Chương : LẬP TRINH HƯỚNG ĐỐI TƯỢNG TRONG JAVA

Java là một ngôn ngữ lập trình hướng đối tượng. Nếu bạn chưa bao giờ dùng một ngôn ngữ lập trình hướng đối tượng trước đây, bạn cần phải hiểu các khái niệm sau : lập trình hướng đối tượng (Object Oriented Programming) là gì ? đối tượng (Object), lớp (class) là gì, mối quan hệ giữa đối tượng và lớp, gởi thông điệp (Messages) đến các đối tượng là gì ?

I. KHÁI NIỆM LẬP TRINH HƯỚNG ĐỐI TƯỢNG

1. Lập trình hướng đối tượng (Object Oriented Programming)

Mỗi một chương trình máy tính đều gồm có 2 phần : phần mã lệnh và phần dữ liệu. Một số chương trình đặt trọng tâm ở phần mã lệnh, số khác đặt trọng tâm ở phần dữ liệu. Từ đó dẫn đến 2 mô hình quyết định nên cấu trúc của chương trình : một trả lời cho câu hỏi "Điều gì đang xảy ra", và một cho "Cái gì đang chịu tác động". Mô hình 1 gọi là mô hình hướng xử lý, nó mô tả như là một chương trình bao gồm một chuỗi các bước thực hiện (mã lệnh). Nhưng khi chương trình càng ngày càng lớn và phức tạp thì khó khăn để sử dụng mô hình thứ nhất.

Vì vậy mô hình thứ 2 được đưa ra, đó là mô hình hướng đối tượng. Chương trình của bạn sẽ xây dựng dựa vào dữ liệu và phần giao diện được định nghĩa cho phần dữ liệu đó. Mô hình này được mô tả như là dữ liệu điều khiển truy xuất đối với mã lệnh.

Ngôn ngữ lập trình hướng đối tượng có các khả năng sau :

- Mô phỏng thế giới thực một cách tự nhiên bởi các đối tượng và mối quan hệ giữa chúng, thuận tiện cho việc thiết kế hệ thống phức tạp
- Thừa kế mã có sẵn một cách dễ dàng, giúp tiết kiệm công sức và nâng cao năng suất của người lập trình, dễ bảo trì, dễ nâng cấp, mở rộng

2. Trừu tượng hoá (Abstraction)

Con người đã đơn giản hoá các vấn đề phức tạp thông qua sự trừu tượng hoá. Ví dụ, người sử dụng máy tính không nhìn máy tính một cách phức tạp. Nhờ sự trừu tượng hoá mà người ta có thể sử dụng máy tính mà không quan tâm đến cấu trúc chi tiết bên trong máy tính. Họ chỉ sử dụng chúng như là một thực thể

Cách tốt nhất để nắm vững kỹ thuật trừu tượng là dùng hệ thống phân cấp. Điều này cho phép bạn phân lớp các thành phần có ý nghĩa của cả hệ thống phức tạp, chia nhỏ chúng thành những phần đơn giản có thể quản lý được. Nhìn bên ngoài máy tính là một đối tượng, nếu nhìn sâu hơn một cấp, máy tính bao gồm một số bộ phận : hộp điều khiển, màn hình, bàn phím, chuột..., các bộ phận này lại bao gồm các bộ phận nhỏ hơn, ví dụ như hộp điều khiển có bảng mạch chính chứa CPU, các mạch giao tiếp gắn trên bảng mạch chính, đĩa cứng, ổ đĩa mềm… Nhờ sự trừu tượng hoá mà bạn không quan tâm đến chi tiết từng bảng mạch, mà chỉ quan tâm mối quan hệ, giao tiếp giữa các bộ phận. Một mạch giao tiếp dù có chức năng ly kỳ thế nào đi nữa, bạn có thể sử dụng không mấy khó khăn nếu được ấn vừa vặn vào khe cắm trên bảng mạch chính.

Sự phân cấp trừu tượng một hệ thống phức tạp có thể áp dụng cho các chương trình máy tính. Phần dữ liệu từ một chương trình hướng xử lý kinh điển có thể trừu tượng hoá thành các đối tượng thành phần. Dãy các xử lý trở thành các thông điệp giữa các đối tượng. Vì thế các đối tượng cần có hoạt động đặc trưng riêng. Bạn có thể coi các đối tượng này như những thực thể độc lập tiếp nhận các yêu cầu từ bên ngoài. Đây là phần cốt lõi của lập trình hướng đối tượng.

II. CƠ CHẾ TRIỂN KHAI MÔ HINH HƯỚNG ĐỐI TƯỢNG

Tất cả các ngôn ngữ lập trình hướng đối tượng đều có các cơ chế cho phép bạn triển khai các mô hình hướng đối tượng. Đó là tính đóng gói, kế thừa, và tính đa hình.

1. Tính đóng gói (Encapsulation)

Đây là cơ chế dùng một vỏ bọc kết hợp phần dữ liệu và các thao tác trên dữ liệu đó (phần mã lệnh) thành một thể thống nhất, tạo nên sự an toàn, tránh việc sử dụng không đúng thiết kế, bảo vệ cho mã lệnh và dữ liệu chống việc truy xuất từ những đoạn mã lệnh bên ngoài.

Trong Java tính đóng gói thể hiện qua khái niệm lớp (Class). Lớp là hạt nhân của Java, tạo nền tảng cho lập trình hướng đối tượng trong Java. Nó định nghĩa dữ liệu và các hành vi của nó (dữ liệu và mã lệnh), gọi là các thành viên của lớp, dùng chung cho các đối tượng cùng loại. Từ sự phân tích hệ thống, người ta trừu tượng nên các lớp. Sau đó các đối tượng được tạo ra theo khuôn mẫu của lớp. Mỗi đối tượng thuộc một lớp có dữ liệu và hành vi định nghĩa cho lớp đó, giống như là sinh ra từ một khuôn đúc của lớp đó. Vì vậy mà lớp là khuôn mẫu của đối tượng, đối tượng là thể hiện của một lớp. Lớp là cấu trúc logic, còn đối tượng là cấu trúc vật lý. Dữ liệu định nghĩa trong lớp gọi là biến, mã lệnh gọi là phương thức. Phương thức định nghĩa cho việc sử dụng dữ liệu như thế nào. Điều này có nghĩa là hoạt động của lớp được định nghĩa thông qua phương thức.

Các đặc trưng của lớp gồm có hai phần chính : thuộc tính (Attribute) và hành vi (Behavior). Giả sử bạn phải tạo ra giao diện với người dùng và cần có những nút nhấn (Button). Thế thì trước hết bạn xây dựng lớp Button với các thuộc tính như nhãn ghi trên nút, chiều rộng, chiều cao, màu của nút, đồng thời quy định hành vi của nút nhấn, nghĩa là nút nhấn cần phản ứng như thế nào khi được chọn, phát yêu cầu gì, có đổi màu hay nhấp nháy chi không. Với lớp Button như vậy, bạn có thể tạo ra nhanh chóng những nút nhấn cụ thể phục vụ cho các mục đích khác nhau

Gói là kỹ thuật của Java, dùng để phân hoạch không gian tên lớp, giao diện thành những vùng dễ quản lý hơn, thể hiện tính đóng gói của Java.

2. Tính kế thừa (Inheritance)

Tính kế thừa là khả năng xây dựng các lớp mới từ các lớp đã có. Tính đóng gói cũng tác động đến tính kế thừa. Khi lớp đóng gói một số dữ liệu và phương thức, lớp mới sẽ kế thừa mọi cấu trúc dữ liệu và các phương thức của lớp mà nó kế thừa. Ngoài ra nó có thể bổ sung các dữ liệu và các phương thức của riêng mình.

Nó rất quan trọng vì nó ứng dụng cho khái niệm cây phân cấp (mô hình TopDown). Không sử dụng cây phân lớp, mỗi lớp phải định nghĩa tất cả các dữ liệu và phương thức của mình một cách rõ ràng. Nếu sử dụng sự kế thừa, mỗi lớp chỉ cần định nghĩa thêm những đặc trưng của mình.

Ví dụ : Xe có thể xem như một lớp và các xe Pergout, BWM, Dream là các đối tượng của lớp xe. Các xe đều có thể lái đi, dừng lại... Từ lớp xe ở trên, ta có thể xây dựng các lớp xe đạp, xe ôtô. Xe ôtô có thêm máy và có thể tự khởi động…

3. Tính đa hình (Polymorphism)

Khi một lớp được kế thừa từ các lớp tổ tiên thì nó có thể thay đổi cách thức làm việc của lớp tổ tiên trong một số phương thức nào đó (nhưng tên, kiểu trả về, danh sách tham đối của phương thức thì vẫn giữ nguyên). Điều này gọi là viết chồng. Như vậy với một tên phương thức, chương trình có thể có các hành động khác nhau tùy thuộc vào lớp của đối tượng gọi phương thức. Đó là tính đa hình

Ví dụ : với một phương thức chạy, xe ôtô, xe máy có thể tăng ga, còn xe đạp thì phải đạp…

Tính đa hình còn thể hiện ở việc một giao diện có thể sử dụng cho các hoạt động của một lớp tổng quát, hay còn gọi là "một giao diện, nhiều phương thức". Có nghĩa là có thể thiết kế một giao diện tổng quát cho một nhóm các hành vi liên quan. Điều này giảm thiểu sự phức tạp bằng cách cho phép một giao diện có thể sử dụng cho các hoạt động của một lớp tổng quát. Trình biên dịch sẽ xác định hoạt động cụ thể nào sẽ được thi hành tùy theo điều kiện. Bạn chỉ cần nhớ các giao diện của lớp tổng quát và sử dụng nó.

Sự kết hợp đúng đắn giữa : đa hình, đóng gói và kế thừa tạo nên một môi trường lập trình có khả năng phát triển tốt hơn rất nhiều so với môi trường không hỗ trợ hướng đối tượng. Một cây phân cấp lớp thiết kế tốt là điều căn bản cho việc sử dụng lại những đoạn mã lệnh mà bạn đã tốn công sức nhiều cho việc phát triển và kiểm tra. Tính đóng gói cho phép bạn sử dụng các đối tượng và ra lệnh thi hành tới chúng mà không phá vỡ cấu trúc các đoạn mã lệnh đã bảo vệ bởi giao diện của các lớp. Sự đa hình cho phép bạn tạo ra những đoạn mã lệnh gọn gàng, dễ đọc, dễ hiểu và có tính ổn định.

Java là ngôn ngữ lập trình hướng đối tượng nên có đầy đủ các tính năng trên, thư viện lớp Java được cung cấp khá đầy đủ cho người lập trình để bắt đầu một dự án mới

Chương : ĐỐI TƯỢNG VÀ LỚP, MẢNG

I. XÂY DỰNG LỚP

Khi định nghĩa một lớp, bạn chỉ ra thuộc tính mà nó chứa được thể hiện bằng biến (Member Variable) và hành vi được thể hiện bởi hàm (Method)

Các biến định nghĩa bên trong một lớp gọi là các biến thành viên (Member Variables). Mã lệnh chứa trong các phương thức (Method). Các phương thức và biến định nghĩa trong lớp gọi chung là thành phần của lớp. Trong hầu hết các lớp, các biến thể hiện được truy cập bởi các phương thức định nghĩa trong lớp đó. Vì vậy, chính các phương thức quyết định dữ liệu của lớp có thể dùng như thế nào. Lớp định nghĩa một kiểu dữ liệu mới, dùng để tạo các đối tượng thuộc kiểu đó.

Dạng đầy đủ của một định nghĩa lớp như sau :

[public]	Lớp được truy xuất chung cho các Package khác, mặc định chỉ có các đoạn mã trong cùng một gói mới có quyền truy xuất nó
[abstract]	Lớp trừu tượng, không thể khởi tạo
[final]	Lớp hằng không có lớp con, không kế thừa
class *ClassName*	Tên lớp
[extends *SuperClass*]	Kế thừa lớp cha SuperClass
[implements *Interfaces*]	Giao diện được cài đặt bởi Class
{ //Member Variables Declarations	Khai báo các biến
// Methods Declarations	Khai báo các phương thức
}	

Ví dụ : Tạo một lớp Box đơn giản với ba biến : width, height, depth

```
/* Định nghĩa lớp
*/
class Box {
        double width;
        double height;
        double depth;
}
```

II. TAO ĐỐI TƯỢNG

1. Khai báo đối tượng

Để có được các đối tượng của một lớp phải qua hai giai đoạn :

- ***ClassName ObjectName;***

Ví dụ : Box myBox

Khai báo biến myBox có kiểu lớp Box. Khai báo này thực ra không cấp phát ký ức đủ chứa đối tượng thuộc lớp Box, mà chỉ tạo ra quy chiếu trỏ đến đối tượng Box. Sau câu lệnh này, quy chiếu myBox xuất hiện trên ký ức chứa giá trị null chỉ ra rằng nó chưa trỏ đến một đối tượng thực tế nào

Khác với câu lệnh khai báo biến kiểu sơ cấp là dành chỗ trên ký ức đủ chứa một trị thuộc kiểu đó :

Ví dụ : int i;

Sau câu lệnh này, biến nguyên i hình thành.

- Sau đó, để thực sự tạo ra một đối tượng và gán địa chỉ của đối tượng cho biến này,

dùng toán tử new

ObjectName* = new *ClassName();

Ví dụ : myBox = new Box();

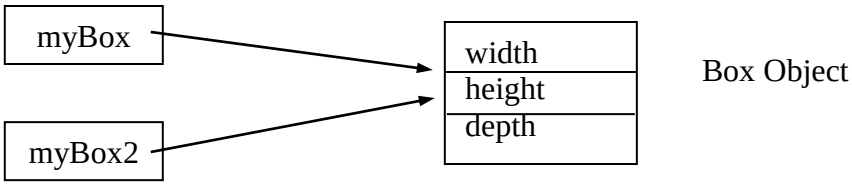
◆ Có thể kết hợp cả hai bước trên vào một câu lệnh :

ClassName ObjectName* = new *ClassName();

Ví dụ : Box myBox = new Box();

Box myBox2 = myBox;

myBox2 tham chiếu đến cùng đối tượng mà myBox tham chiếu



2. Cách truy xuất thành phần của lớp

◆ Biến khai báo trong định nghĩa lớp gồm có hai loại :

- Biến đối tượng (Instance Variable hay Object Variable) : chỉ thuộc tính đối tượng, khi truy xuất phải khởi tạo đối tượng

+ Cách khai báo biến đối tượng :

Type InstanceVar;

+ Cách truy cập biến đối tượng :

ObjectName.InstanceVar

- Biến lớp (Class Variable) : về bản chất là biến toàn cục, là biến tĩnh được tạo lập một lần cùng với lớp, dùng chung cho mọi đối tượng thuộc lớp, khi truy xuất không cần khởi tạo đối tượng, để trao đổi thông tin của các đối tượng cùng lớp

+ Cách khai báo biến lớp :

static *Type ClassVar;*

+ Cách truy cập biến lớp :

ClassName.ClassVar

◆ Hàm khai báo trong định nghĩa lớp gồm có hai loại :

- Hàm đối tượng (Object Method) : cách truy xuất hàm đối tượng như biến đối tượng

***ObjectName.ObjectMethod*(*Parameter-List*)**

- Hàm lớp (Class Method) : thông thường một thành phần của lớp chỉ truy xuất trong sự liên kết với một đối tượng thuộc lớp của nó. Tuy nhiên, có thể tạo ra một thành phần mà có thể dùng một độc lập một mình, không cần tham chiếu đến một đối tượng cụ thể, có thể được truy xuất trước khi bất kỳ đối tượng nào của lớp đó được tạo ra, bằng cách đặt trước khai báo của nó từ khoá static. Cách truy xuất hàm lớp :

***ClassName.ClassMethod*(*Parameter-List*)**

Các hàm toán học của lớp Math trong Package Java.Lang là hàm lớp nên khi gọi không cần phải khởi tạo đối tượng

Ví dụ : double a = Math.sqrt(453.28);

Ví dụ 1: class BaiTho {

```
        static int i;              // Biến lớp
        String s;                  // Biến đối tượng
        BaiTho(String ss) {        // Hàm khởi tạo
            s = ss;
            i++;
```

```
        }
        void content( ) {
            System.out.println(s);
        }
    }
    class UngDung {
        public static void main(String args[]){
            BaiTho p1 = new BaiTho("Chi co thuyen moi hieu");
            BaiTho p2 = new BaiTho("Bien menh mong nhuong nao");
            p1.content();
            p2.content();
            System.out.println("So cau tho la : "+BaiTho.i);
        }
    }
```

Khi tạo đối tượng p1, p2 bởi toán tử new, hàm dựng BaiTho() được gọi, và i tăng lên 1

Ví dụ 2:

```
    class BaiTho2 {
        static int i;
        String s;
        BaiTho2(String ss) {   // Hàm khởi tạo
            s = ss;  i++;
        }
        static int number() {    // Hàm lớp
            return i;
        }
        String content() {       // Hàm đối tượng
            return s;
        }
    }
    class UngDung2 {
        public static void main (String args[]) {
            System.out.println("Bai tho co "+BaiTho2.number()+" cau");
            BaiTho2.p1 = new BaiTho2("Chi co thuyen moi hieu");
            BaiTho2.p2 = new BaiTho2("Bien menh mong nhuong nao");
            System.out.println("Bai tho co "+BaiTho2.number()+" cau");
            System.out.println("Cau tho\n"+p1.content().toUpperCase()+"\nco" +
                p1.content().length() +" ky tu");
            System.out.println("Tu \"tinh yeu\" bat dau sau ky tu thu"+
                p2.content().indexOf("tinh yeu")+" trong cau\n"+
                p2.content().toUpperCase());
        }
    }
```

Gọi hàm lớp BaiTho2.number() lúc chưa gọi hàm dựng BaiTho2 để khởi tạo đối tượng sẽ cho trị 0

p1.content() trả về một đối tượng String

III. GIỚI THIỆU VỀ PHƯƠNG THỨC

1. Khai báo phương thức (hàm)

Dạng tổng quát của một phương thức như sau :

[*acess*]	**điều khiển truy xuất**
[static]	**hàm lớp**
[abstract]	**hàm trừu tượng**
[final]	**hàm hằng**
[*Type*] *MethodName*(*Parameter-List*) throws *exceptions* { **// Body of method }**	

- Type : Kiểu dữ liệu do hàm trả về, có thể là kiểu bất kỳ, kể cả các kiểu lớp do bạn tạo ra. Nếu hàm không trả về giá trị nào, kiểu trả về của nó phải là void.
- Các hàm có kiểu trả về không phải là void sẽ trả về một giá trị cho chương trình gọi nó dùng dạng câu lệnh return như sau :

return biểu thức;

Giá trị của biểu thức được tính và trả về cho hàm
- Tất cả thông tin bạn muốn truyền được gởi thông qua tham số nằm trong hai dấu ( ) ngay sau tên hàm. Nếu không có tham số vẫn phải có ( )

Parameter-List : Danh sách tham đối phân cách bởi các dấu phẩy, mỗi tham đối phải được khai báo kiểu, có thể là kiểu bất kỳ, có dạng : *Type Parameter1, Type Parameter2 ...*

2. Phạm vi truy xuất thành phần của lớp

Các điều khiển truy xuất của Java là public, private và protected. protected chỉ áp dụng khi có liên quan đến kế thừa sẽ xét đến sau

Khi bổ sung tiền tố cho một thành phần của lớp (biến và hàm) là :
- Từ khoá public : chỉ ra rằng thành phần này có thể được truy xuất bởi bất kỳ dòng lệnh nào dù ở trong hay ngoài lớp mà nó khai báo
- private : chỉ có thể được truy xuất trong lớp của nó, mọi đoạn mã nằm ngoài lớp, kể cả những lớp con đều không có quyền truy xuất
- Khi không có điều khiển truy xuất nào được dùng, mặc nhiên là public nhưng chỉ trong gói của nó, không thể truy xuất từ bên ngoài gói của nó

3. Phương thức main()

Khi chạy ứng dụng độc lập, bạn chỉ tên Class muốn chạy, Java tìm gọi hàm main() trước tiên trong Class đó, phương thức main sẽ điều khiển chạy các phương thức khác.

Dạng tổng quát của phương thức main()

```
public static void main(String args[]) {
        // Body of  Method
}
```

- Một chương trình chỉ cần một lớp có phương thức main() gọi là lớp ứng dụng độc lập Primary Class.
- Từ khoá static cho phép hàm main() được gọi khi không cần khởi tạo đối tượng. Vì main() được trình thông dịch của Java gọi trước khi bất kỳ lớp nào được khởi tạo
- Từ khoá void cho biết hàm main() không trả về giá trị
- Từ khoá public chỉ ra rằng hàm này được gọi bởi dòng lệnh bên ngoài lớp khi chương trình khởi động.
- Tham đối String args[ ] khai báo tham số tên args thuộc lớp String, chứa chuỗi ký tự. Tham đối này giữ các tham đối dòng lệnh dùng khi thi hành chương trình.

Ví dụ 1 :

```
class ViDu {
        public static void main (String args[]) {
                for (int i=0; i < args.length; i++) {
```

```
                        System.out.println("Tham doi thu "+i+": "+args[i]);
                    }
                }
        }
```

Khi chạy chương trình :

```
C:\>java ViDu Thu tham doi dong lenh ↵
Tham doi thu 0 : Thu
Tham doi thu 1 : tham ....
C:>java ViDu Thu "tham doi" "dong lenh" ↵
Tham doi thu 0 : Thu
Tham doi thu 1 : tham doi
Tham doi thu 2 : dong lenh
```

Ví dụ 2 :

```
class ViDu2;
public static void main(String args[]) {
        int sum = 0;
        float avg = 0;
        for (int i=0; i<args.length;i++) {
                sum += Integer.parseInt(args[i]);
                }
        System.out.println("Tong ="+sum);
        System.out.println("Trung binh ="+ (float) sum/args.length);
        }
}
```

Khi chạy chương trình :

```
C:\>java ViDu2 1 2 3 ↵
Tong = 6
Trung binh = 2
```

4. Hàm khởi tạo (Constructor)

Có những thao tác cần thực hiện mỗi khi đối tượng lần đầu tiên được tạo như khởi tạo giá trị cho các biến. Các công việc này có thể làm tự động bằng cách dùng hàm khởi tạo.

Hàm khởi tạo có cùng tên với lớp mà nó thuộc về, chỉ được tự động gọi bởi toán tử new khi đối tượng thuộc lớp được tạo. Hàm khởi tạo không có giá trị trả về, khi định nghĩa hàm có thể ghi void hay không ghi.

Ví dụ : - kích thước hộp được khởi tạo tự động khi đối tượng được tạo.

```
class Box {
        double width;
        double height;
        double depth;
        double volume() {
                return width * height * depth;
        }
        Box(double w, double h, double d) {
                width = w;
                height = h;
                depth = d;
        }
}
class BoxDemo {
```

```
public static void main (String args[ ]) {
        Box myBox1 = new Box(10,20,15);
        Box myBox2 = new Box(3,6,9);
        double vol;
        vol = myBox1.volume();
        System.out.println("Thể tích là : "+vol);
        vol = myBox2.volume();
        System.out.println("Thể tích là : "+vol);
        }
}
```

- Khi bạn không định nghĩa tường minh hàm khởi tạo cho một lớp, Java sẽ tạo hàm khởi tạo mặc nhiên cho lớp đó. Vì vậy các chương trình trước đó vẫn làm việc bình thường. Hàm khởi tạo mặc nhiên không có danh sách tham đối, tự động khởi tạo tất cả các biến của đối tượng về trị rỗng theo các quy ước mặc định của Java, trị 0 cho kiểu số, ký tự '\0' cho kiểu ký tự char, trị false cho kiểu boolean, trị null cho các đối tượng
- Hàm khởi tạo cũng có thể được nạp chồng như hàm bình thường (sẽ nói rõ ở phần sau) nghĩa là ta được phép định nghĩa nhiều hàm khởi tạo khác nhau ở danh sách tham đối hay kiểu tham đối

5. Hàm hủy

Các đối tượng cấp phát động bằng toán tử new, khi không tồn tại tham chiếu nào đến đối tượng, đối tượng đó xem như không còn cần đến nữa và bộ nhớ cho nó có thể được tự động giải phóng bởi bộ thu gom rác (garbage collector). Trình thu gom rác hoạt động trong một tuyến đoạn (Thread) độc lập với chương trình của bạn. Bạn không phải bận tâm gì đối với công việc này. Sau này bạn sẽ hiểu rõ tuyến đoạn là thế nào

Tuy nhiên, Java cũng cho phép ta viết hàm hủy, có thể cũng cần thiết cho những trường hợp nào đó. Hàm hủy trong Java chỉ được gọi bởi trình thu gom rác, do vậy bạn khó đoán trước vào lúc nào hàm hủy sẽ được gọi

Dạng hàm hủy như sau :

```
protected void finalize() {
        // Body of Method
}
```

6. Từ khoá this

Nếu biến được định nghĩa trong thân hàm, đó là biến cục bộ chỉ tồn tại khi hàm được gọi. Nếu biến cục bộ như vậy được đặt tên trùng với biến đối tượng hoặc biến lớp, nó sẽ che khuất biến đối tượng hay biến lớp trong thân hàm :

Ví dụ :

```
class ViDu {
        int test = 10;                                  // Biến đối tượng
        void printTest() {
                int test = 20;                          // Biến cục bộ
                System.out.println("test = "+test);     // In biến cục bộ
        }
        public static void main(String args[]) {
                ViDu a = new ViDu();
                a.printTest();
        }
}
```

Từ khoá this có thể dùng bên trong bất cứ phương thức nào để tham chiếu đến đối tượng hiện hành, khi biến đối tượng trùng tên với biến cục bộ.

Ví dụ : Thay dòng lệnh trên :

```
System.out.println("test = "+this.test);          // In biến cục bộ, this chỉ đối tượng a
```

7. Nạp chồng hàm (Overloaded Methods)

Trong cùng một lớp, Java cho phép bạn định nghĩa nhiều hàm trùng tên với điều kiện các hàm như vậy phải có danh sách tham đối khác nhau, nghĩa là khác nhau về số tham đối hoặc kiểu của các tham đối. Khả năng như vậy gọi là sự nạp chồng hàm. Java chỉ phân biệt hàm này với hàm khác dựa vào số tham đối và kiểu của các tham đối, bất chấp tên hàm và kiểu của kết quả trả về.

Ví dụ :

```
//  MyRect.java
import java.awt.Point;
class MyRect {
        int x1 = 0;
        int y1 = 0;
        int x2 = 0;
        int y2 = 0;
        MyRect buildRect(int x1, int y1, int x2, int y2) {
                this.x1 = x1;
                this.y1 = y1;
                this.x2 = x2;
                this.y2 = y2;
                return this;
        }
        MyRect buildRect(Point topLeft, Point bottomRight) {
                x1 = topLeft.x;
                y1 = topLeft.y;
                x2 = bottomRight.x;
                y2 = bottomRight.y;
                return this;
        }
        MyRect buildRect(Point topLeft, int w, int h) {
                x1 = topLeft.x;
                y1 = topLeft.y;
                x2 = x1+w;
                y2 = y1 + h;
                return this;
        }
        void display() {
                System.out.print("Doi tuong MyRect : <" + x1 + ", "+y1);
                System.out.println(", "+x2+", "+y2+">");
        }
}
```

Thật ra, trong gói awt có sẵn lớp Rectangle chuyên dùng để biểu diễn hình chữ nhật. Lớp MyRect của ta chỉ dùng để minh hoạ cho khái niệm nạp chồng hàm. Trong lớp MyRect có những hàm giúp bạn tạo ra đối tượng MyRect với những yếu tố cho trước khác nhau :

- Cho trước toạ độ góc trên trái x1, y1 và toạ độ góc dưới phải x2, y2
- Cho trước góc trên trái và góc dưới phải của hình chữ nhật dưới dạng đối tượng Point
- Cho trước toạ độ góc trên trái của hình chữ nhật dạng đối tượng Point cùng chiều rộng, chiều cao

Nhờ khả năng nạp chồng hàm, bạn chỉ cần nhớ một tên hàm cho các hàm khác nhau cùng chức năng

Chương trình sử dụng lớp MyRect xây dựng ở trên :

```
import java.awt.Point;
class UngDung {
    public static void main(String args[]) {
        MyRect rect = new MyRect();
        rect.buildRect(25,25,50,50);
        rect.display();
        rect.buildRect(new Point(10,10), new Point(20,20));
        rect.display();
        rect.buildRect(new Point(10,10), 50, 50);
        rect.display();
    }
}
```

8. Truyền tham đối

Java dùng cả hai cách truyền tham đối : truyền bằng giá trị và truyền bằng tham chiếu, tùy vào cái gì được truyền

- Khi ta truyền một kiểu sơ cấp cho phương thức, nó sẽ truyền bằng giá trị. Vì vậy những gì xảy ra với tham đối trong phương thức, khi ra khỏi phương thức sẽ hết tác dụng
- Khi ta truyền một đối tượng (kiểu phức hợp) cho phương thức, nó sẽ truyền bằng tham chiếu. Vì vậy, thay đổi ở đối tượng bên trong phương thức ảnh hưởng đến đối tượng dùng làm tham đối.

Ví dụ 1 :

```
class ViDu {
    void tinhToan(int i, int j) {
        i *= 2;
        j /= 2;
    }
}
class UngDung {
    public static void main(String args) {
        ViDu o = new ViDu();
        int a = 15, b = 20;
        System.out.println("a và b trước khi gọi : "+a+ " "+b);
        o.tinhToan(a, b);
        System.out.println("a và b sau khi gọi : "+a+" "+b);
    }
}
```

Kết quả của chương trình :

```
a và b trước khi gọi : 15 20
a và b sau khi gọi : 15 20
```

Ví dụ 2 :

```
class ViDu {
    int a, b;
    ViDu (int i, int j) {
        a = i;
        b = j;
    }
```

```
        void tinhToan(ViDu o) {
            o.a *= 2;
            0.b /= 2;
        }
}
class UngDung {
        public static void main(String args[]) {
            ViDu o = new ViDu(15, 20);
            System.out.println("o.a và o.b trước khi gọi : "+o.a+" "+o.b);
            o.tinhToan(o);
            System.out.println("o.a và o.b sau khi gọi : "+o.a+" "+o.b);
        }
}
```

Kết quả chương trình :

o.a và o.b trước khi gọi : 15 20

o.a và o.b sau khi gọi : 30 10

IV. LỚP KẾ THỪA

1. Khai báo kế thừa

Ta có thể sử dụng tính kế thừa tạo lớp tổng quát có những đặc tính chung đại diện cho một tập hợp các đối tượng có cùng mối quan hệ. Sau đó, lớp này có thể được kế thừa bởi một hay nhiều lớp khác và những đặc tính này trở thành những thành những đặc tính của lớp kế thừa

- Lớp được kế thừa gọi là lớp cha (SuperClass : là lớp cha trực tiếp)
- Lớp kế thừa gọi là lớp con (SubClass)

Lớp con kế thừa tất cả các biến và hàm định nghĩa trong lớp cha

class *ClassName* extends *SuperClass*

{ //Member Variables Declarations, Methods

}

- Mặc dù vậy, lớp con không thể truy xuất các thành phần được khai báo private trong lớp cha
- Một biến tham chiếu của lớp cha có thể gán để tham chiếu đến một lớp con bất kỳ dẫn xuất từ lớp cha. Khi một tham chiếu đến một lớp con được gán cho biến tham chiếu kiểu lớp cha, ta chỉ có quyền truy xuất những phần được định nghĩa bởi lớp cha.

2. Viết chồng hàm hay che khuất hàm (Overriding Methods)

Trong phân cấp lớp, khi một hàm của lớp con có cùng tên, và giống nhau về số lượng và kiểu tham đối cũng như kiểu trả về với một hàm ở lớp cha, thì hàm ở lớp con được gọi là viết chồng hàm trong lớp cha. Khi đó hàm của lớp con sẽ che khuất hàm thừa kế từ lớp cha

Tuy nhiên lớp con không được viết chồng hàm hằng (có khai báo final) và hàm lớp trong lớp cha.

Ví dụ : Tất cả các lớp là hậu duệ của lớp Object. Lớp Object chứa phương thức toString, mà trả về một đối tượng String chứa tên lớp của đối tượng. Hầu hết các lớp con viết chồng phương thức này và in ra một vài điều gì đó có nghĩa cho lớp đó

3. Từ khoá super

Đôi khi bạn không muốn thực hiện viết chồng một phương thức mà chỉ muốn thêm chức năng vào phương thức. Để làm được điều này, bạn gọi phương thức được viết chồng dùng từ khoá super. Từ khoá super dùng khi lớp con cần tham chiếu lớp cha trực tiếp của nó. Super có hai dạng cú pháp :

- Dạng 1 : Hàm khởi tạo lớp cha phải được gọi trước hàm khởi tạo của lớp con. Nếu trong định

nghĩa hàm khởi tạo ở lớp con không có câu lệnh gọi hàm khởi tạo lớp cha, trình biên dịch Java sẽ tự động đưa vào câu lệnh gọi hàm khởi tạo mặc định của lớp cha có dạng : *classname*()
Bạn có thể tự thêm lệnh gọi hàm khởi tạo ở lớp cha có dạng như sau :

super(*Parameter-List*)

Parameter-List là danh sách các tham đối cần thiết cho hàm khởi tạo của lớp cha. super() phải luôn luôn là phát biểu đầu tiên được thực hiện trong hàm khởi tạo của lớp con
Ví dụ :

```
class MyPoint {
        int x, y;
        MyPoint(int x, int y) {
                this.x = x;
                this.y = y;
        }
        void display() {
                System.out.print("x = "+x+", y = "+y+"\n");
        }
}
class MyPoint2 extends MyPoint {
        int z;
        String name;
        MyPoint2(int x, int y, int z, String name) {
                super(x,y);                     // Khởi tạo 2 biến x, y bằng cách gọi
                this.z = z;                     // hàm dựng của lớp cha
                this.name = name;
        }
        void display() {        // Viết chồng hàm kế thừa từ lớp cha
                System.out.print("x = "+x+", y = "+y+", z = "+z+" "+"name
:"+name+"\n");
        }
}
```

- Dạng 2 : dùng để hàm lớp con truy xuất hàm kế thừa từ lớp cha :

super.*Member*

Member có thể là phương thức hay biến của đối tượng
Ví dụ : Viết lại hàm display() trong class MyPoint2, có gọi hàm kế thừa từ lớp cha :

```
        void display() {
                super.display();
                System.out.print(", z = "+z+" "+"name :"+name+"\n");
        }
```

V. LỚP, PHƯƠNG THỨC TRỪU TƯỢNG

Trong trường hợp chúng ta muốn định nghĩa một lớp cha theo một cấu trúc trừu tượng cho trước mà không cần hiện thực đầy đủ các phương thức. Tức là ta muốn tạo một lớp cha có dạng chung cho tất cả các lớp con và để các lớp con hiện thực chi tiết. Khi đó, bạn muốn chắc chắn lớp con có chồng lắp phương thức. Những phương thức phải được chồng lắp trong lớp con gọi là phương thức trừu tượng, được khai báo abstract và không có phần thân phương thức

abstract [*Type*] *MethodName*(*Parameter-List*) ;

Bất kỳ lớp nào chứa một hay nhiều phương thức trừu tượng cũng phải khai báo trừu tượng, sử dụng từ khoá abstract trước từ khoá class. Không thể khởi tạo đối tượng kiểu lớp trừu tượng, vì lớp trừu tượng không được định nghĩa đầy đủ. Do đó, bạn cũng không thể khai báo hàm khởi tạo. Bất kỳ lớp con nào cũng phải hoặc là viết chồng tất cả các phương thức trừu tượng

hoặc chính nó lại được khai báo abstract

Ví dụ : Trong các ứng dụng, bạn có thể vẽ đường tròn, hình chữ nhật, đoạn thẳng, đường cong… Mỗi một đối tượng đồ hoạ này đều chứa các thuộc tính (vị trí, nét viền) và hành vi (di chuyển, thay kích thước, vẽ). Bạn có thể khai báo chúng kế thừa lớp Graphic. Tuy nhiên vẽ một đường tròn là hoàn toàn khác với vẽ một hình chữ nhật, nên lớp Graphic được khai báo là lớp trừu tường, chứa các phương thức đã được hiện thực như moveTo, và phương thức trừu tượng như draw

```
abstract class GraphicObject {
        int x, y;
        . . .
        void moveTo(int newX, int newY) {
        . . .
        }
        abstract void draw();
}
```

Mỗi một lớp con không trừu tượng của lớp Graphic như Circle, Rectangle sẽ phải cài đặt đầy đủ cho phương thức draw

```
class Circle extends GraphicObject {
        void draw() {
        . . .
        }
}
class Rectangle extends GraphicObject {
        void draw() {
        . . .
        }
}
```

VI. LỚP HẰNG (KHÔNG KẾ THỪA), HÀM HẰNG (KHÔNG VIẾT CHỒNG)

1. Sử dụng từ khoá final cấm sự chồng lắp

Mặc dù chồng lắp phương thức là một trong những đặc điểm mạnh nhất của Java, tuy nhiên trong vài trường hợp bạn muốn cấm điều này. Để cấm một phương thức lớp con viết chồng phương thức ở lớp cha, bạn đưa từ khoá final vào đầu khai báo

Ví dụ :

```
class Box {
        double width;
        double height;
        double depth;
        …
        final double volume() {
                return width * height * depth;
        }
           . . .
}
```

2. Sử dụng từ khoá final cấm sự kế thừa

Muốn khai báo một lớp mà không có lớp con kế thừa, bạn sử dụng từ khoá final. Với một lớp final, thì tất cả các phương thức của nó sẽ là final.

Ta không thể khai báo một lớp vừa abstract và final vì một lớp trừu tượng là một lớp chưa hoàn chỉnh và phải có lớp con để hiện thực đầy đủ

Ví dụ : final class Box { . . .
}

VII. LỚP LỒNG NHAU

Có thể định nghĩa một lớp bên trong một lớp khác. Lớp như vậy gọi là lớp lồng (Nested Class) và được cài đặt như sau :

```
class EnclosingClass{ // Lớp bao bên ngoài
    . . .
    static class StaticNestedClass { // Lớp lồng tĩnh
        . . .
    }
    class InnerClass { // Lớp lồng phi tĩnh hay lớp nội bộ
        . . .
    }
}
```

Lớp lồng chỉ được biết bên trong tầm vực của lớp bao bên ngoài. Bộ dịch Java sẽ báo lỗi nếu một đoạn mã bất kỳ bên ngoài lớp bao cố dùng trực tiếp lớp lồng.

Một lớp lồng có quyền truy cập đến các thành viên của lớp bao bên ngoài, thậm chí nếu chúng được khai báo private. Tuy nhiên, lớp bao không thể truy xuất các thành phần của lớp lồng.

Có hai kiểu lớp lồng : tĩnh và phi tĩnh.

Lớp lồng tĩnh (static nested class) được bổ sung từ khoá static. Nó không thể tham chiếu trực tiếp đến biến hay phương thức đối tượng được định nghĩa trong lớp bao, mà chỉ dùng chúng thông qua đối tượng. Vì giới hạn này nên lớp lồng tĩnh ít được dùng. Hầu hết các lớp lồng là lớp nội bộ

Lớp lồng phi tĩnh (nonstatic nested class) không bổ sung từ khoá static, còn được gọi là lớp nội bộ (inner class). Nó có thể truy cập trực tiếp đến các biến và phương thức đối tượng.

```
class Outer {
    int outer_x = 100;
    void test() {
        Inner inner = new Inner();
        inner.display_x();
    }
    class Inner {    // có thể truy xuất trực tiếp biến đối tượng của lớp Outer
        int inner_y = 10;
        void display_x() {
            System.out.println("display : outer_x = " + outer_x);
        }
    }
    void display_y() { // không thể truy xuất biến đối tượng của lớp Inner
        System.out.println("display : inner_y = " + inner_y); // Error
    }
}
class InnerClassDemo {
    public static void main(String args[]) {
        Outer outer = new Outer();
        outer.test();
    }
}
```

VIII. CHUYỂN ĐỔI KIỂU

1. Chuyển đổi giữa các kiểu phức hợp

Java chỉ cho phép chuyển đổi đối tượng thuộc lớp con cháu thành đối tượng của lớp cha ông (Ancestors), và không cho chuyển ngược lại

Giả sử bạn có đối tượng thuộc lớp con Child và cần chuyển đổi thành đối tượng thuộc lớp cha ông Parent. Java cho phép dùng đối tượng Child một cách tự nhiên ở bất cứ chỗ nào dành cho đối tượng Parent, ta không cần làm động tác chuyển đổi nào cả. Đối tượng Child có đầy đủ thuộc tính và hành vi của đối tượng Parent nên có thể "vào vai" đối tượng Parent. Nếu muốn, bạn cũng có thể chuyển đổi đối tượng thuộc lớp con cháu thành đối tượng thuộc lớp cha ông một cách tường minh, nhưng không cần thiết :

```
Child c = new Child();
Parent p = (Parent) c;
```

2. Chuyển đổi kiểu sơ cấp thành kiểu phức hợp

Trong gói java.lang có sẵn những lớp tương ứng với các kiểu sơ cấp, có thể dùng thay cho kiểu sơ cấp : lớp Integer thay cho kiểu int, lớp Boolean cho kiểu boolean, lớp Float cho kiểu float, lớp Double cho kiểu double… Lớp Number là lớp cha của mọi lớp bọc kiểu

Chẳng hạn, muốn cho kiểu int có thể xuất hiện như một đối tượng thuộc lớp Integer :

```
Integer intObj = new Integer(25);
```

Lớp Integer được trang bị những phương thức giúp bạn nhiều việc mà kiểu int không thể đảm đương.

- Lấy giá trị nguyên mà đối tượng intObj nắm giữ :

```
int i = intObj.intValue();
```

IX. MẢNG (ARRAY)

Mảng là một cấu trúc lưu giữ các thành phần có cùng kiểu. Chiều dài một mảng được thiết lập và cố định khi mảng được tạo lúc chạy chương trình. Mỗi thành phần của mảng được truy xuất bởi chỉ số của nó trong mảng

Nếu bạn muốn lưu giữ các thành phần khác kiểu nhau hay kích thước mảng có thể thay đổi động, dùng một Vector thay cho mảng

1. Tạo và sử dụng mảng

- ***Khai báo một biến tham chiếu đến mảng***

ArrayType*[] *ArrayName

Khai báo một biến có kiểu *ArrayType* dùng để tham chiếu đến mảng, nhưng không có mảng nào thật sự tồn tại

ArrayType : là kiểu dữ liệu của các thành phần chứa trong mảng và dấu [] chỉ định đó là một mảng

Kiểu dữ liệu thành phần có thể là bất kỳ kiểu cơ sở, tham chiếu

```
int[] anArrayOfInts;        // Khai báo một mảng số nguyên
float[] anArrayOfFloats;
boolean[] anArrayOfBooleans;
Object[] anArrayOfObjects;
String[] anArrayOfStrings;
```

- ***Tạo một mảng***

Bạn dùng toán tử new để tạo một mảng, nghĩa là cấp phát bộ nhớ cho các thành phần và gán mảng đến biến đã khai báo

ArrayName = new ArrayType[ArraySize]

ArraySize : là số thành phần của mảng
Ví dụ : int[] M; // khai báo biến mảng kiểu số nguyên
M = new int[10]; // tạo một mảng số nguyên
Bạn có thể kết hợp sự khai báo biến mảng và tạo mảng như sau :
ArrayType*[] *ArrayName = new ArrayType[ArraySize]
Có thể viết như sau :
ArrayType ArrayName*[] *= new ArrayType[ArraySize]
Ví dụ : int[] M = new int[10];
int M[] = new int[10];

- ***Truy xuất thành phần của mảng***

ArrayVar[index]
index : chỉ vị trí của thành phần trong mảng cần truy xuất, có thể là giá trị, biến hay biểu thức, và có giá trị từ 0 đến *ArraySize*-1
Ví dụ : M[1] = 20;

- ***Lấy kích thước mảng***

***ArrayName*.length**

- ***Khởi tạo giá trị đầu của mảng***

Mảng có thể khởi tạo khi khai báo. Mảng khởi tạo là danh sách các biểu thức cách nhau bởi dấu phẩy và bao quanh bởi dấu ngoặc móc. Mảng sẽ được khởi tạo tự động để lưu số phần tử mà bạn xác định lúc khởi tạo, không cần sử dụng new. Chiều dài của mảng là số giá trị giữa { và }
Ví dụ : boolean[] answers = { true, false, true, true, false };
int month_days[] = {31,28,31,30,31,30,31,31,30,31,30,31};
Ví dụ 1: Tạo và sử dụng mảng có thành phần kiểu cơ sở

```
public class ArrayDemo {
    public static void main(String[] args) {
        int[] anArray;
        anArray = new int[10];
        for (int i = 0; i < anArray.length; i++) {
            anArray[i] = i;
            System.out.print(anArray[i] + " ");
        }
        System.out.println();
    }
}
```

Ví dụ 2 : Tạo và sử dụng mảng có thành phần kiểu tham lớp String

```
public class ArrayOfStringsDemo {
    public static void main(String[] args) {
        String[] anArray = { "String One", "String Two", "String Three" };
        for (int i = 0; i < anArray.length; i++) {
            System.out.println(anArray[i].toLowerCase());
        }
    }
}
```

2. Mảng đa chiều (Arrays of Arrays)

Mảng có thể chứa các thành phần là mảng. Để khai báo một biến mảng đa chiều cần xác định mỗi chiều của mảng bằng cách sử dụng các cặp dấu ngoặc vuông.
Ví dụ : int M[][] = new int[4][5];
int[][] M = new int[4][5];

M là một mảng 4x5 thành phần là các số nguyên.

Khi cấp phát bộ nhớ cho mảng đa chiều, bạn có thể chỉ định chiều dài của mảng chính, và không chỉ định chiều dài của mảng con cho đến khi tạo chúng

Ví dụ : int M[][] = new int[3][];

```
M[0] = new int[3];
M[1] = new int[4];
M[2] = new int[2];
```

Ví dụ 1 :

```
public class ArrayOfArraysDemo {
    public static void main(String[] args) {
        String[][] cartoons = {
            { "Flintstones", "Fred", "Wilma", "Pebbles", "Dino" },
            { "Rubbles", "Barney", "Betty", "Bam Bam" },
            { "Jetsons", "George", "Jane", "Elroy", "Judy", "Rosie", "Astro" },
            { "Scooby Doo Gang", "Scooby Doo", "Shaggy", "Velma", "Fred",
            "Daphne" }
        };
        for (int i = 0; i < cartoons.length; i++) {
            System.out.print(cartoons[i][0] + ": ");
            for (int j = 1; j < cartoons[i].length; j++) {
                System.out.print(cartoons[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

Chú ý là tất cả mảng con có chiều dài khác nhau. Tên của mảng con là cartoons[0], cartoons[1]...

Ví dụ 2 :

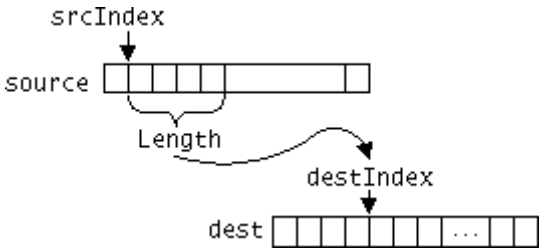
```
public class ArrayOfArraysDemo2 {
    public static void main(String[] args) {
        int[][] aMatrix = new int[4][];
        for (int i = 0; i < aMatrix.length; i++) {
            aMatrix[i] = new int[5];
            for (int j = 0; j < aMatrix[i].length; j++) {
                aMatrix[i][j] = i + j;
            }
        }
        for (int i = 0; i < aMatrix.length; i++) {
            for (int j = 0; j < aMatrix[i].length; j++) {
                System.out.print(aMatrix[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

3. Sao chép mảng (Copying Arrays)

Sử dụng phương thức arraycopy của System sao chép dữ liệu từ một mảng đến một mảng khác. Phương thức arraycopy yêu cầu 5 tham đối :

public static void arraycopy(*ArrayType*[] *source*,
int *srcIndex*,
***ArrayType*[] *dest*,**
int *destIndex*,
int *length*)

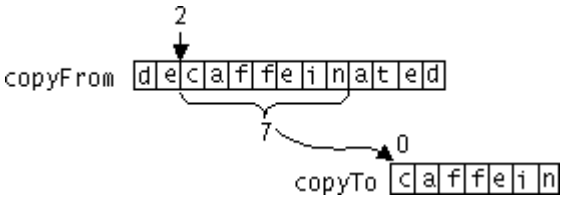
Hai tham đối Object chỉ định mảng nguồn và mảng đích. Ba tham đối int chỉ vị trí bắt đầu trong mỗi mảng nguồn và đích, và số thành phần để sao chép. Biểu đồ này minh hoạ việc sao chép :



Ví dụ :

```
public class ArrayCopyDemo {
    public static void main(String[] args) {
        char[] copyFrom = { 'd', 'e', 'c', 'a', 'f', 'f', 'e',
                'i', 'n', 'a', 't', 'e', 'd' };
        char[] copyTo = new char[7];
        System.arraycopy(copyFrom, 2, copyTo, 0, 7);
        System.out.println(new String(copyTo));
    }
}
```

Biểu đồ sau mô tả cho ví dụ trên :



Chú ý rằng mảng đích phải được cấp phát và phải đủ lớn để chứa dữ liệu được sao chép

Chương : GÓI VÀ GIAO DIỆN

Gói (Package) và giao diện (Interface) là hai thành phần cơ bản trong một chương trình Java.

I. GÓI

Gói là kỹ thuật phân hoạch không gian tên lớp, giao diện thành những vùng dễ quản lý hơn. Ví dụ khi bạn tạo một lớp trong một gói nào đó, bạn không cần phải kiểm tra xem nó có bị trùng tên với một lớp nào đó trong gói khác không.

Gói bao gồm hai kỹ thuật đặt tên và kỹ thuật điều khiển truy xuất. Bạn có thể cấp hay không cấp quyền truy xuất các lớp bên trong gói đối với các đoạn mã nằm ngoài gói. Bạn cũng có thể xác định thành phần nào của lớp mà chỉ có các thành phần trong cùng một lớp mới có quyền truy xuất.

1. Định nghĩa gói

Tạo một gói bằng cách đặt từ khoá package ngay phát biểu đầu tiên của tập tin nguồn Java. Bất kỳ lớp nào khai báo trong tập tin này đều thuộc gói này. Nếu bạn bỏ qua phát biểu Package, các lớp sẽ đặt vào package mặc định

package *PackageName;*

Java sử dụng hệ thống thư mục để lưu trữ các gói. Các lớp sẽ chứa trong thư mục trùng tên PackageName

Có thể tạo các package phân cấp, dùng dấu chấm để phân biệt một package với package cha của nó. Sự phân cấp package phải được ánh xạ vào hệ thống tập tin. Java xem gốc của cây phân cấp gói được định nghĩa bởi biến môi trường CLASSPATH.

package *PackageName1*[.*PackageName2*[.*PackageName3*]];

Ví dụ : package java.awt.image;
được lưu trữ trong Java\awt\image với hệ điều hành Windows.

2. Điều khiển truy xuất

Thông qua phép đóng gói (lớp, gói), ta có thể điều khiển phần nào của chương trình có thể truy xuất các thành phần của lớp

Các điều khiển truy xuất của Java là public, private và protected. protected chỉ áp dụng khi có liên quan đến kế thừa

◆ Khi bổ sung tiền tố cho một thành phần của lớp (biến và hàm) là :

- Từ khoá public : chỉ ra rằng thành phần này có thể được truy xuất bởi bất kỳ dòng lệnh nào dù ở trong hay ngoài lớp, gói (Package) mà nó khai báo
- private : chỉ có thể được truy xuất trong lớp của nó, mọi đoạn mã nằm ngoài lớp, kể cả những lớp con đều không có quyền truy xuất
- Khi không có điều khiển truy xuất nào được dùng, các lớp con cũng như các lớp trong cùng gói đều có thể truy xuất nó, không thể truy xuất từ bên ngoài gói của nó
- protected : liên quan đến sự kế thừa, nếu bạn chỉ cho các lớp con trực tiếp mới có quyền truy xuất các thành phần của lớp, bạn khai báo chúng là protected

◆ Với lớp chỉ có hai mức truy xuất : mặc định và public. Khi một lớp khai báo public, các đoạn mã khác có thể truy xuất được nó. Nếu lớp là truy xuất mặc đinh, chỉ có các đoạn mã trong cùng một gói mới có quyền truy xuất nó

Điều khiển truy xuất thành phần của lớp	Trong lớp	Trong lớp con	Trong Package (Gói)	Toàn bộ
private	X			
protected	X	X*	X	

public	X	X	X	X
Không có	X	X	X	

Ví dụ :

```
package Greek;
public class Alpha {
        protected int i;
        protected void protectedMethod() {
                System.out.println("Protected Metho");
        }
}
class Gamma {
        void accessMethod() {
                Alpha a = new Alpha();
                a.i = 10;                       // Hợp lệ
                a.protectedMethod();            // Hợp lệ
        }
}
package Latin;
import Greek.*;
class Delta extends Alpha {
        void accessMethod (Alpha a, Delta d) {
                a.i = 10;                       // Không hợp lệ
                d.i = 10;                       // Hợp lệ
                a.protectedMethod();            // Không hợp lệ
                d.protectedMethod();            // Hợp lệ
        }
}
```

3. Sử dụng gói

Java đưa ra phát biểu import để những lớp nào đó hay toàn bộ gói có thể thấy được, nghĩa là bạn có thể sử dụng lớp trực tiếp qua tên của nó, không cần dùng dấu chấm truy xuất

Trong tập tin nguồn Java, phát biểu import đặt ngay sau phát biểu package (nếu tồn tại) và trước bất cứ định nghĩa lớp nào

import *PackageName1*[.*PackageName2*].*ClassName*;

import *PackageName1*[.*PackageName2*].*;

Ví dụ :

```
import java.util.Date;
import java.io.*;
```

Tất cả các lớp chuẩn của Java lưu trong gói tên là java. Bạn phải nhập từng gói hay lớp bạn muốn sử dụng, riêng lớp gói java.lang lưu nhiều chức năng thông dụng, được import ngầm định bởi bộ biên dịch cho tất cả các chương trình.

Phát biểu import của gói chỉ có giá trị trên các thành phần khai báo public của nó.

II. GIAO DIỆN

1. Định nghĩa giao diện

Với từ khoá interface, bạn có thể trừu tượng hoàn toàn giao diện của lớp khỏi sự hiện thực của nó. Nghĩa là bạn có thể đặc tả một lớp phải làm gì, nhưng không cần biết làm như thế nào. Giao diện là tập hợp các khai báo phương thức, hằng mà lớp con kế thừa. Giao diện có cú pháp tương tự lớp, nhưng nó không có biến thành viên, chỉ có khai báo hằng và những phương

thức của chúng khai báo không có thân. Trong thực tế, điều này có nghĩa rằng bạn có thể định nghĩa những giao diện mà không cần đảm nhiệm phần hiện thực nó. Số lượng lớp hiện thực một giao diện là tuỳ ý. Một lớp cũng có thể hiện thực số lượng tuỳ ý giao diện.

Để hiện thực một giao diện, một lớp phải cài đặt đầy đủ các phương thức định nghĩa trong giao diện. Với từ khoá interface, Java cho phép bạn có những tiện ích đầy đủ cho đặc điểm "một giao diện, nhiều phương thức" của tính đa hình

Giao diện được thiết kế để hỗ trợ sự quyết định phương thức động lúc thời gian chạy. Thông thường, để lớp này có thể gọi phương thức của lớp kia, cả hai lớp cần hiện diện lúc thời gian dịch. Điều này làm cho môi trường lớp trở nên tĩnh và không có khả năng mở rộng. Trong một hệ thống như vậy cây phân cấp càng ngày càng bị đẩy lên cao. Vì vậy, giao diện được định nghĩa để hạn chế việc ngày càng nhiều lớp con. Nó tách sự định nghĩa một phương thức hay tập các phương thức ra khỏi cây phân cấp kế thừa. Vì các giao diện phân cấp khác các lớp, do đó các lớp không có quan hệ trong sự phân cấp cũng có thể hiện thực cùng một giao diện. Ta có thể thấy đây thực sự là thế mạnh giao diện

***Acess* interface *InterfaceName* {**
***Type MethodName1*(*Parameter-List*);**
...
***Type MethodNamen*(*Parameter-List*);**
Type Final-Var1 = Value;
…
Type Final-Varn = Value;
}

- Acess có thể là public hay không. Khi không chứa đặc tả nào, access là mặc định và giao diện chỉ có giá trị đối với các thành phần khác khai báo trong gói. Khi khai báo public, mọi đoạn mã đều có thể sử dụng giao diện. Tất cả các phương thức và biến hiểu ngầm là public nếu giao diện khai báo là public
- Các phương thức là các phương thức trừu tượng, không có thân, chúng không được hiện thực trong giao diện.
- Các biến có thể khai báo trong khai báo giao diện. Chúng hiểu ngầm là final và static, nghĩa là chúng không thể bị thay đổi bởi sự hiện thực của lớp. Chúng phải được khởi tạo với những giá trị hằng.

2. Hiện thực giao diện

Khi đã định nghĩa giao diện, một hay nhiều lớp có thể hiện thực giao diện. Để hiện thực giao diện, đặt mệnh đề implements trong định nghĩa lớp và sau đó tạo những phương thức định nghĩa trong giao diện :

***Acess* class *ClassName* [extends *SuperClass*]**
[implements *InterfaceName1*,… *InterfaceNamen*] {
// Body of class
}

- Nếu lớp hiện thực hai giao diện có phương thức giống nhau, phương thức được gọi tương ứng với giao diện đó
- Những phương thức hiện thực giao diện phải khai báo public. Hình thức của phương thức hiện thực phải giống hệt khi nó được đặc tả trong định nghĩa interface
- Hiện thực từng phần : Nếu một lớp chứa một giao diện nhưng không hiện thực đầy đủ các phương thức định nghĩa trong giao diện, lớp đó phải khai báo abstract
- Một giao diện giống như một lớp trừu tượng, tuy nhiên Class của bạn không thể kế thừa nhiều lớp, nên dùng giao diện thay cho lớp trừu tượng, một lớp có thể hiện thực nhiều giao diện. Vì vậy mà giao diện cung cấp nhiều sự kế thừa.

Ví dụ 1 :

```
interface KiemTra {
        void inSo(int p);
}
class HienThuc implements KiemTra {
        public void inSo(int p) {
                System.out.println("Giá trị của p là : "+p);
        }
        void boSung() {
                System.out.println("Class hiện thực giao diện có thể định nghĩa thêm "+
                        "thành viên khác hay không");
        }
}
```

Ví dụ 2 :

```
class HTKhac implements KiemTra {
        public void inSo(int p) {
                System.out.println("Bình phương của p là : "+p*p);
        }
}
```

3. Truy xuất hiện thực thông qua tham chiếu giao diện

Bạn có thể khai báo một biến tham chiếu đến đối tượng kiểu giao diện chứ không hẳn là lớp. Khi bạn gọi phương thức thông qua một trong những tham chiếu đến đối tượng kiểu giao diện hay lớp hiện thực giao diện, phiên bản đúng sẽ được gọi dựa trên thể hiện thực sự của giao diện đang tham chiếu đến. Phương thức thực thi được tìm tự động lúc chạy.

Ví dụ :

```
class UngDung {
        public static void main (String args[] {
                KiemTra c = new HienThuc(); //c chỉ biết hàm khai báo trong giao diện
                HienThuc d  = new HienThuc(); //d biết các hàm khai báo trong HienThuc
                HTKhac e = new HTKhac(); //e biết các hàm khai báo trong HTKhac
                c. inSo(50);
                c = d; // c bây giờ tham chiếu đến đối tượng kiểu HienThuc
                c.bo Sung();
                c = e;   // c bây giờ tham chiếu đến đối tượng kiểu HTKhac
                c. inSo(50);
        }
}
```

Kết quả chương trình là :

```
Giá trị của p là : 50
Class hiện thực giao diện có thể định nghĩa thêm thành viên khác hay không
Bình phương của p là : 2500
```

4. Biến trong giao diện

Bạn có thể dùng giao diện để import những hằng dùng chung cho nhiều lớp đơn giản bằng cách khai báo giao diện chứa những biến được khởi tạo bằng những giá trị yêu cầu. Khi bạn đưa giao diện đó vào trong lớp, tất cả những tên biến này có phạm vi như một hằng. Điều này giống như sử dụng tập tin header trong C/C^{++} tạo số lượng lớn hằng bằng #defined hay khai báo const. Nếu giao diện không chứa phương thức nào, lớp chứa giao diện như vậy thực sự không hiện thực điều gì cả. Nó tương tự việc lớp đó import những biến hằng cho không gian lớp như những biến final

5. Kế thừa giao diện

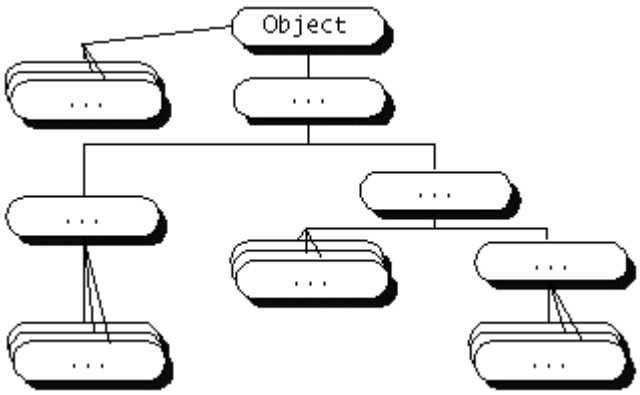
Một giao diện có thể kế thừa giao diện khác bằng cách sử dụng từ khoá extends. Cú pháp giống như lớp kế thừa. Khi một lớp hiện thực một giao diện kế thừa từ một giao diện khác, nó phải cung cấp tất cả các hiện thực cho tất cả các phương thức kể cả phương thức trong danh sách giao diện cha mà giao diện này kế thừa

Chương : LỚP STRING VÀ NUMBER

I. TỔNG QUÁT

Chúng ta đã biết cách dùng từ khoá extends khai báo một lớp là lớp con của một lớp khác. Tuy nhiên bạn chỉ có thể định nghĩa một lớp cha cho lớp con của bạn (Java không ủng hộ nhiều sự kế thừa lớp), và thậm chí bỏ qua từ khoá extends trong một khai báo lớp, lớp của bạn cũng có một lớp cha. Điều này dẫn đến một câu hỏi là các lớp bắt đầu từ đâu ?

Như mô tả trong hình sau, lớp cao nhất, lớp mà từ đó tất cả các lớp xuất phát từ, là lớp Object định nghĩa trong java.lang. Lớp Object định nghĩa và cài đặt các hành vi mà mọi lớp trong Java cần đến.



II. LỚP STRING VÀ STRINGBUFFER

Trong gói java.lang chứa hai lớp lưu trữ và thao tác dữ liệu kiểu ký tự : String và StringBuffer, được khai báo final, nghĩa là không kế thừa

Bạn dùng lớp String khi bạn đang làm việc với chuỗi hằng, nội dung không thể thay đổi. StringBuffer được dùng khi bạn muốn thay đổi nội dung của chuỗi.

Ví dụ : Phương thức reverse dùng cả hai lớp String và StringBuffer để đảo các ký tự của chuỗi.

```
public class ReverseString {
        public static String reverse(String source) {
                int i, len = source.length();
                StringBuffer dest = new StringBuffer(len);
                for (i = (len - 1); i >= 0; i--)
                        dest.append(source.charAt(i));
                return dest.toString();
        }
}
```

1. Lớp String

Tạo một đối tượng

Nhiều String được tạo từ các hằng chuỗi. Khi trình dịch bắt gặp một chuỗi ký tự bao giữa cặp nháy kép, nó tạo ra một đối tượng chuỗi mà có giá trị là chuỗi bao giữa cặp nháy kép. Bạn có thể dùng hằng String ở bất kỳ đâu bạn dùng đối tượng String

Bạn có thể tạo đối tượng chuỗi như bất kỳ đối tượng nào khác của java, dùng từ khoá new

```
String s = new String();
String s = new String("Gobbledygook.");
```

hay có thể viết :

```
String s = "Hola Mundo";
```

- Một số các hàm khởi tạo của lớp String :

String()
String(byte *bytes*[])
String(byte *bytes*[],int *startIndex*, int *numChars*)
String(char *chars*[])
String(char *chars*[], int *startIndex*, int *numChars*)
String(String *s*)
String(StringBuffer *s*)

Ví dụ :

```
byte b[] = {65, 66, 67, 68, 69, 70};
String s1 = new String(b);          // Khởi tạo s với chuỗi ABCDEF
String s2 = new String(b,2,3);      // Khởi tạo s với chuỗi CDE
char c[] = {'a','b','c','d','e','f');
String s3 = new String(c);          // Khởi tạo s với chuỗi abcdef
String s4 = new String(c,2,3);      // Khởi tạo s với các ký tự cde
String s5 = new String(s2);         // Tạo đối tượng s3 chứa cùng dãy ký tự như s2
```

Các phương thức thường dùng của lớp String

- **int length()** : cho chiều dài chuỗi

```
int len = source.length();
int len = "Goodbye Cruel World".length();
```

- **char charAt(int *index*)** : trả về ký tự tại vị trí thứ index

```
char c = source.charAt(1);
char ch = "abc".charAt(0);     //Gán giá trị a cho ch
```

- **boolean equals(String *object*)** : kiểm tra hai chuỗi có bằng nhau không, có phân biệt hoa thường

So sánh phương thức equals() và toán tử = = khác nhau hoàn toàn. Phương thức dùng so sánh các ký tự trong đối tượng String. Toán tử = = so sánh 2 đối tượng có cùng tham chiếu đến cùng một thể hiện.

```
String s1 = "Hello";
String s2 = new String(s1);     // tạo s2 có nội dung như s1, nhưng không trỏ đến cùng
                                // một đối tượng
System.out.println("s1 equals s2 :"+s1.equals(s2));
System.out.println("s1 = = s2 :"+(s1= =s2));
Kết quả là :
s1 equals s2 : true
s1 = = s2 : false
```

- **int compareTo(String str)** : so sánh 2 chuỗi, trả về giá trị :

nếu < 0 : chuỗi nhỏ hơn str
nếu > 0 : chuỗi lớn hơn str
nếu = 0 : chuỗi bằng str

- **int indexOf(int *character*)** : trả về vị trí tìm thấy đầu tiên (cuối cùng) của ký tự *character*
int lastIndexOf(int *character*)
- **int indexOf(int *character*, int *from*)** : trả về vị trí tìm thấy đầu tiên (cuối cùng) của ký tự *character,* kể từ vị trí *from* về cuối chuỗi (hay đầu chuỗi)
int lastIndexOf(int *character*, int *from*)
- **int indexOf(String *string*)** : trả về vị trí tìm thấy đầu tiên (cuối cùng) của chuỗi *string*
int lastIndexOf(String *string*)

- **int indexOf(String *string*, int *from*)** : trả về vị trí tìm thấy đầu tiên (cuối cùng) của chuỗi *string*, kể từ vị trí *from* về cuối chuỗi (hay đầu chuỗi)

int lastIndexOf(String *string*, int *from*)

- **String subString(int *startIndex*, int *endIndex*)** : trả về chuỗi con của một chuỗi bắt đầu từ vị trí startIndex đến vị trí endIndex-1, nếu không có endIndex thì lấy đến cuối chuỗi

```
String org = “This is a test”;
String  result = “”;
result = org.subString(8);
```

- **String replace(char *orginal*, char *replacement*)** : thay thế ký tự replacement cho ký tự orginal

```
String s = “Hello”.replace(‘l’,’w’);    // Cho s bằng “Hewwo”
```

- **String trim()** : cắt bỏ khoảng trống trước và sau chuỗi
- **String toLowerCase()** : đổi chuỗi thành chuỗi thường
- **String toUpperCase()** : đổi chuỗi thành chuỗi hoa

```
String s = “This is a test”;
String upper = s.toUpperCase();
```

- **Toán tử +** : để kết nối hai đối tượng String, hay một đối tượng String và một giá trị khác thành đối tượng String,

```
String s1 = "two";
System.out.println("one" + s1 + "three");
System.out.println("Word v. " + 9+7);
```

Vì đối tượng String không thể thay đổi do đó bất cứ lúc nào bạn muốn thay đổi chúng, bạn phải copy chuỗi vào StringBuffer, nhưng với toán tử +, bạn có thể viết như sau vì Java tự chuyển sang StringBuffer và thay đổi chuỗi

```
s1 = s1 + “three”;
```

Vì vậy, có thể viết lại chương trình đảo chuỗi trên, không cần thiết phải chuyển sang StringBuffer

```
public class ReverseString {
    public static String reverse(String source) {
        int i, len = source.length();
        String dest = “”;
        for (i = (len - 1); i >= 0; i--)
            dest = dest + source.charAt(i);
        return dest;
    }
}
```

- **static String valueOf(object/var *x*)** :

là hàm lớp, trả về một chuỗi để chuyển đổi các biến kiểu sơ cấp hay đối tượng x thành một String

Ví dụ : PI là một biến lớp của lớp Math, để in giá trị của sô PI :

```
System.out.println(String.valueOf(Math.PI));
```

2. Lớp StringBuffer

Tạo một đối tượng StringBuffer

Phương thức khởi tạo của lớp StringBuffer có dạng :

StringBuffer() : dùng cho chuỗi 16 ký tự

StringBuffer(int *length*) : dùng cho chuỗi length ký tự

Ví dụ : StringBuffer dest = new StringBuffer(25);

Các phương thức thường dùng của lớp StringBuffer

- **int length()** : cho chiều dài chuỗi
- **char charAt(int *index*)** : trả về ký tự tại vị trí thứ index
- **void setCharAt(int *index*, char *ch*)** : đặt ký tự ch vào StringBuffer, tại vị trí index
- **StringBuffer append(object/var *x*)** : bổ sung đối tượng hay biến x kiểu bất kỳ vào cuối StringBuffer. Dữ liệu được chuyển thành chuỗi trước khi bổ sung vào StringBuffer

```
int a = 20;
StringBuffer sb = new StringBuffer(40);
String s = sb.append("a =").append(a).toString();
```

- **StringBuffer insert(int *index*, object/var *x*)** : chèn một đối tượng hay biến x kiểu bất kỳ vào vị trí thứ index

```
StringBuffer sb = new StringBuffer("I Java!");
sb.insert(3, "like ");
System.out.println(sb);          // Cho chuỗi "I like Java"
```

- **StringBuffer reverse()** : đảo ngược các ký tự của chuỗi

```
StringBuffer sb = new StringBuffer("I Java!");
sb.reverse();
```

- **StringBuffer delete(int *startIndex*, int *endIndex*)** : xoá chuỗi con từ startIndex đến endIndex-1
- **StringBuffer deleteCharAt(int *index*)** : xoá ký tự tại vị trí index
- **StringBuffer subString(int *startIndex*, int *endIndex*)** : trả về chuỗi con của một chuỗi bắt đầu từ vị trí startIndex đến vị trí endIndex-1, nếu không có endIndex thì lấy đến cuối chuỗi
- **StringBuffer replace(int *startIndex*, int *endIndex*, String *str*)** : thay thế chuỗi str vào vị trí bắt đầu là startIndex đến endIndex-1 của chuỗi

Bạn hãy xem thêm java.lang.String and java.lang.StringBuffer để có được định nghĩa đầy đủ các phương thức và biến cài đặt cho hai lớp này

III. LỚP NUMBERS

Như đã nói ở trên, trong gói java.lang có sẵn những lớp tương ứng với các kiểu sơ cấp, có thể dùng thay cho kiểu sơ cấp : lớp Integer thay cho kiểu int, lớp Boolean cho kiểu boolean… Lớp Number là lớp cha của mọi lớp bọc kiểu

Các lớp bọc kiểu số: Byte, Double, Float, Integer, Long, Short

Các lớp bao bọc cho các kiểu dữ liệu khác : Boolean, Character, Void, Math

1. Tạo một đối tượng

```
Float f = new Float(25.5);
Float f = new Float("24.5");
```

- Các hàm khởi tạo của các lớp bọc kiểu số

Float(double *n*)
Float(float *n*)
Float(string *str*)
Double(double *n*)
Double(string *str*)

Tương tự với các lớp bọc kiểu số khác

2. Các phương thức thường dùng cho các lớp kiểu số

- Các phương thức trả về giá trị của các đối tượng tương ứng với các dạng số khác nhau

byte byteValue() // trả về dạng số byte
short shortValue()

int intValue()
long longValue()
double doubleValue()
float floatValue()

```
Integer intObj = new Integer(25);
int i = intObj.intValue(); // cho i = 25
```

- Các phương thức lớp chuyển đổi một chuỗi thành giá trị số tương ứng

static byte parseFloat(String *str*)
static short parseShort(String *str*)
static int parseInt(String *str*)
static long parseLong(String *str*)
static double parseDouble(String *str*)
static float parseFloat(String *str*)

```
String s = "42";
int i =Integer.parseInt(s);       // cho i = 42
```

- **int compareTo(floatObj/floatVar *f*)** : so sánh giá trị của đối tượng số với đối tượng hay biến số f, trả về giá trị :

nếu = 0 : bằng nhau
nếu = số âm : giá trị của đối tượng nhỏ hơn f
nếu = số dương : giá trị của đối tượng lớn hơn f

- **boolean equals(floatObj *f*)** : nếu = true nghĩa là giá trị của đối tượng bằng f
- **string toString()** : chuyển một đối tượng thành String.

Tất cả các lớp kế thừa toString từ lớp Object và nhiều lớp khác trong gói java.lang viết đè phương thức này để cung cấp một cài đặt mà có ý nghĩa với class đó. Chẳng hạn, các lớp bọc kiểu Character, Integer, Boolean… đều viết đè toString

Ví dụ :

```
Integer i = new Integer(20);
System.out.println(i.toString());
```

- **static String toString(var *n*)** : đây là hàm lớp để chuyển biến số n thành chuỗi

```
String s = Integer.toString(25)          // Cho chuỗi s là "25"
```

- **static Float valueOf(String *str*)** : đây là hàm lớp trả về đối tượng Float của giá trị str, tương tự với các lớp bọc kiểu số khác

```
String s = "42.5";
Integer i = Integer.valueOf(s);
```

Ví dụ : Viết đoạn chương trình nhập : họ tên không quá 20 ký tự, năm sinh >1970 và <2100, có kiểm tra cho đến khi người dùng nhập đúng

```
import java.io.*;
public class Nhap {
    public static void main (String [] args) throws IOException {
        DataInputStream kbd = new DataInputStream (System.in);
        String s = null;
        String ns = null;
        // Nhap ho ten
        while  (true) {
            System.out.print("Nhap Ho va ten : ");
            s = kbd.readLine();
            if ( (s.length() <=20) && (s.length() != 0) ) {
                System.out.println("Ho va ten la : "+ s);
                break;
            }
```

```
                        System.out.println("Phai nhap ho ten khong qua 20 ky tu");
                }
                // Nhap nam sinh
                while (true) {
                        try {
                                System.out.print("Nhap nam sinh : ");
                                s = kbd.readLine();
                                int i =Integer.parseInt(s);
                                if ((i<=1970) || (i>=2100)) throw new
                                        NumberFormatException();
                                System.out.println("Nam sinh la : "+i);
                                break;
                        }
                        catch (IOException e){}
                        catch(NumberFormatException e){
                                System.out.println("Ban Phai nhap lai nam sinh trong
khoang
                                        1970 den 2100");
                        }
                }
        }
}
```

Bạn hãy xem thêm java.lang.Byte, java.lang.Short, java.lang.Integer, java.lang.Double, java.lang.Float để có được định nghĩa đầy đủ các phương thức và biến cài đặt cho các lớp này

MỤC LỤC