1. **Tạo project trong java.**

* chọn “File” -> “New” -> “Java Project” -> đặt tên cho project
* trong phần source code của project “src” nhấp chuột phải để tùy chọn tạo class hay package,...

1. **Cấu trúc cơ bản của một chương trình java.**

|  |
| --- |
| **class** Hello\_World  {  **public** **static** **void** main(String args[])  {  System.***out***.print("Nguyen Tan Dung");  }  } |

1. **Chuỗi kí tự và số.**

* Trong java chúng ta có thể sử dụng các giá trị cố định chúng được gọi là “**HẰNG KÝ TỰ”.**
* Số nguyên(int).
  + Không có phần thập phân.
* Số thực dấu phẩy động độ chính xác kép(double).
  + Số thực phẩy động là các số có phần thập phân.
    - (số thực dấu phẩy động còn có thể sử dụng float nhưng trong thực tế thì nó không hiệu quả bằng double, vì thế chúng ta sẽ sử dụng double từ đây về sau.)
* Chuỗi ký tự
  + Trong java, các ký tự nằm trong cặp dấu ngoặc kép được gọi là các chuỗi ký tự.

1. **Cấu trúc in số và ký tự.**

* Chúng ta sử dụng câu lệnh

|  |
| --- |
| System.***out***.println("Nguyen Tan Dung"); |

* + System: là một lớp trong java đóng gói các hàm chức năng xử lí dữ liệu trong đó có xử lí dữ liệu vào ra từ bàn phím và màn hình.
  + out: trường đại diện cho dòng đầu ra của hệ thống (trường hợp này là màn hình).
  + println: là câu lệnh in ra chuỗi kí tự sau đó thực hiện xuống dòng.
* In nhiều biến đồng thời:

|  |
| --- |
| **int** A = 10;  **double** B;  B = 10.1;  String C = "nguyentandung";  System.***out***.println("nguyentandung" + A); |

1. **Chú thích trong java.**

Trong khi lập trình đôi khi chúng ta sẽ phải chú thích ý nghĩa cho những dòng code ấy.

* Chú thích trên 1 hàng: chúng ta sử dụng // ... ( sau dấu // chúng ta có thể viết dòng chú thích của mình trên cùng một hàng.).
* Chú thích xuống dòng: chúng ta có thể sử dụng cặp/\* ... \*/ ở đầu và cuối của đoạn chúng ta chú thích.

Khi chương trình được biên dịch nó sẽ bỏ qua toàn bộ những đoạn được comment.

1. **Biến trong java.**

* Khai báo biến:

|  |
| --- |
| **int** A;  **double** B;  String C; |

* Gán giá trị cho biến:

|  |
| --- |
| **int** A = 10;  **double** B;  B = 10.1;  String C = "nguyentandung"; |

1. **Các kiểu dữ liệu trong java.**

* Kiểu int – số nguyên không có phần thập phân.
* kiểu Double – số dấu phẩy động có phần thập phân.
* kiểu char – ký tự.
* boolean – chỉ có hai giá trị là true và false hoặc 0 và 1.
  + “boolean a = true;”
* String – chuỗi kí tự.
  + Truy cặp chuỗi kí tự trong java
    - String s = “12345”;
    - System.out.println(s.charAt(2));

ép kiểu dữ liệu trong java.

* + Nối hai chuỗi trong java.
    - Sử dụng phương thức chuẩn thư viện concat(chuỗi\_ cần nối);
    - text1 = text1.concat(text2);
    - Chúng ta có thể sử dụng toán tử “+” để nối hai chuỗi lại với nhau.
  + So sánh hai chuỗi có bằng nhau hay không.
    - Boolean result = s1.equals(s2);
  + Thay thế một kí tự trong chuỗi.
    - ta phải chuyển chuỗi đó thành một mảng kí tự rồi mới xử lí được.
* chúng ta chỉ có thể ép kiểu dữ liệu số sang số, kí tự hoặc ngược lại.
  + int s = 10;
  + char c = (int)s;
  + double s = 11.1;
  + int a = (int)s;
  + ...

1. **Toán tử số học.**

* +: cộng.
* -: trừ.
* \*: nhân.
* /: chia.
* %: chia lấy phần dư.
* toán tử ++.
  + int number = 1;
  + ++number;
  + ( điều này tương đương number = number + 1;)
* toán tử --.
  + int number = 1;
  + --number;
  + ( điều này tương đương number = number - 1;)

1. **Nhập đầu vào từ bàn phím trong java.**

|  |
| --- |
| **package** Hello\_World;  **import** java.util.Scanner;  **class** Hello\_World  {  **public** **static** **void** main(String args[])  {  Scanner input = **new** Scanner(System.***in***);  **int** age = input.nextInt();  System.***out***.println(age);  input.close();  }  } |

* Nhập lớp Scanner.
  + “import java.util.Scanner” -> câu lệnh này nhập lớp Scanner vào chương trình, lớp này được sử dụng để lấy các giá trị đầu vào trong java.
* Tạo đối tượng Scanner.
  + “Scanner input = new Scanner(System.in);” -> đoạn code này có tác dụng tạo ra một đối tượng của lớp Scanner.
* Nhập đầu vào từ người dùng.
  + Số nguyên:
    - “int age = intput.nextInt();” -> đoạn code này có tác dụng lấy thông tin từ người dùng nhập vào (trong trường hợp này là số nguyên) rồi gán vào cho biến “age”.
  + Số thực dấu phẩy động:
    - “double age = input.nextDouble();” -> đoạn code này có tác dụng lấy thông tin người dùng nhập vào(trong trường hợp này là số thực dấu phẩy động) rồi gán vào biến “age”.
* Đóng Scanner.
  + “input.close();” -> bước này không bắt buộc nhưng nó là một điều cần thiết khi lập trình.
* Nhập int -> input.nextInt();
* Nhập double-> input.nextDouble();
* Nhập String ->
  + input.next(); : cú pháp này kết thục khi nhận khoảng trắng.
  + input.nextLint(); : cú pháp này kết thúc khi chúng ta xuống hàng.
* Trong thời gian

1. **Các toán tử so sánh trong java.**

* a == b: kiểm tra xem a có bằng b không.
* a != b: kiểm tra xem a có khác b không.
* a > b: kiểm tra xem a có lớn hơn b không.
* a >= b: kiểm tra xem a có lớn hơn hoặc bằng b không.
* a < b: kiểm tra xem a có nhỏ hơn b không.
* a <= b: kiểm tra xem a có nhỏ hơn hoặc bằng b không.

1. **Random in Java.**

* **Max = y;**
* **Min = x;**

int randomNumber = random.nextInt(y - x + 1) + x;

1. **Toán tử logic trong java.**

* a && b: true nếu cả a và b đều true.
* a || b: true nếu a hoặc b là true.
* ! (a): true nếu a là false.
* Toán tử 3 ngôi:
  + String result = (a > b) ? “Pass” : “Fail”;.

Nếu như điều kiện (a > b) là đúng thì “Pass” và ngược lại.

1. **vòng lặp**

* for:
* while:
* do...while:

1. **câu lệnh**

* if:
* else:
* break:
* continue:
* switch: câu lệnh switch này có tác dụng giống như if else nhưng nó chỉ được sử dụng với những trường hợp cụ thể.

1. **Mảng.**

* Khai báo mảng trong java:
  + int[] A = {1, 2, 3, 4};
  + int[] A = new int[4];
  + int A[] = {1, 2, 3 ,4};

1. **Một số hàm có sẵn tích hợp trong java.**

* Math.pow(a, b): hàm này được sử đụng để có thể tính lũy thừa b của a.
  + Đối với những hàm liên quan đến toán học thì chúng ta cần import thư viện “import java.lang.Math;”

(int a = Math.pow(1, 2);)

* (kiểu dữ liệu)(Math.random() \* a + b): hàm này có tác dụng trả về một số ran dom trong đoạn từ b đến a với kiểu dữ liệu đã được chọn.

1. **Phương thức trong java.**

* Bất kì phương thức nào cũng được tạo ở bên trong một lớp nhất định. Vì vậy, khi chúng ta sử dụng một phương thức chúng ta cần phải tạo đối tượng của lớp đó rồi dùng đối tượng đó để truy cặp vào phương thức cần sử dụng.

|  |
| --- |
| **// Replace \_\_\_ with your code below**  **class Main {**  **// create printNumbers method**  **void printNumbers() {**  **// print 5**  **System.out.println(5);**  **// print 100**  **System.out.println(100);**  **}**  **public static void main(String[] args) {**  **// creating object of Main class**  **Main obj = new Main();**    **// call printNumbers()**  **obj.printNumbers();**    **// call printNumbers() again**  **obj.printNumbers();**  **}**  **}** |

* Hàm main cũng là một phương thức trong java.
* Phương thức thư viện chuẩn:
  + là những phương thức chúng ta chưa định nghĩa gì cho nó nhưng chúng ta vẫn có thể sử dụng nó
    - ví dụ:

Scanner input = new Scanner(System.in);

int a = input.nextInt();

->nextInt là một phương thức thư viện chuẩn trong java.

1. **Biến.**

* Biến cục bộ: bất kì một biến nào được tạo ở trong phương thức đều được gọi là biến cục bộ.

giả sử biến cục bộ a nằm trong phương thức A thì sau khi A thực hiện xong thì biến cục bộ a bị hủy hoàn toàn.

1. **Sự khác nhau giữa double và float.**

* double: 8 byte.
* float: 4 byte.

**LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG (OOP)**

1. **Tạo một lớp trong java.**

* class Tên\_Class

{

//..............

}

ví dụ:

class A

{

}

1. **Tạo một đối tượng trong java.**

* Tên\_Class Tên\_đối\_tượng = new Tên\_Class();

A a = new A();

1. **Trường bên trong class.**

* Mỗi class đều có các trường khác nhau

Có hai trường bên trong class:

* + Trường biến:

|  |
| --- |
| class Student {    // add fields  String name = "Jackie";  int score = 40;    public static void main(String[] args) {    // create object of Student  Student student1 = new Student();    // access fields using object  System.out.println("Name: " + student1.name);  System.out.println("Score: " + student1.score);  }  } |

* + Trường phương thức.

|  |
| --- |
| class Student {    // add method  void checkPassFail(int score) {  if (score >= 35) {  System.out.println("Passed");  }  else {  System.out.println("Failed");  }  }  } |

1. **Hàm tạo trong java.(constructor)**

* Trong java hàm tạo hay hàm khởi tạo trong java tương tự như các phương thức, nhưng nó không có kiểu trả về và có cùng tên với lớp.

|  |
| --- |
| class Student {    // constructor  Student() {  System.out.println("Calling Constructor");  }  }  class Main {  public static void main(String[] args) {    // create an object of Student  // this calls the constructor  Student student1 = new Student();    }  } |

* Hàm khởi tạo mặc định:
  + Nếu chúng ta không tạo bất kì hàm khởi tạo nào thì java tự động tạo hàm khởi tạo mặc định cho nó. (hàm này là một hàm khởi tạo không có đối số.)
* Hàm khởi tạo không có tham số:
  + Hàm khởi tạo không có tham số là hàm khởi tạo do chúng ta tự định nghĩa, và đặc biệt là nó không nhận bất kì tham số nào.
* Hàm khởi tạo có đối số:
  + Gioongs như các hàm khởi tạo khác thì hàm khởi tạo có đối số chỉ khác một chỗ là nó có sử dụng đối số ở đầu vào.
  + Khi chúng ta sử dụng hàm khởi tạo có đối số, nhưng không hề khai báo hàm khởi tạo không có đối số thì nếu như chúng ta khai báo như sau chương trình sẽ bị lỗi:”Main A = new Main();” bởi vì bây giờ hàm khởi tạo mặc định sẽ không được áp dụng vì nó chỉ được áp dụng khi không có hàm khởi tạo nào trong class.

1. **Từ khóa “this”.**

* Khi chúng ta sử dụng từ khóa this trong bất kì phương thức nào hay hàm khởi tạo của một lớp bất kì thì từ khóa “this” này đang tham chiếu đến lớp đó.

Vd:

|  |
| --- |
| class Student {    String name;    void displayName() {  System.out.println("Name using this: " + this.name);  }    public static void main(String[] args) {  Student student1 = new Student();    student1.name = "Rosie”;  student1.displayName();    // print name using object  System.out.println("Name using object: " + student1.name);  }  } |

từ khóa “this” trong ví dụ trên được sử dụng trong hàm displayName(); và hàm displayName() này nằm trong class Student, vì thế từ khóa this này đang tham chiếu tới đối tượng student này.

1. **Các chỉ định truy cập.**

* Public: cho phép truy cập từ các lớp khác.
* Private: ngăn chặn truy cập từ các lớp khác
  + Vì không thể truy cặp cũng như thay đổi từ những lớp khác nên chúng ta có thể sử dụng hai phương thức getter và setter ở chỉ định truy cập public để lấy và cũng như sửa đổi đối với các trường biến private.

1. **Tính kế thừa.**

* n lớp được gọi là kế thừa của 1 lớp A nếu như n lớp này có các trường giống nhau và các trường giống nhau này được đặt trong lớp A. (n > 1).
* Cú pháp của một lớp kế thừa một lớp.

|  |
| --- |
| //parent class  class Animal {  void eat() {  System.out.println("I can eat.");  }  }  //the Dog class is derived from Animal  class Dog extends Animal {  void bark() {  System.out.println("I can bark.");  }  } |

trong ví dụ trên thì lớp Dog đang kế thừa lớp Animal.

Bây giờ lớp Dog có thể kế thừa tất cả các trường của lớp Animals.

* Mục đích của tính kế thừa: giảm thiểu code, dễ sửa đổi, hạn chế lỗi.

1. **Ghi đè phương thức.**

* Trong quá trình kế thừa lớp con kế thừa lớp cha (kế thừa cả các phương thức). Trong lớp con chúng ta vẫn có thể tạo ra một phương thức với tên giống như phương thức trong lớp cha và sửa đổi theo ý của chúng ta. Khi chúng ta tạo ra đối tượng thuộc lớp con và gọi đến phương thức này thì chương trình sẽ hiểu phương thức đó là phương thức chúng ta vừa định nghĩa.

1. **Từ khóa super.**

* Trong quá trình kế thừa của lớp con với lớp cha, lớp con sẽ ghi đè phương thức của lớp cha. Gỉa sử chúng ta muốn sử dụng lại phương thức của lớp cha thì chúng ta có thể sử dụng từ khóa super.

|  |
| --- |
| class Polygon {  void displayInfo() {  System.out.println("A polygon is a two dimensional shape with straight lines.");  }    int getPerimeter(int[] sides) {  int perimeter = 0;  // find sum of all sides  for (int side : sides) {  perimeter = perimeter + side;  }    return perimeter;  }  }    class Triangle extends Polygon {  void displayInfo() {  super.displayInfo();  System.out.println("A triangle is a polygon with 3 sides.");  }  }    class Main {  public static void main(String[] args) {    // create an object of Triangle  Triangle t1 = new Triangle();    // sides of triangle  int[] triangleSides = {8, 5, 11};    // call getPerimeter using t1  int perimeter = t1.getPerimeter(triangleSides);  System.out.println("Triangle Perimeter: " + perimeter);    // call displayInfo() method  t1.displayInfo();  }  } |

* Biến thể khác của từ khóa super là có thể gọi cả phương thức khởi tạo của lớp cha.

|  |
| --- |
| class Person {    void display() {  System.out.println("I am a person");  }  }    class Student extends Person {    void display() {    // call  super.display();  System.out.println("I am a student");  }  }    class Main {  public static void main(String[] args) {    Student s1 = new Student();  s1.display();  }  } |

* từ khóa super còn được sử dụng để truy cập trường thuộc tính.

|  |
| --- |
| //replace \_\_\_ with your code  class Person {  // info field inside Person  String info = "Person";  }  //create a Student class inheriting from Person  class Student extends Person {  // create the field with same name as of the parent class  String info = "Student";  // create a method named getSuperInfo()  String getSuperInfo() {  // use super keyword to access the info field of Person  return super.info;  }  }  class Main {  public static void main(String[] args) {  // create an object of Student named student1  Student student1 = new Student();  // use student1 to access the info field of Student  System.out.println(student1.info);  // call the getSuperInfo() method using student1  System.out.println(student1.getSuperInfo());  }  } |

1. **Lớp trừu tường và phương thức trừu tượng.**

* Cả lớp trừu tượng và phương thức trừu tượng đều bắt đầu bằng từ khóa abstract.
* phương thức trừu tượng thay {} bằng dấu;. => Phần thân của phương thức trừu tượng không có gì cả.
* Chúng ta không thể nào tạo một đối tượng của lớp trừu tượng.
* Lớp trừu tượng:

|  |
| --- |
| abstract class Polygon {  // code inside the class  }    class Main {  public static void main(String[] args) {    // create object of Polygon  Polygon obj = new Polygon();  }  } |

* + Chúng ta không thể tạo đối tượng thuộc lớp trừu tượng, vì thế chúng ta phải tạo các lớp con kế thừa lớp trừu tượng.
  + Trong lớp trừu tượng có cả phương thức trừu tượng và phương thức không trừu tượng. Vì thế khi tạo lớp con chúng ta phải ghi đè và triển khai code hết toàn bộ các lớp trừu tượng ở bên trong lớp trừu tượng.
* Phương thức trừu tượng:

|  |
| --- |
| class Polygon {  // code inside the class  abstract void display();  }    class Main {  public static void main(String[] args) {    // create object of Polygon  Polygon obj = new Polygon();  }  } |

* Mục đích của các lớp này là nhằm để cho các lớp con tự triển khai theo ý thích của mình. Gỉa sử chúng ta có các lớp đa giác như tam giác, hình vuông, chữ nhât,... mỗi lớp đều có công thức tính diện tích, chu vi riêng nên là điều này mới được sinh ra.

1. **Interface.**

* Đối với abstract đây là một lớp bán trừu tượng vì nó bao gồm cả hai phương thức trừu tượng và không trừu tượng. Trong java chúng ta có thể tạo một lớp hoàn toàn trừu tượng, lớp này được gọi là một lớp interface. Trong interface này nó chỉ có thể khai báo những trừu tượng.
* Gioongs như lớp abstract chúng ta cũng không thể nào khởi tạo một đối tượng cho interface mà nhờ những lớp con làm điều đó.
* Khác với abstract và kế thừa, các lớp con sử dụng từ khóa “implements” thay vì “extends”.

|  |
| --- |
| interface Animal {    abstract public void play();  abstract public void makeSound();  }    class Dog implements Animal {    // implement play()  public void play() {  System.out.println("I play ball fetch");  }    // implement makeSound()  public void makeSound() {  System.out.println("Woof Woof");  }  } |

1. **Tính trừu tượng.**

* Như chúng ta thấy, thì tính trừu tượng rất là quan trọng vì nó góp phần đặt ra những quy tắc, những tiêu chuẩn để cho chúng ta tuân theo. Nó giúp cho chúng ta giám bớt sự phức tạp của cả chương trình: giúp ẩn những thông tin không cần thiết, chỉ hiển thị những thông tin cần thiết.

1. **Tính đa hình.**

* Tính đa hình được hiểu là việc cơ động trong việc một hình thái có thể thực hiện được nhiều tác vụ khác nhau.
* Nạp chồng phương thức.

1. **Tính đóng gói.**

* Trong một lớp chúng ta phải bao gồm tất cả thông tin của nó các trường thuộc tính và trường phương thức.
* sử dụng private để có thể ẩn đi dữ liệu => tính quan trọng của tính đóng gói.

|  |
| --- |
| class Rectangle {    // fields to calculate area  private int length;  private int breadth;  } |

1. **Toán tử instanceof. (kiểm tra xem đối tượng này có phải thuộc lớp nào đó hay không).**

* Toán tử này được sử dụng để kiểm tra xem một đối tượng có phải là đối tượng của một lớp cụ thể hay không.

Main obj = new Main();

boolean result = obj instanceof Main;

đoạn code trên kiểm tra xem đối tượng obj có phải là một đối tượng của lớp Main hay không.

* Toán tử này còn giúp cho chúng ta kiểm tra xem một lớp con có phải là lớp con của lớp cha không.

|  |
| --- |
| **class Animal {**    **}**    **// inherits the Animal class**  **class Dog extends Animal {**    **}**    **class Main {**  **public static void main(String[] args) {**    **// create an object of the subclass**  **Dog dog = new Dog();**    **// check if dog is an instance of the subclass**  **System.out.println(dog instanceof Dog); // true**    **// check if dog is an instance of the superclass**  **System.out.println(dog instanceof Animal); // true**  **}**  **}** |

1. **Từ khóa Static.**

* Thông thường chúng ta sẽ tạo một đối tượng của một lớp và sử dụng đối tượng này để truy cập vào các trường của lớp đó.
* khi chúng ta sử dụng từ khóa static trước một trường thì chúng ta không cần phải tạo đối tượng của lớp đó truy cập vì lúc này trường này thuộc về lớp đó chứ không còn thuộc về đối tượng.

|  |
| --- |
| class Company {    static String name;  }    class Main {    public static void main(String[] args) {    Company.name = "Programiz";  }  } |

1. **Từ khóa final.**

* Sử dụng với:
  + Biến: chúng ta không thể thay đổi giá trị của biến.
  + Phương thức: chúng ta không thể ghi đè.
  + Lớp: chúng ta không thể kế thừa.

Nếu những điều cắm mà được thực hiện thì chương trình sẽ báo lỗi.

**JAVA NÂNG CAO**

1. **Framework Collections**

* Framework Collections cung cấp nhiều lớp và interface để triển khai những kiểu cấu trúc dữ liệu khác nhau.
  + ArrayList:
    - Mảng thông thường chúng ta chỉ có thể tạo ra số phần tử cố định cho nó. Đối với ArrayList nó như một mạng động, có thể thêm hoặc xóa các phần tử.
    - Cú pháp:
      * Khai báo thư viện: “import java.util.ArrayList;
      * Khởi tạo đối tượng: “ArrayList<Kiểu dữ liệu> names = new ArrayList<>();”
        + chuỗi: String.
        + Số nguyên: Interger.
        + Số thực dấu chấm động: Double.
        + char: Character.
    - Các phương thức để thao tác với ArrayList.
      * Thêm phần tử vào ArrayList.(Cú pháp này sẽ thêm phần tử vào vị trí cuối Arraylist.

names.add(Value);

* + - * Thêm phần tử tại vị trí xác định trong ArrayList.

names.add(position, value);

* + - * Lấy giá trị của một phần tử tại vị trí xác đinh.

names.get(position);

* + - * Thay đổi giá trị của một phần tử.

names.set(position, Value); position là vị trí cần thay đổi, value là giá trị thay đổi cho phần tử đó.

* + - * Xóa phần tử ra khỏi ArrayList.

names.remove(position);

* + - * Sắp xếp phần tử theo cách mong muốn.

|  |
| --- |
| **package** Hello\_World;  **import** java.util.Scanner;  **import** java.util.ArrayList;  **import** java.util.Collections;  **import** java.util.Comparator;  **class** Hello\_World  {  **public** **static** **void** main(String args[])  {  ArrayList<Integer> Dynamic\_Array = **new** ArrayList<>();  **int** Number\_Of\_Elements;  Scanner input = **new** Scanner(System.***in***);  Number\_Of\_Elements = input.nextInt();  **for** (**int** i = 0; i < Number\_Of\_Elements; i++)  {  **int** value = input.nextInt();  Dynamic\_Array.add(value);  }  Collections.*sort*(Dynamic\_Array, **new** Comparator<Integer>()  {  **public** **int** compare(Integer a, Integer b)  {  **if** (a > b)  {  **return** -1;  }  **return** 1;  }  // a đứng trước b nếu hàm này trả về -1  });  System.***out***.println(Dynamic\_Array);  }  } |

* + LinkedList:
    - Lớp LinkeList trong java dùng để tạo một danh sách liên kết đôi.
    - Một thành phần trong lInkeList chứa 3 trường.
      * Prev: Lưu trữ địa chỉa của phần tử tước đó. Bằng “null” đối với phần tử đầu tiên.
      * Next: Lưu trữ địa chỉ của phần tử tiếp theo. Bằng “null” đối với phần tử cuối cùng.
      * Data: lưu trữ dữ liệu của nút.
    - Cú pháp: giống như ArrayList nhưng chỉ cần thay đổi “ArrayList” thành “LinkedList”.
    - Các phương thức khác của LinkedList cũng giống với ArrayList.
  + Iterator:
    - được sử dụng để duyệt qua các phần tử ở trong các dữ liệu mà không thể duyệt bằng cách thông thường như mảng.
    - Cú pháp ví dụ:

|  |
| --- |
| HashSet<Integer> HS = new HashSet<>();  Iterator<Integer> iterator = HS.iterator(); // khởi tạo một iterator để liên kết với HashSet. |

* + - Duyệt qua các phần tử.

|  |
| --- |
| **while** (**true**) {  **if** (iterator.hasNext() == **false**)  {  **break**;  }  **int** x = iterator.next();  System.***out***.println(x);  } |

Chúng ta hãy tưởng tượng các cấu trúc này lưu dưới dạng như sau:

* + - * Ô đầu tiên: chứa địa chỉ của cả cấu trúc dữ liệu đó và mang giá trị null.
      * Ô cuối cùng: luôn mang giá trị null để có thể thêm phần tử vào. Thêm vào rồi thì lại tăng thêm 1 ô cuối và mang giá tị null để có thể thêm phần tử khác vào.
      * Các ô ở giữa là lưu các giá trị.
      * Cú pháp iterator.hasNext() dùng để kiểm tra xem phần tử tiếp theo có tồn tại hay không.
      * Cú pháp iterator.next() có tác dụng trả về giá trị của phần tử tiếp theo.
    - Xóa phần tử.

|  |
| --- |
| **while** (**true**) {  **if** (iterator.hasNext() == **false**)  {  **break**;  }  **int** x = iterator.next();  **if** (x == 10)  {  iterator.remove();  }  } |

* + HashSet:
    - Cấu trúc dữ liệu này giống như Set trong C++.
    - Các giá trị trong Set chỉ được lưu một lần duy nhất.
    - Set sẽ lưu các giá trị theo thứ tự tăng dần.
  + HashMap:
    - Tạo ra một danh sách mà mỗi phần tử trong danh sách đó gồm hai thứ: giá trị và khóa.
    - Cú pháp:

HashMap<Integer, String> HM = new HashMap<>();

* + - Thêm một cặp khóa vào.
      * HM.put(1, “abc”);
    - Lấy giá trị tương ứng khóa.
      * String a = HM.get(1);
    - xóa một cặp khóa.
      * HM.remove(1);
    - duyệt và sử lý HashMap.

|  |
| --- |
| HashMap<Integer, String> HM = **new** HashMap<>();  HM.put(1, "Nguyen");  HM.put(2, "Han");  Iterator<HashMap.Entry<Integer, String> > iterator = HM.entrySet().iterator();  **while**(**true**)  {  **if**(iterator.hasNext() == **false**)  {  **break**;  }  HashMap.Entry<Integer, String> entry = iterator.next();  System.***out***.println(entry.getKey() + " " + entry.getValue());  } |

* Thuật toán binarySearch().

|  |
| --- |
| **package** Hello\_World;  **import** java.util.Scanner;  **import** java.util.HashSet;  **import** java.util.ArrayList;  **import** java.util.Collections;  **import** java.util.Comparator;  **import** java.util.LinkedList;  **import** java.util.Map;  **import** java.util.Iterator;  **import** java.util.HashMap;  **class** Hello\_World  {  **public** **static** **void** main(String args[])  {  ArrayList<Integer> AL = **new** ArrayList<>();  AL.add(12);  AL.add(34);  AL.add(67);  AL.add(100);  Scanner input = **new** Scanner(System.***in***);  **int** Element = input.nextInt();  System.***out***.println(Collections.*binarySearch*(AL,Element));  input.close();  }  } |

* Đảo ngược vị trí của các phần tử “Collections.reverse(tên của Collection cần đảo ngược);”
* Đổi chỗ hai phần tử: “Collections.swap(tên collection, vị trí 1, vị trí 2).

**XỬ LÍ NGOẠI LỆ TRONG JAVA.**

* Khi một đoạn code bị lỗi mà chúng ta vẫn muốn chương trình chạy tiếp tục và bỏ qua lỗi đó thì chúng ta sẽ phải sử dụng xử lí ngoại lệ.
* Cú pháp:

|  |
| --- |
| try  {  // đoạn mã mà chúng ta cho là người dùng có thể sử dụng sai cách hoặc là chúng ta có thể code sai cách.  }  catch (Exception e)  {  // nếu như mã code ở trong phần try bị lỗi thì ngay lập tức sẽ chuyển đến khối mã của //phần catch này và đến những câu lệnh ở ngoài khối mã try catch.  }  finally  { // trong phần này code có tác dụng thực thi khi mà khối try có bị lỗi hay không. (Riêng cá nhân thì thấy không cần thiết bị nếu như chúng ta không sử dụng khối finally cũng chả sao.)  } |

* Lỗi do hệ thống định nghĩa: trong phần catch chúng ta sử dụng biến e – “System.out.println(e);”.
* Lỗi do chúng ta định nghĩa: trong phần catch chúng ta sử dụng bất kì điều gì ta muốn.

**Throw và Throws.**

* Từ khóa Throw được sử dụng để ném ra một ngoại lệ. (xuất ra ngoại lệ)
* Từ khóa Throws được sử dụng để thông báo một phương thức có khả năng xuất hiện ngoại lệ. (thông báo như thế để chúng ta dễ dàng nhận biết và nên xử lí try catch và lưu ý đến những phương thức này)
  + Trong một phương thức có thể khai báo một throws với nhiều kiểu ngoại lệ khác nhau.
* Chúng ta cũng có thể tự định nghĩa một Exception riêng.

**FILE TRONG JAVA**

1. **Ba bước làm việc với file.**

* kết nối với file.
* thực hiện thao tác với file.
* đóng file.

1. **Phải làm việc file với khối try...catch.**
2. **Thư viện sử dụng cho file**

“java.io.File;”

1. **Thao tác với File.**

* Tạo File: chúng ta có thể tạo một File trong máy tính bằng chương trình java.

Ví dụ tạo một File test.docx bên trong đường dẫn cụ thể.

|  |
| --- |
| File file = new File("C:\\Users\\user\\Downloads\\java\\Test chương trình\\test.docx");  // trying to create a file  try {  boolean value = file.createNewFile();  // check if file is created  if (value) {  System.*out*.println("The new file is created.");  }  else {  System.*out*.println("The file already exists.");  }  }  catch (Exception e) {  System.*out*.println(e);  } |

* + Cú pháp: “File file = new File(“....”); // tạo ra một đối tượng file với đối số là “...”, lúc này file vẫn chưa được tạo ra.
  + Sau đó chúng ta phải sử dụng một khối lệnh try catch để vừa tạo vừa kiểm tra xem “...” đã tồn tại trên thiết bị chưa.
  + file.createNewFile(); trả về false nếu như “...” đã tồn tại trên máy. True là file không tồn tại và đã được tạo.
* Đọc File:

|  |
| --- |
| ArrayList<Character> fileContents = new ArrayList<>();  try {  FileReader fileReader = new FileReader("C:\\Users\\user\\Downloads\\java-code\\Hello\_World\\src\\Hello\_World\\input.INP");  while (true)  {  int value = fileReader.read();  if (value == -1)  {  break;  }  fileContents.add((char)value);  }  fileReader.close();  } catch (Exception e) {  System.***out***.println(e);  }  // Print the contents from the ArrayList  for (int i = 0; i < fileContents.size(); i++)  {  System.***out***.print(fileContents.get(i));  } |

* + B1: tạo một ArrayList.
  + B2: tất cả quá trình xử lí file đều bỏ vào trong try..catch.
  + B3: khởi tạo một đối tượng của FileReader với thư viện (java.io.FileReader). Đối tượng này giúp cho chúng ta đọc file theo từng kí tự.
  + B4: đọc từng kí tự trong file. Chú ý rằng cú pháp khi truy cập vào phương thức “fileReader.read();” kí tự được trỏ tới sẽ tăng lên một đơn vị. Điều này tương đương với việc hasNext() trong Iterator.
  + B5: sau khi không đọc được nữa thì cần phải đóng đối tượng FileReader lại.
* Ghi nội dung vào File.

Để ghi nội dung vào File, chúng ta sử dụng phương thức write() của lớp FileWriter. (thư viện của thao tác này là “import java.io.FileWriter;”

* + Nếu ghi nội dung vào File không tồn tại, một File mới sẽ được tạo.
  + Nếu File đã tồn tại, nội dung cũ trong File này sẽ bị xóa và nội dung mới sẽ được thêm vào file.

|  |
| --- |
| ArrayList<Character> AL = new ArrayList<>();  AL.add('1');  AL.add('2');  try {  // craeate a Writer using FileWriter  FileWriter output = new FileWriter("C:\\Users\\user\\Downloads\\java-code\\Main\\src\\Main\\output.OUT");  // write string to the file  for (int i = 0; i < AL.size(); i++)  {  output.write(AL.get(i));  }  System.***out***.println("Data is written to the file.");  // close FileWriter  output.close();  }  catch (Exception e) {  System.***out***.println(e);  } |

* + - Để nội dung trong File vẫn được dữ nguyên mỗi khi ghi vào File chúng ta sẽ phải truyển tham số “true” cho đối tượng của FileWriter

|  |
| --- |
| FileWriter output = new FileWriter("C:\\Users\\user\\Downloads\\java-code\\Main\\src\\Main\\output.OUT", true); |

* Xóa File.
  + Chúng ta sử dụng phương thức delete() của File để xóa một File.

|  |
| --- |
| File file = new File("message.txt");  try {  // delete the file  boolean value = file.delete();  if (value) {  System.out.println("The File is deleted.");  } else {  System.out.println("The File is not deleted.");  }  }  catch (Exception e) {  System.out.println(e);  } |

* Tạo một Folder trong java.

|  |
| --- |
| File file = new File ("C:\\Users\\user\\Downloads\\java\\Test chương trình\\test");  try  {  boolean x = file.mkdir();  if (x == true)  {  System.out.println("Directory is created successfully");  }  else  {  System.out.println("Error");  }  }  catch (Exception e)  {  System.out.println(e);  }  System.out.println("nguyentandung"); |

* Sao chép một file sang file khác.
  + Đọc nội dung từ File sau đó lại ghii nội dung đó vào File khác.
* Xuất ra toàn bộ file trong một folder.

|  |
| --- |
| File file = new File ("C:\\Users\\user\\Downloads\\java\\Test chương trình");  try  {  String AL[] = file.list();  for (int i = 0; i < AL.length; i++)  {  System.out.println(AL[i]);  }  }  catch (Exception e)  {  System.out.println(e);  }  System.out.println("nguyentandung"); |

* Đổi tên file.

Chương trình bên dưới là đổi tên của file sang ten của file1. (file 1 sẽ không được tạo ra)

|  |
| --- |
| File file = new File ("C:\\Users\\user\\Downloads\\java\\Test chương trình\\abc.docx");  File file1 = new File ("C:\\Users\\user\\Downloads\\java\\Test chương trình\\abcd.docx");  try  {  boolean x = file.renameTo(file1);  }  catch (Exception e)  {  System.out.println(e);  }  System.out.println("nguyentandung"); |

**Gói do một người tự định nghĩa.**

Gỉa sử hiện tại chúng ta đang ở package Main1 và class Main1. Chúng ta sẽ tạo một package Main2 và class Main2 trong package Main2 và tạo một hàm void display(). Và chúng ta sẽ tạo một đối tượng của lớp Main2 trong lớp Main1 và sử dụng hàm display() vừa rồi ở ngay trong lớp Main1.

* Lớp Main2.

|  |
| --- |
| package Main2;  class Main2  {  public void display()  {  System.out.println(“Nguyen Tan Dung”); }  } |

* Lớp Main1.

|  |
| --- |
| package Main1;  import Main2.Main2; // Main2 đầu tieenn là tên gói, Main2 thứ hai là class.  class Main1  {  public static void main(String args[])  {  Main2 M = new Main2();  M.dislay();  }  } |

**Anotation**

Anotation phổ biến là override.

Xét ví dụ sau đây:

* Chúng ta tạo một class Animal và class Dog kế thừa class Animal.
* Gỉa sử class Animal có phương thức A và class Dog sẽ ghi đè phương thức A này của lớp Animal.
* Chúng ta sẽ sử dụng “@Override” ở ngay trên phương thức ghi đè của class Dog.

=> Nó có tác dụng là chỉ định phương thức này chính là phương thức ghi đè của class Animal. Nếu như chúng ta ghi sai cú pháp của phương thức A thì “@Override” sẽ thực hiện ngay chức năng của nó.

|  |
| --- |
| class Animal  {  public void display()  {  System.out.println("Animal");  }  }  class Dog extends Animal  {  @Override  public void display()  {  System.out.println("Dog");  }  }  public class Main {  public void display()  {  Dog d = new Dog();  d.display();  }        } |

**Lớp lồng lớp**

Gỉa sử chúng ta có lớp A, lớp lồng lớp có nghĩa là có một lớp B ở bên trong lớp A.

Sau đây là cách thức truy cập vào lớp B và sử dụng các phương thức cũng như thuộc tính của nó.

|  |
| --- |
| class Song  {  class Artist  {  public void get(String name)  {  System.out.println(name);  }  }  }  class Main  {  public static void main(String args[])  {  Song s = new Song();  Song.Artist A = s.new Artist();  Scanner input = new Scanner(System.in);  String name = input.nextLine();  A.get(name);  input.close();  }  } |

**Lớp Generics.**

Java Generic cho phép chúng ta tạo một lớp làm việc với các đối tượng khác nhau. Dễ hình dung thì HashMap, ArrayList là các lớp Generic và chúng làm việc với các kiểu dữ liệu String, Integer, ...

|  |
| --- |
| class Generic<T>  {  public Generic(T data)  {  System.out.println(data);  }  }  class Main  {  public static void main(String args[])  {  Generic<Integer> Gn = new Generic<>(10);  }  } |

Phương thức Generic.

|  |
| --- |
| class Main  {  public <T> void display(T data)  {  System.out.println(data);  }  public static void main(String args[])  {  Main obj = new Main();  obj.<String>display("java");  }  } |

**BIỂU THỨC LAMBDA**

**AUTOBOXING VÀ UNBOXING.**

Sau đây chúng ta sẽ tìm hiểu về cách thức hoạt động của Arraylist tượng trưng cho các Collections.

Xét chương trình sau đây.

|  |
| --- |
| import java.util.ArrayList;    class Main {  public static void main(String[] args) {  ArrayList<Integer> list = new ArrayList<>();    // insert value to the list  // autoboxing  list.add(5);  list.add(6);  System.out.println("ArrayList: " + list);    // access value from arraylist  // unboxing  int number = list.get(1);  System.out.println("Value at index 1: " + number);  }  } |

* Chúng ta để ý thấy rằng lớp Arraylist sử dụng đối Integer thay vì int.
  + int được xem là một thành phần ở trong Integer. Integer giúp đóng gói một giá tị int thành một đối tượng và từ đó chúng ta có thể thực hiện các việc chuyển kiểu,...
* Cú pháp “list.add(5);”: tự động chuyển giá trị 5 (int) thành một đối tượng Integer.
  + - * Autoboxing.
* Cú pháp “int number = list.get(1);”: chúng ta chủ động chuyển đối tượng Integer (list.get(1)) thành kiểu int.
  + - * Unboxing.

**Lớp enum (Enumeration)**

* **Khái quát về lớp enum:** là một lớp dùng để định nghĩa một tập hợp các giá trị không thay đổi. Mỗi phần tử này cũng được xem là một class cùng tên với tên của lớp enum và có các đối tượng
  + Các phần tử được định nghĩa hằng trong enum đều ở dạng String.
* **Cú pháp:**

|  |
| --- |
| enum Month  {  January(31),  Febuary(30),  March(31);  private int Number\_Of\_Days;  Month(int Number\_Of\_Days)  {  this.Number\_Of\_Days = Number\_Of\_Days;  }  public void display() {  System.out.println(Number\_Of\_Days);  }  }  class Main  {  public static void main(String args[])  {  Scanner input = new Scanner(System.in);  String m = input.nextLine();  try  {  Month M = Month.valueOf(m);  M.display();  }  catch(Exception e)  {  System.out.println("Month is wrong");  }  }  } |

* + Như chúng ta thấy trong enum Month chúng ta có cài đặt sẵn 3 hằng số “Month” là January, Febuary and March.
  + **Cú pháp**

|  |
| --- |
| private int Number\_Of\_Days;  Month(int Number\_Of\_Days)  {  this.Number\_Of\_Days = Number\_Of\_Days;  }  public void display() {  System.out.println(Number\_Of\_Days);  } |

Đây là một thuộc tính, constructor, Method trong enum Month

Như chúng ta đã nêu ra ở phần khái quát ở phía trên thì các Attribute, Constructor and Method trong enum đều thuộc về các phần tử bên trong các phần tử.

* + Cú pháp:

|  |
| --- |
| Month M = Month.valueOf(m); |

cú pháp dùng để tạo ra một đối tượng M của class tương ứng với phần tử có giá trị bằng m.

* Truy cập vào một phần tử hằng trong enum.

|  |
| --- |
| enum Card  {  CLUB, DIAMOND, SPADE, HEART;  }  class Main  {  public static void main(String args[])  {  String CardType = Card.DIAMOND.name();  }  } |

**ÉP KIỂU NGẦM ĐỊNH TRONG JAVA.**

* Trong java chúng ta không có ép kiểu ngầm định như trong C++ và C# mà chúng ta phải sử dụng ép kiểu tường minh.
* Gỉa sử chúng ta muốn chuyển từ số nguyên sang phân số thì chúng ta chỉ cần khởi tạo một constructor, lúc đó chương trình sẽ tự động hiểu rằng 1 số nguyên có thể làm một phân số nhất định.
* Trong java chúng ta không thể làm như vậy mà phải thực hiện chuyển từ một số nguyên sang một phân số.

**TÍNH ĐA HÌNH TRONG JAVA.**

|  |
| --- |
| import java.util.Scanner;  class SinhVien  {      private String MSSV;      private String Ten;      public void input()      {          Scanner input = new Scanner(System.in);          System.out.println("Nhap mssv: ");          MSSV = input.nextLine();          System.out.println("Nhap Ten: ");          Ten = input.nextLine();      }      public void output()      {          System.out.println("mssv: " + MSSV);          System.out.println("ten: " + Ten);      }      public String display\_instance()      {          return "SinhVien";      }  }  class SinhVienTN extends SinhVien  {      private int HSLuong;      @Override      public void input()      {          super.input();          System.out.println("HSLuong: ");          Scanner input = new Scanner(System.in);          this.HSLuong = input.nextInt();      }      public void output()      {          super.output();          System.out.println("HSLuong: " + HSLuong);      }      @Override      public String display\_instance()      {          return "Sinhvien TN";      }  }  class Hello\_World  {      public static void main(String[] args)      {          Scanner input = new Scanner(System.in);          SinhVien[] sv = new SinhVien[3];          for (int i = 0; i < 3; i++)          {              System.out.println("Nhap Loai sinh vien: ");              String s = input.nextLine();              if (s.equals("TN"))              {                  sv[i] = new SinhVienTN();                  sv[i].input();              }              else              {                  sv[i] = new SinhVien();                  sv[i].input();              }          }          for (int i = 0; i < 3; i++)          {              if (sv[i].display\_instance().equals("SinhVien"))              {                  System.out.println("Sinhvien");              }              else if (sv[i].display\_instance().equals("Sinhvien TN"))              {                  System.out.println("SinhvienTN");              }          }          input.close();      }  } |

**Java Date.**

* Lớp java Date cung cấp một số thứ về xử lí thời gian ngày tháng.
* Trong phần này tôi sẽ không xử lí nhiều mà chỉ nêu ra cách lấy các thông tin hữu ích như sau:
  + Ngày.
  + Tháng.
  + Năm.
  + giờ.
  + Phút.
  + giây.

|  |
| --- |
| import java.util.Calendar;  public class Hello\_World {      public static void main(String[] args) {          Calendar calendar = Calendar.getInstance(); //dòng này là tạo một thể hiện của lớp Calendar về thời gian hiện tại.          // Lấy thông tin về ngày, tháng, năm và thứ          int ngay = calendar.get(Calendar.DAY\_OF\_MONTH);          int thang = calendar.get(Calendar.MONTH) + 1; // Tháng được đếm từ 0, nên cần +1          int nam = calendar.get(Calendar.YEAR);          int thu = calendar.get(Calendar.DAY\_OF\_WEEK);          int gio = calendar.get(Calendar.HOUR\_OF\_DAY);          int phut = calendar.get(Calendar.MINUTE);          int giay = calendar.get(Calendar.SECOND);          System.out.println("Ngày: " + ngay);          System.out.println("Tháng: " + thang);          System.out.println("Năm: " + nam);          System.out.println("Thứ: " + thu);          System.out.println("Giờ: " + gio);          System.out.println("Phút: " + phut);          System.out.println("Giây: " + giay);      }  } |

* Khi chương trình cứ chạy liên tục mà chúng ta muốn sử dụng một số thứ đề cập ở trên (ngày, tháng, năm, ...) thì chúng ta bắt buộc phải dừng chương trình tại thời điểm đó và lấy ra giá trị của (ngày, tháng, năm, ...) mà chúng ta cần.

Ví dụ: viết chương trình in ra phút và giây sau cứ sau mỗi 2 s.

**DỪNG CHƯƠNG TRÌNH TRONG MỘT VÀI GIÂYS**

|  |
| --- |
| import java.util.Calendar;  class Hello\_World  {  public static void main(String[] args)  {  Calendar calendar = Calendar.getInstance(); // tạo một thể hiện của lớp Calendar lưu những giá trị của thời gian hiện tại.  int x = 0;  while (true) {  try {  calendar = Calendar.getInstance(); // cập nhật lại trạng thái hiện tại của đối tượng calendar.  int phut = calendar.get(Calendar.MINUTE);  int giay = calendar.get(Calendar.SECOND);    System.out.println(phut + " : " + giay);  x++;  if (x == 10) {  break;  }  Thread.sleep(2 \* 1000); // cú pháp này giúp chương trình dừng lại sau mỗi 2 giây. Và chính là do cú pháp này chúng ta cần phải để trong khối try catch.  } catch (Exception e) {  System.out.println(e);  }  }  }  } |

**Đo lường thời gian chạy của chương trình.**

|  |
| --- |
| public static void main(String args[])     {       long Start = System.currentTimeMillis();          for (int i = 0; i < 100000; i++)        {          // System.out.println(i);        }        long End = System.currentTimeMillis();        System.out.println(End - Start);     } |

**InputStream và OutputStream**

Hai đối tượng này cần phải được phân biệt kĩ.

* InputStream: dịch đại khái có nghĩa là dòng vào. Chúng ta dùng nó và những lớp liên quan đến đó để định nghĩa dòng vào của dữ liệu. Dễ hiểu hơn là chúng ta lấy dữ liệu từ đâu, ví dụ như: lấy dữ liệu từ mảng A, từ người dùng nhập vào, từ một file nào đó.
* OutputStream: dịch đại khái có nghĩa là dòng ra. Chúng ta dùng nó và những lớp liên quan đến nó để định nghĩa dòng dữ liệu ra. Dễ hiểu hơn là sau khi lấy dòng dữ liệu vào thì chúng dựa vào đó để lấy dòng dữ liệu ra, dùng nó để ghi vào file, để xuất ra màn hình hoặc lưu trữ ở đâu đó hoặc có thể làm dữ liệu vào của một đối tượng khác.

**ByteArrayInputStream và ByteArrayOutputStream**

* Hai lớp trên cho phép chúng ta đọc và ghi dữ liệu nhị phân từ người dùng vào chương trình và bắt đầu biên dịch.
* ByteArrayInputStream: biên dịch dòng vào.
* ByteArrayOutputStream: dùng để ghi dữ liệu vào một cái gì đó ví dụ như là mảng.

(**có nhiều phương thức và cách sử dụng cho nó, sau đây sẽ là một cách hoạt động đơn giản)**

|  |
| --- |
| import java.io.\*;  public class Hello\_World {     public static void main(String args[])throws IOException {        /\*          tạo một đối tượng của lớp ByteArrayOutputStream và set up cho nó có dung lượng là 12 byte.        \*/        ByteArrayOutputStream bOutput = new ByteArrayOutputStream(12);        /\*          nhận dữ liệu từ người dùng và lưu vào trong đối tượng bOutput với điều kiện là dòng dữ liệu vào chỉ được          bé hơn 10 byte.        \*/        while( bOutput.size()!= 10 )        {           // Nhan input tu nguoi dung           bOutput.write(System.in.read());        }        // chuyển đổi dòng dữ liệu vào sang một mảng byte.        byte b [] = bOutput.toByteArray();        System.out.println("In noi dung");        // in ra giá trị của các kí tự trong dòng dữ liệu chúng ta đã nhập ở trên.        for(int x= 0 ; x < b.length; x++)        {          // giá trị của b[x] chính là mã ASCII của các từng kí tự trong dòng dữ liệu chúng ta đã nhập vào.           System.out.print(b[x]  + "   ");        }        System.out.println("   ");        /\*          Tới đây chúng ta có thể sử dụng trực tiếp mảng byte b[] để xử lí cho mọi thứ nhưng          ByteArrayInputStream giống như là một lớp tiện ích giúp chúng ta xử lí vấn đề về byte nhanh hơn.        \*/        /\*          tạo một đối tượng bInput của lớp ByteArrayInputStream và setup cho nó xử lí mảng byte b[].        \*/        ByteArrayInputStream bInput = new ByteArrayInputStream(b);        System.out.println("Chuyen doi cac ky tu thanh chu hoa " );        while (true)        {          /\*            phương thức bInput.read() có tác dụng là trả về mã ASCII của kí tự hiện tại và            đồng thời dịch chuyển con trỏ đến kí tự tiếp theo vì thế chúng ta mới có            int c = bInput.read().          \*/          int c = bInput.read();          if (c == -1)          {            break;          }          System.out.println(Character.toUpperCase((char)c));        }        bInput.reset(); // phương thức này có tác dụng dịch chuyển con trỏ về lại vị trí ban đầu của mảng byte b[].     }  } |

1. **Ghi dữ liệu vào bộ đệm khi hoàn thành rồi xuất ra.**

|  |
| --- |
| public static void main(String args[]) {          try          {              // tạo một bộ đệm bằng đối lớp ByteArrayOutputStream.              ByteArrayOutputStream Buffer = new ByteArrayOutputStream();              // wirte\_Data được sử dụng để ghi dòng dữ liệu vào Buffer.              DataOutputStream write\_Data = new DataOutputStream(Buffer);              // Buffered\_Reader được sử dụng để lấy dữ liệu từ đầu vào.              BufferedReader Buffered\_Reader = new BufferedReader(new InputStreamReader(System.in));              String s = "";              while (true)              {                  /\*                    s được gán giá trị bằng dòng đang thực hiện                    và con trỏ của đối tượng Buffered\_Reader trỏ đến dòng tiếp                    theo.                   \*/                  s = Buffered\_Reader.readLine();                  if (s.equals("End"))                  {                      break;                  }                  // ghi dữ liệu dưới dạng byte vào bên trong Buffer.                  write\_Data.writeUTF(s);                  // thêm "/n" vì s tương đương với một dòng cần có kí tự xuống dòng.                  write\_Data.writeUTF("\n");              }              /\*               sau khi không cần phải nhập vào nữa thì chuyển đổi bộ đệm               Buffer sang một mảng byte để dễ dàng xử lí.              \*/              byte b[] = Buffer.toByteArray();              // định nghĩa dòng dữ liệu vào để xử lí bằng đối tượng ByteArrayInputStream.              ByteArrayInputStream binput = new ByteArrayInputStream(b);              int x = 0;              while (true)              {                  /\*                   giá trị của b[i] là mã ASCII của từng kí tự được nhập                   vào ở trên.                   x được gán giá trị bởi mã ASCII hiện tại đang xét.                   con trỏ của binput trỏ đến b[i] tiếp theo.                   \*/                  x = binput.read();                  if (x == -1)                  {                      break;                  }                  System.out.print((char)x);              }              // reset lại con trỏ của binput.              binput.reset();              // đóng đối tượng write\_Data.              write\_Data.close();              // đóng đối tượng Buffered\_Reader.              Buffered\_Reader.close();          }          catch (Exception e)          {              System.out.println(e);          } |

* Chúng ta có thể thay thế BufferedReader ở trên bằng DataInputStream để lấy luongf dữ liệu vào.
* Chúng ta có thể thấy ở trên là một chương trình đa luồng. Sau khi chờ chương trình ngoại vi thì nó đã xử lí dữ liệu ở phía trước đó và đưa vào bộ nhớ đệm.

**PHƯƠNG THỨC ToString() cho các đối tượng.**

* **Phương thức này cho phép chúng ta chuyển tất cả các đối tượng theo dạng chuỗi.**
* **Khi chúng ta muốn một đối tượng được xuất ra dưới dạng chuỗi thì chúng ta có thể sử dụng phương thức này.**

|  |  |
| --- | --- |
| import java.util.Scanner;  class Fraction  {    private int Numerator;    private int Denominator;    public Fraction()    {      this.Numerator = 0;      this.Denominator = 1;    }    public void input()    {      Scanner input = new Scanner(System.in);      this.Numerator = input.nextInt();      this.Denominator = input.nextInt();    }    // ghi đè phương thức toString() của hệ thống.    @Override    public String toString()    {      return this.Numerator + " / " + this.Denominator;    }  } | class Hello\_World  {    public static void main(String args[])    {      Fraction f = new Fraction();      f.input();      System.out.println(f);    }  } |

**LẬP TRÌNH MẠNG.**

1. **Chuyển đổi một hostname sang địa chỉ IP.**

* Hostname là những tên miền ví dụ như: ([www.facebook.com](http://www.facebook.com)).

|  |
| --- |
| import java.net.InetAddress;  import java.net.UnknownHostException;  public class GetIPfromHostNAME{     public static void main(String[] args) {        InetAddress address = null;        try {          address = InetAddress.getByName           ("www.facebook.com");        }      catch (UnknownHostException e) {           System.out.println("khoong ton tai");        }        System.out.println(address.getHostName() + "="        + address.getHostAddress());        System.exit(0);     }  } |

1. **Kết nối đến máy chủ server của một trang web.**

|  |
| --- |
| import java.net.InetAddress;  import java.net.Socket;  public class Hello\_World {     public static void main(String[] args) {        try {           InetAddress addr;           Socket sock = new Socket("www.facebook.com", 80);           addr = sock.getInetAddress();           System.out.println("Connected to " + addr);           sock.close();        } catch (Exception e) {           System.out.println("Can't connect to this web");        }     }  } |

1. **Kiểm tra một file trên server được sửa đổi khi nào.**

|  |
| --- |
| import java.net.URL;  import java.net.URLConnection;  import java.sql.Date;  import java.text.SimpleDateFormat;  import java.io.File;  import java.net.URI;  public class Hello\_World {     public static void main(String[] argv) throws Exception     {         // Đường dẫn tới thư mục chứa tệp My\_Java.docx          File folder = new File("C:\\Users\\user\\Downloads\\java"); // Thay thế bằng đường dẫn thư mục của bạn            // Tên tệp và đường dẫn thư mục          String filename = "My\_Java.docx";          String folderPath = folder.getAbsolutePath();          // Tạo URI bằng cách kết hợp đường dẫn thư mục và tên tệp          URI uri = new URI("file:///" + folderPath.replace("\\", "/") + "/" + filename);           // chuyển đường dẫn file sang URL.          URL u = uri.toURL();          System.out.println(u);          // mở kết nối với URL đó.          URLConnection uc = u.openConnection();          uc.setUseCaches(false);          long timestamp = uc.getLastModified();           /\*              biến tiemstamp đo thời gian từ thời điểm ngày và giờ của hệ thống bắt đầu (1/1/1970)              đến lúc thời gian sửa đổi lần cuối.           \*/          // chuyển đổi timestamp sang định dạng ngày tháng năm.           Date lastModifiedDate = new Date(timestamp);          SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");          String formattedDate = sdf.format(lastModifiedDate);          System.out.println("The last modification time of My\_Java.docx is: " + formattedDate);     }  } |

Chúng ta có thể hiểu đơn giản là cú pháp trên đang tham chiếu đến máy chủ localhost, 127.0.0.1

**SOCKET**

|  |
| --- |
| Socket |

client

Server

|  |
| --- |
| Socket |

* Client và Server muốn trao đổi với nhau thì dữ liệu đầu tiên phải đi ra và cũng phải đi vào Socket. Socket đóng vai trò giống như một cách cửa.
* Socket sẽ có nhiều cánh cửa khác nhau. Server sẽ chọn một cách cửa bất kì để mở cho client kết nối. Bên phía Client phải mở cánh cửa kết nối giống tên với cánh cửa bên Server. (Cánh cửa này gọi là cổng(Port)).

1. **Hiển thị các cổng Port đang mở và lắng nghe.**

Vào cmd và gõ netstat -an | find "LISTENING"

1. **Mô hình cơ bản về socket.**

**Mô hình1:** Trong mô hình này chúng ta sẽ tạo một máy chủ, một client. Client sẽ gửi một thông điệp tới máy chủ và máy chủ sẽ hiển thị thông điệp đó ngay lập tức.

* Class\_Client.

|  |
| --- |
| package Package\_Client;  // A Java program for a Client  import java.io.\*;  import java.net.\*;  public class class\_Client {      /\*       chúng ta cần phải khởi tạo các giá trị của các thuộc tính trước vì       không biết được trong quá trình kết nối có bị lỗi hay không.       \*/      private Socket socket = null;      private DataInputStream input = null;      private DataOutputStream out = null;      private BufferedReader buff\_read = null;      /\*       tạo một Constructor gồm hai đối số là địa chỉ IP của Server và cổng       Port mà Server mở ra để kết nối.       \*/      public class\_Client(String address, int port)      {          try {              // khởi tạo socket để kết nối đến socket của Server.              socket = new Socket(address, port);              // dòng này dùng để hiển thị cho biết rằng đã kết nối thành công.              // nếu như không kết nối thành công thì sẽ không thể xuất được dòng này.              System.out.println("Connected");              // tạo dòng dữ liệu bằng cách nhập từ terminal.              input = new DataInputStream(System.in);              /\*               sau khi nhập vào dữ liệu từ terminal thì dữ liệu đó sẽ nhảy vào               bộ đệm, chúng ta cần phải lấy dữ liệu từ bộ đệm lên để xử lí.               \*/              buff\_read = new BufferedReader(new InputStreamReader(input));              /\*               tạo một đối tượng DataOutputStream để có thể ghi dữ liệu cho               luồng đầu ra cho socket, và dữ liệu đầu ra này sẽ gửi đến cho               Server.               \*/              out = new DataOutputStream(socket.getOutputStream());          }          catch (UnknownHostException u) {              System.out.println(u);              return;          }          catch (IOException i) {              System.out.println(i);              return;          }          String line = "";            while (!line.equals("Over")) {              try {                  /\*                  dùng đối tượng buff\_read (đọc dữ liệu ở bộ đệm) để gán dữ liệu                  cho line.                   \*/                    line = buff\_read.readLine();                  /\*                   dòng này dùng để xuất ra line để chúng ta biết rằng đây                   là một chương trình đa luồng khi chúng ta nhập một thông tin                   vào terminal, dữ liệu sẽ đi xuống bộ đệm. Trong khi chờ ngoại                   vi thì nó sẽ xử lí dữ liệu và gửi nó đến Server.                   \*/                  System.out.println(line);                  /\*                   ghi dữ liệu vào đầu ra cho socket.                   \*/                  out.writeUTF(line);              }              catch (IOException i) {                  System.out.println(i);              }          }          // close the connection          try {              // kết thúc kết nối bằng cách đóng hết các đối tượng.              input.close();              out.close();              socket.close();              buff\_read.close();          }          catch (IOException i) {              System.out.println(i);          }      }      public static void main(String args[])      {          class\_Client client = new class\_Client("127.0.0.1", 5000);      }  } |

* Class\_Server

|  |
| --- |
| package Package\_Server;  // A Java program for a Server  import java.net.\*;  import java.io.\*;  public class Class\_Server  {      /\*       chúng ta cần phải khởi tạo trước các thuộc tính này để có thể phòng       ngưà các trường hợp trường trình bị lỗi kết nối.       \*/      private Socket       socket = null;      private ServerSocket server = null;      private DataInputStream in   = null;      /\*       khởi tạo constructor với đối số Port là cổng dữ liệu mà Server muốn       mở ra cho các Clients kết nối.       \*/      public Class\_Server(int port)      {          try          {              /\*               Mở cổng Port cho client kết nối.               \*/              server = new ServerSocket(port);              /\*               Hai dòng dưới là để thông báo Server đã khởi động thành công.               \*/              System.out.println("Server started");              System.out.println("Waiting for a client ...");              // chấp nhận kết nối từ Client.              socket = server.accept();              // Hiển thị dòng dưới để cho biết chấp nhận kết nối thành công.              System.out.println("Client accepted");              /\*               Dữ liệu được gửi từ socket của client gửi đến bộ đệm của socket               Server. Chúng ta cần phải khởi tạo luồng đầu vào bằng cách               lấy dữ liệu từ dưới bộ đệm.               \*/              in = new DataInputStream(new BufferedInputStream(socket.getInputStream()));              String line = "";              while (!line.equals("Over"))              {                  try                  {                      // đọc dữ liệu từ luồng đầu vào.                      line = in.readUTF();                      System.out.println(line);                  }                  catch(IOException i)                  {                      System.out.println(i);                  }              }              System.out.println("Closing connection");              // đóng kết nối lại.              socket.close();              in.close();          }          catch(IOException i)          {              System.out.println(i);          }      }      public static void main(String args[])      {          Class\_Server server = new Class\_Server(5000);      }  } |

**MÔ HÌNH 2:** gửi dữ liệu 200 OK về cho máy khách khi kết nối thành công.

* Class\_Server.

|  |
| --- |
| package Package\_Server;  import java.io.\*;  import java.net.\*;  class Class\_Server  {      private DataOutputStream output = null;      private Socket socket\_Server = null;      public Class\_Server(int Port)      {          try {              /\*               khởi tạo cổng Port cho client kết nối vào.               Nếu như cổng này không mở được thì sẽ bị lỗi liền.               \*/              ServerSocket Open\_Port = new ServerSocket(Port);              /\*          Mở cổng chờ người dùng kết nối.               \*/              System.out.println("Dang cho ket noi.");              /\*          Khi có người dùng đến thì chấp nhận kết nối.      Nếu như việc kết nối có vấn đề thì bị lỗi liền.               \*/              socket\_Server = Open\_Port.accept();              /\*          Tạo hiệu ứng chờ.               \*/              Thread.sleep(10\*1000);              System.out.println("Ket noi thanh cong");              System.out.println("dang gui ma");              /\*              Tạo hiệu ứng chờ.               \*/              Thread.sleep(5\*1000);              /\*              Tạo một luồng dữ liệu ra để ghi vào socket\_Server.               \*/              output = new DataOutputStream(socket\_Server.getOutputStream());              /\*              Ghi dữ liệu vào socket server.               \*/              output.writeUTF("200 OK");              System.out.println("Gui ma thanh cong");              /\*              Kết thúc tiến trình.               \*/              socket\_Server.close();              Open\_Port.close();              output.close();          } catch (Exception e) {              // TODO Auto-generated catch block              e.printStackTrace();          }      }      public static void main(String[] args)      {          Class\_Server sv = new Class\_Server(5000);      }  } |

* Class\_Client

|  |
| --- |
| package Package\_Client;  import java.net.\*;  import java.io.\*;  class class\_Client  {      private DataInputStream input = null;      public class\_Client(String Address, int Port)      {          try {              /\*              Tạo socket để kết nối tới socket của máy chủ.               \*/              Socket socket = new Socket(Address, Port);              System.out.println("Dang ket noi");              /\*              Lấy luồng dữ liệu vào là bộ đệm của socket          do máy chủ gửi dữ liệu về thì lưu trong bộ đệm của socket.               \*/              this.input = new DataInputStream(new BufferedInputStream(socket.getInputStream()));              // đọc dữ liệu từ bộ đệm.              String s = input.readUTF();              // xuất dữ liệu ra.              System.out.println(s);              // kết thúc quá trình.              socket.close();              input.close();          } catch (Exception e) {              // TODO Auto-generated catch block              e.printStackTrace();          }      }      public static void main(String[] args)      {          class\_Client cl = new class\_Client("127.0.0.1", 5000);      }  } |

**MÔ HÌNH 3: Chat box đơn giản giữa người dùng và máy khách.**

Trong mô hình này, chúng ta sẽ phải thực hiện kết nối tới máy chủ nhằm mục đích hỏi về cách giải quyết vấn đề về kết nối.

* Client: Yêu cầu nhắn tin với máy chủ.
* Server: Chấp nhận yêu cầu máy khách.
* Client: nhắn tin (Nhắn trên một dòng).
* Server: trả lời(Trên một dòng).s
* Cilent: kết thúc( nhấn End).