu trữ trong mảng String args[]. Đặc tính "length" xác định số các phần tử trong mảng args[]. Mảng args[] là kiểu String. Vì thế, các phần tử phải được đổi sang kiểu dữ liệu int trước khi cộng chúng. Quá trình chuyển đổi được thực hiện với sự giúp đỡ của lớp trình bao bọc "Integer". Phương thức "parseInt()" trong lớp "Integer" thực hiện quá trình chuyển đổi của kiểu dữ liệu chuỗi sang kiểu dữ liệu số.

Tất cả các lớp trình bao bọc, ngoại trừ lớp "Character" có một phương thức tĩnh "valueOf()" được gọi để tách một chuỗi, và trả về một giá trị số nguyên được bao bọc. Các lớp trình bao bọc của byte, int, long, và short cung cấp các hằng số MIN_VALUE và MAX_VALUE. Các lớp trình bao bọc của double và long cũng cung cấp các hằng POSITIVE_INFINITY và NEGATIVE_INFINITY.

4.6.1 Lớp String (lớp chuỗi)

Các chuỗi là hàng loạt các ký tự. Lớp String cung cấp hàng loạt các phương thức để thao tác với các chuỗi. Nó cung cấp các phương thức khởi tạo (constructor) khác nhau. Dưới đây là một vài phương thức đã được cho:

```
String str1 = new String( );
```

//str1 chứa một dòng trống.

```
String str2 = new String("Hello World");
```

//str2 chứa dòng "Hello World"

```
char ch[] = {'A','B','C','D','E'};
```

```
String str3 = new String(ch);
```

//str3 chứa "ABCDE"

```
String str4 = new String(ch,0,2);
```

//str4 chứa "AB" vì 0- tính từ ký tự bắt đầu, 2- là số lượng ký tự kể từ ký tự bắt đầu.

Toán tử "+" được cung cấp để cộng chuỗi khác đến một chuỗi đang tồn tại. Toán tử "+" này được gọi như là "thao tác nối chuỗi". Ở đây, nối chuỗi được thực hiện thông qua lớp "StringBuffer". Chúng ta sẽ thảo luận tiến trình này ngay sau đó trong chương này. Phương thức "concat()" của lớp String cũng có thể thực hiện việc nối chuỗi. Không giống như toán tử "+", phương thức này không thường xuyên nối hai chuỗi tại vị trí cuối cùng của chuỗi đầu tiên. Thay vào đó, phương thức này trả về một chuỗi mới, chuỗi mới đó sẽ chứa giá trị của cả hai chuỗi ban đầu. Điều này có thể được gán cho chuỗi đang tồn tại. Ví dụ:

```
String strFirst, strSecond, strFinal;
```

```
strFirst = "Charlie";
```

```
strSecond = "Chaplin";
```

//....bằng cách sử dụng phương thức `concat()` để gán với một chuỗi đang tồn tại.

```
StrFinal = strFirst.concat(strSecond);
```

Phương thức `concat()` chỉ làm việc với hai chuỗi tại một thời điểm.

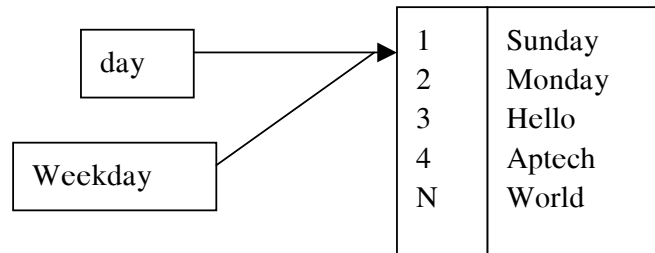
4.6.2 Chuỗi mặc định (String pool)

Một chương trình Java có thể chứa nhiều chuỗi bằng chữ. “String Pool” đại diện cho tất cả các chữ được tạo trong chương trình. Mỗi khi một chuỗi bằng chữ được tạo, String Pool tìm kiếm để nhìn thấy nếu chuỗi bằng chữ tồn tại. Nếu nó tồn tại, một thể hiện mới được gán đến một chuỗi mới. Việc này sẽ chiếm nhiều không gian bộ nhớ. Ví dụ:

```
String day = “Monday”;
```

```
String weekday = “Monday”;
```

Ở đây, một thể hiện cho biến “day”, biến đó có giá trị là “Monday”, được tạo trong String Pool. Khi chuỗi bằng chữ “weekday” được tạo, việc lưu giữ các giá trị giống nhau như của biến “day”, một thể hiện đang tồn tại được gán đến biến “weekday”. Vì cả hai biến “day” và “weekday” cũng đều nhằm chỉ vào chuỗi tương tự trong String Pool. Hình ảnh sau minh họa khái niệm của “String Pool”.



Hình 4.1 Khái niệm của String Pool.

4.6.3 Các phương thức lớp String

Trong phần này, chúng ta sẽ xem xét các phương thức của lớp String.

➤ **CharAt()**

Phương thức này trả về một ký tự tại một vị trí đặc biệt trong một chuỗi.

Ví dụ:

```
String name = new String(“Java Language”);
```

```
char ch = name.charAt(5);
```

Biến “ch” chứa giá trị “L”, từ đó vị trí các số bắt đầu từ 0.

➤ **startsWith()**

Phương thức này trả về giá trị kiểu logic (Boolean), phụ thuộc vào chuỗi có bắt đầu với một giá trị đặc biệt không. Ví dụ:

```
String strname = “Java Language”;
```

```
boolean flag = strname.startsWith(“Java”);
```

Biến “flag” chứa giá trị true.

➤ **endsWith()**

Phương thức này trả về một giá trị kiểu logic (boolean), có chăng phụ thuộc vào chuỗi kết thúc với một giá trị đặc biệt, Ví dụ:

```
String strname = "Java Language";  
boolean flag = strname.endsWith("Java");
```

Biến “flag” chứa giá trị false.

➤ **copyValueOf()**

Phương thức này trả về một chuỗi được rút ra từ một mảng ký tự được truyền như một đối số. Phương thức này cũng lấy hai tham số nguyên. Tham số đầu tiên chỉ định vị trí từ nơi các ký tự phải được rút ra, và tham số thứ hai chỉ định số ký tự được rút ra từ mảng. Ví dụ:

```
char name[] = {'L', 'a', 'n', 'g', 'u', 'a', 'g', 'e'};  
String subname = String.copyValueOf(name, 5, 2);
```

Bây giờ biến “subname” chứa chuỗi “ag”.

➤ **toCharArray()**

Phương thức này lấy một chuỗi, và chuyển nó vào một mảng ký tự. Ví dụ:

```
String text = new String("Hello World");  
Char textArray[] = text.toCharArray();
```

➤ **indexOf()**

Phương thức này trả về thứ tự của một ký tự đặc biệt, hoặc một chuỗi trong phạm vi một chuỗi. Các câu lệnh sau biểu diễn các cách khác nhau của việc sử dụng hàm.

```
String day = new String("Sunday");  
int index1 = day.indexOf('n');  
//chứa 2
```

```
int index2 = day.indexOf('z', 2);  
//chứa -1 nếu “z” không tìm thấy tại vị trí 2.
```

```
int index3 = day.indexOf("Sun");  
//chứa mục 0 của mẫu tự 1st
```

➤ **toUpperCase()**

Phương thức này trả về chữ hoa của chuỗi thông qua hàm.

```
String lower = new String("good morning");  
System.out.println("Uppercase: " + lower.toUpperCase());
```

➤ **toLowerCase()**

Phương thức này trả về chữ thường của chuỗi thông qua hàm.

```
String upper = new String("APTECH");  
System.out.println("Lowercase: " + upper.toLowerCase());
```

➤ trim()

Phương thức này cắt bỏ khoảng trắng trong đối tượng String. Hãy thử đoạn mã sau để thấy sự khác nhau trước và sau khi cắt bỏ khoảng trắng.

```
String space = new String("    Spaces    ");
System.out.println(spaces);
System.out.println(spaces.trim()); //Sau khi cắt bỏ khoảng trắng
```

➤ equals()

Phương thức này so sánh nội dung của hai đối tượng chuỗi.

```
String name1 = "Aptech", name2 = "APTECH";
boolean flag = name1.equals(name2);
```

Biến "flag" chứa giá trị false.

4.6.4 Lớp StringBuffer

Lớp StringBuffer cung cấp các phương thức khác nhau để thao tác một đối tượng dạng chuỗi. Các đối tượng của lớp này rất mềm dẻo, đó là các ký tự và các chuỗi có thể được chèn vào giữa đối tượng StringBuffer, hoặc nối thêm dữ liệu vào tại vị trí cuối. Lớp này cung cấp các phương thức khởi tạo nạp chồng. Chương trình sau biểu diễn làm thế nào để sử dụng các phương thức khởi tạo khác nhau để tạo ra các đối tượng của lớp này.

Chương trình 4.6

```
class StringBufferCons
{
    public static void main(String args[])
    {
        StringBuffer s1 = new StringBuffer();
        StringBuffer s2 = new StringBuffer(20);
        StringBuffer s3 = new StringBuffer("StringBuffer");

        System.out.println("s3 = "+ s3);
        System.out.println(s2.length()); //chứa 0
        System.out.println(s3.length()); //chứa 12
        System.out.println(s1.capacity()); //chứa 16
        System.out.println(s2.capacity()); //chứa 20
        System.out.println(s3.capacity()); //chứa 28
    }
}
```

"length()" và "capacity()" của đối tượng StringBuffer là hoàn toàn khác nhau. Phương thức "length()" đề cập đến số các ký tự mà đối tượng đưa ra, trong khi "capacity()" trả về tổng dung lượng mặc định của một đối tượng (16), và số các ký tự trong đối tượng StringBuffer.

Dung lượng của bộ đệm chuỗi có thể thay đổi với phương thức “ensureCapacity()” được cung cấp trong lớp. Đối số int đã được truyền đến phương thức này, và phù hợp với một dung lượng mới được tính toán như sau:

$$\text{New Capacity} = \text{Old Capacity} * 2 + 2$$

Trước khi dung lượng của bộ nhớ trung gian được cấp phát dung lượng được tính toán mới, điều kiện sau sẽ được kiểm tra:

- Nếu dung lượng mới lớn hơn đối số được truyền đến phương thức “ensureCapacity()”, thì dung lượng bộ nhớ đệm được cấp phát

Một dung lượng được tính toán mới.

- Nếu dung lượng mới nhỏ hơn đối số được truyền đến phương thức “ensureCapacity()”, thì dung lượng bộ nhớ đệm được cấp phát giá trị của đối số được truyền đến.

Chương trình 4.7 minh họa làm thế nào dung lượng được tính toán và được cấp phát.

Chương trình 4.7

```
class test{  
    public static void main(String args[]){  
        StringBuffer s1 = new StringBuffer(5);  
        System.out.println("Dung lượng của bộ nhớ đệm = "+s1.capacity()); //chứa 5  
        s1.ensureCapacity(8);  
        System.out.println("Dung lượng của bộ nhớ đệm = "+s1.capacity()); //chứa 12  
        s1.ensureCapacity(30);  
        System.out.println("Dung lượng của bộ nhớ đệm = "+s1.capacity()); //chứa 30  
    }  
}
```

Trong đoạn mã trên, dung lượng ban đầu của s1 là 5. Câu lệnh

s1.ensureCapacity(8);

Thiết lập dung lượng của s1 đến 12(5*2+2) bởi vì dung lượng trên lý thuyết là (8) thì nhỏ hơn dung lượng được tính toán là (12) .

s1.ensureCapacity(30);

Thiết lập dung lượng của “s1” đến 30 bởi vì dung lượng trên lý thuyết là (30) thì lớn hơn dung lượng được tính toán (12*2+2).

4.6.5 Các phương thức lớp StringBuffer

Trong phần này, chúng ta sẽ xem xét các phương thức của lớp StringBuffer với một chương trình.

➤ append()

Phương thức này nối thêm một chuỗi hoặc một mảng ký tự tại vị trí cuối cùng của một đối tượng StringBuffer. Ví dụ:

```
StringBuffer s1 = new StringBuffer("Good");
```

```
s1.append("evening");
```

Giá trị trong s1 bây giờ là "goodevening".

➤ **insert()**

Phương thức này lấy hai tham số. Tham số đầu tiên là vị trí chèn. Tham số thứ hai có thể là một chuỗi, một ký tự (char), một giá trị nguyên (int), hay một giá trị số thực (float) được chèn vào. Vị trí chèn sẽ lớn hơn hay bằng đến 0, và nhỏ hơn hay bằng chiều dài của đối tượng StringBuffer. Bất kỳ đối số nào, trừ ký tự hoặc chuỗi, được chuyển vào biểu mẫu chuỗi, và sau đó được chèn vào. Ví dụ:

```
StringBuffer str = new StringBuffer("Java sion");
```

```
str.insert(1, 'b');
```

Biến "str" chứa chuỗi "Java sion".

➤ **charAt()**

Phương thức này trả về một giá trị ký tự trong đối tượng StringBuffer tại vị trí được chỉ định. Ví dụ:

```
StringBuffer str = new StringBuffer("James Gosling");
```

```
char letter = str.charAt(6); //chứa "G"
```

➤ **setCharAt()**

Phương thức này được sử dụng để thay thế ký tự trong một StringBuffer với những cái khác tại một vị trí được chỉ định.

```
StringBuffer name = new StringBuffer("Java");
```

```
name.setCharAt(2, 'v');
```

Biến "name" chứa "Java".

➤ **setLength()**

Phương thức này thiết lập chiều dài của đối tượng StringBuffer. Nếu chiều dài được chỉ định nhỏ hơn chiều dài nguyên thủy của bộ nhớ trung gian, thì các ký tự thừa sẽ bị cắt bớt. Nếu chiều dài chỉ định nhiều hơn chiều dài nguyên thủy của bộ nhớ đệm, các ký tự null được thêm vào tại vị trí cuối cùng của bộ nhớ đệm.

```
StringBuffer str = new StringBuffer(10);
```

```
str.setLength(str.length() + 10);
```

➤ **getChars()**

Phương thức này được sử dụng để trích ra các ký tự từ đối tượng StringBuffer, và sao chép chúng vào một mảng. Phương thức getChars() lấy bốn tham số sau:

Mục bắt đầu: vị trí bắt đầu, từ nơi mà ký tự được lấy vào.

Mục kết thúc: vị trí kết thúc

Mảng: Mảng đích, nơi mà các ký tự được sao chép.

Nơi gởi tới mục bắt đầu: Các ký tự được sao chép trong mảng đích từ vị trí này.

Ví dụ:

```
StringBuffer str = new StringBuffer("Leopard");
```

```
char ch[] = new char[10];
```

```
str.getChars(3,6,ch,0);
```

Bây giờ biến "ch" chứa "par"

➤ **reverse()**

Phương thức này đảo ngược nội dung của một đối tượng StringBuffer, và trả về một đối tượng StringBuffer. Ví dụ:

```
StringBuffer str = new StringBuffer("devil");
```

```
StringBuffer strrev = str.reverse();
```

Biến "strrev" chứa "lived".

4.6.6 Lớp java.lang.Math

Lớp này chứa các phương thức tĩnh để thực hiện các thao tác toán học. Chúng được mô tả như sau:

\$ Cú pháp là toán học.<tên hàm>

➤ **abs()**

Phương thức này trả về giá trị tuyệt đối của một số. Đối số được truyền đến nó có thể là kiểu int, float, double, hoặc long. Kiểu dữ liệu byte và short được chuyển thành kiểu int nếu chúng được truyền tới như là một đối số. Ví dụ:

```
int num = -1;
```

```
Math.abs(num) //trả về 1.
```

➤ **ceil()**

Phương thức này tìm thấy số nguyên lớn hơn hoặc bằng đối số được truyền đến ngay tức thời.

➤ **floor()**

Phương thức này trả về số nguyên nhỏ hơn hoặc bằng đối số được truyền vào ngay tức thời.

```
System.out.println(Math.ceil(8.02)); //trả về 8.0
```

```
System.out.println(Math.ceil(-1.3)); //trả về -1.0
```

```
System.out.println(Math.ceil(100)); //trả về 100.0
```

```
System.out.println(Math.floor(-5.6)); //trả về -6.0
```

```
System.out.println(Math.floor(201.1)); //trả về 201
```

```
System.out.println(Math.floor(100)); //trả về 100
```

➤ **max()**

Phương thức này tìm giá trị lớn nhất trong hai giá trị được truyền vào. Các đối số được truyền vào có thể là kiểu int, long, double, và float.

➤ **min()**

Phương thức này tìm giá trị nhỏ nhất trong hai giá trị được truyền vào. Các đối số được truyền vào có thể là kiểu int, long, double và float.

➤ *round()*

Phương thức này làm tròn đối số có dấu phẩy động. Ví dụ, câu lệnh `Math.round(34.5)` trả về 35.

➤ *random()*

Phương thức này trả về một số ngẫu nhiên giữa 0.0 và 1.0 của kiểu `double`.

➤ *sqrt()*

Phương thức này trả về bình phương của một số. Ví dụ, câu lệnh `Math.sqrt(144)` trả về 12.0.

➤ *sin()*

Phương thức này trả về sine của một số, nếu góc được truyền đến bằng radian. Ví dụ: **`Math.sin(Math.PI/2)`** trả về 1.0, giá trị của `sin 45`.

`Pi/2 radians = 90 độ`. Giá trị của “pi” bắt nguồn từ hằng số được định nghĩa trong lớp “`Math.PI`”.

➤ *cos()*

Phương thức này trả về cos của một số, nếu góc được truyền đến bằng radian.

➤ *tan()*

Phương thức này trả về tan của một số, nếu góc được truyền đến bằng radian.

4.6.7 Lớp Runtime (Thời gian thực hiện chương trình)

Lớp Runtime được gói gọn trong môi trường Runtime. Lớp này được sử dụng cho việc quản lý bộ nhớ, và việc thực thi của các quá trình xử lý gia tăng. Mỗi chương trình Java có một thể hiện đơn của lớp này, để cho phép ứng dụng giao tiếp với môi trường. Nó không thể được khởi tạo, khi mà một ứng dụng không thể tạo ra một minh dụ của riêng mình thuộc lớp này. Tuy nhiên, chúng ta có thể tạo ra một minh dụ hiện hành trong lúc thực hiện chương trình từ việc dùng phương thức `Runtime().garbage`

Bây giờ, chúng ta biết rằng việc thu gom các dữ liệu không thích hợp trong Java là một tiến trình tự động, và chạy một cách định kỳ. Để kích hoạt một cách thủ công bộ thu thập dữ liệu không thích hợp, ta gọi phương thức `gc()` trên minh dụ thời gian thời gian thực hiện hành. Để quyết định chi tiết cấp phát bộ nhớ, sử dụng các phương thức `totalMemory()` và `freeMemory()`.

```
Runtime r = Runtime.getRuntime();
```

```
.....
```

```
.....
```

```
long freemem = r.freeMemory();
```

```
long totalmem = r.totalMemory();
```

```
r.gc();
```

Bảng sau biểu diễn một vài phương thức được sử dụng chung của lớp này:

Method	Purpose
<code>exit(int)</code>	Dừng việc thực thi, và trả về giá trị của đoạn mã đến hệ điều hành. Việc ngắt thông thường tại 0; giá trị khác 0 cho biết việc ngắt khác thường.
<code>freeMemory()</code>	Quyết định số lượng sẵn có của bộ nhớ trống

	đến hệ thống thời gian chạy của Java trong giới hạn của các byte
<code>getRuntime()</code>	Trả về thể hiện thời gian chạy hiện hành.
<code>gc()</code>	Gọi những bộ phận thu thập dữ liệu vô nghĩa.
<code>totalMemory()</code>	Để quyết định tổng số lượng bộ nhớ sẵn có của chương trình.
<code>Exec(String)</code>	Thực thi một chương trình phân cách của tên được gọi.

Bảng 4.4 Lớp Runtime**Chương trình 4.7***class RuntimeDemo*

```
{  
    public static void main(String args[])  
    {  
        Runtime r = Runtime.getRuntime();  
        Process p = null;  
        try  
        {  
            p = r.exec("calc.exe");  
        }  
        catch(Exception e)  
        {  
            System.out.println("Error executing calculator");  
        }  
    }  
}
```

Bạn có thể đạt được minh dụ thời Runtime hiện hành thông qua phương thức `Runtime.getRuntime()`.

Sau đó, bạn có thể tham chiếu đến chương trình thi hành `calc.exe`, và lưu trữ trong một đối tượng của tiến trình.

4.6.8 Lớp hệ thống (System)

Lớp System cung cấp các điều kiện thuận lợi như là, xuất, nhập chuẩn và các luồng lỗi. Nó cũng cung cấp một giá trị trung bình để các thuộc tính truy cập được kết hợp với hệ thống thời gian chạy của Java, và các thuộc tính môi trường khác nhau như là, phiên bản, đường dẫn, hay các dịch vụ, v.v.. Các trường của lớp này là **in**, **out**, và **err**, các trường này tiêu biểu cho xuất, nhập và lỗi chuẩn tương ứng.

Bảng sau mô tả các phương thức của lớp này:

Phương thức	Mục đích
-------------	----------

Core Java

Exit(int)	Dừng việc thực thi, và trả về giá trị của đoạn mã. 0 cho biết có thể thoát ra một cách bình thường.
gc()	Khởi tạo tập hợp các dữ liệu vô nghĩa.
getProperties()	Trả về thuộc tính được kết hợp với hệ thống thời gian chạy của Java.
setProperty()	Thiết lập các đặc tính hệ thống hiện hành.
currentTimeMillis()	Trả về thời gian hiện tại trong mili giây (ms), được đo lường lúc nửa đêm vào tháng giêng năm 1970.
arraycopy(Object, int, Object, int, int)	Sao chép một mảng.

Bảng 4.5 Lớp System.

Lớp System không thể khai báo để tạo các đối tượng.

Đoạn mã trong chương trình sau truy lục và hiển thị một vài các thuộc tính môi trường liên quan đến Java.

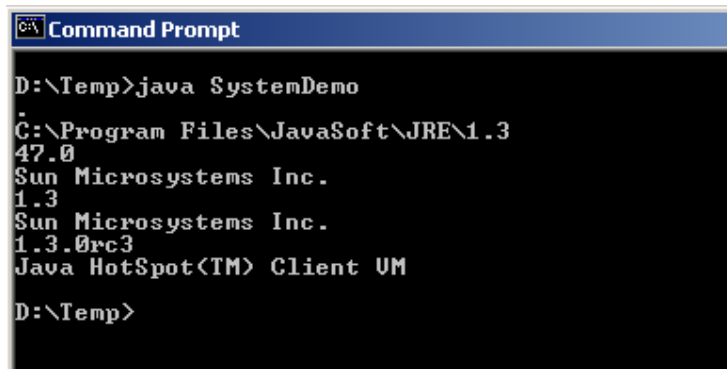
Chương trình 4.9

Class SystemDemo

```
{
    public static void main(String args[])
    {
        System.out.println(System.getProperty("java.class.path"));
        System.out.println(System.getProperty("java.home"));
        System.out.println(System.getProperty("java.class.version"));
        System.out.println(System.getProperty("java.specification.vendor"));
        System.out.println(System.getProperty("java.specification.version"));
        System.out.println(System.getProperty("java.vendor"));
        System.out.println(System.getProperty("java.vendor.url"));
        System.out.println(System.getProperty("java.version"));
        System.out.println(System.getProperty("java.vm.name"));
    }
}
```

Mỗi thuộc tính mà được yêu cầu để được in, được cung cấp như một tham số chuỗi đến phương thức System.getProperty(). Phương thức này lần lượt sẽ trả về thông tin có liên quan đến phương thức System.out.println().

Quá trình xuất ra của các thao tác xử lý được tự động tạo ra sẽ trông giống hình dưới đây:



```
C:\ Command Prompt
D:\Temp>java SystemDemo
C:\Program Files\JavaSoft\JRE\1.3
47.0
Sun Microsystems Inc.
1.3
Sun Microsystems Inc.
1.3.0rc3
Java HotSpot(TM) Client VM
D:\Temp>
```

Hình 4.2 Lớp System xuất

4.6.9 Lớp Class

Các minh dụ của lớp này bao bọc trạng thái thời gian thực hiện của một đối tượng trong một ứng dụng Java đang chạy. Điều này cho phép chúng ta truy cập thông tin về đối tượng trong suốt thời gian chạy.

Chúng ta có thể lấy một đối tượng của lớp này, hoặc một minh dụ bằng một trong ba cách sau:

Sử dụng phương thức `getChar()` trong một đối tượng.

- Sử dụng phương thức tĩnh `forName()` của lớp để lấy một thể hiện của lớp thông qua tên của lớp đó.
- Sử dụng một đối tượng `ClassLoader` tùy thích để nạp một lớp mới.

Không có phương thức xây dựng cho lớp.

Các chương trình sau minh họa làm sao để bạn có thể sử dụng phương thức của một lớp để truy lục thông tin của lớp đó:

Chương trình 4.10

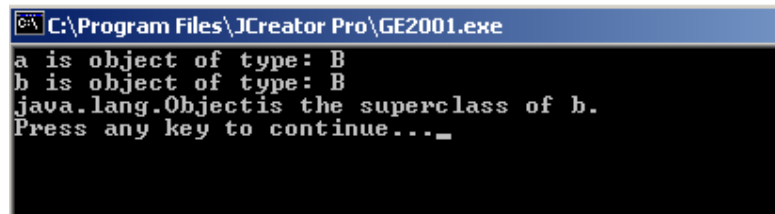
```
interface A
{
    final int id = 1;
    final String name = "Diana";
}
class B implements A
{
    int deptno;
}
class ClassDemo
{
    public static void main(String args[])
    {
```

```

A a = new B();
B b = new B();
Class x;
x = a.getClass();
System.out.println("a is object of type: "+x.getName());
x = b.getClass();
System.out.println("b is object of type: "+x.getName());
x=x.getSuperclass();
System.out.println(x.getName()+ "is the superclass of b.");
}
}

```

Quá trình xuất ra các kết quả được mô tả như hình dưới đây:



Hình 4.3 Quá trình xuất ra các kết quả của lớp Class.

4.6.10 Lớp Object

Lớp Object là một lớp cha của tất cả các lớp. Dù là một lớp do người dùng định nghĩa không mở rộng bất kỳ một lớp nào khác, theo mặc định nó mở rộng lớp đối tượng.

Một vài các phương thức của lớp Object được biểu diễn bên dưới:

Phương thức	Mục đích
equals(Object)	So sánh thể hiện đối tượng hiện tại với đối tượng đã cho, và kiểm tra nếu chúng bằng nhau.
finalize()	Mặc định hình thức của phương thức cuối cùng. Thông thường bị phủ bởi lớp con.
notify()	Thông báo dòng (thread) mà hiện thời trong trạng thái đang chờ trên màn hình của đối tượng này.
notifyAll()	Thông báo tất cả các dòng (thread) hiện hành trong trạng thái chờ trên màn hình của đối tượng này.
toString()	Trả về một chuỗi đại diện cho đối tượng.
wait()	Tạo ra dòng hiện hành để nhập vào trạng thái đang chờ.

Bảng 4.6 Lớp Object.

Trong chương trình sau, chúng ta không khai báo bất kỳ lớp hoặc gói nào. Bây giờ, chúng ta có thể tạo bằng cách sử dụng phương thức equals(). Bởi vì, theo mặc định lớp ObjectDemo mở rộng lớp Object.

Chương trình 4.11*Class ObjectDemo*

```
{  
    public static void main(String args[])  
    {  
        if (args[0].equals("Aptech"));  
        System.out.println("Yes, Aptech is the right choice!");  
    }  
}
```

4.7 Gói java.util

Gói Java.util cung cấp một vài lớp Java hữu ích nhất, được cần đến thường xuyên trong tất cả các loại chương trình ứng dụng. Nó giới thiệu các lớp phi trừu tượng sau:

- Hashtable
- Random
- Vector
- StringTokenizer

4.7.1 Lớp Hashtable

Lớp Hashtable mở rộng lớp trừu tượng Dictionary, lớp này cũng được định nghĩa trong gói java.util. *Hashtable* được sử dụng để ánh xạ các khoá đến các giá trị. Ví dụ, nó có thể được sử dụng để ánh xạ các tên đến tuổi, những người lập trình đến những dự án, các tiêu đề công việc đến các lương, và cứ tiếp tục như vậy.

Hashtable mở rộng kích thước khi các phần tử được thêm vào. Khi đó việc tạo một bảng băm mới, bạn có thể chỉ định một dung lượng ban đầu và các yếu tố nạp vào. Điều này sẽ làm cho *hashtable* tăng kích thước lên, bất cứ lúc nào việc thêm vào một phần tử mới sẽ làm thay đổi giới hạn của *hashtable* cũ. Giới hạn của bảng băm là dung lượng được nhân lên bởi các yếu tố được nạp vào. Ví dụ: một bảng băm với dung lượng 100, và một yếu tố nạp vào là 0.75 sẽ có một giới hạn là 75 mục. Các phương thức xây dựng cho bảng băm được biểu diễn trong bảng sau:

Constructor	Purpose
Hashtable(int)	Xây dựng một bảng mới với dung lượng ban đầu được chỉ định.
Hashtable(int, float)	Xây dựng một lớp mới với dung lượng ban đầu được chỉ định và yếu tố nạp vào.
Hashtable()	Xây dựng một lớp mới bằng cách sử dụng giá trị mặc định cho dung lượng ban đầu và yếu tố nạp vào.

Bảng 4.7 Các phương thức xây dựng Hashtable.

Hashtable hash1 = new Hashtable(500,0,80);

Trong trường hợp này, Bảng băm “hash1” sẽ lưu trữ 500 phần tử. Khi bảng băm lưu trữ vừa đầy 80% (một yếu tố nạp vào của .80), kích thước tối đa của nó sẽ được tăng lên.

Mỗi phần tử trong một hashtable bao gồm một khoá và một giá trị. Các phần tử được thêm vào bảng băm bằng cách sử dụng phương thức put(), và được truy lục bằng cách sử dụng phương thức get(). Các phần tử có thể được xoá từ một bảng băm với phương thức remove(). Các phương thức contains() và containsKey() có thể được sử dụng để tra cứu một giá trị hoặc một khoá trong bảng băm. Một vài phương thức của Hashtable được tóm tắt trong bảng sau:

Phương thức	Mục đích
clear()	Xoá tất cả các phần tử từ bảng băm.
Clone()	Tạo một bảng sao của Hashtable.
contains(Object)	Trả về True nếu bảng băm chứa các đối tượng được chỉ định.
containsKey(Object)	Trả về True nếu bảng băm chứa khoá được chỉ định.
elements()	Trả về một bảng liệt kê các yếu tố trong bảng băm.
get(Object key)	Truy lục đối tượng được kết hợp với khoá được chỉ định.
isEmpty()	Trả về true nếu bảng băm trống.
keys()	Trả về một bảng liệt kê các khoá trong bảng băm.
put(Object, Object)	Thêm một phần tử mới vào bảng băm bằng cách sử dụng khoá và giá trị được chỉ định.
rehash()	Thay đổi bảng băm thành một bảng băm lớn hơn.
remove(Object key)	Xoá một đối tượng được cho bởi khoá được chỉ định.
size()	Trả về số phần tử trong bảng băm.
toString()	Trả về đại diện chuỗi được định dạng cho bảng băm.

Bảng 4.8 Các phương thức lớp Hashtable.

Chương trình sau sử dụng lớp Hashtable. Trong chương trình này, tên của các tập ảnh là các khoá, và các năm là các phần tử.

“contains” được sử dụng để tra cứu phần tử nguyên 1969, để thấy có danh sách chứa bất kỳ các tập ảnh từ 1969.

“containsKey” được sử dụng để tìm kiếm cho khoá “Animals”, để nhìn thấy nếu tập ảnh đó tạo nên danh sách.

Phương thức “get()” được sử dụng để truy lục tập ảnh “Wish You Were Here” có trong bảng băm không. Phương thức get() trả về phần tử kết hợp với khoá, cả hai tên và năm được hiển thị tại điểm này.

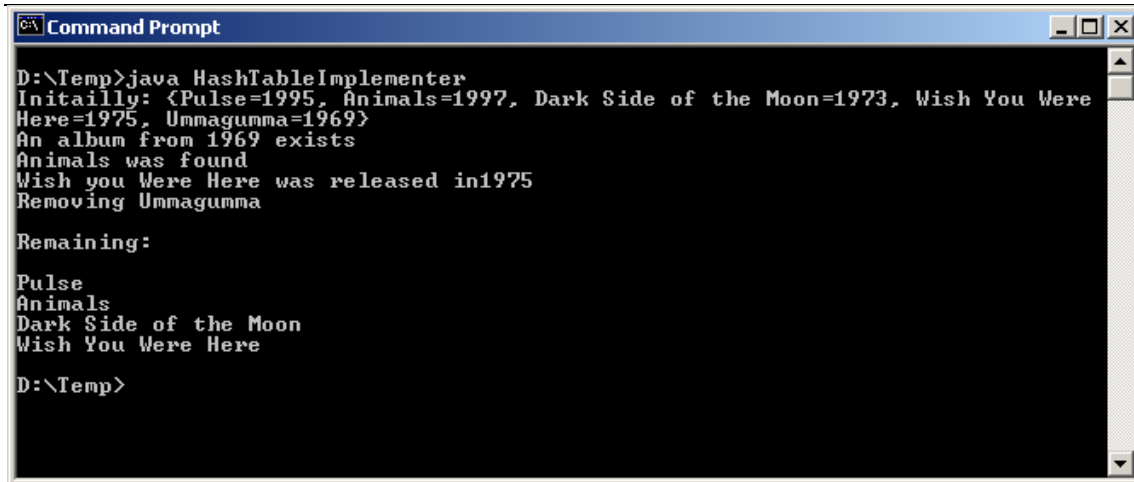
Chương trình 4.12

```
import java.util.*;
```

```
public class HashTableImplementer
```

```
{  
    public static void main(String args[])  
    {  
        //tạo một bảng băm mới  
        Hashtable ht = new Hashtable();  
        //thêm các tập ảnh tốt nhất của Pink Floyd  
        ht.put("Pulse", new Integer(1995));  
        ht.put("Dark Side of the Moon", new Integer(1973));  
        ht.put("Wish You Were Here", new Integer(1975));  
        ht.put("Animals", new Integer(1997));  
        ht.put("Ummagumma", new Integer(1969));  
        //Hiển thị bảng băm  
        System.out.println("Initailly: "+ht.toString());  
        //kiểm tra cho bất kỳ tập ảnh nào từ 1969  
        if(ht.contains(new Integer(1969)))  
            System.out.println("An album from 1969 exists");  
        //kiểm tra cho tập ảnh các con thú  
        if(ht.containsKey("Animals"));  
        System.out.println("Animals was found");  
        //Tìm ra  
        Integer year = (Integer)ht.get("Wish You Were Here");  
        System.out.println("Wish you Were Here was released in "+year.toString());  
        //Xoá một tập ảnh  
        System.out.println("Removing Ummagumma\r\n");  
        ht.remove("Ummagumma");  
        //Di chuyển thông qua một bảng liệt kê của tất cả các khoá trong bảng.  
        System.out.println("Remaining:\r\n");  
        for(Enumeration enum = ht.keys(); enum.hasMoreElements();)  
            System.out.println((String)enum.nextElement());  
    }  
}
```

Quá trình hiển thị kết quả sẽ được mô tả dưới đây:



```

D:\Temp>java HashTableImplementer
Initailly: <Pulse=1995, Animals=1997, Dark Side of the Moon=1973, Wish You Were
Here=1975, Ummagumma=1969>
An album from 1969 exists
Animals was found
Wish you Were Here was released in1975
Removing Ummagumma

Remaining:

Pulse
Animals
Dark Side of the Moon
Wish You Were Here
D:\Temp>
  
```

Hình 4.4 Quá trình hiển thị kết quả của HashTableImplementer

4.7.2 Lớp random

Lớp này đại diện một bộ tạo số giả ngẫu nhiên (pseudo-random). Hai phương thức xây dựng được cung cấp. Một trong những phương thức xây dựng này lấy giá trị khởi đầu như một tham số. Phương thức xây dựng khác thì không lấy giá trị như một tham số, và sử dụng thời gian hiện tại như một giá trị khởi đầu. Việc xây dựng một bộ tạo số ngẫu nhiên với một giá trị khởi đầu là một ý kiến hay, trừ khi bạn muốn bộ tạo số ngẫu nhiên luôn tạo ra một tập các giá trị giống nhau. Mặt khác, thỉnh thoảng nó hữu dụng để tạo ra trình tự giống nhau của các số random. Điều này có ý nghĩa trong việc gỡ rối một chương trình. Một khi bộ tạo số ngẫu nhiên được tạo ra, bạn có thể sử dụng bất kỳ các phương thức sau đây để truy lục một giá trị từ nó:

- nextDouble()
- nextFloat()
- nextGaussian()
- nextInt()
- nextLong()

Các phương thức xây dựng và các phương thức của lớp Random được tóm tắt trong bảng sau:

Phương thức	Mục đích
random()	tạo ra một bộ tạo số ngẫu nhiên mới
random(long)	Tạo ra một bộ tạo số ngẫu nhiên mới dựa trên giá trị khởi tạo được chỉ định.
nextDouble()	Trả về một giá trị kiểu double kế tiếp giữa 0.0D đến 1.0D từ bộ tạo số ngẫu nhiên.
nextFloat()	Trả về một giá trị kiểu float kế tiếp giữa 0.0F và 1.0F từ bộ tạo số ngẫu nhiên.
nextGaussian()	Trả về kiểu double được phân phối Gaussian kế tiếp từ bộ tạo số ngẫu nhiên. Tạo ra các giá trị Gaussian sẽ có một giá trị trung bình của 0, và một độ lệch tiêu chuẩn của 1.0.
nextInt()	Trả về giá trị kiểu Integer kế tiếp từ một bộ tạo số ngẫu nhiên.

nextLong()	Trả về giá trị kiểu long kế tiếp từ một bộ tạo số ngẫu nhiên.
setSeed(long)	Thiết lập giá trị khởi tạo từ bộ tạo số ngẫu nhiên.

Bảng 4.9 Các phương thức lớp Random.

4.7.3 Lớp Vector

Một trong các vấn đề với một mảng là chúng ta phải biết nó lớn như thế nào khi chúng ta tạo nó. Nó thì không thể xác định kích thước của mảng trước khi tạo nó.

Lớp Vector của Java giải quyết vấn đề này. Nó cung cấp một dạng mảng với kích thước ban đầu, mảng này có thể tăng thêm khi nhiều phần tử được thêm vào. Một lớp Vector lưu trữ các item của kiểu Object, nó có thể dùng để lưu trữ các thể hiện của bất kỳ lớp Java nào. Một lớp Vector đơn lẻ có thể lưu trữ các phần tử khác nhau, các phần tử khác nhau này là thể hiện của các lớp khác nhau.

Tại bất kỳ thời điểm, một lớp Vector có dung lượng để lưu trữ một số nào đó của các phần tử. Khi một lớp Vector biết về dung lượng của nó, thì dung lượng của nó được gia tăng bởi một số lượng riêng cho Vector đó. Lớp Vector cung cấp ba phương thức xây dựng khác nhau mà có thể chúng ta chỉ định dung lượng khởi tạo, và tăng số lượng của một Vector, khi nó được tạo ra. Các phương thức xây dựng này được tóm tắt trong bảng sau:

Phương thức Constructor	Mục đích
Vector(int)	Tạo ra một lớp Vector mới với dung lượng ban đầu được chỉ định.
Vector(int, int)	Tạo ra một lớp Vector mới với dung lượng ban đầu được chỉ định, và tăng số lượng.
Vector()	Tạo ra một lớp Vector mới với dung lượng khởi tạo mặc định, và tăng số lượng.

Bảng 4.10 các phương thức xây dựng của lớp Vector.

Một mục (item) được thêm vào một lớp Vector bằng cách sử dụng hàm addElement(). Tương tự, một phần tử có thể được thay thế bằng cách sử dụng hàm setElementAt(). Một lớp Vector có thể tìm kiếm bằng cách sử dụng phương thức contains(), phương thức này trông có vẻ dễ dàng cho một lần xuất hiện của một đối tượng (Object). Phương thức elements() thì hữu dụng bởi vì nó trả về một bảng liệt kê của các đối tượng được lưu trữ trong lớp Vector. Các phương thức này và các phương thức thành viên khác của lớp Vector được tóm tắt trong bảng dưới đây:

Phương thức	Mục đích
addElement(Object)	Chèn các phần tử được chỉ định vào lớp Vector.
capacity()	Trả về số phần tử mà sẽ vừa đủ cho phần được cấp phát hiện thời của lớp Vector.
Clone()	Bắt chước vector, nhưng không phải là các phần tử của nó.
contains(Object)	Trả về True nếu lớp Vector chứa đối tượng được chỉ định.
copyInto(Object [])	Sao chép các phần tử của lớp Vector vào mảng được chỉ định.
elementAt(int)	Truy lục phần tử được cấp phát tại chỉ mục được chỉ định.

elements()	Trả về một bảng liệt kê của các phần tử trong lớp Vector.
ensureCapacity(int)	Chắc chắn rằng lớp Vector có thể lưu trữ ít nhất dung lượng tối thiểu được chỉ định.
firstElement()	Trả về phần tử đầu tiên trong lớp Vector.
indexOf(Object)	Tìm kiếm lớp Vector, và trả về chỉ mục zero cơ bản cho khớp với đối tượng đầu tiên.
indexOf(Object, int)	Tìm kiếm lớp Vector đang bắt đầu tại số chỉ mục được chỉ định, và trả về chỉ mục zero cơ bản cho khớp với đối tượng kế tiếp.
insertElementAt(Object, int)	Thêm các đối tượng được chỉ định tại chỉ mục được chỉ định.
isEmpty()	Trả về True nếu lớp Vector không có phần tử.
lastElement()	Trả về phần tử cuối cùng trong lớp Vector.
lastIndexOf(Object)	Tìm kiếm lớp Vector, và trả về chỉ mục zero cơ bản cho khớp với đối tượng cuối cùng.
lastIndexOf(Object, int)	Tìm kiếm lớp Vector đang bắt đầu tại số chỉ mục được chỉ định, và trả về chỉ mục zero cơ bản cho khớp với đối tượng trước.
removeAllElements()	Xoá tất cả các phần tử từ lớp Vector.
removeElement(Object)	Xoá đối tượng được chỉ định từ lớp Vector.
removeElementAt(int)	Xoá đối tượng tại chỉ mục được chỉ định.
setElementAt(Object, int)	Thay thế đối tượng tại chỉ mục được chỉ định với đối tượng được chỉ định.
setSize(int)	Thiết lập kích thước của lớp Vector thành kích thước mới được chỉ định.
setSize(int)	Thiết lập kích thước của lớp Vector thành kích thước mới được chỉ định.
Size()	Trả về số của các phần tử hiện thời trong lớp Vector.
toString()	Trả về một đại diện chuỗi được định dạng nội dung của lớp Vector.
trimToSize()	Định lại kích thước của lớp Vector để di chuyển dung lượng thừa trong nó.

Bảng 4.11 Các phương thức lớp Vector

Chương trình sau tạo ra một lớp Vector “vect”. Nó chứa 6 phần tử: “Numbers In Words”, “One”, “Two”, “Three”, “Four”, “Five”. Phương thức removeElement() được sử dụng để xoá các phần tử từ “vect”.

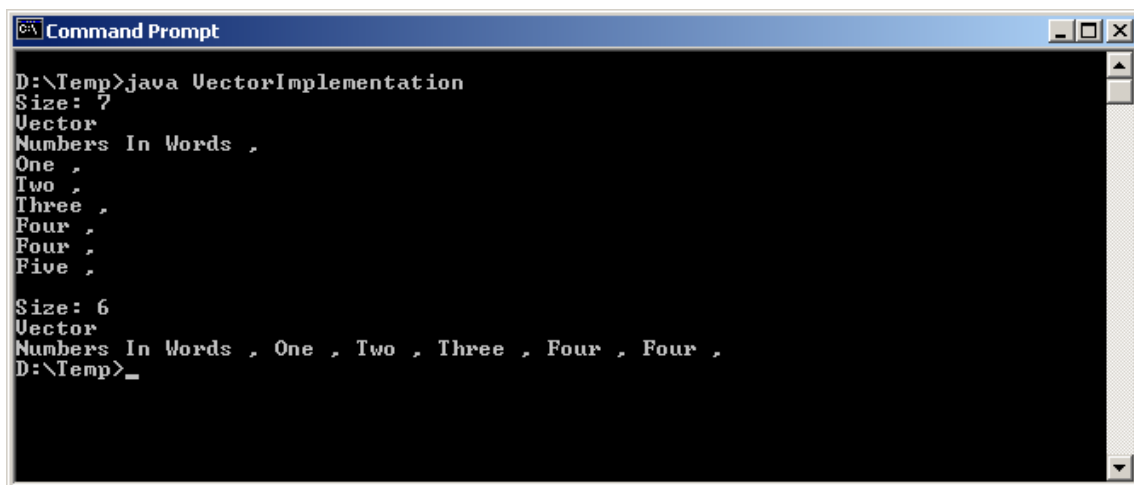
Chương trình 4.13

```
import java.util.*;

public class VectorImplementation
{
    public static void main(String args[])
    {
        Vector vect = new Vector();
        vect.addElement("One");
    }
}
```

```
vect.addElement("Two");
vect.addElement("Three");
vect.addElement("Four");
vect.addElement("Five");
vect.insertElementAt("Numbers In Words",0);
vect.insertElementAt("Four",4);
System.out.println("Size: "+vect.size());
System.out.println("Vector ");
for(int i = 0; i<vect.size(); i++)
{
    System.out.println(vect.elementAt(i)+" , ");
}
vect.removeElement("Five");
System.out.println("");
System.out.println("Size: "+vect.size());
System.out.println("Vector ");
for(int i = 0;i<vect.size();i++)
{
    System.out.print(vect.elementAt(i)+ " , ");
}
}
}
```

Quá trình hiển thị kết quả sẽ được mô tả như hình dưới.



```
Command Prompt
D:\Temp>java VectorImplementation
Size: 7
Vector
Numbers In Words ,
One ,
Two ,
Three ,
Four ,
Four ,
Five ,
Size: 6
Vector
Numbers In Words , One , Two , Three , Four , Four ,
D:\Temp>
```

Hình 4.5 Quá trình hiển thị kết quả của chương trình lớp Vector.

4.7.4 Lớp StringTokenizer

Một lớp StringTokenizer có thể sử dụng để tách một chuỗi thành các phần tử token của nó. Ví dụ, mỗi từ trong một câu có thể coi như là một token. Tuy nhiên, lớp StringTokenizer đã đi xa hơn việc phân tách của các câu. Để tạo nên một mã thông báo đầy đủ theo yêu cầu, bạn có thể chỉ định một bộ dấu phân cách token, khi lớp StringTokenizer được tạo ra. Dấu phân cách khoảng trắng mặc định thì thường có khả năng để tách văn bản. Tuy nhiên, chúng ta có thể sử dụng tập các toán tử toán học (+, *, /, và -) trong khi phân tách một biểu thức. Các ký tự phân cách có thể chỉ định khi một đối tượng StringTokenizer mới được xây dựng. Bảng sau tóm tắt 3 phương thức xây dựng có sẵn:

Phương thức xây dựng	Mục đích
StringTokenizer(String)	Tạo ra một lớp StringTokenizer mới dựa trên chuỗi chỉ định được thông báo.
StringTokenizer	Tạo ra một lớp StringTokenizer mới dựa trên (String, String) chuỗi chỉ định được thông báo, và một tập các dấu phân cách.
StringTokenizer(String, String, Boolean)	Tạo ra một lớp StringTokenizer dựa trên chuỗi chỉ định được thông báo, một tập các dấu phân cách, và một cờ hiệu cho biết nếu các dấu phân cách sẽ được trả về như các token.

Bảng 4.12 Các phương thức xây dựng của lớp StringTokenizer.

Các phương thức xây dựng ở trên được sử dụng trong các ví dụ sau:

```
StringTokenizer st1 = new StringTokenizer("A Stream of words");
```

```
StringTokenizer st2 = new StringTokenizer("4*3/2-1+4", "+/-", true);
```

```
StringTokenizer st3 = new StringTokenizer("aaa,bbbb,ccc", ",", true);
```

Trong câu lệnh đầu tiên, StringTokenizer của "st1" sẽ được xây dựng bằng cách sử dụng các chuỗi được cung cấp và các dấu phân cách mặc định. Các dấu phân cách mặc định là khoảng trắng, tab, dòng mới, và các ký tự xuống dòng. Các dấu phân cách này thì hữu dụng khi phân tách văn bản, như với "st1".

Câu lệnh thứ hai trong ví dụ trên xây dựng một lớp StringTokenizer cho các biểu thức toán học bằng cách sử dụng các ký hiệu *, +, /, và -.

Câu lệnh thứ 3, StringTokenizer của "st3" sẽ thông báo chuỗi được cung cấp chỉ bằng cách sử dụng ký tự dấu phẩy như một dấu phân cách.

Lớp StringTokenizer thực thi giao diện bằng liệt kê. Vì thế, nó bao gồm các phương thức hasMoreElements() và nextElement(). Các phương thức non-private của lớp StringTokenizer được tóm tắt trong bảng sau:

Phương thức	Mục đích
countTokens()	Trả về số các token còn lại.
hasMoreElements()	Trả về True nếu nhiều phần tử đang được đánh dấu trong chuỗi. Nó thì giống hệt như hasMoreTokens.
hasMoreTokens()	Trả về True nếu nhiều tokens đang được đánh dấu trong chuỗi. Nó thì giống hệt như hasMoreElements.

nextElement()	Trả về phần tử kế tiếp trong chuỗi. Nó thì giống như nextToken.
nextToken()	Trả về Token kế tiếp trong chuỗi. Nó thì giống như nextElement.
nextToken(String)	Thay đổi bộ dấu phân cách đến chuỗi được chỉ định, và sau đó trả về token kế tiếp trong chuỗi.

Bảng 4.13 Các phương thức lớp StringTokenizer.

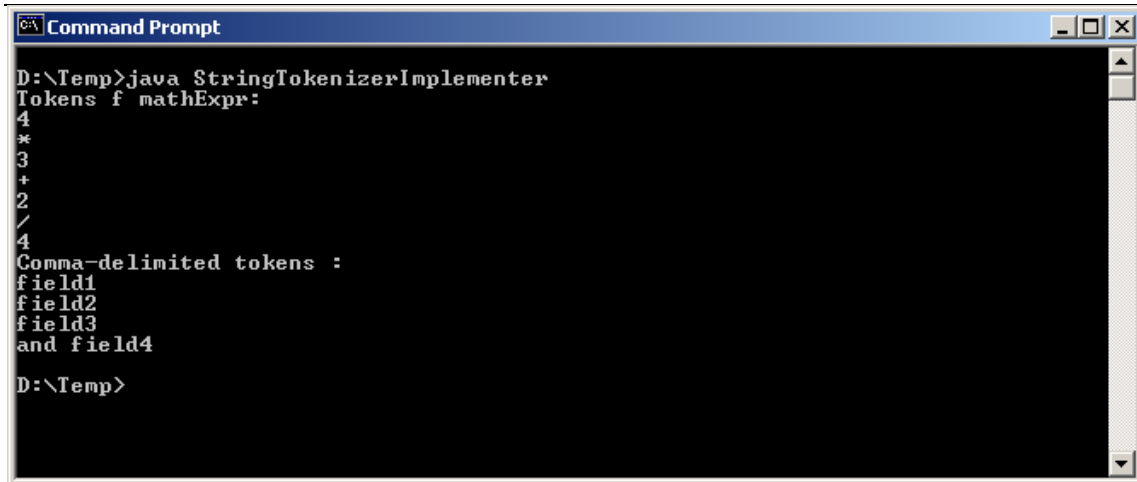
Hãy xem xét chương trình đã cho ở bên dưới. Trong ví dụ này, hai đối tượng StringTokenizer đã được tạo ra. Đầu tiên, “st1” được sử dụng để phân tách một biểu thức toán học. Thứ hai, “st2” phân tách một dòng của các trường được phân cách bởi dấu phẩy. Cả hai tokenizer, phương thức hasMoreTokens() và nextToken() được sử dụng để lặp đi lặp lại thông qua tập các token, và sau đó được hiển thị.

Chương trình 4.13

```
import java.util.*;

public class StringTokenizerImplementer
{
    public static void main(String args[])
    {
        // đặt một biểu thức toán học trong một chuỗi và tạo một tokenizer cho chuỗi đó.
        String mathExpr = "4*3+2/4";
        StringTokenizer st1 = new StringTokenizer(mathExpr, "*+/-", true);
        //trong khi có các token trái, hãy hiển thị mỗi token
        System.out.println("Tokens f mathExpr: ");
        while(st1.hasMoreTokens())
            System.out.println(st1.nextToken());
        //tạo một chuỗi của các trường được phân cách bởi dấu phẩy và tạo một tokenizer cho chuỗi.
        String commas = "field1,field2,field3,and field4";
        StringTokenizer st2 = new StringTokenizer(commas, ",", false);
        //trong khi có các token trái, hãy hiển thị mỗi token.
        System.out.println("Comma-delimited tokens : ");
        while (st2.hasMoreTokens())
            System.out.println(st2.nextToken());
    }
}
```

Quá trình hiển thị kết quả sẽ được mô tả như hình dưới.



```
C:\> Command Prompt

D:\Temp>java StringTokenizerImplementer
Tokens f mathExpr:
4
*
3
+
2
/
4
Comma-delimited tokens :
field1
field2
field3
and field4

D:\Temp>
```

Hình 4.6 Quá trình hiển thị kết quả của lớp StringTokenizer.

Tóm tắt bài học

- Khi không có sự thi hành để thừa kế, một giao diện được sử dụng thay cho một lớp ảo.
- Một gói là một thư mục mà bạn tổ chức các giao diện và các lớp của bạn.
- Một CLASSPATH là một danh sách của các thư mục để giúp đỡ tìm kiếm cho tập tin lớp tương ứng.
- Lớp java.lang.Math cung cấp các phương thức để thực hiện các hàm toán học.
- Các kiểu dữ liệu nguyên thủy có thể được vận dụng và được truy cập thông qua các lớp trình bao bọc của chúng.
- Các lớp String được sử dụng để tạo và chế tác các chuỗi, các chuỗi có thể được gán, có thể được so sánh và được nối vào nhau.
- Một chuỗi mặc định đại diện cho tất cả các chữ đã tạo ra trong một chương trình.
- Lớp StringBuffer cung cấp các phương thức khác nhau để vận dụng một đối tượng chuỗi. Các đối tượng của lớp này thì linh động. Đó là, các ký tự hoặc các chuỗi có thể được chèn vào giữa đối tượng StringBuffer, hoặc được nối vào vị trí cuối cùng của chuỗi.
- Lớp Runtime đóng gói môi trường thời gian chạy.
- Lớp System cung cấp các thuận lợi như là, xuất, nhập chuẩn, và các luồng lỗi.
- java.util chứa các lớp phi trừu tượng sau:
 - Hashtable
 - Random
 - Vector
 - StringTokenizer
- Lớp Hashtable có thể được sử dụng để tạo một mảng của các khoá và các giá trị. Nó cho phép các phần tử được tra cứu bởi khoá hoặc giá trị.
- Lớp Random là một bộ tạo số ngẫu nhiên giả mà có thể trả về các giá trị kiểu integer, dấu phẩy động (floating-point), hoặc Gaussian-distributed.

- Lớp Vector có thể sử dụng để lưu trữ bất kỳ các đối tượng nào. Nó có thể lưu trữ các đối tượng của nhiều hơn một kiểu trong các vector tương tự.
- Lớp StringTokenizer cung cấp một cơ chế động cho việc phân tách các chuỗi.

Kiểm tra kiến thức của bạn

1.sẽ luôn đến đầu tiên giữa các gói, các khai báo và lớp trong chương trình Java.
2. Một giao diện có thể chứa nhiều các phương thức. **True/False**
3. Trong khi việc tạo một gói, thì mã nguồn phải hiện có trong thư mục tương tự, thư mục đó chính là tên của gói bạn. **True/False**
4. Một.....là một danh sách của các thư mục, mà giúp đỡ tìm kiếm cho các tập tin lớp tương ứng.
5. Lớp bao bọc cho các kiểu dữ liệu double và long cung cấpvà.....các hằng.
6.phương thức được sử dụng để thay thế một ký tự trong lớp StringBuffer, với một ký tự khác tại vị trí được chỉ định.
7. Một.....được sử dụng để ánh xạ các khoá thành các giá trị.
8.Phương thức của lớp StringTokenizer trả về số của các token còn lại.

Bài tập

1. Tạo một giao diện và sử dụng nó trong một chương trình của Java để hiển thị bình phương và lũy thừa 3 của một số.
2. Tạo một gói và viết một hàm, hàm đó trả về giai thừa của một đối số được truyền đến trong một chương trình.
3. Viết một chương trình bằng cách sử dụng các hàm của lớp Math để hiển thị bình phương của các số lớn nhất và nhỏ nhất của một tập các số được nhập vào bởi người sử dụng bằng dòng lệnh.
4. Hãy tạo ra sổ ghi nhớ của chính bạn, nơi mà những con số được nhập vào như sau:

Joy	34543
Jack	56765
Tina	34567

Bảng 4.14**Chương trình phải làm như sau:**

- Kiểm tra xem số 3443 có tồn tại trong sổ ghi nhớ của bạn hay không.
- Kiểm tra xem mẫu tin của Jack có hiện hữu trong sổ ghi nhớ của bạn hay không.
- Hiển thị số điện thoại của Tina.
- Xóa số điện thoại của Joy.
- Hiển thị các mẫu tin còn lại.

5. Viết một chương trình mà nhập vào một số điện thoại tại dòng lệnh, như một chuỗi có dạng (091) 022-6758080. Chương trình sẽ hiển thị mã quốc gia (091), mã vùng (022), và số điện thoại (6758080) (Sử dụng lớp StringTokenizer).