LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Bài 3:Java nâng cao

Trinh Thi Van Anh – PTIT

Nội dung

- Các kỹ thuật xây dựng lớp
- Nạp chồng
- Lớp bao
- Các kỹ thuật thao tác với chuỗi
- Tập hợp trong java (Collections)

Lớp (Class)

- Lớp (Class) là cách phân loại (classify) các đối tượng dựa trên đặc điểm chung của các đối tượng đó.
- Lớp có thể coi là khuôn mẫu để tạo các đối tượng
  - Ví dụ: Người, Sinh Vật, Màu sắc, tài khoản, đăng ký…
- Lớp chính là kết quả của quá trình trừu tượng hóa dữ liệu
  - Lớp định nghĩa một kiểu dữ liệu mới, trừu tượng hóa một tập các đối tượng
  - Một đối tượng gọi là một thể hiện của lớp

Thuộc tính

- Thuộc tính
  - Một thuộc tính của một lớp là một trạng thái chung được đặt tên của tất cả các thể hiện của lớp đó có thể có
  - Ví dụ: Lớp Ô tô có các thuộc tính
    - Màu sắc
    - Tên Hãng,….
- Các thuộc tính của nó cũng là các giá trị trừu tượng. Mỗi đối tượng có bản sao các thuộc tính của riêng nó
  - Ví dụ: một chiếc Ô tô đang đi có thể có màu đen, thuộc hãng HONDA CIVIC

Phương thức

- Phương thức
  - Xác định các hoạt động chung mà tất cả các thể hiện của lớp có thể thực hiện được.
  - Xác định cách một đối tượng đáp ứng lại một thông điệp
  - Thông thường các phương thức sẽ hoạt động trên các thuộc tính và thường làm thay đổi các trạng thái của lớp.
  - Bất kỳ phương thức nào cũng phải thuộc về một lớp nào đó
  - Ví dụ: Lớp Ô tô có các phương thức
    - Tăng tốc
    - Giảm tốc, …

Phạm vi

- Phạm vi nhìn thấy được xác định khả năng nhìn thấy được của một thành phần của chương trình với các thành phần khác của chương trình
- Đối với lớp
  - Phạm vi nhìn thấy được có thể được áp dụng cho các thành phần của lớp
    - private: chỉ truy cập được bên trong lớp đó
    - public: có thể truy cập được tại mọi nơi
    - protected: chỉ các class con, kế thừa từ class con mới được truy cập method của class cha

Gói (package)

- Gói (package) giống như thư mục giúp:
  - Tổ chức và xác định vị trí lớp dễ dàng và sử dụng các lớp một cách phù hợp.
  - Tránh cho việc đặt tên lớp bị xung đột (trùng tên)
    - Các package khác nhau có thể chứa các lớp có cùng tên
  - Bảo vệ các lớp, dữ liệu và phương thức ở mức rộng hơn so với mối quan hệ giữa các lớp.
- Một package cũng có thể chứa các package khác

Gói trong Java (1)

- Java đã xây dựng sẵn một số package
  - java.lang
  - javax.swing
  - java.io
  - …
- Có thể tự tạo ra các gói để tổ chức các lớp
  - Cú pháp:

```
package <tên gói>;
```

Gói trong Java (2)

- Tên gói phải được viết trên cùng của file mã nguồn
- Chỉ được phép có 1 câu khai báo gói trong mỗi file mã nguồn, và khai báo này sẽ được áp dụng cho tất cả các dữ liệu trong file đó.
- Một gói có thể được đặt trong một gói khác
  - Phân cách bằng dấu .
  - Ví dụ package `model.entity;`

Quy ước đặt tên gói

- Tên gói được viết toàn bộ bằng chữ thường để tránh xung đột với tên lớp hay giao diện
- Đối với các công ty có tên miền Internet: Sử dụng tên miền đảo để đặt tên gói
  - Ví dụ: Một lập trình viên tại công ty example.com sẽ đặt tên gói là `com.example.mypackage`

Các package cơ bản trong Java

- java.lang (không cần `import` vào)
  - Cung cấp các lớp cơ bản cho thiết kế ngôn ngữ lập trình Java
  - Bao gồm wrapper classes, String và StringBuffer, Object, ...
  - **Import ngầm định** vào tất cả các lớp
- java.util
  - Bao gồm tập hợp framework, mô hình sự kiện, date time, và nhiều tiện ích khác.
- java.io
  - Cung cấp khả năng vào/ra hệ thống với các luồng dữ liệu và hệ thống file.

- java.math
  - Cung cấp các lớp thực thi các phép toán với số nguyên và các phép toán thập phân
- java.sql
  - Cung cấp các API cho phép truy nhập và xử lý dữ liệu được lưu trữ trong một nguồn dữ liệu (thường sử dụng cơ sở dữ liệu quan hệ)
- javax.swing
  - Cung cấp các lớp và giao diện cho phép tạo ra các ứng dụng đồ họa.
- ...

Khai báo lớp

- Cú pháp: sử dụng từ khóa class

```
class <ClassName>{
    <kiểu dữ liệu> <field_1>;
    <kiểu dữ liệu> <field_2>;
    constructor // hàm khởi tạo
    method_1 // phương thức của lớp
    method_2
}
```

Cú pháp khai báo lớp sử dụng chỉ định truy cập:

```
accessmodifier class <Tên Lớp> {
// Nội dung lớp
}
```

Chỉ định truy cập

- public: Lớp có thể được truy cập từ bất cứ đâu, kể cả bên ngoài package chứa lớp đó.
- private: Lớp chỉ có thể được truy cập trong phạm vi lớp đó
- protected: chỉ các class con, kế thừa từ class con mới được truy cập method của class cha
- mặc định: Lớp có thể được truy cập từ bên trong package chứa lớp đó.

Modifier	Class	Package	Subclass	World
`public`	✓	✓	✓	✓
`protected`	✓	✓	✓	✗
`no modifier*`	✓	✓	✗	✗
`private`	✓	✗	✗	✗

- public: toàn cục
- protected: chỉ các class con, kế thừa từ class con mới được truy cập method của class cha
- no modifier*: ta hiểu là mặc định, tức không có từ khóa tầm vực đứng phía trước
- private: chỉ trong nội bộ class mới gọi được.
- class: trong cùng 1 class
- package: trong cùng 1 gói (package)
- subclass: gọi trong class kế thừa (class con)
- world: toàn cục (nơi bất kỳ trong ứng dụng)

Thuộc tính

- Bản chất của các thuộc tính là các thành phần dữ liệu của đối tượng
- Thuộc tính của lớp được khai báo bên trong lớp

```
class <ClassName>
{
// khai báo những thuộc tính của lớp
//<quyền truy xuất> <kiểu dữ liệu> <tên>;
// …
}
```

- Thuộc tính có thể được khởi tạo khi khai báo. Các giá trị mặc định sẽ được sử dụng nếu không được khởi tạo.
- Ví dụ:

```
private String hoten;
private float hsluong = 2.06f;
```

Thành viên tĩnh trong OOP

- Trong lập trình hướng đối tượng
- Các thành viên bình thường là thành viên thuộc về đối tượng
- Các thành viên tĩnh (**static**) là các thành viên **thuộc về lớp** (mang thông tin chung của một lớp)
- Cú pháp khai báo thành viên static:

```
<chỉ định truy cập> static <kiểu> tên biến;
```

- Chỉ có 1 bản copy biến này được chia sẻ cho tất cả các đối tượng của lớp
- Sự thay đổi giá trị của biến này sẽ ảnh hưởng tới tất cả các đối tượng của lớp.

Ví dụ

```
public class Circle {
    public float radius=0;
    public static float PI=3.14f;
    public Circle(float r){
        radius=r;
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Circle c1=new Circle(11.5f);
        Circle c2=new Circle(5.8f);
       System.out.println("c1 radius: "+c1.radius);
        System.out.println("c1 PI: "+c1.PI);
     System.out.println("\n c2 radius: "+c2.radius);
        System.out.println("c2 PI: "+c2.PI);
        System.out.println("\nPI: "+Circle.PI);
        c1.radius=45.6f;
        c1.PI=10.5f;
      System.out.println("\nc1 radius: "+c1.radius);
        System.out.println("c1 PI: "+c1.PI);
      System.out.println("\nc2 radius: "+c2.radius);
        System.out.println("c2 PI: "+c2.PI);
    }}
```

???

```
public class SinhVien {
    private int id;
    public  String hoten;
    private  int tuoi;
    public   static float hs4 = 0.4f;
    …….
}
SinhVien s1 = new SinhVien();
s1.hoten="Do My Linh";
S1.tuoi=20;
System.out.println("id ="+s1.id);
System.out.println("Ho ten:"+s1.hoten);
System.out.println("H S:"+s1.hs4);
System.out.println("HS:"+SinhVien.hs4);
```

Phương thức của lớp

- Phương thức xác định các hoạt động của lớp
- Bất kỳ phương thức nào cũng phải thuộc về một lớp nào đó
- Có hai loại phương thức trong ngôn ngữ Java:
  - Hàm khởi tạo (Constructor)
  - Các phương thức get/set và khác
    - Phương thức thể hiện (Instance Method)
    - Gọi phương thức và truyền tham số kiểu trị (Passing Arguments by Value).
    - Gọi phương thức và truyền tham số kiểu tham chiếu (Passing Arguments by Reference).
    - Phương thức tĩnh (Static Methods)
    - Phương thức tham số biến (Variable Argument Methods)

Hàm khởi tạo (Constructor)

- Constructor là phương thức đặc biệt được gọi khi tạo object
- Mục đích: Khởi động trị cho biến instance của class.
- A constructor phải thỏa 2 điều kiện:
  - Cùng tên class
  - Không giá trị trả về
- Một lớp có thể có nhiều Constructors
- Nếu không viết Constructor, trình biên dịch tạo default constructor
  - Default constructor không thông số và không làm gì cả

```
public class TaiKhoan {
    private String tenTaiKhoan;
    private String matKhau;
    public TaiKhoan() {
    }
    public TaiKhoan(String
  tenTaiKhoan, String matKhau) {
      this.tenTaiKhoan = tenTaiKhoan;
      this.matKhau = matKhau;
    }
…….
}
```

```
public class Matrix{
   private double[][] A;
   private int row, column;
   public Matrix (int row, int column) {
      this.row = row;
      this.column = column;
      A = new double[row][column];
   }
   public Matrix (double[][] A) {
      row = A.length;
      column = A[0].length;
      this.A = A;
   }
   public Matrix (double[][] A, int row, int
  column) {
      this.A = A;
      this.row = row;
      this.column = column;
   }
   ……
```

Phương thức get và set

- Các thành phần bên trong cần giới hạn truy cập (private), vì thế sinh ra việc sử dụng hàm get/set.
- Quy luật bất biến của module hoá, mỗi hàm chỉ làm và thực hiện một việc duy nhất. Vậy thì hàm **SET** thì chỉ có **đặt** giá trị vào, còn **GET** thì chỉ **lấy** giá trị ra. Không thể nhét cả **GET/SET** vô trong một hàm.
- Trong lập trình, hàm viết càng nhỏ càng tốt, càng ít logic càng tốt.

Ví dụ Get/set

```
public class Author{
        private String tenTacGia;
        private String maTacGia;
        private int dienThoai;
    public Author() {    }
    public int getDienThoai() {
        return dienThoai;
    }
    public void setDienThoai(int dienThoai) {
        this.dienThoai = dienThoai;
    }
    public String getMaTacGia() {
        return maTacGia;
    }
```

```
    public void setMaTacGia(String maTacGia) {
            this.maTacGia = maTacGia;
       }
    public String getTenTacGia() {
       return tenTacGia;    }
    public void setTenTacGia(String tenTacGia) {
       this.tenTacGia = tenTacGia;
      }
      @Override
    public String toString() {
       return
    (tenTacGia+”,”+maTacGia+”,”+Integer.toString(dien
    Thoai));
      }
}
```

Ví dụ lớp ma trận

```
public class Matrix{
   private double[][] A;
   private int row, column;
   public Matrix (int row, int column) {
      this.row = row;
      this.column = column;
      A = new double[row][column];   }
   public Matrix (double[][] A) {
      row = A.length;
      column = A[0].length;
      this.A = A;   }
   public Matrix (double[][] A, int row, int column) {
      this.A = A;
      this.row = row;
      this.column = column;   }
```

```
public double[][] getArray() { // getA
      return A;   }
   public int getRow () {
      return row;   }
   public int getColumn() {
      return column;   }
   public double getValue (int i, int j) {
      return A[i][j];   }
   public void setRow(int row){
       this.row = row;   }
   public void setColumn(int column){
       this.column = column;   }
   public void setArray(double[][] b){//setA
       for (int i = 0; i < row; i++) {
         for (int j = 0; j < column; j++) {
            A[j][i] = b[i][j]; }
      }
   }
```

Định nghĩa phương thức

- Cú pháp trên bao gồm

```
modifier kieuTraVe tenPhuongThuc(Danh
  sach tham so) {
    // Than phuong thuc }
```

  - **modifier:** Nó định nghĩa kiểu truy cập của phương thức và nó là tùy ý để sử dụng.
  - **kieuTraVe:** Phương thức có thể trả về một giá trị.
  - **tenPhuongThuc:** Đây là tên phương thức.
  - **Danh sach tham so:** Danh sách các tham số, nó là kiểu, thứ tự, và số tham số của một phương thức. Đây là tùy ý, phương thức có thể không chứa tham số nào.
  - **Than phuong thuc:** Phần thân phương thức định nghĩa những gì phương thức đó thực hiện với các lệnh.

Gọi phương thức

- Các giá trị từ phương thức gọi (calling method) sẽ được truyền như đối số tới phương thức được gọi (called method).
- Bất kỳ sự thay đổi của đối số trong phương thức được gọi đều **không ảnh hưởng** đến các giá trị được truyền từ phương thức gọi.
- Các biến có giá trị **kiểu nguyên thủy** (primitive types int, float …) sẽ được truyền theo kiểu này.

Tham số biến

```
public class Caculator {
    int sum(int... a){
       int t=0;
       for(int x:a)
          t+=x;
       return t;
    }
    double sum(double... a){
       double t=0.0;
       for(double x:a)
          t+=x;
       return t;
    }
    public static void main(String[] args) {
        Caculator c = new Caculator();
        System.out.println("Tong: " +
c.sum(2,3,4));
        System.out.println("Tong: " +
c.sum(2,3,4,8,7));
        System.out.println("Tong thuc : " +
c.sum(2.5,3.6,4.8,8,7));
    } }
```

- Phương thức tham số biến. Phương thức này cho phép gọi phương thức với số tham số **thay đổi.**

Ví dụ tài khoản

```
public class BankAccount {
  private String owner;
  private double balance;
  public boolean withdraw(double amount){
      if (amount > balance)
       return false;
      else {
         balance -= amount;
         return true;
      }
  }
  public void deposit(double amount){
      balance += amount;
  }
}
```

```
public Matrix transpose () {
      Matrix X = new Matrix(column,row);
      double[][] C = X.getArray();
      for (int i = 0; i < row; i++) {
         for (int j = 0; j < column; j++) {
            C[j][i] = A[i][j];  }}
      return X;
   }
   public Matrix plus (Matrix B) {
      Matrix X = new Matrix(row,column);
      double[][] C = X.getArray();
      for (int i = 0; i < row; i++) {
         for (int j = 0; j < column; j++) {
            C[i][j] = this.A[i][j] + B.A[i][j];
         }  }
      return X;
   }
```

Ví dụ ma trận

```
public Matrix minus (Matrix B) {
      Matrix X = new Matrix(row,column);
      double[][] C = X.getArray();
      for (int i = 0; i < row; i++) {
         for (int j = 0; j < column; j++) {
            C[i][j] = A[i][j] - B.A[i][j];
         }
      }
      return X;
   }
public double sum () {
      double t = 0;
     for (int i = 0; i < row; i++) {
         for (int j = 0; j < column; j++) {
            t += A[i][j];
      }
      return t;
   }
```

```
public Matrix multiply (double[][] b) {
   int r = b.length;
   int c = b[0].length;
   if (column != r) throw new
  RuntimeException("can't multiply 2
  matrix.");
   double[][] C = new double[row][c];
   for (int i = 0; i < row; i++)
      for (int j = 0; j < c; j++)
         for (int k = 0; k < column; k++)
           C[i][j] += A[i][k] * b[k][j];
   Matrix X = new Matrix(C,row,c);
   return X;
   }
```

Phương thức tĩnh (Static Methods)

- Là những **phương thức** được gọi thông qua **tên Lớp** (**không** cần đối tượng).
- Khai báo phương thức thêm từ khóa **`static`**.
- Chỉ có thể truy xuất 1 cách trực tiếp tới các biến tĩnh (**`static`**) và các phương thức tĩnh khác của lớp.
- Không thể truy xuất đến các phương thức và biến không tĩnh (**`non-static`**).

Phương thức tĩnh (2)

- Việc sử dụng phương thức tĩnh
  - Khi phương thức không truy xuất tới các trạng thái của đối tượng.
  - Khi phương thức chỉ quan tâm đến các biến tĩnh

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

    public int sub(int a, int b)
    {
        return (a - b);
    }
}
```

```
public static void main(String[] args)
{
    Calculator cal = new Calculator();
    int s = cal.sub(3, 5);

    System.out.println("Subtraction result: " + s);

    //calling static method
    int a = Calculator.add(3,5);

    System.out.println("Addition result: " + a);
```

```
Console
<terminated> Calculator [Java Application] C:\Program Fil
Subtraction result: -2
Addition result: 8
```

Thành viên lớp và thành viên đối tượng

Thành viên đối tượng	Thành viên lớp (static)
Thuộc tính/phương thức chỉ được **truy cập** thông qua **đối tượng**	Thuộc tính/phương thức có thể được **truy cập** thông qua **lớp**
Mỗi đối tượng có **1 bản sao riêng** của 1 thuộc tính đối tượng	Các đối tượng có **chung 1 bản sao** của 1 thuộc tính lớp
Giá trị của **1 thuộc tính** đối tượng của các đối tượng khác nhau là **khác nhau**.	Giá trị của **1 thuộc tính** lớp của các đối Tượng khác nhau là **giống nhau**.

Ví dụ

- Lớp `JOptionPane` trong `javax.swing`
- Cú pháp

```
public static void
  showMessageDialog(Component
  parentComponent, Object message,
  String title, int messageType)
```

- Ví dụ

```
JOptionPane.showMessageDialog(null,
"Ban da thao tac loi","Thong bao
  loi",JOptionPane.ERROR_MESSAGE);
```

Thành viên hằng

- Một thuộc tính/phương thức không thể thay đổi giá trị/nội dung trong quá trình sử dụng.
- Cú pháp khai báo: Sử dụng từ khóa **final**

```
<chỉ định truy cập> final <kiểu> tên
  hằng = giá trị;
```

- Ví dụ:

```
public  final double PI= 3.141592653589793;
public final int VAL_THREE = 39;
private final int[] A={ 1, 2, 3, 4, 5, 6 };
```

Khai báo

- Trong OOP, lớp có thể coi là kiểu dữ liệu trừu tượng do người dùng định nghĩa và đối tượng chính là biến của kiểu dữ liệu đó
- Khai báo: tương tự khai báo biến
  - Cú pháp: <Tên Lớp> tên đối tượng;
  - Ví dụ:

```
// Đối tượng acc là một BankAccount
BankAccount acc;
```

Tạo đối tượng của lớp

- Tạo đối tượng (object) dùng toán tử **new**
- Cú pháp

```
// gọi tới contructor mặc định
ClassName objectName = new ClassName();
// gọi tới constructor có tham số
ClassName objectName1 = new ClassName(ts1,
  ts2, …);
```

- Truy xuất các thuộc tính và phương thức
  - Khi tạo object bằng toán tử new, vùng nhớ được cấp phát cho mỗi thuộc tính và phương thức của class.
  - Để truy cập các thuộc tính và phương thức của đối tượng dùng toán tử dấu chấm (dot operator).

- Nếu một class không có constructor, trình biên dịch tạo ra constructor mặc định **không** có tham số.
- Nếu class có một hoặc nhiều constructor, bất kể tham số kiểu gì, trình biên dịch sẽ không thêm mặc định constructor nữa.
- Ví dụ: lớp không có constructor mặc định

```
Book b = new Book();
```

```
public class Sach {
   private String
   ten,tacGia,chuyenNghanh;
   private int ma,nam,soLuong;
   public Sach(String ten, String
   tacGia, String chuyenNghanh, int ma,
   int nam, int soLuong) {
       this.ten = ten;
       this.tacGia = tacGia;
       this.chuyenNghanh = chuyenNghanh;
       this.ma = ma;
       this.nam = nam;
       this.soLuong = soLuong;
     }
   ……
```

Tự tham chiếu

- Sử dụng từ khóa **this**
- Cho phép truy cập vào đối tượng hiện tại của lớp.
- Quan trọng khi hàm/phương thức thành phần thao tác trên hai hay nhiều đối tượng.
- Xóa đi sự nhập nhằng giữa một biến cục bộ, tham số với thành phần dữ liệu của lớp
- Không dùng bên trong các khối lệnh static

this trong hàm tạo contructor

```
class HinhChuNhatThis
{
      double cdai, crong;
      public HinhChuNhatThis (double cd, double cr)
      {
            cdai = cd;
            crong = cr;
      }
      public HinhChuNhatThis()
      {
          this(0,0);
      }
      public double DienTich()
      {
          return cdai * crong;
      }
      public double ChuVi()
      {
          return (cdai + crong)*2;
      }
}
```

this (tt)

- Dùng **this** đại diện cho đối tượng
- Đại diện cho đối tượng, dùng để truy xuất một thành phần của đối tượng

  `this.tênThànhPhần.`
- Khi tham số trùng với tên thuộc tính thì nhờ từ khóa **this** để phân biệt rõ thuộc tính với tham số.

```
public void setTen(String ten) {
        this.ten = ten;
    }
```

Tham chiếu đến lớp khác gói

- Đối với lớp trong cùng một gói: chỉ cần tên lớp
  - Ví dụ: `BankAccount`
- Đối với lớp khác gói: phải cung cấp đầy đủ tên lớp và tên gói
  - Ví dụ trong Java:

  `model.entity.BankAccount`
- Sử dụng lệnh **`import`** để khai báo các package hoặc các lớp để khi sử dụng không cần nêu tên đầy đủ.

```
import model.entity.BankAccount;
```

Từ khóa final trong Java

- Từ khóa final trong Java được sử dụng để hạn chế người dùng. Từ khóa final có thể được sử dụng trong nhiều ngữ cảnh: với biến, với phương thức và với lớp.
- Biến final trong Java
  - Biến là final: không thể thay đổi giá trị của biến (hằng số).

Ví dụ

```
class Bike{
 final int speedlimit=90;//bien final
 void run(){
    speedlimit=400;//Lỗi
 }
 public static void main(String
  args[]){
 Bike obj=new  Bike();
 obj.run();
 }
}
```

Biến final trống

- Một biến final mà không được khởi tạo tại thời điểm khai báo được gọi là biến final trống. Nếu bạn muốn tạo một biến mà được khởi tạo tại thời điểm tạo đối tượng và một khi nó đã được khởi tạo thì không thể bị thay đổi, thì biến final trống là hữu ích trong trường hợp này.
- Nó chỉ có thể được khởi tạo **trong** Constructor

Ví dụ

```
class Bike10{
   //bien final trong
  final int speedlimit;
  Bike10(){
      speedlimit=70;
      System.out.println(speedlimit);
  }
  public static void main(String
  args[]){
       new Bike10();
} }
```

Biến static final trống trong Java

- Một biến static final mà không được khởi tạo tại thời điểm khai báo thì đó là biến static final trống. Nó chỉ có thể được khởi tạo **trong** khối **static**.

```
class A{
   //bien static final trong
   static final int data;
   static{ data=50;}
   public static void main(String
   args[]){
       System.out.println(A.data); } }
```

Tham số final

- Nếu khai báo bất cứ tham số nào là final, thì ta không thể thay đổi giá trị của nó.

```
class Bike{
  int cube(final int n){
     n=n+2;//Lỗi
     n*n*n; }
   public static void main(String
  args[]){
     Bike b = new Bike();
     b.cube(5); } }
```

Phương thức final

- Nếu tạo bất cứ phương thức nào là final, thì ta không thể ghi đè nó.

```
class Bike{
  final void run(){
    System.out.println("running");} }
class Honda extends Bike{
  void run(){
    System.out.println("Chay an toan voi
  100kmph");}
  public static void main(String args[]){
  Honda honda= new Honda(); honda.run(); }
  }
```

Lớp final

- Nếu tạo bất cứ lớp nào là final thì ta **không** thể **kế thừa** nó.

```
final class Bike{}
class Honda extends Bike{//Lỗi
  void run(){
     System.out.println("Chay an toan
  voi 100kmph");}
  public static void main(String
  args[]){
    Honda honda= new Honda();
    honda.run();
} }
```

Nạp chồng (overloading)

- Nạp chồng hay chồng phương thức (method overloading): Các phương thức trong cùng một lớp có thể **trùng tên**
  - Số lượng tham số khác nhau
  - Nếu cùng số lượng tham số thì kiểu dữ liệu các tham số phải khác nhau hoặc trật tự các tham số khác nhau
- Mục đích:
  - Tên trùng nhau để mô tả bản chất công việc
  - Thuận tiện cho lập trình vì không cần phải nhớ quá nhiều tên phương thức mà chỉ cần nhớ một tên và lựa chọn các tham số cho phù hợp.

- Trình biên dịch so sánh danh sách thông số thực để quyết định gọi phương thức nào.
- Kiểu giá trị trả về của phương thức không được tính vào dấu hiệu của overloading method
- Các constructors có thể được overloaded. Một overloaded constructor cho ta nhiều cách khác nhau để tạo ra một đối tượng mới.

```
class Calculation{
  void sum(int a,int b){
    System.out.println(a+b);}
  void sum(int a,int b,int c){
    System.out.println(a+b+c);}
  public static void main(String args[]){
    Calculation obj=new Calculation();
    obj.sum(10,10,10);
    obj.sum(20,20); } }
```

Ví dụ

```
class MyDate {
  int year, month, day;
  public boolean setMonth(int m) { …}
  public boolean setMonth(String s) { …}
}
public class Test{
  public static void main(String args[]){
    MyDate d = new MyDate();
    d.setMonth(9);
    d.setMonth(”September”);
}
}
```

Chú ý

- Nạp chồng phương thức là **không thể** bằng cách thay đổi **kiểu trả về** của phương thức. Bởi vì việc này có thể gây ra tính lưỡng nghĩa, mơ hồ (ambiguity).

```
class Calculation{
 int sum(int a,int b){
    return (a+b);}
 double sum(int a,int b){
    return (double)(a+b);}
 public static void main(String args[]){
    Calculation obj=new Calculation();
 //Gây ra Compile Time Error
    int result=obj.sum(20,20); } }
```

Ví dụ về nạp chồng trong Java

- Phương thức println() trong System.out.println() có 10 khai báo với các tham số khác nhau: boolean, char[], char, double, float, int, long, Object, String, và một không có tham số.
- Không cần sử dụng các tên khác nhau (chẳng hạn printString hoặc printDouble) cho mỗi kiểu dữ liệu muốn hiển thị.

Lớp bao (wrapper class)

- Các kiểu dữ liệu nguyên thủy không có các phương thức liên quan đến nó.
- Mỗi kiểu dữ liệu nguyên thủy có một lớp tương ứng gọi là lớp bao (wrapper class)
- Các lớp bao sẽ "gói" dữ liệu nguyên thủy và cung cấp các phương thức thích hợp cho dữ liệu đó.
- Mỗi đối tượng của lớp bao đơn giản là lưu trữ một biến đơn và đưa ra các phương thức để xử lý nó.
- Các lớp bao là một phần của Java API

Wrapper trong Java

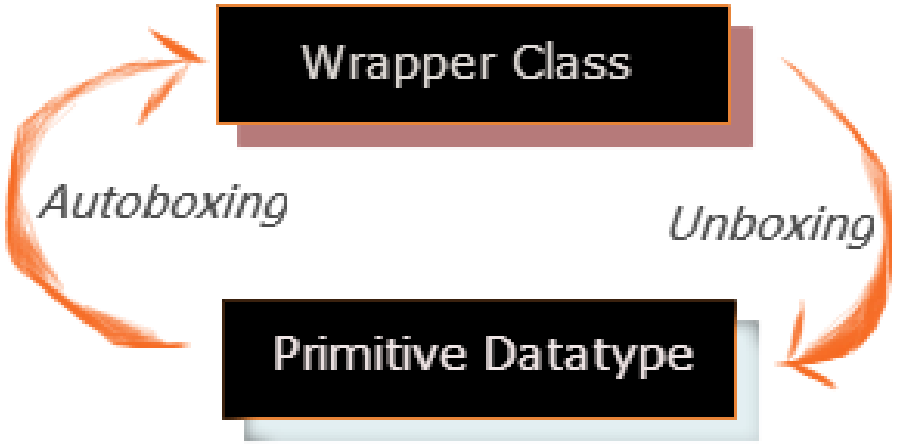
Kiểu gốc	Lớp Wrapper
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

Lớp bao

- Các lớp bao là không thay đổi được
  - Sau khi đã được gán một giá trị, thể hiện của lớp đó không được phép thay đổi giá trị nữa.
- Các lớp bao là final
  - Không thể kế thừa từ các lớp bao
- Tất cả các phương thức của các lớp bao là static

Autoboxing và Unboxing

- Java tự động convert kiểu dữ liệu primitvies (thường dịch là nguyên thủy) sang một class wrapper tương ứng thì được gọi là *Autoboxing*. Và ngược lại, việc convert từ wrapper sang kiểu dữ liệu primitives gọi là *Unboxing*.



Trả về biến của kiểu dữ liệu nguyên thủy

- Để trả về các kiểu dữ liệu nguyên thủy: sử dụng phương thức **`typeValue()`**
  - Không có tham số
  - Mỗi kiểu số có 6 phương thức `typeValue()` tương ứng
  - Ví dụ

```
// make a new wrapper object
Integer i2 = new Integer(42);
// chuyển vê`byte
byte b = i2.byteValue();
// chuyển vê`short
short s = i2.shortValue();
// chuyển vê`double
double d = i2.doubleValue();
```

Chuyển đổi từ String sang các kiểu dữ liệu nguyên thủy

- Dùng các phương thức static của lớp bao

```
static <type> parseType(String s)
```

- Ví dụ

```
String s = "123";
//chuyển sang int
int i = Integer.parseInt(s);
// chuyển sang short
short j = Short.parseShort(s)
Float x =Float.parseFloat(txt);//
  String txt
```

Java.lang.Integer Class

1. static int MAX_VALUE: $2^{31}-1$.
2. static int MIN_VALUE : -2^{31}.
3. Integer(int value), Integer(String s)
4. java.lang.Integer.compare()
5. Integer obj1 = new Integer("25"); Integer obj2 = new Integer("10");
6. int **retval** = obj1.compareTo(obj2);
7. int **retval** = Integer.compare(obj1,obj2);
8. byte byteValue(), double doubleValue(), float floatValue()…
9. static int parseInt(String s), String toString(), static String toString(int i)
10. boolean equals(Object obj)

Java.lang.Double Class

- `Double(double value), Double(String s)`
- `byte byteValue(), double doubleValue(), float floatValue(), int intValue(), ..`
- `static int compare(double d1, double d2), int compareTo(Double anotherDouble)`
- `boolean isNaN(), static double parseDouble(String s), String toString(), static String toString(double d)`

Các kỹ thuật thao tác với chuỗi

- Kiểu String là một lớp và không phải là kiểu dữ liệu nguyên thủy
- Một String được tạo thành từ một dãy các ký tự nằm trong dấu nháy kép:

```
String a = "A String";
String b = "";
```

- Đối tượng String có thể khởi tạo theo nhiều cách:

```
String c = new String();
String d = new String("Another String");
String e = String.valueOf(1.23);
String f = null;
```

Ghép xâu

- Toán tử + có thể nối các String:

```
String a = "This" + " is a " + "String";
//a = “This is a String”
```

- Các kiểu dữ liệu cơ bản sử dụng trong lời gọi println() được chuyển đổi tự động sang kiểu String

```
System.out.println("answer="+1 + 2 + 3);
System.out.println("answer="+(1+2+3));
System.out.println("So: "+1.45);
```

Một số phương thức của xâu

- String name = “Ly Lao Lo";
- name.toLowerCase(); // "ly lao lo"
- name.toUpperCase(); // "LY LAO LO"
- “   Ly Lao Lo ".trim();// "Ly Lao Lo"
- “ Ly Lao Lo".indexOf(‘L'); // 1
- "Ly Lao Lo".length(); // 9
- "Ly Lao Lo".charAt(5); // ‘o'
- "Ly Lao Lo".substring(5); // "o Lo"
- "Ly Lao Lo".substring(2,5); // " La“

- int **compareTo**(String anotherString)
- int **compareToIgnoreCase**(String str)
- String **concat**(String str)
- boolean **endsWith**(String suffix)

```
String str = "www.tutorialspoint.com";
String endstr1 = ".com";
String endstr2 = ".org";
boolean retval1 = str.endsWith(endstr1);
boolean retval2 = str.endsWith(endstr2);
```

- boolean **equals**(Object anObject)
- boolean **equalsIgnoreCase**(String anotherString)
- int **indexOf**(String str)
- int **lastIndexOf**(String str)

boolean matches(String regex)

TT	Biểu thức chính quy	Mô tả
1	.	Khớp (match) với bất kỳ ký tự nào
2	^regex	Biểu thức chính quy phải khớp tại điểm bắt đầu
3	regex$	Biểu thức chính quy phải khớp ở cuối dòng.
4	[abc]	Thiết lập định nghĩa, có thể khớp với a hoặc b hoặc c.
5	[abc][vz]	Thiết lập định nghĩa, có thể khớp với a hoặc b hoặc c theo sau là v hay z.
6	[^abc]	Khi dấu ^ xuất hiện như là nhân vật đầu tiên trong dấu ngoặc vuông, nó phủ nhận mô hình. bất kỳ ký tự nào ngoại trừ a hoặc b hoặc c.
7	[a-d1-7]	Phạm vi: phù hợp với một chuỗi giữa a và điểm d và con số từ 1 đến 7.
8	X|Z	Tìm X hoặc Z.
9	XZ	Tìm X và theo sau là Z.
10	$	Kiểm tra kết thúc dòng.

- Mẫu regex mô tả một ký tự bất kỳ.

```
String regex = ".";
```

- Mẫu regex mô tả ký tự dấu chấm.

```
String regex = "\\.";
```

11	`\d`	Số bất kỳ, viết ngắn gọn cho `[0-9]`
12	`\D`	Ký tự không phải là số, viết ngắn gon cho `[^0-9]`
13	`\s`	Ký tự khoảng trắng, viết ngắn gọn cho `[ \t\n\x0b\r\f]`
14	`\S`	Ký tự không phải khoản trắng, viết ngắn gọn cho `[^\s]`
15	`\w`	Ký tự chữ, viết ngắn gọn cho `[a-zA-Z_0-9]`
16	`\W`	Ký tự không phải chữ, viết ngắn gọn cho `[^\w]`
17	`\S+`	Một số ký tự không phải khoảng trắng (Một hoặc nhiều)
18	`\b`	Ký tự thuộc a-z hoặc A-Z hoặc 0-9 hoặc _, viết ngắn gọn cho `[a-zA-Z0-9_]`

19	`*`	Xuất hiện 0 hoặc nhiều lần, viết ngắn gọn cho `{0,}`
20	`+`	Xuất hiện 1 hoặc nhiều lần, viết ngắn gọn cho `{1,}`
21	`?`	Xuất hiện 0 hoặc 1 lần, `?` viết ngắn gọn cho `{0,1}`.
22	`{X}`	Xuất hiện X lần, `{}`
23	`{X,Y}`	Xuất hiện trong khoảng X tới Y lần.
24	`*?`	* có nghĩa là xuất hiện 0 hoặc nhiều lần, thêm ? phía sau nghĩa là tìm kiếm

Ví dụ

1. "B11CNTT345".
   matches("^[Bb]{1}\\d{2}[A-Z]{4}\\d{3}$");
2. "b11CNTT345".
   matches("^[Bb]{1}\\d{2}[A-Z]{4}\\d{3}$");
3. “Bb11CNTT345".
   matches("^[Bb]{1}\\d{2}[A-Z]{4}\\d{3}$");
4. “N234BN”.
   matches("^N{1}|X{1}\\d{3}[A-Za-z]{4}$");

Cắt chuỗi split()

- public String[] **split**(String regex, int limit)
- public String[] **split**(String regex)
- **1.** String s1="Vi du ve viec tach tu";

```
String[] st1=s1.split("\\s+");
for(String w1:st1){
    System.out.println(w1); }
```

- **2.** String s2="Ban la ai? ban co xinh khong? chac xinh! dung xinh.";

```
String[] st2=s2.split("[\\.\\?\\!]");
for(String w2:st2){
    System.out.println(w2); }
```

replaceAll() và replaceFirst()

- public String **replaceAll**(String regex, String replacement)
- public String **replaceFirst**(String regex, String replacement)

1. “Hom qua toi buon qua!”. replaceAll("\\s+",“ ");
2. String s3="Ban la ai.ban co xinh khong. chac xinh.dung xinh."; System.out.println(s3.replaceAll("\\ .\\s*", ". "));
3. "Toi sinh ra o Thai Binh. Toi hoc o Nguyen Duc Canh. Toi la sinh vien Bach Khoa.“. replaceAll("Toi","Ban");

StringBuilder

- String là kiểu bất biến, đối tượng không thay đổi giá trị sau khi được tạo ra.
- Lớp StringBuilder trong Java được sử dụng để tạo chuỗi có thể thay đổi.
- Lớp StringBuilder là giống như lớp StringBuffer ngoại trừ rằng nó là **không** đồng bộ.
- Các Constructor quan trọng
  - `StringBuilder()`
  - `StringBuilder(String str)`
  - `StringBuilder(int capacity)`

Phương thức	Miêu tả
public StringBuilder append(String s)	Được sử dụng để phụ thêm (append) chuỗi đã cho với chuỗi này. Phương thức append() được nạp chồng giống dạng append(char), append(boolean), append(int), append(float), append(double) ...
public StringBuilder insert(int offset, String s)	Được sử dụng để chèn chuỗi đã cho với chuỗi này tại vị trí đã cho. Phương thức insert() được nạp chồng giống dạng insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) ...
public StringBuilder replace(int startIndex, int endIndex, String str)	Được sử dụng để thay thế chuỗi từ chí mục ban đầu startIndex và chí mục kết thúc endIndex đã cho
public StringBuilder delete(int startIndex, int endIndex)	Được sử dụng để xóa chuỗi từ chí mục startIndex và endIndex đã cho
public StringBuilder reverse()	Được sử dụng để đáo ngược chuỗi
public int capacity()	Được sử dụng để trả về dung lượng capacity hiện tại

public char charAt(int index)	Được sử dụng để trả về ký tự tại vị trí đã cho
public int length()	Được sử dụng để trả về độ dài của chuỗi (chẳng hạn như tổng số ký tự)
public String substring(int beginIndex)	Được sử dụng để trả về chuỗi con từ chỉ mục bắt đầu beginIndex đã cho
public String substring(int beginIndex, int endIndex)	Được sử dụng để trả về chuỗi con từ beginIndex đến endIndex đã cho

```
1. StringBuilder s=new
   StringBuilder("D14- ");
   s.append("CN");//D14- CN
2. s.insert(4,"CQ");//D14-CQ CN
3. s.delete(3,5);// D14 CN
4. s.reverse();
```

StringBuffer

- Các Constructor
  - `StringBuffer()`
  - `StringBuffer(String str)`
  - `StringBuffer(int capacity)`
- Các phương thức
  - `public synchronized StringBuffer append(String s)`
  - `public synchronized StringBuffer insert(int offset, String s)`
  - `public synchronized StringBuffer replace(int startIndex, int endIndex, String str)`
  - ……..

StringBuffer và StringBuilder

Lớp StringBuffer	Lớp StringBuilder
Lớp StringBuffer là đồng bộ (*synchronized*), tức là an toàn luồng (thread safe). Nghĩa là hai Thread không thể gọi đồng thời các phương thức của lớp StringBuffer	StringBuilder là không đồng bộ (*non-synchronized*) tức là không an toàn luồng. Nghĩa là hai Thread có thể gọi đồng thời các phương thức của lớp StringBuilder
StringBuffer là *kém hiệu quả hơn* StringBuilder	StringBuilder là *hiệu quả hơn* StringBuffer

```
public class ConcatTest{
  public static void main(String[] args){
    long startTime = System.currentTimeMillis();
    StringBuffer sb1 = new StringBuffer("Java");
    for (int i=0; i<10000; i++){
      sb1.append(“D14CN-PTIT"); }
      System.out.println("Thoi gian tieu ton boi
   StringBuffer: " + (System.currentTimeMillis() -
   startTime) + "ms");
      startTime = System.currentTimeMillis();
      StringBuilder sb2 = new
   StringBuilder("Java");
    for (int i=0; i<10000; i++){
       sb2.append(“D14CN-PTIT"); }
    System.out.println("Thoi gian tieu ton boi
   StringBuilder: " + (System.currentTimeMillis() -
   startTime) + "ms");
 } }
```

Thoi gian tieu ton boi StringBuffer: 15ms

Thoi gian tieu ton boi StringBuilder: 0ms

Lớp StringTokenizer

- Lớp `java.util.StringTokenizer` cho phép ta chia một chuỗi thành các token. Đây là cách đơn giản để chia chuỗi.
- Các Constructor
  - `StringTokenizer(String str)`: Tạo StringTokenizer với chuỗi string đã cho
  - `StringTokenizer(String str, String delim)`: Tạo StringTokenizer với chuỗi string và dấu phân tách delimeter đã cho

Các phương thức StringTokenizer

Phương thức public	Miêu tả
boolean hasMoreTokens()	Kiểm tra xem có nhiều token có sẵn không
String nextToken()	Trả về token tiếp theo từ đối tượng StringTokenizer
String nextToken(String delim)	Trả về token tiếp theo dựa trên dấu phân tách delim
Object nextElement()	Giống như nextToken() nhưng kiểu trả về của nó là Object
int countTokens()	Trả về tổng số token

Ví dụ

```
import java.util.StringTokenizer;
public class Simple{
public static void main(String
  args[]){
   StringTokenizer st = new
   StringTokenizer("Toi lam. viec o.
   HaNoi“,”\\.”);
   while(st.hasMoreTokens()) {
   System.out.println(st.nextToken());
   } } }
```

- By default StringTokenizer breaks String

```
String str = "I am sample string and will be
  tokenized on space";
StringTokenizer dt=new StringTokenizer(str);
while (dt.hasMoreTokens()){
  System.out.println(dt.nextToken());}
```

- Multiple delimiters

```
String s =“Toi la ai? Lan ban cua ban. Toi
yey ban! Ban yeu toi? Tat nhien roi!”;
StringTokenizer mt = new
StringTokenizer(s, “.?!");
while (mt.hasMoreTokens()){
  System.out.println(mt.nextToken());
}
```

Java Collections

- Tập hợp dùng lưu trữ, thao tác trên một nhóm các đối tượng.
- Tập hợp hoạt động gần giống với array ngoại trừ việc nó có thể thay đổi kích thước
- Tập hợp là đối tượng có khả năng chứa các đối tượng khác.
- Các đối tượng của tập hợp có thể thuộc nhiều loại dữ liệu khác nhau
- Các thao tác thông thường trên tập hợp
  - Thêm/Xoá đối tượng vào/ra tập hợp
  - Kiểm tra một đối tượng có ở trong tập hợp hay không
  - Lấy một đối tượng từ tập hợp
  - Duyệt các đối tượng trong tập hợp
  - Xoá toàn bộ tập hợp

Tập hợp (tt)

- Collections Framework (từ Java 1.2)
  - Là một kiến trúc hợp nhất để biểu diễn và thao tác trên các collection.
  - Giúp cho việc xử lý các collection độc lập với biểu diễn chi tiết bên trong của chúng.
- Một số lợi ích của Collections Framework
  - Giảm thời gian lập trình
  - Tăng cường hiệu năng chương trình
  - Dễ mở rộng các collection mới
  - Sử dụng lại mã chương trình

Collections Framework

- Collections Framework bao gồm:
  - Interfaces: Là các interface thể hiện tính chất của các kiểu collection khác nhau như List, Set, Map.
  - Implementations: Là các lớp collection có sẵn được cài đặt các collection interfaces.
  - Algorithms: Là các phương thức tĩnh để xử lý trên collection, ví dụ: sắp xếp danh sách, tìm phần tử lớn nhất...

Tập hợp và mảng

Mảng	Tập hợp
Mảng truy xuất 1 cách **tuần tự**	tập hợp (có thể) truy xuất theo dạng **ngẫu nhiên**
Mảng chứa **1 loại** đối tượng/dữ liệu nhất định	Tập hợp có thể chứa **nhiều loại** đối tượng/dữ liệu khác nhau
Dùng tổ chức dữ liệu theo mảng **phải lập trình** hoàn toàn	dùng theo kiểu tập hợp **xây dựng sẵn** của Java chỉ khai báo và gọi những phương thức đã được định Nghĩa
Duyệt các phần tử mảng tuần tự thông qua **chỉ số** mảng	Duyệt các phần tử tập hợp thông qua **Iterator**

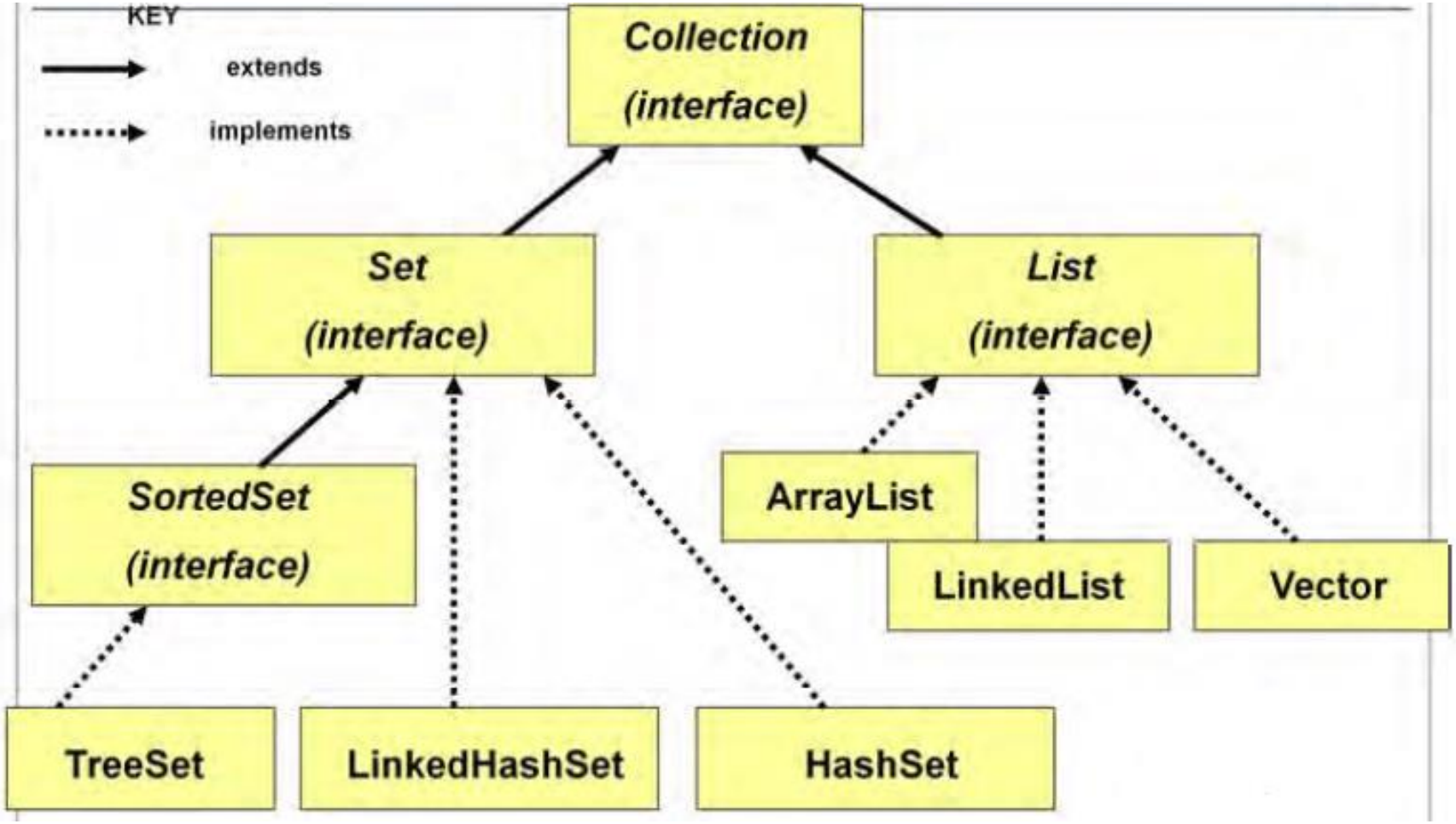
Duyệt tập hợp

- Iterator cho phép duyệt các phần tử của một collection.
- Các phương thức của Iterator:
  - `boolean hasNext();`
  - `Object next();`
  - `void remove();`

Ví dụ

```
Collection name =new ArrayList();
 name.add("Lan anh");
 name.add("Hoa bao");
 Iterator i = name.iterator();
 String s;
 while(i.hasNext()){
      s = (String)i.next();
      System.out.println(s);
 }
```

Các interfaces của tập hợp



ArrayList

- Lớp ArrayList trong Java kế thừa **AbstractList** và cài đặt từ **List** Interface. Lớp ArrayList hỗ trợ các mảng động mà có thể tăng kích cỡ nếu cần.
- ArrayList được tạo với một kích cỡ ban đầu. Khi kích cỡ này bị vượt, collection tự động được tăng. Khi các đối tượng bị gỡ bỏ, ArrayList có thể bị giảm kích cỡ.
- Constructor
  - `ArrayList( )`: một danh sách mảng trống.
  - `ArrayList(Collection c)`: một Array List mà được khởi tạo với các phần tử của collection c.
  - `ArrayList(int capacity)`: một Array List mà có dung lượng ban đầu được xác định.

Các phương thức

Phương thức	Miêu tả
void add(int index, Object element)	Chèn phần tử đã cho tại index đã xác định trong list này.
boolean add(Object o)	Thêm vào phần tử đã xác định vào cuối list này
boolean addAll(Collection c)	Thêm vào tất cả phần tử trong collection đã xác định vào cuối list này,
void clear()	Gỡ bỏ tất cả phần tử trong list này
boolean contains(Object o)	Trả về true nếu list này chứa phần tử đã cho.
Object get(int index)	Trả về phần tử tại vị trí index.

Phương thức	Miêu tả
protected void removeRange(int fromIndex, int toIndex)	Gỡ bỏ từ list tất cả phần tử mà có index từ fromIndex đến toIndex
Object set(int index, Object element)	Thay thế phần tử tại vị trí index bởi phần tử element.
int size()	Trả về số phần tử trong list này
int indexOf(Object o) **int lastIndexOf(Object o)**	Trả về index trong list này của sự xuất hiện đầu tiên (cuối) của phần tử đã cho, hoặc -1 nếu list không chứa phần tử này
Object remove(int index)	Gỡ bỏ phần tử tại index đã cho.

Ví dụ

```
public class ArrayListDemo {
   public static void main(String
   args[]){
      ArrayList al = new ArrayList();
      System.out.println("Size ban dau
        la: " + al.size());
      // them cac phan tu
      al.add("C");      al.add("A");
      al.add("E");      al.add("B");
      al.add("D");      al.add("F");
      al.add(1, "A2");
```

```
        System.out.println("Size sau khi
        them la: " + al.size());
        System.out.println("Noi dung la: "
        + al);
        al.remove("F"); al.remove(2);
        System.out.println("Size sau khi
        xoa la: " + al.size());
        System.out.println("Noi dung la: "
        + al);
    }
}
```

Kết quả

```
Size ban dau la: 0
Size sau khi them la: 7
Noi dung la: [C, A2, A, E, B, D, F]
Size sau khi xoa la: 5
Noi dung la: [C, A2, E, B, D]
```

Generic trong java

- **Generics** là một khái niệm được đưa vào Java từ phiên bản 5.
- Thuật ngữ "*Generics*" nghĩa là **tham số hóa kiểu dữ liệu**. Tham số hóa kiểu dữ liệu rất quan trọng vì nó cho phép chúng ta tạo ra và sử dụng một class, interface, method với **nhiều kiểu dữ liệu khác nhau**.
- Một class, interface hay một method mà thực hiện trên **một kiểu tham số xác định** thì gọi là **generic**.

```
1.public class BeforeJ5Example {
2.    public static void main(String[] args) {
3.       //Tạo một đt ds chứa tên
4.       ArrayList userNames = new ArrayList();
5.       // Thêm String vào danh sách
6.       userNames.add("Ha Anh");
7.       userNames.add("Thuy Vi");
8.       //Thêm một phần tử không phải String
9.       userNames.add(new Integer(24));
10.        // Và lấy ra phần tử đầu
11.        Object obj1 = userNames.get(0);
12.        //Ép kiểu về String
13.        String userName1 = (String) obj1;
14.      System.out.println("userName1 = " + userName1);
15.        // Lấy ra phần tử thứ 2
16.        String userName2 = (String) userNames.get(1);
17.      System.out.println("userName2 = " + userName2);
18.        // Lấy ra phần tử thứ 3
19.        // (Lỗi ép kiểu xẩy ra tại đây).
20.        String userName3 = (String) userNames.get(2);
21.      System.out.println("userName3 = " + userName3);
22.   }}
```

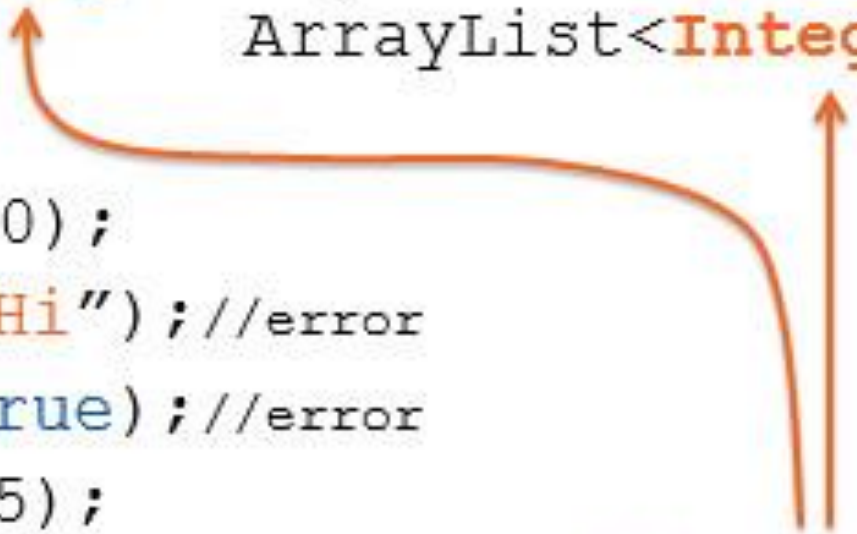
Nhận xét

- Khi tạo ra một đối tượng **ArrayList** với mục đích chỉ chứa các phần tử có kiểu **String**, tuy nhiên tại nơi nào đó trong chương trình ta thêm vào danh sách này một phần tử không phải **String** (Việc này hoàn toàn có thể), khi lấy ra các phần tử đó và ép kiểu về **String**, một ngoại lệ sẽ bị ném ra.
- **Java 5** đưa vào khái niệm **Generics**. Với sự trợ giúp của **Generics**, chúng ta có thể tạo ra một đối tượng **ArrayList** chỉ cho phép chứa các phần tử có kiểu **String**, và không cho phép chứa các phần tử có kiểu khác.
- Khi tạo một đối tượng **ArrayList<String>**, nó chỉ chứa các phần tử có kiểu **String**, trình biên dịch của Java không cho phép đối tượng này chứa các phần tử có kiểu khác **String**.

Ví dụ

```
ArrayList<Integer> mylist = new
                        ArrayList<Integer>();

mylist.add(10);
mylist.add("Hi");//error
mylist.add(true);//error
mylist.add(15);
```



Generic

Ưu điểm của generic

- Kiểm tra kiểu dữ liệu trong thời điểm biên dịch
- Không cần ép kiểu dữ liệu, ví dụ
- ```
  ArrayList list = new ArrayList();
  list.add("hello");
  String s=(String)list.get(0);//phải ép kiểu
  ```
- ```
  ArrayList<String> list = new ArrayList<String>();
  list.add("hello");
  String s=list.get(0);  //không ép kiểu
  ```

Ví dụ QL thí sinh

- Hai đối tượng
  - Thí sinh dự thi: số báo danh, họ tên, điểm toán, điểm lý, điểm hóa
  - Phòng thi: mã phòng thi, phòng thi và số lượng sinh viên

Menu bài toán

1. Thêm thí sinh
2. Sửa thông tin thí sinh
3. Xóa thí sinh
4. Tìm kiếm thí sinh (theo nhiều tiêu chí)
5. Sắp xếp thí sinh (theo nhiều tiêu chí)
6. In danh sách (theo nhiều tiêu chí)
7. Nhập điểm thí sinh

0 . Thoát

Lớp ThiSinh

```
public class ThiSinh {
    private String sobaodanh;
    private String hoten;
    private float toan;
    private float ly;
    private float hoa;
    public ThiSinh() {
    }
```

```
public ThiSinh(String sobaodanh, String
  hoten,float toan, float ly, float hoa) {
        this.sobaodanh = sobaodanh;
        this.hoten = hoten;
        this.toan = toan;
        this.ly = ly;
        this.hoa = hoa;
    }
public ThiSinh(String sobaodanh) {
        this.sobaodanh=sobaodanh;
        this.hoten="";
        toan=0.0f;
        ly=0.0f;
        hoa=0.0f;
    }
```

```
// Các phương thức getter và setter
// cho 5 thuộc tính của đối tượng

public String toString() {
  return sobaodanh+"\t"+hoten+"\t"+
  toan+"\t"+ly+"\t"+hoa;
}
```

Phương thức contains(Object o)

- Trong java, có các kiểu dữ liệu như int, char, String, Date … đã được dựng sẵn, ta chỉ cần sử dụng. Nhưng đôi khi chính các đối tượng này được xây dựng sẵn, khiến cho ta hay bị lầm lẫn giữa các khái niệm như: bằng nhau, lớn – nhỏ hơn khi ta tự xây dựng các đối tượng.
- Với các kiểu dữ liệu có sẵn là int, char… thì có thể dùng dấu == để so sánh. Còn với các đối tượng khác, như String hay Date thì gọi hàm **equals(),** hai cách đem lại cùng một kết quả.
- Tất cả các đối tượng trong Java đều có 1 gốc đối tượng cha duy nhất là Object, bản thân đối tượng Object có 2 phương thức: equals() và **hashCode()**. equals() sẽ dùng để so sánh các đối tượng. Các lớp dẫn xuất từ Object có thể định nghĩa thế nào là bằng nhau nhờ Override hàm này.
- Vậy, muốn đối tượng bằng nhau trọn vẹn khi và chỉ khi: hàm equals() trả về true, và hàm hashCode() trả về cùng một giá trị.

```
1. @Override
2. public boolean equals(Object o) {
3.    if (this == o)
4.          return true;
5.    if (o == null)
6.          return false;
7.    if (getClass() != o.getClass())
8.          return false;
9.    ThiSinh other = (ThiSinh) o;
10.   if (sobaodanh == null) {
11.         if (other.sobaodanh != null)
12.               return false;
13.   } else
    if(!sobaodanh.equals(other.sobaodanh))
14.         return false;
15.   return true;
16.     }
```

Lớp PhongThi

```
public class PhongThi {
    private String mapt;
    private String phongthi;
    private int soluong;
    private ArrayList<ThiSinh> list;
    public PhongThi() { }
    public PhongThi(String mapt,
       String phongthi, int soluong) {
        this.mapt = mapt;
        this.phongthi = phongthi;
        this.soluong = soluong;
        list = new ArrayList<ThiSinh>();
    }
// getter and setter
```

```
public boolean themThiSinh() {
    Scanner input=new Scanner(System.in);
    System.out.println("Nhap thong tin thi
  sinh");
    System.out.print("\n Nhap so bao danh:
  ");
    String sobaodanh = input.nextLine();
    ThiSinh ts = new ThiSinh(sobaodanh);
    if(list.contains(ts))
        return false;
    System.out.print("\n Nhap Ho ten: ");
    String hoten=input.nextLine();
    ts.setHoten(hoten);
    return list.add(ts);
     }
```

```
public boolean xoaThiSinh(){
   Scanner input=new
  Scanner(System.in);
   System.out.print("\n Nhap so bao
  danh thi sinh can xoa: ");
   String sobaodanh = input.nextLine();
   ThiSinh ts = new
  ThiSinh(sobaodanh,"",0f,0f,0f);
  if(!list.contains(ts))
     return false;
  return list.remove(ts);
     }
```

```
public boolean suaThiSinh() {
   Scanner input=new Scanner(System.in);
   System.out.print("\n Nhap so bao danh
   thi sinh can sua: ");
   String sobaodanh = input.nextLine();
   ThiSinh ts = new
   ThiSinh(sobaodanh,"",0f,0f,0f);
   if(!list.contains(ts))
      return false;
   ts = list.get(list.indexOf(ts));
   System.out.print("\n Nhap ho ten: ");
   String hoten = input.nextLine();
   ts.setHoten(hoten);
   return true;
     }
```

```
public ThiSinh layThiSinh() {
    Scanner input=new
  Scanner(System.in);
    System.out.print("\n Nhap so bao
  danh: ");
    String sobaodanh = input.nextLine();
    ThiSinh ts = new
  ThiSinh(sobaodanh,"",0f,0f,0f);
    if(!list.contains(ts))
      return null;
   ts = list.get(list.indexOf(ts));
   return ts;
    }
```

```
public ArrayList<ThiSinh>
  layThiSinhByName(String ten) {
        ArrayList<ThiSinh> l= new
  ArrayList<>();
        int i=0;
        for(ThiSinh ts:list){
  if(ts.getHoten().equalsIgnoreCase(ten)){
                l.add(ts);
                i++;
            }
        }
        if(i==0)
            return null;
        else
      return l;
    }
```

```
public ArrayList<ThiSinh>
  layThiSinhByDiem(float from, float to) {
        ArrayList<ThiSinh> l= new
  ArrayList<>();
        int i=0;
        for(ThiSinh ts:list){
            if(ts.getToan()>=from &&
  ts.getToan()<=to){
                l.add(ts);
                i++;
            }
        }
        if(i==0)
            return null;
        else
      return l;
    }
```

```
public  void inDanhSach(){
  System.out.println("=DANH SACH
  SINH VIEN=");
   System.out.println("So BD  Ho va
  ten Diem Toan  Diem Ly    Diem
  Hoa");
   for (ThiSinh sv: list)
  System.out.println(sv.toString());
  System.out.println("==============
  ======================");
     }
```

```
public void nhapDiem(){
        Scanner in=new Scanner(System.in);
        inDanhSach();
        float dt,dl,dh;
        int i=0;
        for(ThiSinh t:list){
            System.out.println("So bao
  danh:"+t.getSobaodanh());
            System.out.print("Nhap diem toan:");
            dt=Float.parseFloat(in.nextLine());
            System.out.print("\n Nhap diem ly:");
            dl=Float.parseFloat(in.nextLine());
            System.out.print("\n Nhap diem hoa:");
            dh=Float.parseFloat(in.nextLine());
            t=list.get(i); t.setToan(dt);
            t.setLy(dl);   t.setHoa(dh);
            i++;
        }
    }
```

java.util.Comparator

- Việc sắp xếp một các đối tượng trong 1 ArrayList có thể có nhiều cách. Ta có thể áp dụng các phương pháp cơ bản. Java đã hỗ trợ sẵn 1 cách sắp xếp ngắn gọn, đó là phương thức sắp xếp Collection.sort().
- Để Collections.sort() có thể làm việc được ta phải truyền vào một đối tượng implements java.util.Comparator (interface), trong đó định nghĩa qui tắc sắp xếp.

Sắp xếp theo xâu

```
1. public void sapXepByTen(){
2.  Collections.sort(list, new
   Comparator<ThiSinh>() {
3.  @Override
4.  public int compare(ThiSinh sv1,
   ThiSinh sv2) {
5.      return
   (sv1.getHoten().compareTo(sv2.getHot
   en()));
6.  }
7.  });
8.  }
```

Sắp xếp theo số

```
1. public void sapXepByDiemLy(){
2.  Collections.sort(list, new
   Comparator<ThiSinh>() {
3.  @Override
4.  public int compare(ThiSinh sv1, ThiSinh sv2) {
5.    if (sv1.getLy() < sv2.getLy()) {
6.       return 1;
7.    } else {
8.        if (sv1.getLy() == sv2.getLy()) {
9.           return 0;
10.       } else
11.          return -1;
12.        } }      }
13.         );
14.    }
```

Vector

- Lớp Vector trong Java khai triển một mảng động. Nó tương tự như ArrayList, nhưng được **đồng bộ**.
- Lớp Vector trong Java được chứng minh rất hữu ích nếu bạn không biết kích cỡ của mảng
- Constructor:
  - Vector( ): tạo một vector mặc định có kích cỡ khởi tạo là 10
  - Vector(int size): tạo một vector mà dung lượng khởi tạo là size
  - Vector(Collection c): tạo một vector mà chứa các phần tử của collection c

Các phương thức

Phương thức	Miêu tả
void add(int index, Object element)	Chèn *element* đã xác định tại vị trí đã cho trong Vector này
boolean add(Object o)	Thêm vào phần tử đã cho vào cuối của Vector này
boolean addAll(Collection c)	Thêm vào tất cả phần tử trong Collection đã xác định ở cuối của Vector này, để mà chúng được trả về bởi Iterator của Collection đã cho đó
void clear()	Gỡ bỏ tất cả phần tử từ Vector này
Object clone()	Trả về một mô phỏng của Vector này
boolean contains(Object elem)	Kiểm tra nếu đối tượng đã cho là một phần tử trong Vector này

Phương thức	Miêu tả
boolean containsAll(Collection c)	Trả về true nếu Vector này chứa tất cả phần tử trong Collection đã cho
Object elementAt(int index)	Trả về phần tử tại index đã cho
void copyInto(Object[] anArray)	Sao chép các thành phần của Vector này vào trong mảng đã cho
Enumeration elements()	Trả về một bản liệt kê các phần tử của Vector này
boolean equals(Object o)	So sánh Object đã cho với Vector này về sự cân bằng
Object firstElement()	Trả về phần tử đầu tiên trong Vector này
int indexOf(Object elem) **int indexOf(Object elem, int index)**	Tìm kiếm sự xuất hiện đầu tiên của tham số đã cho (bắt đầu tìm kiếm tại index)
void insertElementAt(Object obj, int index)	Chèn đối tượng đã cho như là một phần tử vào Vector này tại index đã cho

Phương thức	Miêu tả
boolean isEmpty()	Kiểm tra tính rỗng của Vector này
Object lastElement()	Trả về phần tử cuối cùng của Vector này
Object remove(int index) **void removeElementAt(int index)**	Gỡ bỏ phần tử tại vị trí đã cho trong Vector này
boolean remove(Object o)	Gỡ bỏ sự xuất hiện đầu tiên của phần tử đã cho trong Vector này
boolean removeAll(Collection c)	Gỡ bỏ tất cả phần tử, mà chứa trong Collection đã cho, từ Vector này
boolean removeElement(Object obj)	Gỡ bỏ sự xuất hiện đầu tiên
Object set(int index, Object element) **void setElementAt(Object obj, int index)**	Thay thế phần tử tại vị trí đã cho trong Vector này với phần tử đã xác định

Phương thức	Miêu tả
void setSize(int newSize)	Thiết lập kích cỡ của Vector này
int size()	Trả về số phần tử trong Vector này
Object[] toArray()	Trả về một mảng chứa tất cả phần tử trong Vector này theo đúng thứ tự
String toString()	Trả về một biểu diễn chuỗi của Vector này, chứa biểu diễn chuỗi của mỗi phần tử

```
import java.util.*;
public class VectorDemo {
   public static void main(String args[]) {
      // capacity ban dau la 3, incr la 2
      Vector v = new Vector(3, 2);
      System.out.println("Size ban dau: " +
  v.size());
      System.out.println("Capacity ban dau: " +
      v.capacity());
```

```
v.addElement(new Integer(1));v.addElement(new Integer(2));
v.addElement(new Integer(3));v.addElement(new Integer(4));
   System.out.println("Capacityla: "+v.capacity());
   v.addElement(new Double(5.45));
   System.out.println("Capacity : " + v.capacity());
   v.addElement(new Double(6.08));
   v.addElement(new Integer(7));
   System.out.println("Capacity : " + v.capacity());
   System.out.println("Phan tu dau tien: " +
      (Integer)v.firstElement());
   System.out.println("Phan tu cuoi cung: " +
      (Integer)v.lastElement());
   if(v.contains(new Integer(3)))
      System.out.println("Vector chua 3.");
   Enumeration vEnum = v.elements();
   System.out.println("\nCac phan tu trong Vector:");
   while(vEnum.hasMoreElements())
      System.out.print(vEnum.nextElement() + " ");
   System.out.println();
}}
```

ViDuVector (phân số)

```
public class Fraction{
    private int tu;
    private int mau;
    public Fraction(int t, int m) {
  if(m != 0){
      tu = t;
      mau = m;
  }
  else
      System.exit(0);
  }
// getter/setter
```

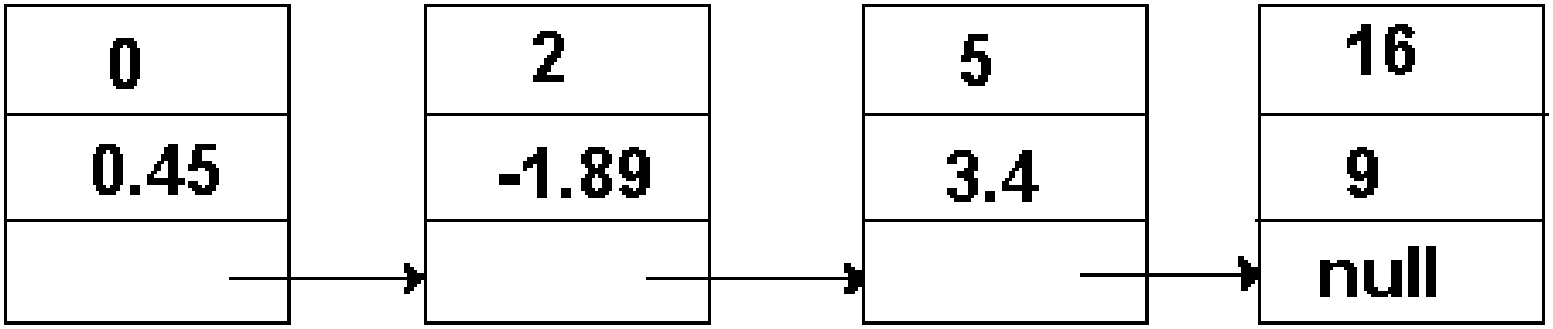
```
private static int ucln(int x, int y) {
   int mod;
   if(x < y){
       mod = x;
       x = y;
       y = mod;    }
   int r = x % y;
   while (r != 0) {
       x = y;
       y = r;
       r = x % y;  }
   return y;
     }
private Fraction rutGon(int t, int m){
     int uclnNum = ucln(t,m);
     m = m / uclnNum;
     t = t / uclnNum;
     return new Fraction(t,m);
     }
```

```
public Fraction tong(Fraction b){
    int num1 = (this.tu * b.mau) + (b.tu * this.mau);
    int num2 = this.mau * b.mau;
    return rutGon(num1,num2);   }
public Fraction hieu(Fraction b){
    int num1 = (this.tu * b.mau) - (b.tu * this.mau);
    int num2 = this.mau * b.mau;
    return rutGon(num1,num2);    }
public Fraction tich(Fraction b){
    int num1 = this.tu * b.tu;
    int num2 = this.mau * b.mau;
    return rutGon(num1,num2);    }
public Fraction thuong(Fraction b){
    int num1 = this.tu * b.mau;
    int num2 = this.mau * b.tu;
    return rutGon(num1, num2);    }
public String toString(){
    if(tu > mau && mau > 1)
      return (tu + "/" + mau + " or " + (tu/mau) + " " +
   (tu % mau) + "/" + mau);
    else
      return(tu + "/" + mau);   }}
```

```
Vector vt = new Vector();
Fraction ps1 = new Fraction(1,2);
Fraction ps2 = new Fraction(3,2);
Fraction ps3 = new Fraction(1,7);
Fraction ps4, pscout;
vt.addElement(ps1);
vt.addElement(ps2);
vt.addElement(ps3);
System.out.println("\n Cac phan tu
vector:");
Enumeration list = vt.elements();
while (list.hasMoreElements()) {
  pscout = (Fraction) list.nextElement();
  System.out.println("Phan so: " + pscout);
}
ps4 = ps1.tong(ps2);
System.out.println("Tong la: " + ps4);
```

LinkedList

- Lớp LinkedList trong Java cài đặt từ List Interface. Nó cung cấp một cấu trúc dữ liệu linked-list (dạng danh sách được liên kết).
- Constructor:
  - LinkedList( )
  - LinkedList(Collection c)
- $0.45 - 1.89\ x^2 + 3.4\ x^5 + 9\ x^{16}$



Polynomial: ViDuLinkedList

```
public class Polynomial {
    private Node first = new Node(0, 0);
    private Node last  = first;
    private static class Node {
        int coef; // coefficient – hệ số
        int exp;  // Exponent – số mũ
        Node next;
        Node(int coef, int exp) {
            this.coef = coef;
            this.exp  = exp;
        }
    }
    private Polynomial() { }
    // a * x^b - - 1.89 x^2
    public Polynomial(int coef, int exp) {
        last.next = new Node(coef, exp);
        last = last.next;
    }
```

```
// return c = a + b
public Polynomial plus(Polynomial b) {
  Polynomial a = this;
  Polynomial c = new Polynomial();
  Node x = a.first.next;
  Node y = b.first.next;
  while (x != null || y != null) {
     Node t = null;
     if(x == null){ t = new Node(y.coef, y.exp);  y = y.next; }
     else if(y == null){t = new Node(x.coef, x.exp);  x = x.next; }
     else if(x.exp > y.exp){t=new Node(x.coef, x.exp);x = x.next; }
     else if(x.exp < y.exp){t=new Node(y.coef, y.exp); y=y.next; }
          else {
             int coef = x.coef + y.coef;
             int exp  = x.exp;
             x = x.next;
             y = y.next;
             if (coef == 0) continue;
             t = new Node(coef, exp);}
            c.last.next = t;
             c.last = c.last.next;
        }
        return c;
    }
```

```
// return c = a * b
public Polynomial multiply(Polynomial b) {
   Polynomial a = this;
   Polynomial c = new Polynomial();
   for(Node x=a.first.next; x!= null; x= x.next)
  {
      Polynomial temp = new Polynomial();
      for (Node y = b.first.next; y!= null; y =
  y.next) {
         temp.last.next = new Node(x.coef *
  y.coef, x.exp + y.exp);
         temp.last = temp.last.next;
            }
         c = c.plus(temp);
        }
        return c;
    }
```

```
public String toString() {
  String s = "";
  for(Node x = first.next; x != null; x = x.next)
  {
     if(x.coef > 0) s = s + " + " +   x.coef  +
  "x^" + x.exp;
     else if (x.coef < 0) s = s + " - " + (-
  x.coef) + "x^" + x.exp;}
    return s;
    }
public static void main(String[] args) {
        Polynomial zero = new Polynomial(0, 0);
 // 4x^3 + 3x^2 + 1
        Polynomial p1   = new Polynomial(4, 3);
        Polynomial p2   = new Polynomial(3, 2);
        Polynomial p3   = new Polynomial(1, 0);
        Polynomial p    = p1.plus(p2).plus(p3);
```

```
 // 3x^2 + 5
    Polynomial q1   = new Polynomial(3, 2);
    Polynomial q2   = new Polynomial(5, 0);
    Polynomial q    = q1.plus(q2);

    Polynomial r    = p.plus(q);
    Polynomial s    = p.multiply(q);
    System.out.println("zero(x) =     " +
   zero);
    System.out.println("p(x) =        " + p);
    System.out.println("q(x) =        " + q);
    System.out.println("p(x) + q(x) = " + r);
    System.out.println("p(x) * q(x) = " + s);
     }
}
```

List

- List Interface trong Java kế thừa Collection và khai báo các hành vi của một collection mà lưu giữ một dãy các phần tử.

```
List listA = new ArrayList();
List listB = new LinkedList();
List listC = new Vector();
listA.add("element 1");
listA.add("element 2");
listA.add(0, "element 0");
String element1 = listA.get(1);
Collections.sort(listA);
```

- Truy xuất qua Iterator

```
Iterator iterator = listA.iterator();
while(iterator.hasNext(){
      String element = (String)
  iterator.next(); }
```

- Truy xuất qua (for-loop)

```
for(Object object : listA) {
String element =(String) object; }
```

- Object remove(int index)

```
List list = new ArrayList();
list.add("object 1");
list.add("object 2");
list.add("object 3");
list.remove(1);
int size = list.size();
list.clear();
```

- List<MyObject> list = new ArrayList<MyObject>();

Chuyển đổi ArrayList và Array

- Để chuyển đổi ArrayList thành Array và ngược lại trong Java thay vì bạn phải viết vòng lập để chuyển đổi thì Java có cung cấp hai phương thức để convert ArrayList to Array hoặc convert Array to ArrayList.
- Khi convert **ArrayList to Array** các bạn sử dụng phương thức **toArray()** và ngược lại khi convert **Array to ArrayList** thì sử dụng phương thức **asList()**.

toArray(): ArrayList to Array

```
List<String> list = new ArrayList();
    // thêm lần lượt 4 phần tử.
    list.add("Hoc Vien");
    list.add("Cong Nghe");
    list.add("Buu Chinh");
    list.add("Vien Thong");
    //convert listString tới array.
String[] array = list.toArray(new
  String[list.size()]);
    System.out.println("\n
  "+Arrays.toString(array));
```

asList() : Array to ArrayList

- Arrays.asList(T... a)

```
List<String> listNames =
  Arrays.asList("John", "Peter",
  "Tom", "Mary");
List<Integer> listNumbers =
  Arrays.asList(1, 3, 5, 7, 9, 2,
  4, 6, 8);
System.out.println(listNames);
System.out.println(listNumbers);
```

Sử dụng Iterator

- Ta muốn duyệt các phần tử trong một tập hợp. Cách đơn giản là dùng một Iterator
- Iterator: ý tưởng là liệt kê mỗi phần tử của 1 collection
- 3 phương thức sau: boolean hasNext(); Object next() ; void remove() ;

```
List<Number> linkedNumbers = new
  LinkedList<>();
linkedNumbers.add(new Integer(123));
linkedNumbers.add(new Float(3.1415));
linkedNumbers.add(new Double(299.988));
Iterator<Number> iterator =
  linkedNumbers.listIterator();
while (iterator.hasNext()) {
    System.out.println(iterator.next());
}
```