



# Lập trình hướng đối tượng - Java

---

Chương IV

## **Interfaces & Packages**

Biên soạn: Lý Quỳnh Trân



# Nội dung..

---

- Định nghĩa Package
- Phạm vi truy cập
- Sử dụng package
- Các package trong thư viện
- Khái niệm Interface
- Thiết kế & thực thi interface
- Khái niệm & ý nghĩa Package

# Packages

- Package được sử dụng để chia các class và interface thành các gói khác nhau.
- Package là tên thư mục chứa các class & interface
- Trong package có thể chứa package khác
  - Khai báo **import** cho phép tham chiếu đến tên lớp từ các package khác.
- Trong NetBeans, ta chọn new Project tên **MyPackage** thì thư mục mypackage tự động được tạo
  - Tất cả các lớp sau đó sẽ nằm trong cấu trúc thư mục mang tên mypackage
  - Tên **myPackage** nên bắt đầu bằng **chữ thường** để phân biệt với tên lớp

## • Tạo package

```
package myPackage;  
class A  
{ .....  
}
```

## • Sử dụng package

```
import myPackage.A;  
class B  
{ .....  
    void method3()  
    {   A obj = new A();  
        ..... *  
    }  
}
```

# Truy cập thành phần trong Packages

- Truy nhập các thành phần bên trong package
  - Các **class** dự định sử dụng bên **ngoài package** thì khai báo là **public**
  - Các package khác nhau mà có các class trùng tên với nhau thì khi sử dụng bắt buộc phải import đầy đủ tên package và tên class
- Các class trong package được khai báo dựa trên 4 kiểu

**private:** Chỉ có thể được truy cập bởi chính class đó.

**default:** Được truy cập bởi các class cùng package.

**protected:** Được truy cập bởi các class cùng trong package và các class là sub-class của class này.

**public:** Được truy cập bởi tất cả các class ở cùng package hay khác package.

# Ví dụ về Packages

```
12 package accessmodifierpackage;
13 //private access modifier
14 class A{
15     private int data=40;
16     private void msg()
17     {
18         System.out.println("Hello java");
19     }
20 }
21
22 public class AccessModifierPackage {
23
24     public static void main(String[] args) {
25         // TODO code application logic here
26         A obj=new A();
27         System.out.println(obj.data);//Compile Time Error
28         obj.msg();//Compile Time Error
29     }
30
31 }
```

run:

```
Exception in thread "main" java.lang.RuntimeException: Uncompilable source code - data has private access in accessmodifierpackage.A
    at accessmodifierpackage.AccessModifierPackage.main(AccessModifierPackage.java:27)
C:\Users\Quynh Tran Ly\AppData\Local\Temp\beans(Cache\0.2)\executor-snippets\run.xml:53: Java returned: 1
BUILD FAILED (total time: 1 second)
```

```
12 package DefaultDemo;
13 class A {
14     void msg()
15     {
16         System.out.println("Hello");
17     }
18 }
19
20 package accessmodifierpackage;
21 import DefaultDemo.*;
22 class B {
23
24     public static void main(String[] args) {
25         // TODO code application logic here
26         A obj = new A();//Compile Time Error
27         obj.msg();//Compile Time Error
28     }
29
30 }
```

run:

```
Exception in thread "main" java.lang.RuntimeException: Uncompilable source code - msg() is not public in DefaultDemo.A; cannot be accessed from outside package
    at accessmodifierpackage.B.main(B.java:20)
C:\Users\Quynh Tran Ly\AppData\Local\Temp\beans(Cache\0.2)\executor-snippets\run.xml:53: Java returned: 1
BUILD FAILED (total time: 1 second)
```

# Ví dụ về Packages

```
12 package DefaultDemo;
13
14 public class A1 {
15     protected void msg()
16     {
17         System.out.println("Hello");
18     }
19 }
20
21 package accessmodifierpackage;
22 import DefaultDemo.*;
23 public class B1 extends A1 {
24
25     public static void main(String[] args) {
26         // TODO code application logic here
27         B1 obj = new B1();
28         obj.msg();
29     }
30
31 }
```

```
run:
Hello
BUILD SUCCESSFUL (total time: 1 second)
```

```
12 package DefaultDemo;
13 public class A2 {
14     public void msg()
15     {
16         System.out.println("Hello");
17     }
18 }
19
20 package accessmodifierpackage;
21 import DefaultDemo.*;
22 public class B2 {
23
24     /**
25      * @param args the command line arguments
26      */
27     public static void main(String[] args) {
28         // TODO code application logic here
29         A2 obj = new A2();
30         obj.msg();
31     }
32
33 }
```

```
run:
Hello
BUILD SUCCESSFUL (total time: 1 second)
```



# Các Packages thư viện

Tên Package	Mô tả
java.lang	Chứa các class như Integer, String, System... và được tự động import vào mỗi chương trình Java.
java.util	Các các Java collections như List, Set, Map ...
java.io	Chứa các class liên quan đến việc nhập, xuất dữ liệu như File, Reader, Writer...
java.awt và java.swing	Chứa các class liên quan đến việc trình bày giao diện đồ họa và xử lý sự kiện.
...	

# Khai báo biến static

- Từ khóa static được sử dụng để định nghĩa cho khối và các thành viên tĩnh:
  - lớp nội, phương thức tĩnh, biến tĩnh
- Biến tĩnh trong một lớp là biến dùng chung cho tất cả các đối tượng thuộc lớp đó.
- Biến tĩnh static gắn với lớp đối tượng
- Khi khai báo biến tĩnh static thì phiên bản copy được tạo và chia sẻ giữa tất cả các đối tượng thuộc lớp đó hàm.
- Biến tĩnh có thể coi là biến toàn cục
- Khai báo **biến tĩnh khi thuộc tính đó là thuộc tính dùng chung** cho tất cả các đối tượng.
  - Ví dụ: **Students có chung college.**
- Dùng với biến, hàm, khối chỉ thành phần dùng chung
  - Hàm static không được sử dụng ***this***
  - Hàm static không cho phép định nghĩa lại (overridden) bởi hàm non-static'

```
public class MyClass{  
    static public int X;  
    static{  
        X+=100;  
    }  
    static public void method(){  
        X+=200;  
    }  
    static class MyInnerClass{  
    }  
}
```

MyClass.X = 700;  
MyClass.method()



# Ví dụ về biến static

```
16 class Student
17 {
18     String name;
19     int rollNo;
20     // static variable
21     static String cllgName;
22     // static counter to set unique roll no
23     static int counter = 0;
24
25     public Student(String name)
26     {
27         this.name = name;
28         this.rollNo = setRollNo();
29     }
30
31     // getting unique rollNo
32     // through static variable(counter)
33     static int setRollNo()
34     {
35         counter++;
36         return counter;
37     }
38
39     // static method
40     static void setCllg(String name){
41         cllgName = name ;
42     }
43
44     // instance method
45     void getStudentInfo(){
46         System.out.println("name : " + this.name);
47         System.out.println("rollNo : " + this.rollNo);
48         // accessing static variable
49         System.out.println("cllgName : " + cllgName);
50     }
51 }
52 }
```

```
54 //Driver class
55 public class StaticDemo
56 {
57     public static void main(String[] args)
58     {
59         // calling static method
60         // without instantiating Student class
61         Student.setCllg("XYZ");
62
63         Student s1 = new Student("Alice");
64         Student s2 = new Student("Bob");
65
66         s1.getStudentInfo();
67         s2.getStudentInfo();
68     }
69 }
--
```

```
run:
name : Alice
rollNo : 1
cllgName : XYZ
name : Bob
rollNo : 2
cllgName : XYZ
BUILD SUCCESSFUL (total time: 0 seconds)
```



# Phương thức static

---

- Phương thức static tồn tại độc lập với các đối tượng
  - được gọi thẳng từ lớp mà **không cần** đến một tham chiếu đối tượng nào.
  - Ta dùng từ khóa static khi khai báo phương thức lớp.
  - Phương thức static có thể truy cập thành viên dữ liệu static và có thể thay đổi giá trị của nó.

# Lớp nội static

- Lớp nội là lớp được khai báo bên trong một lớp khác
- Từ khóa **static** được sử dụng để định nghĩa lớp nội
- Có hai loại: lớp nội tĩnh và lớp nội thông thường
- Lớp bên trong chỉ có thể xác định trong phạm vi lớp ngoài cùng và có thể truy cập các thành viên của lớp bao nó.

```
public class MyClass{  
    static public class MyInnerStaticClass{}  
    public class MyInnerClass{}  
}
```

Sử dụng lớp nội

```
MyClass.MyInnerStaticClass x = new MyClass.MyInnerStaticClass();  
MyClass.MyInnerClass y = new MyClass().new MyInnerClass();
```

# Ví dụ về lớp nội static

```
12 class OuterClass{
13     private static String msg = "Hello Students studying Java!";
14
15     // Static nested class
16     public static class NestedStaticClass{
17
18         // Only static members of Outer class is directly accessible in nested
19         // static class
20         public void printMessage() {
21
22             // Try making 'message' a non-static variable, there will be
23             // compiler error
24             System.out.println("Message from nested static class: " + msg);
25         }
26     }
27
28     // non-static nested class - also called Inner class
29     public class InnerClass{
30
31         // Both static and non-static members of Outer class are accessible in
32         // this Inner class
33         public void display(){
34             System.out.println("Message from non-static nested class: " + msg);
35         }
36     }
37 }
```

```
38 class StaticClassDemo
39 {
40     // How to create instance of static and non static nested class?
41     public static void main(String args[]){
42
43         // create instance of nested Static class
44         OuterClass.NestedStaticClass printer = new OuterClass.NestedStaticClass();
45
46         // call non static method of nested static class
47         printer.printMessage();
48
49         // In order to create instance of Inner class we need an Outer class
50         // instance. Let us create Outer class instance for creating
51         // non-static nested class
52         OuterClass outer = new OuterClass();
53         OuterClass.InnerClass inner = outer.new InnerClass();
54
55         // calling non-static method of Inner class
56         inner.display();
57
58         // we can also combine above steps in one step to create instance of
59         // Inner class
60         OuterClass.InnerClass innerObject = new OuterClass().new InnerClass();
61
62         // similarly we can now call Inner class method
63         innerObject.display();
64     }
65 }
```

run:

```
Message from nested static class: Hello Students studying Java!
Message from non-static nested class: Hello Students studying Java!
Message from non-static nested class: Hello Students studying Java!
BUILD SUCCESSFUL (total time: 0 seconds)
```



# Khai báo hằng final

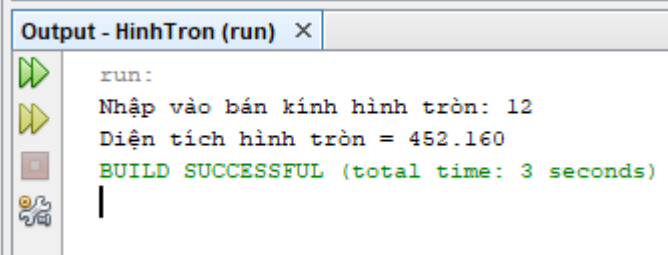
- Trong java có 3 loại hằng
  - **Lớp hằng** là lớp không cho phép thừa kế
  - **Phương thức hằng** là phương thức không cho phép ghi đè
  - **Biến hằng** là biến không cho phép thay đổi giá trị
- Sử dụng **từ khóa final** để định nghĩa hằng

```
final public class MyFinalClass{...}
```

```
public class MyClass{  
    final public double PI = 3.14  
    final public void method(){...}  
}
```

# Ví dụ về khai báo hằng final

```
7 import java.text.DecimalFormat;
8 import java.util.Scanner;
9 public class HinhTron {
10
11     public static void main(String[] args) {
12         // TODO code application logic here
13         int banKinh;
14         final double soPI = 3.14;
15         double area;
16         Scanner scanner = new Scanner(System.in);
17         System.out.print("Nhập vào bán kính hình tròn: ");
18         banKinh = scanner.nextInt();
19         area = soPI * banKinh * banKinh;
20         System.out.printf("Diện tích hình tròn = %.3f ",area);
21         System.out.println();
22     }
23 }
```



```
Output - HinhTron (run) ×
run:
Nhập vào bán kính hình tròn: 12
Diện tích hình tròn = 452.160
BUILD SUCCESSFUL (total time: 3 seconds)
```

# Giao diện Interface

- **interface** là một khai báo nhằm tạo khuôn mẫu cho một kiểu tham chiếu mới.
- Trong **interface** chỉ có các biến, hằng dữ liệu và **phương thức trừu tượng** (abstract method).
- Khi một class thực thi một interface nó phải viết lại **override** tất cả các method trong interface.
- Khai báo interface bằng từ khóa public hoặc default
- Interface có thể có kế thừa
- Một interface có thể được thực hiện bởi nhiều class và một class có thể thực thi nhiều interface

```
19 access_modifier interface interface_name {
20     // Variables
21     data_type var_name_1 = value_1;
22     data_type var_name_2 = value_2;
23     ...
24     data_type var_name_N = value_N;
25     // Method declarations
26     return_type method_name_1(parameter_list);
27     return_type method_name_2(parameter_list);
28     ...
29     return_type method_
30 }
```

# Thực thi interface

## Cú pháp khai báo

- sử dụng từ khóa **implements**
- `public class HìnhChuNhat implements Shape{... }`
- Lớp triển khai phải định nghĩa tất cả các phương thức đã khai báo trong các interface.
- Biến khai báo kiểu interface có thể nhận giá trị thể hiện của lớp thực thi tương ứng
- Thể hiện đặc điểm đa kế thừa trong Java
  - 1 class chỉ được kế thừa 1 lớp cơ sở, nhưng có thể kế thừa nhiều interface
  - Sử dụng interface thay cho abstract class

```
package mypackage;
class myclass implements myinterface{

    public void mymethod1(){ //phải là public
        System.out.println("Override my method 1");
    }

    public void mymethod2(){
        System.out.println("Override my method 2");
    }

    void mymethod3(){//không là method trong interface
        System.out.println("My method 3");
    }
}
```



# Ví dụ về interface

```
9 import java.io.*;
10
11 interface Vehicle {
12     // all are the abstract methods.
13     void changeGear(int a);
14     void speedUp(int a);
15     void applyBrakes(int a);
16 }
17
18 class Bicycle implements Vehicle{
19     int speed;
20     int gear;
21     // to change gear
22     @Override
23     public void changeGear(int newGear){
24         gear = newGear;
25     }
26     // to increase speed
27     @Override
28     public void speedUp(int increment){
29         speed = speed + increment;
30         System.out.println("Bicycle Speed up " + speed+ " km/h");
31     }
32     // to decrease speed
33     @Override
34     public void applyBrakes(int decrement){
35         speed = speed - decrement;
36     }
37     public void printStates() {
38         System.out.println("after apply Brakes " + speed
39             + " gear: " + gear);
40     }
41 }
42
```

```
43 class Car implements Vehicle {
44     int speed;
45     int gear;
46     // to change gear
47     @Override
48     public void changeGear(int newGear){
49         gear = newGear;
50     }
51     // to increase speed
52     @Override
53     public void speedUp(int increment){
54         speed = speed + increment;
55         System.out.println("Car Speed up " + speed+ " km/h");
56     }
57     // to decrease speed
58     @Override
59     public void applyBrakes(int decrement){
60         speed = speed - decrement;
61     }
62     public void printStates() {
63         System.out.println("Speed after apply Brakes: " + speed
64             + " gear: " + gear);
65     }
66 }
67 class VehicleInterface {
68
69     public static void main (String[] args) {
70         // creating an instance of Bicycle
71         // doing some operations
72         Bicycle bicycle = new Bicycle();
73         bicycle.changeGear(2);
74         bicycle.speedUp(10);
75         bicycle.applyBrakes(1);
76
77         System.out.println("Bicycle present state :");
78         bicycle.printStates();
79         // creating instance of bike.
80         Car ca = new Car();

```

```
81         ca.changeGear(1);
82         ca.speedUp(80);
83         ca.applyBrakes(3);
84
85         System.out.println("Car present state :");
86         ca.printStates();
87     }
88 }
89
90
```



```
Output - vehicleInterface (run) x
run:
Bicycle Speed up 10 km/h
Bicycle present state :
after apply Brakes 9 gear: 2
Car Speed up 80 km/h
Car present state :
Speed after apply Brakes: 77 gear: 1
BUILD SUCCESSFUL (total time: 0 seconds)
```



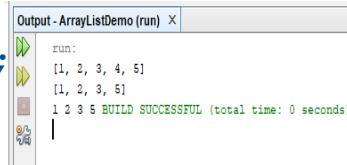
# Các Packages thư viện

Tên Package	Mô tả
java.lang	Chứa các class như Integer, String, System... và được tự động import vào mỗi chương trình Java.
java.util	Các các Java collections như List, Set, Map ...
java.io	Chứa các class liên quan đến việc nhập, xuất dữ liệu như File, Reader, Writer...
java.awt và java.swing	Chứa các class liên quan đến việc trình bày giao diện đồ họa và xử lý sự kiện.
...	

# Lớp thư viện ArrayList

- ArrayList là lớp được viết sẵn trong thư viện chuẩn của Java
- ArrayList được sử dụng để lưu tập các phần tử có kiểu dữ liệu tương tự nhau, tương tự như Array trong Java
- Chiều dài mảng của ArrayList tự thay đổi tùy thuộc số phần tử thêm vào hoạt lấy ra, không cần khai báo chiều dài mảng khi khởi tạo,
- ArrayList còn là cấu trúc có tham số kiểu, ta có thể tạo ArrayList với kiểu dữ liệu tùy ý.

```
ArrayList<DataType> myList =  
    new ArrayList<DataType>();
```



```
Output - ArrayListDemo (run) x  
run:  
[1, 2, 3, 4, 5]  
[1, 2, 3, 5]  
1 2 3 5 BUILD SUCCESSFUL (total time: 0 seconds)
```

```
8 import java.util.ArrayList;  
9  
10 public class ArrayListDemo {  
11  
12     public static void main(String[] args) {  
13         // TODO code application logic here  
14         // size of ArrayList  
15         int n = 5;  
16  
17         //declaring ArrayList with initial size n  
18         ArrayList<Integer> arrli = new ArrayList<Integer>(n);  
19  
20         // Appending the new element at the end of the list  
21         for (int i=1; i<=n; i++)  
22             arrli.add(i);  
23  
24         // Printing elements  
25         System.out.println(arrli);  
26  
27         // Remove element at index 3  
28         arrli.remove(3);  
29  
30         // Displaying ArrayList after deletion  
31         System.out.println(arrli);  
32  
33         // Printing elements one by one  
34         for (int i=0; i<arrli.size(); i++)  
35             System.out.print(arrli.get(i)+" ");  
36     }  
37 }
```



# Phương thức ArrayList

<code>add (value)</code>	Thêm value vào cuối list
<code>add (index, value)</code>	Thêm value trước phần tử index, di chuyển các phần tử qua phải 1 đơn vị
<code>clear ()</code>	Xóa tất cả các phần tử của list
<code>indexOf (value)</code>	Trả về giá trị index của phần tử đầu tiên bằng value tìm thấy trong list (-1 nếu không tìm thấy)
<code>get (index)</code>	Trả về value của phần tử vị trí index
<code>remove (index)</code>	xóa/trả về value tại vị trí index, di chuyển các phần tử còn lại qua bên trái
<code>set (index, value)</code>	Thay thế giá trị value tại index
<code>size ()</code>	Trả về số phần tử của list
<code>toString ()</code>	Ép kiểu, trả về kiểu dữ liệu string của list như là "[ 3, 42, -7, 15 ]"



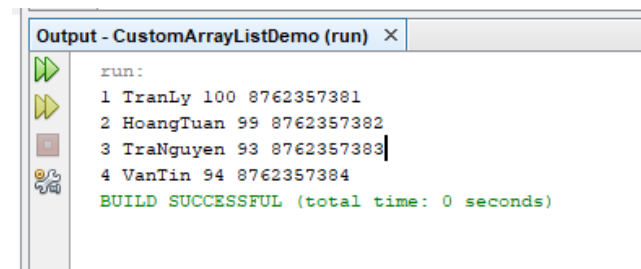
# Phương thức ArrayList

<code>addAll(<b>list</b>)</code>	Thêm tất cả phần tử của list vào list tại vị trí cuối list
<code>addAll(<b>index</b>, <b>list</b>)</code>	Thêm vào sau vị trí index
<code>contains(<b>value</b>)</code>	Trả về true, nếu value được tìm thấy ở list
<code>containsAll(<b>list</b>)</code>	Trả về true, nếu list chứa tất cả các phần tử của given list
<code>equals(<b>list</b>)</code>	Trả về true, nếu list chứa cùng phần tử
<code>iterator()</code> <code>listIterator()</code>	Trả về đối tượng dùng để thực thi nội dung list
<code>lastIndexOf(<b>value</b>)</code>	Trả về index của phần tử cuối cùng có giá trị value (-1 nếu không tìm thấy)
<code>remove(<b>value</b>)</code>	Tìm và xóa phần tử value trong list
<code>removeAll(<b>list</b>)</code>	Xóa tất cả các phần tử tìm thấy trong given list ở list cần xóa
<code>retainAll(<b>list</b>)</code>	Xóa các phần tử không tìm thấy trong given list ở list
<code>subList(<b>from</b>, <b>to</b>)</code>	Trả về list con có vị trí index từ <b>from</b> đến <b>to</b>
<code>toArray()</code>	Trả về các phần tử trong list dưới dạng mảng array

# Ví dụ về ArrayList

```
8 import java.util.ArrayList;
9
10 class Customer
11 {
12     // global variables of Customer
13     int roll;
14     String name;
15     int marks;
16     long phone;
17
18     // constructor has type of data that is required
19     Customer(int roll, String name, int marks, long phone)
20     {
21         // initialize the input variable from main
22         // function to the global variable of the class
23         this.roll = roll;
24         this.name = name;
25         this.marks = marks;
26         this.phone = phone;
27     }
28 }
29
30 public class CustomArrayListDemo {
31     /**
32     * @param args the command line arguments
33     */
34     int n=4;
35     public void addValues(int roll[], String name[], int marks[],
36         long phone[])
37     {
38         ArrayList<Customer> list=new ArrayList<>();
39         for (int i = 0; i < n; i++)
40         {
41             list.add(new Customer(roll[i], name[i], marks[i], phone[i]));
42         }
43         printValues(list);
44     }
45 }
```

```
46
47 public void printValues(ArrayList<Customer> list)
48 {
49     for (int i = 0; i < n; i++)
50     { Customer data = list.get(i);
51         System.out.println(data.roll+" "+data.name+" "
52             +data.marks+" "+data.phone);
53     }
54 }
55 public static void main(String[] args) {
56     // TODO code application logic here
57     int roll[] = {1, 2, 3, 4};
58     String name[] = {"TranLy", "HoangTuan", "TraNguyen", "VanTin"};
59     int marks[] = {100, 99, 93, 94};
60     long phone[] = {8762357381L, 8762357382L, 8762357383L,
61         8762357384L
62     };
63
64     // Create an object of the class
65     CustomArrayListDemo custom = new CustomArrayListDemo();
66     custom.addValues(roll, name, marks, phone);
67 }
68
69 }
```



```
Output - CustomArrayListDemo (run) X
run:
1 TranLy 100 8762357381
2 HoangTuan 99 8762357382
3 TraNguyen 93 8762357383
4 VanTin 94 8762357384
BUILD SUCCESSFUL (total time: 0 seconds)
```



# Lớp thư viện: Vector

- Vector là lớp được viết sẵn trong thư viện chuẩn của Java, hoạt động tương tự như Array truyền thống trong Java
- Lưu trữ tập các phần tử có kiểu dữ liệu Object
- Chiều dài **Vector thay đổi tùy thuộc số phần tử thêm vào hoặc lấy ra**, không cần khai báo kích cỡ khi khởi tạo Vector
- Thành phần mảng của vector có thể truy cập theo chỉ số **index**.
- **Lớp vector hỗ trợ bốn hàm dựng như sau:**
  - Vector mặc định, kích cỡ 10 phần tử Vector();
  - Vector kích thước size phần tử Vector(int size)
  - Vector kích thước size phần tử và lượng thay đổi incr Vector(int size, int incr)
  - Vector mà chứa các phần tử của collection c: Vector(Collection c)



# Phương thức Vector

<b>add(int index, Object element)</b>	Chèn <i>element</i> đã xác định tại vị trí đã cho trong Vector này
<b>boolean add(Object o)</b>	Thêm phần tử đã cho vào cuối của Vector này
<b>void addElement(Object obj)</b>	Thêm phần tử đã cho tới cuối của Vector này, tăng kích cỡ nó thêm 1
<b>int capacity()</b>	Trả về dung lượng hiện tại của Vector này
<b>void clear()</b>	Gỡ bỏ tất cả phần tử từ Vector này
<b>Object elementAt(int index)</b>	Trả về phần tử tại index đã cho
<b>Object firstElement()</b>	Trả về phần tử đầu tiên (tại chỉ mục 0) của Vector này
<b>int indexOf(Object elem)</b>	Tìm kiếm sự xuất hiện đầu tiên của tham số đã cho, kiểm tra tính cân bằng bởi sử dụng phương thức equals
<b>int indexOf(Object elem, int index)</b>	Tìm kiếm sự xuất hiện đầu tiên của tham số đã cho, bắt đầu tìm kiếm tại index, kiểm tra tính cân bằng bởi sử dụng phương thức equals

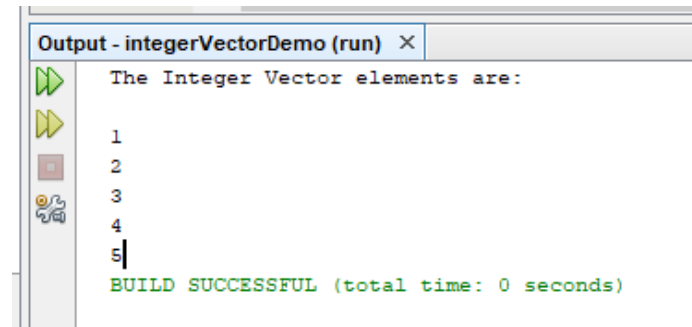


# Phương thức Vector

<b>void insertElementAt(Object obj, int index)</b>	Chèn đối tượng đã cho như là một phần tử vào Vector này tại index đã cho
<b>boolean isEmpty()</b>	Kiểm tra nếu Vector này không có phần tử
<b>Object lastElement()</b>	Trả về phần tử cuối cùng của Vector này
<b>int lastIndexOf(Object elem)</b>	Trả về chỉ mục của sự xuất hiện cuối cùng của đối tượng đã cho trong Vector này
<b>int lastIndexOf(Object elem, int index)</b>	Tìm kiếm ngược về sau cho đối tượng đã cho, bắt đầu từ index đã xác định, và trả về một chỉ mục
<b>Object remove(int index)</b>	Gỡ bỏ phần tử tại vị trí đã cho trong Vector này
<b>boolean remove(Object o)</b>	Gỡ bỏ sự xuất hiện đầu tiên của phần tử đã cho trong Vector này. Nếu Vector này không chứa phần tử đó, nó không bị thay đổi
<b>boolean removeAll(Collection c)</b>	Gỡ bỏ tất cả phần tử, mà chứa trong Collection đã cho, từ Vector này
<b>void removeAllElements()</b>	Gỡ bỏ tất cả phần tử từ Vector này và thiết lập kích cỡ về 0

# Ví dụ 1 về vector

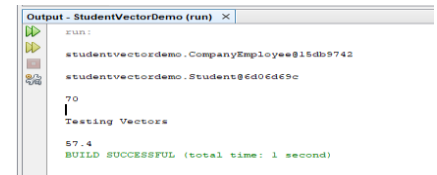
```
8 import java.util.Vector;
9 public class IntegerVectorDemo {
10
11     public static void main(String []args) {
12
13         Vector<Integer> intVector = new Vector<Integer>(); //Declaring a Vector in Java
14         intVector.add(1);
15         intVector.add(2);
16         intVector.add(3);
17         intVector.add(4);
18         intVector.add(5);
19         //Displaying vector elements
20         System.out.println("The Integer Vector elements are:");
21         System.out.println("");
22         for (int currIntVector : intVector) {
23             System.out.println(currIntVector);
24         }
25     }
26 }
```



```
Output - integerVectorDemo (run) X
The Integer Vector elements are:
1
2
3
4
5
BUILD SUCCESSFUL (total time: 0 seconds)
```

# Ví dụ 2 về Vector

```
9 import java.util.Vector;
10
11 class Student
12 {
13     //data members of class
14     private int rollno;
15     private String name;
16     private String schoolCode;
17
18     //Constructor
19     public Student(int rollno,String name,String schoolcode)
20     {
21         this.rollno=rollno;
22         this.name=name;
23         this.schoolCode=schoolcode;
24     }
25     //putStudent() to print the values of the student Object
26     public void putStudent()
27     {
28         System.out.println("RollNo :"+this.rollno+"\nName :"+this.rollno+"\nSchoolCode :"+this.schoolCode);
29     }
30 }
31
32 //class representing the employee
33 class CompanyEmployee
34 {
35     //data members of the class
36     public int id;
37     public String name;
38     public long salary;
39     //constructor
40     public CompanyEmployee(int id,String name,long salary)
41     {
42         this.id=id;
43         this.name=name;
44         this.salary=salary;
45     }
46
47     public void putEmployee()
48     {
49         System.out.println("Id :"+this.id+"\nName :"+this.name+"\nSalary :"+this.salary);
50     }
51 }
52
53 public class StudentVectorDemo {
54
55     public static void main(String[] args) {
56         // TODO code application logic here
57         Vector vecStuEmp=new Vector(); //Vector class to hold the objects
58         vecStuEmp.add(new CompanyEmployee(100,"Harendra Chekkur",34000)); // adding CompanyEmployee Object to Vector
59         vecStuEmp.add(new Student(10,"Madhav Singh","CB2025")); // adding Student Object to Vector
60         vecStuEmp.add(new Integer(70)); // adding Integer as Object to Vector
61         vecStuEmp.add(new String("Testing Vectors")); // adding String as an Object to Vector
62         vecStuEmp.add(new Float(57.4)); // adding employee as Object to Vector
63
64         //iterating the vector to print the Objects
65         for(Object objStuEmp:vecStuEmp)
66         {
67             /* as Vector holds heterogeneous data objects thus we have to cast the object to its type
68             in order to do this we are using getName() function which gets the name of the class of the given object
69             and compares it with the given class , if it matches then typecast the object to that class */
70             if(objStuEmp.getClass().getName().equals("logicProgramming.CompanyEmployee"))
71             {
72                 System.out.println();
73                 ((CompanyEmployee)objStuEmp).putEmployee();
74             }
75             else if(objStuEmp.getClass().getName().equals("logicProgramming.Student"))
76             {
77                 System.out.println();
78                 ((Student)objStuEmp).putStudent();
79             }
80             // if no match is found that is we will simply print the Object
81             else
82             {
83                 System.out.println();
84                 System.out.println(objStuEmp);
85             }
86         }
87     }
88 }
```



```
Output - StudentVectorDemo (run) x
run:
studentvectordemo.CompanyEmployee@15db9742
studentvectordemo.Student@6d06d9c
70
Testing Vectors
57.4
BUILD SUCCESSFUL (total time: 1 second)
```



# Bài tập

---

- Sinh viên hoàn thành bài tập chương IV (Theo giáo trình)