**Bài 2.2. Các kiểu dữ liệu trong Java**

**Nội dung bài học**

1. [Các kiểu dữ liệu trong Java](https://braniumacademy.net/courses/java/lessons/java-data-types/#s1)
2. [Các đặc trưng](https://braniumacademy.net/courses/java/lessons/java-data-types/#s2)
3. [Các lớp bao](https://braniumacademy.net/courses/java/lessons/java-data-types/#s3)
4. [Mục đích sử dụng](https://braniumacademy.net/courses/java/lessons/java-data-types/#s4)
5. [Lấy giá trị cực đại của kiểu](https://braniumacademy.net/courses/java/lessons/java-data-types/#s5)
6. [Lấy kích thước của kiểu](https://braniumacademy.net/courses/java/lessons/java-data-types/#s6)

**Các kiểu dữ liệu trong Java**

* Java là một ngôn ngữ mạnh về kiểu, mọi thành phần đều có một kiểu cụ thể.
* Ta coi các dữ liệu như là các tài nguyên đất, nước thì kiểu dữ liệu được coi như vật chứa các tài nguyên đó.
* Mỗi kiểu có kích thước khác nhau -> sức chứa khác nhau, chứa được các loại dữ liệu khác nhau.
* Các loại dữ liệu lưu trữ trên máy tính phổ biến như: văn bản, hình ảnh, âm thanh…
* Kiểu dữ liệu trong Java chia làm hai nhóm chính:
  + Nguyên thủy.
  + Tham chiếu.

Ảnh có chứa văn bản, ảnh chụp màn hình, biểu đồ, Phông chữ

Mô tả được tạo tự động

* Là các kiểu nền tảng của Java chia thành hai nhóm:
  + Các kiểu số.
  + Kiểu boolean.
* Kiểu số chia làm hai loại:
  + Các số nguyên: char, byte, short, int, long.
  + Các số thực: float, double.
* Phạm vi phần này ta chỉ quan tâm tới các kiểu nguyên thủy. Các kiểu tham chiếu ta sẽ tìm hiểu ở phần sau khóa học.
* Có tổng cộng 8 kiểu nguyên thủy trong Java.

**Các đặc trưng**

* Kiểu nguyên thủy là kiểu dữ liệu nhỏ nhất trong Java, không thể chia nhỏ hơn nữa.
* Các kiểu này có kích thước không đổi trong mọi chương trình và mọi hệ điều hành.
* Các giá trị của kiểu nguyên thủy không chia sẻ chung trạng thái với các kiểu nguyên thủy khác.
* Mỗi kiểu nguyên thủy có một lớp bao tương ứng của nó.
* Lớp bao được sử dụng nhằm chuyển đổi giá trị kiểu nguyên thủy sang đối tượng tương ứng khi cần thực hiện những thao tác hướng đối tượng.

**Các lớp bao**

* Sau đây là danh sách các lớp bao trong Java:

|  |  |
| --- | --- |
| **Kiểu nguyên thủy** | **Lớp bao tương ứng** |
| char | Character |
| byte | Byte |
| short | Short |
| int | Integer |
| long | Long |
| float | Float |
| double | Double |
| boolean | Boolean |

**Mục đích sử dụng**

* Thông số cơ bản về các kiểu nguyên thủy trong Java:

|  |  |  |  |
| --- | --- | --- | --- |
| **Tên kiểu** | **Độ lớn(bit)** | **Khoảng lưu trữ** | **Ví dụ** |
| byte | 8 | -128 đến 127 | 100, -55 |
| char | 16 | Các kí tự UTF-16 từ ‘\u0000’ tới ‘\ufff’ | ‘a’, ‘A’, ‘4’, ‘\*’ |
| short | 16 | -32768 đến 32767 | 32760, -3700 |
| int | 32 | -2147483648 tới 2147483647 | 2147480000 |
| long | 64 | -9223372036854775808 đến 9223372036854775807 | 9999999999999 |
| float | 32 | 1.4E-45 đến 3.4028235E38 | 3.14f |
| double | 64 | 4.9E-324 đến 1.7976931348623157E308 | 3.1415926 |
| boolean |  | true hoặc false | true |

* Sử dụng kiểu char để lưu trữ một kí tự đơn.
* Sử dụng kiểu boolean để lưu trữ trạng thái true/false hoặc đúng/sai.
* Sử dụng kiểu số nguyên short, byte, int, long để lưu các số nguyên âm hoặc dương.
* Sử dụng kiểu long để lưu trữ một con số nếu kiểu int không chứa nổi số đó.
* Sử dụng kiểu float và double để lưu trữ số thực.
* Kiểu float có độ chính xác đến 6 hoặc 7 con số sau phần lẻ thập phân.
* Nếu muốn độ chính xác gấp đôi, lên 14-15 chữ số phần lẻ thập phân ta sử dụng kiểu double.
* Các giá trị kiểu float có thêm hậu tố **f** ở cuối. Ví dụ 3.14f là giá trị kiểu float.
* Các giá trị kiểu double không có hậu tố hoặc sử dụng hậu tố **d**. Ví dụ 3.14 hoặc 3.14d là giá trị của kiểu số thực.
* Khi không đòi hỏi độ chính xác lớn hơn 6 chữ số, sử dụng kiểu float. Nếu cần độ chính xác cao ta sử dụng kiểu double.
* Kiểu float và double có thêm các giá trị NaN, NEGATIVE\_INFINITY, POSITIVE\_INFINITY.

**Lấy giá trị cực đại của kiểu**

* Ta có thể sử dụng hằng số MIN\_VALUE, MAX\_VALUE để lấy giá trị tối thiểu và tối đa mà một kiểu có thể chứa.
* Lưu ý cú pháp lấy giá trị cực đại cực tiểu này phải sử dụng lớp bao để thực hiện.

|  |  |  |
| --- | --- | --- |
| **Kiểu** | **Giá trị nhỏ nhất** | **Giá trị lớn nhất** |
| Byte | Byte.MIN\_VALUE | Byte.MAX\_VALUE |
| Short | Short.MIN\_VALUE | Short.MAX\_VALUE |
| Int | Integer.MIN\_VALUE | Integer.MAX\_VALUE |
| Long | Long.MIN\_VALUE | Long.MAX\_VALUE |
| Float | Float.MIN\_VALUE | Float.MAX\_VALUE |
| Double | Double.MIN\_VALUE | Double.MAX\_VALUE |
| Char | Character.MIN\_VALUE | Character.MAX\_VALUE |

* Code mẫu:

public class Lesson22 {

public static void main(String[] args) {

System.out.println("Giá trị tối đa kiểu int: " + Integer.MAX\_VALUE);

System.out.println("Giá trị tối thiểu kiểu int: " + Integer.MIN\_VALUE);

System.out.println("Giá trị tối đa kiểu short: " + Short.MAX\_VALUE);

System.out.println("Giá trị tối thiểu kiểu short: " + Short.MIN\_VALUE);

System.out.println("Giá trị tối đa kiểu byte: " + Byte.MAX\_VALUE);

System.out.println("Giá trị tối thiểu kiểu byte: " + Byte.MIN\_VALUE);

System.out.println("Giá trị tối đa kiểu long: " + Long.MAX\_VALUE);

System.out.println("Giá trị tối thiểu kiểu long: " + Long.MIN\_VALUE);

System.out.println("Giá trị tối đa kiểu float: " + Float.MAX\_VALUE);

System.out.println("Giá trị tối thiểu kiểu float: " + Float.MIN\_VALUE);

System.out.println("Giá trị tối đa kiểu double: " + Double.MAX\_VALUE);

System.out.println("Giá trị tối thiểu kiểu double: " + Double.MIN\_VALUE);

}

}

Giá trị tối đa kiểu int: 2147483647

Giá trị tối thiểu kiểu int: -2147483648

Giá trị tối đa kiểu short: 32767

Giá trị tối thiểu kiểu short: -32768

Giá trị tối đa kiểu byte: 127

Giá trị tối thiểu kiểu byte: -128

Giá trị tối đa kiểu long: 9223372036854775807

Giá trị tối thiểu kiểu long: -9223372036854775808

Giá trị tối đa kiểu float: 3.4028235E38

Giá trị tối thiểu kiểu float: 1.4E-45

Giá trị tối đa kiểu double: 1.7976931348623157E308

Giá trị tối thiểu kiểu double: 4.9E-324

**Lấy kích thước của kiểu**

* Để kiểm tra kích thước của một kiểu dữ liệu, ta sử dụng cú pháp: **Type.SIZE**.
* Kết quả sẽ cho ta giá trị số nguyên thể hiện kích thước của kiểu dữ liệu đang xét tính theo số bit.
* Ví dụ:
  + Byte.SIZE: 8.
  + Short.SIZE: 16.
  + Integer.SIZE: 32.
  + Long.SIZE: 64.
  + Float.SIZE: 32.
  + Double.SIZE: 64.
  + Character.SIZE: 16.
* Code mẫu:

public class Lesson22 {

public static void main(String[] args) {

System.out.println("Kích thước của kiểu short: " + Short.SIZE);

System.out.println("Kích thước của kiểu int: " + Integer.SIZE);

System.out.println("Kích thước của kiểu long: " + Long.SIZE);

System.out.println("Kích thước của kiểu float: " + Float.SIZE);

System.out.println("Kích thước của kiểu double: " + Double.SIZE);

}

}

Kích thước của kiểu short: 16

Kích thước của kiểu int: 32

Kích thước của kiểu long: 64

Kích thước của kiểu float: 32

Kích thước của kiểu double: 64

**Bài 2.3. Các biến trong Java**

**Nội dung bài học**

1. [Khái niệm và đặc điểm](https://braniumacademy.net/courses/java/lessons/java-variables/#s1)
2. [Cú pháp khai báo biến](https://braniumacademy.net/courses/java/lessons/java-variables/#s2)
3. [Khởi tạo giá trị cho biến](https://braniumacademy.net/courses/java/lessons/java-variables/#s3)
4. [Gán giá trị cho biến](https://braniumacademy.net/courses/java/lessons/java-variables/#s4)
5. [Hằng số](https://braniumacademy.net/courses/java/lessons/java-variables/#s5)
6. [Các từ khóa](https://braniumacademy.net/courses/java/lessons/java-variables/#s6)
7. [Nhập dữ liệu cho biến](https://braniumacademy.net/courses/java/lessons/java-variables/#s7)
8. [Xuất dữ liệu ra màn hình](https://braniumacademy.net/courses/java/lessons/java-variables/#s8)

**Khái niệm và đặc điểm**

* Biến là tên được đặt cho vùng nhớ dùng để lưu trữ dữ liệu tạm thời nhằm phục vụ các hoạt động của chương trình.
* Biến sẽ bị hủy khi chương trình kết thúc để thu hồi bộ nhớ RAM cấp phát cho các chương trình khác.
* Biến được cấp phát bộ nhớ trong RAM, khi chương trình hoạt động, chương trình được nạp vào RAM. Khi nó bị hủy đi thì nó sẽ bị xóa khỏi RAM máy tính, điện thoại…
* Biến được chia thành các nhóm chính:
  + Biến cục bộ: biến được khai báo nội bộ trong một hàm, một khối code nào đó.
  + Biến của lớp: là các trường, các thuộc tính của lớp. Chi tiết ta sẽ tìm hiểu loại biến này trong chương hướng đối tượng.
  + Hằng số là các biến chỉ được gán giá trị duy nhất 1 lần sau đó KHÔNG ĐỔI trong toàn bộ thời gian mà chương trình tồn tại.

**Cú pháp khai báo biến**

* Cú pháp tổng quát: *kiểu danh\_sách\_biến;*
* Trong đó:
  + Kiểu là các kiểu dữ liệu hợp lệ trong ngôn ngữ Java, gồm các kiểu nguyên thủy và kiểu tham chiếu như int, float, bool hay các class, interface.
  + Danh sách biến có thể gồm 1 hoặc nhiều biến, nếu có nhiều biến thì các biến sẽ phân tách nhau bởi dấu phẩy.
  + Tên biến thường là danh từ hoặc cụm danh từ đặt tên sao cho tự bản thân biến mang ý nghĩa mà nó mô tả.
  + Tên biến không trùng với các từ khóa và các từ dành riêng trong Java.
  + Tên biến luôn bắt đầu bằng chữ cái thường, sau đó có thể là chữ cái, chữ số, dấu gạch dưới. Không bao giờ bắt đầu tên biến bằng một kí tự khác chữ cái thường.
  + Nếu tên biến có nhiều từ cấu thành thì viết liền nhau và viết hoa chữ cái đầu từ.
  + Tên biến phân biệt chữ hoa chữ thường. Tức là *age* khác với *Age* và khác *aGE.*
  + Luôn kết thúc câu lệnh khai báo bằng 1 dấu chấm phẩy.
  + Khuyến nghị khai báo mỗi biến trên 1 dòng.
* Từ phiên bản Java 9 ta có thể sử dụng keyword var để khai báo các biến cục bộ. Cú pháp:

var variableName = value;

* Java sẽ tự động gắn kiểu cho biến dựa vào giá trị value ở phía bên phải dấu bằng ta gán cho biến.
* Trong cú pháp khai báo biến với keyword var, **bắt buộc phải gán giá trị cho biến ngay khi khai báo**. Nếu không ta sẽ bị lỗi.
* Các giá trị của kiểu dữ liệu ta đã nói ở bài học 2.2. Bạn có thể xem lại. Sau đây là một số ví dụ:

var age = 20; // biến age kiểu int, nhận giá trị ban đầu là 20

var pi = 3.14; // biến pi có kiểu dữ liệu là double với giá trị ban đầu là 3.14

var letter = 'A'; // biến letter có kiểu dữ liệu là char, giá trị ban đầu là kí tự A hoa.

* Ví dụ:

int age; // tuổi

float avgGrade; // điểm TB

double interestRate; // lãi suất

long ballance; // số dư

String fullName; // họ tên đầy đủ

String address; // địa chỉ

// sau đây là các biến không hợp lệ:

int 5year; // tên bắt đầu bởi số

float my bonus; // trong tên có dấu cách

double max-volume; // trong tên có dấu -

**Khởi tạo giá trị cho biến**

* Mọi biến trong Java phải khai báo trước khi nó có thể được sử dụng
* Trong quá trình khai báo biến mà ta gán cho biến 1 giá trị ban đầu cùng kiểu thì ta gọi là khởi tạo giá trị cho biến
* Cú pháp tổng quát: *kiểu tên\_biến = giá\_trị;*
* Trong đó:
  + Kiểu là các kiểu dữ liệu hợp lệ trong ngôn ngữ Java như int, float, boolean…
  + Tên biến đặt theo quy tắc đặt tên biến
  + Dấu = gọi là toán tử gán
  + Giá trị là các giá trị hợp lệ cùng kiểu với kiểu của biến
* Ví dụ:

int age = 20; // tuổi

String fullName = ""; // họ tên

double interestRate = 0.0; // lãi suất

String address = "Hà Nội"; // địa chỉ

long balance = 15\_600\_000; // số dư

// các khởi tạo biến KHÔNG hợp lệ:

int year = "1500"; // sai kiểu dữ liệu

float gpa = 3.25; // sai kiểu dữ liệu

String email = 12345; // sai kiểu dữ liệu

* Cú pháp khởi tạo giá trị cho biến luôn kèm theo kiểu của biến ở đầu

**Gán giá trị cho biến**

* Khi ta đã khai báo biến, sau đó mới cung cấp giá trị cùng kiểu cho các biến đó thì ta gọi cú pháp này là gán giá trị cho biến
* Cú pháp tổng quát: *tên\_biến = giá\_trị;*
* Trong đó:
  + Tên biến là các tên biến đã có trước đó
  + Dấu = gọi là toán tử gán, cả câu lệnh gồm hai vế trái và phải + toán tử gán gọi là gán giá trị cho biến
  + Giá trị là các giá trị cùng kiểu với kiểu của biến
* Ví dụ:

int age = 0; // khai báo và khởi tạo tuổi bằng 0

float avgGrade = 0.0f; // khai báo và khởi tạo điểm TB = 0.0

double interestRate; // lãi suất

age = 20; // gán giá trị 20 cho biến age

avgGrade = 3.25f; // gán giá trị 3.25 cho điểm TB

interestRate = 5.25; // gán giá trị cho lãi suất

**Hằng số**

* Hằng số là các biến mà giá trị của nó đã biết trước, không có nhu cầu thay đổi và không thể thay đổi kể từ sau khi được gán giá trị
* Cú pháp tổng quát: final kiểu TÊN\_HẰNG\_SỐ = giá\_trị;
* Trong đó:
  + *final* là một từ khóa mặc định
  + Kiểu là bất kì kiều hợp lệ nào trong Java
  + Tên hằng số luôn viết hoa hoàn toàn, mang ý nghĩa nào đó. Nếu tên có nhiều từ cấu thành thì phân tách nhau bởi dấu gạch dưới.
  + Mọi hằng số không bắt buộc phải gán giá trị ngay khi khai báo nhưng sau khi đã gán giá trị lần đầu tiên thì giá trị của hằng số không thể thay đổi hoặc là chương trình sẽ báo lỗi.
* Ví dụ hằng số:

final double PI = 3.141592653589793; // số PI

final double E = 2.718281828459; // số E

final int MAX\_VALUE;

MAX\_VALUE = 100; // gán giá trị 100 cho hằng số MAX\_VALUE

MAX\_VALUE = 500; // error! giá trị của hằng số không thể thay đổi

final String MONDAY = "MONDAY"; // ok

MONDAY = "FRIDAY"; // error!

* Ta có thể khai báo hằng số trước rồi gán giá trị sau:

final String SEASON;

SEASON = "Spring"; // ok

* Ví dụ sử dụng hằng số để tính chu vi đường tròn:

public class Lesson23 {

public static void main(String[] args) {

final double PI = 3.141592653;

float r = 235.6f;

System.out.println("Chu vi đường tròn r = " + r + " là: " + (2 \* PI \* r));

}

}

// kết quả:

// Chu vi đường tròn r = 235.6 là: 1480.3184964431198

**Các từ khóa**

* Từ khóa, từ dành riêng là các từ được gán sẵn 1 chức năng cụ thể nào đó và chỉ sử dụng với mục đích thực hiện chức năng đó mà thôi.
* Các từ khóa và từ dành riêng luôn viết thường hoàn toàn

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| abstract | continue | for | new | switch | assert |
| if | package | synchronized | boolean | do | goto |
| this | break | double | implements | protected | throw |
| else | import | public | throws | case | enum |
| return | transient | catch | extends | int | short |
| char | final | interface | static | void | class |
| long | strictfp | volatile | const | float | native |
| while | default | private | byte | instanceof | super |
| try | finally | \_ | const | goto |  |

* Sau đây là các từ khóa và từ dành riêng trong Java. Khi khai báo biến mà thấy tên biến in đậm lên hoặc có màu khác thường thì đồng nghĩa với việc tên biến đã vi phạm vào các từ khóa, từ dành riêng.
* Dấu gạch dưới \_ là một từ dành riêng sẽ được sử dụng trong tương lai
* Hai giá trị *true, false* không phải từ khóa nhưng nó là giá trị của riêng kiểu boolean cũng không được phép sử dụng làm tên biến
* Biểu diễn giá trị *null* cũng không được dùng làm tên biến
* *var, yiel* cũng không được dùng làm tên biến
* Các từ khóa bị giới hạn khác là: *open, module, requires, transitive, exports, opens, to, uses, provides, with*

**Nhập dữ liệu cho biến**

* Để nhập dữ liệu vào từ bàn phím và gán cho các biến, ta sử dụng đối tượng của lớp Scanner: Scanner input = new Scanner(System.in);
* in là chuẩn đầu vào dữ liệu đọc từ bàn phím
* Để đọc dữ liệu kiểu nào ta dùng phương thức nextX trong đó X là tên kiểu của biến cần lấy giá trị. Cụ thể cho trong bảng sau:

|  |  |  |
| --- | --- | --- |
| **Kiểu biến** | **Phương thức tương ứng** | **Ví dụ** |
| char | next().charAt(0) | char c = input.next().charAt(0); |
| byte | nextByte() | byte b = input.nextByte(); |
| short | nextShort() | short s = input.nextShort(); |
| int | nextInt() | int age = input.nextInt(); |
| long | nextLong() | long longNumber = input.nextLong(); |
| float | nextFloat() | float avgGrade = input.nextFloat(); |
| double | nextDouble() | double interestRate = input.nextDouble(); |
| boolean | nextBoolean() | boolean isPassed = input.nextBoolean(); |
| String | Đọc 1 từ: next()  Đọc cả dòng: nextLine() | String name = input.next();  String fullName = input.nextLine(); |

* Thường đưa ra lời nhắc cho người dùng nhập đúng dữ liệu của kiểu ta muốn nhận vào.
* Khi đọc vào 1 từ hoặc các giá trị khác, việc đọc sẽ dừng khi gặp khoảng trắng như dấu cách, dấu tab, hoặc khi bạn ấn enter xuống dòng(dấu hiệu kết thúc dòng).
* Ví dụ đọc vào một kí tự:

import java.util.Scanner;

public class Lesson23 {

public static void main(String[] args) {

var scanner = new Scanner(System.in);

// Nhập vào một kí tự đơn

char c;

System.out.println("Nhập vào một kí tự: ");

c = scanner.next().charAt(0);

System.out.println("Kí tự bạn vừa nhập là: " + c);

}

Nhập vào một kí tự:

H

Kí tự bạn vừa nhập là: H

* Ví dụ đọc vào các loại dữ liệu khác:

import java.util.Scanner;

public class Lesson23 {

public static void main(String[] args) {

var scanner = new Scanner(System.in); // biến nhập dữ liệu từ bàn phím

// nhập vào một giá trị kiểu khác

System.out.println("Nhập tuổi: ");

var age = scanner.nextInt();

scanner.nextLine();

System.out.println("Họ và tên: ");

var fullName = scanner.nextLine();

System.out.println("Mức lương: ");

var salary = scanner.nextLong();

// hiển thị ra màn hình:

System.out.println("Họ và tên: " + fullName);

System.out.println("Tuổi: " + age);

System.out.println("Mức lương: " + salary);

}

}

Nhập tuổi:

25

Họ và tên:

Lê Hoàng Nam

Mức lương:

65200000

Họ và tên: Lê Hoàng Nam

Tuổi: 25

Mức lương: 65200000

**Xuất dữ liệu ra màn hình**

* Để xuất dữ liệu trong các biến ra màn hình ta dùng phương thức print, printf hoặc println của System.out
* out là chuẩn đầu ra dữ liệu mặc định của Java – màn hình console
* Cú pháp để thực hiện việc xuất dữ liệu ra với công cụ IntelliJ như sau:

|  |  |  |
| --- | --- | --- |
| **Phím tắt** | **Phương thức** | **Ý nghĩa sử dụng** |
| sout+Tab | System.*out*.println(); | In ra nội dung cho trước sau đó đưa con trỏ chuột xuống đầu dòng kế tiếp |
| Soutf+Tab | System.*out*.printf(); | In ra nội dung cho trước theo định dạng cụ thể tương ứng giống như trong ngôn ngữ C |
|  | System.*out*.print(); | In ra nội dung cho trước và sau đó con trỏ chuột ở nguyên vị trí kết thúc không xuống dòng mới |

* Một số kí tự định dạng với printf:

|  |  |
| --- | --- |
| **Kiểu** | **Kí tự chuyển đổi định dạng** |
| Char | %c |
| String | %s |
| Int, byte, short, long | %d |
| Float, double | %f |

* Ví dụ áp dụng:

import java.text.DecimalFormat;

import java.util.Scanner;

public class Lesson23 {

public static void main(String[] args) {

var scanner = new Scanner(System.in); // biến nhập dữ liệu từ bàn phím

System.out.println("Họ và tên: ");

var fullName = scanner.nextLine();

System.out.println("Điểm gpa: ");

var gpa = scanner.nextFloat();

System.out.println("Mức lương: ");

var salary = scanner.nextDouble();

// hiển thị ra màn hình:

System.out.printf("Họ và tên: %-30s\n", fullName);

System.out.printf("Điểm GPA: %-10.2f\n", gpa);

// định dạng giá trị số phân tách bởi dấu phẩy ở phần nghìn, lẻ thập phân

var formatter = new DecimalFormat("#,###.##");

System.out.printf("Mức lương: %sđ\n", formatter.format(salary));

}

}

* Kết quả:

Họ và tên:

Lê Công Nam

Điểm gpa:

3.2564

Mức lương:

12560000.265

Họ và tên: Lê Công Nam

Điểm GPA: 3.26

Mức lương: 12,560,000.27đ

# Bài 2.4. Các toán tử, phép toán

### Nội dung bài học

1. [Khái niệm và quy ước](https://braniumacademy.net/courses/java/lessons/java-operators/#s1)
2. [Các toán tử số học](https://braniumacademy.net/courses/java/lessons/java-operators/#s2)
3. [Các toán tử gán](https://braniumacademy.net/courses/java/lessons/java-operators/#s3)
4. [Các toán tử so sánh](https://braniumacademy.net/courses/java/lessons/java-operators/#s4)
5. [Các toán tử logic](https://braniumacademy.net/courses/java/lessons/java-operators/#s5)

### Khái niệm và quy ước

* Toán tử: hiểu đơn giản là phép toán.
* Toán tử 1 ngôi: toán tử chỉ cần 1 toán hạng là có thể hoàn tất biểu thức.
* Toán tử 2 ngôi: toán tử cần có 2 toán hạng trái và phải phép toán thì mới hoàn thành được biểu thức.
* Nếu phép toán có nhiều hơn 1 kí tự thì phải viết liền các kí tự sát nhau.
* Luôn phân tách giữa toán tử hai ngôi và các toán hạng của nó bằng 1 dấu cách cho chương trình dễ đọc.

### Các toán tử số học: +, -, \*, /, %, ++, —

#### Toán tử +, –

* Thứ tự ưu tiên thực hiện phép toán từ trái qua phải.
* Nếu là chúng được sử dụng như toán tử 1 ngôi thì phải đặt trước toán hạng.
* Kiểu của toán hạng phải là các kiểu số hoặc kiểu lớp bao của kiểu số nếu không chương trình sẽ báo lỗi.
* Ví dụ:

int a = 5;

int b = -a; // dùng - như toán tử 1 ngôi

int c = +b; // dùng + như toán tử 1 ngôi

* Nếu sử dụng như toán tử hai ngôi thì hai toán hạng trái phải của toán tử phải là kiểu số hoặc kiểu có thể chuyển sang số.
* Khi thực hiện cộng trừ có thể xảy ra vấn đề tràn số nếu giá trị kết quả lớn hơn hoặc nhỏ hơn khả năng lưu trữ của kiểu.
* Nếu toán hạng có chứa kiểu float hoặc double thì:
  + Nếu 1 trong hai toán hạng có giá trị là NaN thì kết quả là NaN. NaN là một hằng số trong lớp bao của kiểu float, double ám chỉ giá trị đáng lẽ ra là số nhưng lại không phải số. Cú pháp lấy NaN: Float.NaN hoặc Double.NaN.
  + Tổng của hai giá trị vô cực trái dấu là NaN.
  + Tổng của hai giá trị vô cực cùng dấu là giá trị vô cực cùng dấu.
  + Tổng của một giá trị giới hạn và một giá trị vô cực là vô cực.
  + Tổng của hai giá trị 0 trái dấu là +0.
  + Tổng của giá trị 0 và giá trị khác 0 là giá trị khác 0.
  + Tổng của hai giá trị cùng độ lớn nhưng trái dấu là +0.
* Mặc dù có thể xảy ra tràn số, mất thông tin nhưng việc thực hiện các phép toán trên không gây ra việc ném các ngoại lệ run-time exception.
* Ví dụ:

public class Lesson24 {

public static void main(String[] args) {

var a = 100;

var b = 250.25;

var sum = a + b; // cộng hai số bình thường

var diff = a - b; // trừ số a cho số b

// hiện kết quả:

System.out.println(a + " + " + b + " = " + sum);

System.out.println(a + " - " + b + " = " + diff);

// cộng âm vô cực với dương vô cực

var sumInfinity = Float.NEGATIVE\_INFINITY + Float.POSITIVE\_INFINITY;

System.out.println(sumInfinity);

// cộng vô cực cùng dấu -> vô cực cùng dấu

System.out.println(Float.NEGATIVE\_INFINITY + Float.NEGATIVE\_INFINITY);

System.out.println(Float.POSITIVE\_INFINITY + Float.POSITIVE\_INFINITY);

// cộng vô cực với số cụ thể -> vô cực

System.out.println(Float.POSITIVE\_INFINITY + a);

System.out.println(Float.NEGATIVE\_INFINITY + a);

// trừ vô cực cho vô cực đuợc NaN

System.out.println(Float.NEGATIVE\_INFINITY - Float.NEGATIVE\_INFINITY);

System.out.println(Float.POSITIVE\_INFINITY - Float.POSITIVE\_INFINITY);

// trừ vô cực với số cụ thể -> vô cực

System.out.println(Float.POSITIVE\_INFINITY - a);

System.out.println(Float.NEGATIVE\_INFINITY - a);

}

}

100 + 250.25 = 350.25

100 - 250.25 = -150.25

NaN

-Infinity

Infinity

Infinity

-Infinity

NaN

NaN

Infinity

-Infinity

#### Trường hợp phép + với String:

* Nếu áp dụng phép + với ít nhất 1 toán hạng là kiểu String thì kết quả là một String. Do đó toán tử + còn gọi là toán tử nối chuỗi kí tự.
* Ví dụ: String str = “Hello”; str + 5 + 6 sẽ cho kết quả “Hello56” không phải “Hello11”.
* Để có kết quả chính xác ta có thể gộp nhóm với cặp () ví dụ str + (5 + 6).
* Cụ thể về lớp String sẽ tìm hiểu trong bài học về sau.
* Ví dụ minh họa:

public class Lesson24 {

public static void main(String[] args) {

var a = 100;

var b = 250.25;

// phép nối chuỗi

System.out.println("Result: " + (a + b));

System.out.println("Result: " + a + b);

}

}

Result: 350.25

Result: 100250.25

#### Toán tử \*:

* Thứ tự ưu tiên thực hiện biểu thức là từ trái qua phải.
* Đây là các toán tử hai ngôi với hai toán hạng bắt buộc phải là kiểu số hoặc kiểu lớp bao của các kiểu số nếu không chương trình sẽ báo lỗi.
* Kết quả thực hiện của các phép toán trên thường được lưu vào một biến có kiểu là kiểu lớn nhất trong kiểu của các toán hạng.
* Ví dụ giá trị kiểu int nhân với kiểu long thì kết quả sẽ ở kiểu long vì kiểu long có kích thước lớn hơn int tính theo số bit hoặc số byte.
* Nếu hai toán hạng ở kiểu nguyên và xảy ra việc tràn số khi tính toán, giá trị kết quả thu được có thể khác dấu với giá trị đúng khi thực hiện cùng phép toán đó.
* Nếu nhân mà 1 trong các toán hạng là kiểu float hoặc double thì:
  + Nếu 1 toán hạng là NaN thì kết quả là NaN.
  + Nếu kết quả không phải NaN thì kết quả cho dấu + khi hai toán hạng cùng dấu và ngược lại cho dấu âm khi hai toán hạng trái dấu.
  + Nhân giá trị NaN với vô cực cho kết quả là NaN.
  + Nhân giá trị giới hạn với giá trị vô cực cho kết quả vô cực có dấu được xét theo quy tắc nhân hai số bình thường.
  + Nếu cả hai toán hạng đều khác vô cực và NaN thì kết quả thực hiện theo quy tắc nhân bình thường và lưu vào kiểu lớn nhất trong kiểu của hai toán hạng.
* Ví dụ minh họa:

public class Lesson24 {

public static void main(String[] args) {

var a = 100; // int

var b = -250.25; // double

var c = -999L; // long

var result = a \* b \* c;

System.out.println(a + " \* " + b + " \* " + c + " = " + result);

System.out.println(b \* Float.NaN);

System.out.println(Double.NaN \* Float.NaN);

System.out.println(Double.POSITIVE\_INFINITY \* Float.NaN);

System.out.println(Double.POSITIVE\_INFINITY \* Double.NEGATIVE\_INFINITY);

System.out.println(Double.POSITIVE\_INFINITY \* Double.POSITIVE\_INFINITY);

System.out.println(Double.NEGATIVE\_INFINITY \* Double.NEGATIVE\_INFINITY);

}

}

100 \* -250.25 \* -999 = 2.4999975E7

NaN

NaN

NaN

-Infinity

Infinity

Infinity

#### Phép chia /:

* Nếu tử số và mẫu số của phép chia đều là giá trị kiểu nguyên, kết quả sẽ làm tròn về phần nguyên của phép chia.
* Nếu cả hai toán hạng của phép chia cùng là kiểu nguyên và mẫu số bằng 0 thì sẽ xảy ra ngoại lệ chia cho 0: ArithmeticException.
* Nếu ít nhất 1 trong 2 toán hạng là kiểu float hoặc double thì:
  + Nếu 1 trong hai toán hạng là NaN thì kết quả là NaN.
  + Nếu kết quả khác NaN thì dấu của kết quả được xác định theo quy tắc nhân hai số có dấu trong toán học.
  + Chia vô cực cho vô cực được NaN.
  + Chia vô cực cho giá trị giới hạn được giá trị vô cực và dấu xác định theo quy tắc nhân.
  + Chia giá trị giới hạn cho vô cực được giá trị 0 có dấu xác định theo quy tắc nhân.
  + Chia 0 cho 0 được NaN. Nếu tử số là 0 và mẫu số khác 0 thì dấu xác định theo quy tắc nhân.
  + Chia giá trị khác 0 cho mẫu số bằng 0 được giá trị vô cực có dấu xác định theo quy tắc nhân.
  + Các trường hợp khác tính toán bình thường, kết quả làm tròn chữ số cuối cùng đến giá trị gần nhất, tức làm tròn lên nếu giá trị sau nó >= 0.5 và ngược lại.
* Phép chia có thể gây tràn số và khi đó giá trị cũng như dấu của kết quả sẽ không như mong muốn ban đầu.
* Dù có thể xảy ra việc tràn số và mất thông tin, chia cho 0 nhưng việc thực hiện\*/ không ném ngoại lệ tại thời điểm chạy chương trình.

public class Lesson24 {

public static void main(String[] args) {

var a = 100; // int

var b = -250.25; // double

var c = -999L; // long

var result = a / b;

System.out.println(result);

System.out.println("1 / 2 = " + (1 \* 1.0 / 2));

System.out.println(-1f/0); // -infinity

System.out.println(1f/0); // +infinity

System.out.println(5 / 0); // error

}

}

-0.3996003996003996

1 / 2 = 0.5

-Infinity

Infinity

Exception in thread "main" java.lang.ArithmeticException: / by zero

at Lesson24.main(Lesson24.java:12)

#### Phép chia lấy dư %:

* Là một toán tử hai ngôi.
* Lấy phần dư sau khi thực hiện phép chia /.
* Áp dụng cho cả kiểu số thực.
* Lưu ý mẫu số khác 0 để đảm bảo không xảy ra ngoại lệ chia cho 0.
* Dấu của kết quả chỉ phụ thuộc vào dấu của tử số.
* Nếu thao tác với số thực thì:
  + Nếu 1 trong hai toán hạng là NaN thì kết quả là NaN.
  + Nếu kết quả khác NaN dấu chỉ phụ thuộc dấu của tử số.
  + Nếu tử số là vô cực hoặc mẫu số là 0 hoặc cả hai thì kết quả là NaN.
  + Nếu tử số là giá trị giới hạn và mẫu là giá trị vô cực thì kết quả là tử số.
  + Nếu tử số bằng 0 và mẫu là giá trị khác 0 thì kết quả là tử số.
  + Các trường hợp khác lấy dư bình thường.
* Thực hiện thao tác với phép toán này cũng không xảy ra ngoại lệ trong thời gian chạy chương trình.
* Ví dụ minh họa:

public class Lesson24 {

public static void main(String[] args) {

var a = 100; // int

var b = -250.25; // double

var c = -999L; // long

System.out.println(-20.333 % -3);

System.out.println(a % b);

System.out.println(a % c);

System.out.println(b % 0);

System.out.println(a % 0);

}

}

-2.3329999999999984

100.0

100

NaN

Exception in thread "main" java.lang.ArithmeticException: / by zero

at Lesson24.main(Lesson24.java:11)

#### Toán tử ++, –:

* Là toán tử 1 ngôi thực hiện việc tăng hoặc giảm giá trị của một số đi 01 đơn vị.
* Có thể đặt trước hoặc sau toán hạng đều được.
* Toán hạng của hai toán tử này thường là các kiểu số hoặc kiểu lớp bao của các kiểu số nếu không chương trình sẽ báo lỗi.
* Kiểu của kết quả phép toán ++ hoặc — sẽ có kiểu trùng với kiểu của biến.
* Các phép toán này chỉ ảnh hưởng đến giá trị của biến.
* Các hằng số hay các biến có từ khóa final trong khai báo không thể áp dụng với các toán tử này.
* **Lưu ý** khi sử dụng kết hợp giữa phép toán ++/– với phép gán:
  + Nếu ta gán x = a++; lệnh tương đương sẽ là x = a; a++; tức gán giá trị gốc của a cho x trước; sau đó a mới tăng 1 đơn vị. Lúc này giá trị của x < a sau khi thực hiện biểu thức.
  + Nếu ta gán x = ++a; lúc này lệnh tương đương là a = a + 1; sau đó gán a cho x. Kết quả là giá trị của x và a sau khi kết thúc lệnh là a == x.
  + Đây là câu hỏi hay được sử dụng trong các bài test, phỏng vấn. Các bạn nên cẩn trọng.
* Ví dụ minh họa:

public class Lesson24 {

public static void main(String[] args) {

var a = 100; // int

var afterA = ++a; // a = a + 1; afterA = a;

var b = 200;

var afterB = b++; // afterB = b; b = b + 1;

System.out.println("Sau khi thực hiện phép toán ++, --");

System.out.println("a = " + a);

System.out.println("afterA = " + afterA);

System.out.println("b = " + b);

System.out.println("afterB = " + afterB);

}

}

Sau khi thực hiện phép toán ++, --

a = 101

afterA = 101

b = 201

afterB = 200

### Các toán tử gán: =, +=, -=, \*=, /=, %=

* Các toán tử gán là toán tử hai ngôi dùng để gán giá trị cho một biến cùng kiểu hoặc có kiểu tương thích
* Các biến kiểu lớn hơn luôn có thể chứa được giá trị của các kiểu nhỏ hơn. Thứ tự này là**byte -> short -> int -> long -> float -> double**.
* Nếu muốn thực hiện gán giá trị của kiểu lớn hơn cho kiểu nhỏ hơn, ta phải ép kiểu và chấp nhận mất giá trị. Nội dung này sẽ trình bày chi tiết trong bài tiếp theo.
* Thứ tự ưu tiên sẽ được thực hiện từ phải qua trái tức là nếu a = b = c thì đầu tiên gán c cho b sau đó gán b cho a.
* Toán tử gán có thể kết hợp với các toán tử số học, toán tử thao tác với bit(không học, bạn đọc có thể tự tìm hiểu thêm).
* Toán hạng trái của phép gán luôn là biến không phải hằng số, nếu là hằng số chỉ gán được duy nhất lần đầu tiên.
* Toán hạng phải của phép gán có thể là các biến, hằng số, các giá trị cụ thể của kiểu tương ứng, biểu thức, …
* Sau khi thực hiện phép gán thành công thì giá trị của biểu thức gán là giá trị của toán hạng trái. Nếu trong quá trình gán mà xảy ra lỗi thì kết quả của toán hạng trái không thay đổi
* Sau đây là bảng mô tả phép gán kết hợp với các toán tử số học với a = 5, b = 6:

[Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, số

Mô tả được tạo tự động](https://braniumacademy.net/wp-content/uploads/2020/08/k%E1%BA%BFt-h%E1%BB%A3p-ph%C3%A9p-g%C3%A1n-v%E1%BB%9Bi-to%C3%A1n-t%E1%BB%AD-s%E1%BB%91-h%E1%BB%8Dc.png)

* Ví dụ minh họa:

public class Lesson24 {

public static void main(String[] args) {

int a = 5;

int b = 6;

System.out.println("a = " + a);

a = a + 5; // tương đương a += 5

System.out.println("a = " + a);

a = a - 5; // tương đương a -= 5

System.out.println("a = " + a);

a = a \* 5; // tương đương a \*= 5

System.out.println("a = " + a);

a = a / 5; // tương đương a /= 5

System.out.println("a = " + a);

a = a % 5; // tương đương a %= 5

System.out.println("a = " + a);

}

}

a = 5

a = 10

a = 5

a = 25

a = 5

a = 0

### Các toán tử so sánh: >, >=, <, <=, ==, !=

* Là các toán tử hai ngôi dùng để so sánh mối quan hệ giữa hai toán hạng được áp dụng.
* Thường chỉ sử dụng các phép so sánh với các kiểu số nên các toán hạng của các phép so sánh phải là kiểu có thể chuyển về kiểu số nếu không thì chương trình sẽ bị lỗi.
* Kết quả của phép so sánh luôn ở kiểu boolean theo đúng mối quan hệ về giá trị độ lớn của hai toán hạng.
* Việc so sánh hai số thực kiểu float và double luôn cho kết quả chính xác.
* Với các số thực:
  + Nếu 1 trong hai toán hạng là NaN thì kết quả là false.
  + Tất cả các giá trị khác theo thứ tự: **âm vô cực < giá trị giới hạn < dương vô cực**.
  + -0.0 và 0.0 là tương đương nhưng phương thức min, max của lớp Math coi -0.0 < 0.0.

#### Với toán tử == và !=:

* Thứ tự thực hiện từ trái sang phải.
* Hai toán hạng phải cùng kiểu để đảm bảo không bị lỗi.
* Kết quả của phép so sánh ở kiểu boolean.
* Phép == cho kết quả true khi và chỉ khi hai biểu thức trái và phải của nó là tương đương nhau.
* Phép != chỉ cho kết quả true khi và chỉ khi biểu thức trái và phải của nó không tương đương nhau về mặt giá trị.
* Biểu thức a != b tương đương !(a == b).
* Với số thực:
  + Nếu 1 trong các toán hạng là NaN thì kết quả so sánh == luôn false nhưng != là true. Nói cách khác, x != x true khi và chỉ khi x có giá trị là NaN.
  + Có thể dùng Float.isNaN hoặc Double.isNaN để kiểm tra giá trị của biến nào đó có phải NaN hay không.
  + 0 âm và 0 dương được coi là tương đương nhau, tức -0.0 == 0.0 cho kết quả true.
  + Các trường hợp khác đánh giá như bình thường.
* Ví dụ minh họa:

public class Lesson24 {

public static void main(String[] args) {

int a = 5;

int b = 6;

System.out.println("a > b ? " + (a > b));

System.out.println("a >= b ? " + (a >= b));

System.out.println("a < b ? " + (a < b));

System.out.println("a <= b ? " + (a <= b));

System.out.println("a == b ? " + (a == b));

System.out.println("a != b ? " + (a != b));

// số thực

System.out.println(Float.NaN == Float.NaN);

System.out.println(Float.POSITIVE\_INFINITY == Float.POSITIVE\_INFINITY);

System.out.println(Float.POSITIVE\_INFINITY != Float.POSITIVE\_INFINITY);

System.out.println(Float.POSITIVE\_INFINITY != Float.NEGATIVE\_INFINITY);

System.out.println(Float.POSITIVE\_INFINITY == Float.NEGATIVE\_INFINITY);

System.out.println(-0 == +0);

System.out.println(-0 != +0);

}

}

a > b ? false

a >= b ? false

a < b ? true

a <= b ? true

a == b ? false

a != b ? true

false

true

false

true

false

true

false

### Các toán tử logic: &&, ||, !

#### Toán tử và(&&):

* Thứ tự ưu tiên thực hiện biểu thức từ trái qua phải. Tức là nếu toán hạng trái false thì sẽ không tiếp tục xét tới toán hạng phải.
* Là toán tử hai ngôi với hai toán hạng trái phải cùng có kiểu boolean hoặc Boolean hoặc chương trình sẽ báo lỗi.
* Toán hạng phải chỉ được đánh giá khi toán hạng trái là true.
* Khi kết hợp các biểu thức, hãy đưa biểu thức có khả năng sai nhiều hơn ra làm toán hạng trái của toán tử &&.
* Kết quả thu được luôn ở kiểu boolean, tức chỉ có thể là true hoặc false.

#### Toán tử hoặc(||):

* Thứ tự ưu tiên thực hiện biểu thức từ trái qua phải.
* Là toán tử hai ngôi với cả hai toán hạng trái và phải cùng kiểu boolean hoặc Boolean. Ngược lại chương trình sẽ báo lỗi.
* Toán hạng phải chỉ được đánh giá khi toán hạng trái là false.
* Khi sử dụng, hãy đưa biểu thức có khả năng đúng nhiều nhất ra trước để giảm bớt thời gian thực hiện đánh giá thêm các điều kiện.
* Kết quả thu được là true hoặc false.

#### Toán tử phủ định(!):

* Toán tử ! luôn đứng trước một toán hạng có kiểu boolean hoặc Boolean nếu không thì chương trình sẽ báo lỗi.
* Kết quả của biểu thức với toán tử ! luôn có kiểu boolean.
* Với toán hạng x có kiểu boolean thì:
  + Nếu x = true, !x = false.
  + Nếu x = false, !x = true.
* Bảng tóm tắt các giá trị chân lý với 1 tương đương true, 0 tương đương false:

[Ảnh có chứa ảnh chụp màn hình, hàng, văn bản, số

Mô tả được tạo tự động](https://braniumacademy.net/wp-content/uploads/2020/08/b%E1%BA%A3ng-gi%C3%A1-tr%E1%BB%8B-ch%C3%A2n-l%C3%BD.png)  
  
Ví dụ minh họa:

public class Lesson24 {

public static void main(String[] args) {

int a = 5;

int b = 6;

// gpa >= 3.6 && rank(gpa) <= 100 && điểm rèn luyện >= 90đ

// (gpa >= 3.6 && điểm rèn luyện >= 90đ) || rank(gpa) <= 100 || đạt giải xuất sắc cuộc thi ACM

// !true = false; !false = true

System.out.println(!(a == b));

System.out.println(!(a < b));

}

}

true

false

# Bài 2.5. Thao tác ép kiểu, làm tròn số

### Nội dung bài học

1. [Khái niệm ép kiểu](https://braniumacademy.net/courses/java/lessons/java-types-cast/#s1)
2. [Lý do phải ép kiểu](https://braniumacademy.net/courses/java/lessons/java-types-cast/#s2)
3. [Có những cách ép kiểu nào](https://braniumacademy.net/courses/java/lessons/java-types-cast/#s3)
4. [Làm tròn số](https://braniumacademy.net/courses/java/lessons/java-types-cast/#s4)
5. [Ví dụ minh họa](https://braniumacademy.net/courses/java/lessons/java-types-cast/#s5)

### Khái niệm ép kiểu

* Ép kiểu là hành động đưa giá trị của một kiểu sang một kiểu khác.
* Kiểu ở đây là các kiểu dữ liệu hợp lệ trong ngôn ngữ Java như các kiểu nguyên thủy, tham chiếu.
* Ví dụ ta ép kiểu giá trị kiểu:
  + int sang float
  + float sang int
  + double sang float
  + double sang int
* Hành động ép kiểu có thể thể hiện tường minh, tức chỉ rõ kiểu đích cần ép sang. Hoặc ép kiểu ngầm định, tức không cần chỉ rõ kiểu đích, chương trình tự biết thực hiện.

### Lý do phải ép kiểu

* Trong một ngữ cảnh nào đó, ta sẽ cần một kiểu dữ liệu cụ thể để thao tác được chính xác. Để đảm bảo điều này có thể thực hiện được, ta cần phải ép kiểu.
* Ví dụ:
  + Để có giá trị chính xác của phép chia 2/3 ta phải ép kiểu tử hoặc mẫu số sang kiểu số thực float hoặc double.
  + Để lấy phần nguyên của giá trị 3.14159 ta ép kiểu nó sang số nguyên kiểu int chẳng hạn.
  + Khi muốn các giá trị nguyên thủy có thể thao tác như các đối tượng, ta ép kiểu sang kiểu lớp bao của nó. Ví dụ: Integer intObj = 200;

### Có những cách ép kiểu nào

* Có nhiều cách ép kiểu khác nhau. Chúng ta tập trung vào ép kiểu các kiểu nguyên thủy trước:
  + Ép sang cùng kiểu
  + Ép sang các kiểu rộng hơn
  + Ép sang các kiểu hẹp hơn
  + Ép sang kiểu rộng hơn sau đó sang kiểu hẹp hơn
  + Ép kiểu đóng hộp(boxing)
  + Ép kiểu mở hộp(unboxing)
* Ngoài ra còn có các thao tác ép kiểu nâng cao khác sẽ tìm hiểu ở các bài học tương ứng về sau. Ví dụ ép kiểu với chuỗi kí tự(String), ép kiểu với các kiểu tham chiếu…

#### Ép sang cùng kiểu:

* Đây là cách ép kiểu hay còn gọi chuyển đổi phổ thông nhất. Bạn có thể gán giá trị của các biến cùng kiểu cho nhau. Hành động này không xảy ra lỗi nào cả. Ép kiểu sang giá trị cùng kiểu mặc định là ép kiểu ngầm định. Ta vẫn có thể thể hiện ép kiểu cách này một cách tường minh nếu muốn.
* Khi ép kiểu theo cách này, dữ liệu gốc được bảo toàn.
* Ví dụ:

public class Lesson25 {

public static void main(String[] args) {

String x = "Hello";

String y = x; // tương đương String y = (String)x;

System.out.println(y.toUpperCase());

int a = 200;

int b = a; // tương đương int b = (int)a;

b++;

a--;

System.out.println("a = " + a);

System.out.println("b = " + b);

}

}

HELLO

a = 199

b = 201

#### Ép sang các kiểu rộng hơn:

* Mặc định khi ta ép kiểu từ kiểu có kích thước nhỏ hơn sang kiểu có kích thước lớn hơn hoặc từ kiểu con sang kiểu cha thì nó thuộc loại ép kiểu này.
* Các thao tác ép kiểu nhóm này sẽ thực hiện một cách ngầm định. Không cần phải chỉ rõ kiểu đích.
* Khi ép kiểu theo cách này, dữ liệu gốc được bảo toàn. Các trường hợp thỏa mãn là:
  + Ép từ kiểu nguyên nhỏ hơn sang kiểu nguyên lớn hơn.
  + Ép từ byte, short, char sang kiểu số thực.
  + Ép từ int sang double.
  + Từ float sang double ở biểu thức strictfp.
* Ví dụ ép kiểu:
  + Ép từ kiểu byte sang short, int, long, float, double.
  + Ép từ kiểu short sang kiểu int, long, float, double.
  + Ép kiểu từ int sang long, float hoặc double.
  + Ép kiểu float sang double.

public class Lesson25 {

public static void main(String[] args) {

int a = 99;

long newA = a;

float fA = a;

double dA = fA;

dA = newA;

var result = 99 \* dA; // ép sang kiểu double

}

}

* Ví dụ chứng minh mất dữ liệu khi ép kiểu:

int a = 1234567890;

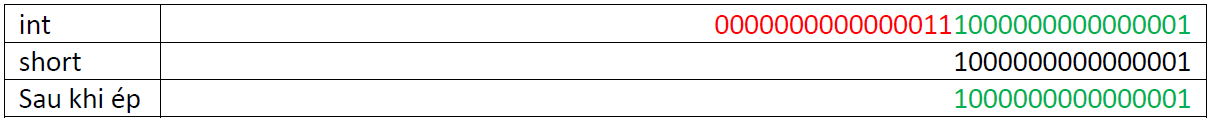
float f = a;

System.out.println(a - (int)f); // Kết quả -46

Ví dụ này chỉ ra rằng. Nếu việc ép kiểu từ kiểu nhỏ hơn sang kiểu lớn hơn mà thực hiện chính xác, thì việc ép kiểu ngược lại cũng sẽ cho cùng giá trị. Nhưng không phải tất cả mọi ép kiểu loại này đều đúng như vậy.

#### Ép sang kiểu hẹp hơn

* Ép kiểu loại này xảy ra khi thực hiện ép kiểu từ kiểu lớn hơn sang kiểu nhỏ hơn.
* Để ép kiểu thành công, ta sẽ phải thực hiện tường minh.
* Ép kiểu loại này sẽ gây mất dữ liệu. Đây là điều không thể tránh khỏi. Lý do mất dữ liệu là kiểu nhỏ hơn chỉ chứa được n bít thấp(màu xanh) của các bit kiểu gốc trước khi ép sang. Cụ thể hơn:

[](https://braniumacademy.net/wp-content/uploads/2020/08/l%C3%BD-do-%C3%A9p-ki%E1%BB%83u-m%E1%BA%A5t-d%E1%BB%AF-li%E1%BB%87u.png)

* Ví dụ:
  + Ép từ kiểu int sang kiểu short.
  + Ép từ kiểu double sang float hoặc long, int…
* Cụ thể:

public class Lesson25 {

public static void main(String[] args) {

int a = 99;

int bound = (int) Math.sqrt(a);

float pi = 3.141f;

int intPi = (int) pi;

System.out.println("Phần nguyên của căn(99): " + bound);

System.out.println("Phần nguyên của số pi: " + intPi);

}

}

Phần nguyên của căn(99): 9

Phần nguyên của số pi: 3

#### Ép sang kiểu rộng hơn sau đó sang kiểu hẹp hơn

* Trường hợp này xảy ra khi ta ép kiểu byte sang char. Các bước thực hiện là:
  + Đầu tiên byte sẽ chuyển sang kiểu int(ép kiểu rộng)
  + Sau đó từ int sẽ ép xuống kiểu char(ép kiểu hẹp)

#### Ép kiểu đóng hộp

* Điều này xảy ra khi ta đưa giá trị nguyên thủy sang giá trị kiểu lớp bao tương ứng của nó.
* Ví dụ:
  + Ép từ kiểu boolean sang Boolean.
  + Ép từ byte sang Byte.
  + Ép từ short sang Short.
  + Ép từ char sang Character.
  + Ép từ int sang Integer.
* Phương pháp hay cách ép kiểu này thường xuyên được áp dụng qua lại trong chương trình.
* Mục đích của việc ép kiểu theo phương pháp này là để sử dụng giá trị kiểu nguyên thủy như là một đối tượng. Từ đó có thể dễ dàng thực hiện các thao tác cần thiết như gọi phương thức, so sánh…

int x = 200; // x là kiểu nguyên thủy

Integer xObject = x; // xObject là kiểu tham chiếu

System.out.println(xObject.toString()); // ok

System.out.println(xObject.compareTo(200)); // ok

System.out.println(x.toString()); // error

System.out.println(x.compareTo(200)); // error

// ví dụ trong video:

public class Lesson25 {

public static void main(String[] args) {

int x = 200;

Integer integer = x;

double d = integer.doubleValue();

integer.toString();

Double doubleX = d;

System.out.println(doubleX.longValue());

System.out.println(doubleX.isNaN());

int y = doubleX.intValue();

long l = doubleX.longValue();

float f = doubleX.floatValue();

double dd = doubleX;

}

}

#### Ép kiểu mở hộp

* Khi thực hiện việc đưa giá trị của kiểu lớp bao nào đó về kiểu nguyên thủy tương ứng. Nói cách khác, khi đưa giá trị của kiểu tham chiếu về kiểu nguyên thủy tương ứng, ta gọi là ép kiểu mở hộp(unboxing).
* Cùng với ép kiểu đóng hộp(boxing) thì ép kiểu mở hộp thường xuyên được thực hiện trong hệ thống của Java khi cần thực hiện các thao tác cần thiết.
* Ví dụ:
  + Ép từ Boolean sang kiểu boolean.
  + Ép từ kiểu Integer sang int.
  + Ép từ Float sang kiểu float.
* Cụ thể:

Integer xObject = 500; // xObject là kiểu tham chiếu

int x = xObject.intValue(); // x là kiểu nguyên thủy

// tương đương:

x = xObject;

Double dObject = 123.456;

double d = dObject.doubleValue();

// tương đương:

d = dObject;

### Làm tròn số

* Việc làm tròn số thường xuyên xảy ra với các giá trị số thực. Để làm tròn số trong Java có nhiều cách khác nhau:
  + Sử dụng phương thức Math.ceil() để làm tròn lên số nguyên gần nhất.
  + Sử dụng phương thức Math.floor() để làm tròn xuống số nguyên gần nhất.
  + Sử dụng phương thức Math.round() để làm tròn tự động lên hoặc xuống giá trị nguyên gần nhất. Quy ước mọi phần thập phân lẻ >= 0.5 sẽ làm tròn lên, ngược lại làm tròn xuống.
  + Làm tròn đến k chữ số với printf() và %n.kf. Trong đó n là độ rộng tính theo số kí tự để in ra kết quả. Còn k là số chữ số cần làm tròn phần thập phân.
  + Sử dụng đối tượng của lớp DecimalFormat. Trong đó mỗi chữ số sẽ được đánh dấu bởi dấu thăng phân tách phần nguyên, thập phân bởi dấu .
* Ví dụ:

double x = 2;

double y = 3;

double result = x / y; // kết quả 0.6666666(6)

System.out.println(Math.ceil(result)); // kq = 1.0, lấy phần nguyên được 1

System.out.println(Math.floor(result)); // kq = 0.0, lấy phần nguyên được 0

System.out.println(Math.round(result)); // kq = 1

// sử dụng DecimalFormat

NumberFormat numberFormat = new DecimalFormat("###.#########");

System.out.println(numberFormat.format(result));

// sử dụng printf và chuỗi chuyển đổi định dạng:

System.out.printf("%10.9f", result);

### Ví dụ minh họa

import java.text.DecimalFormat;

import java.text.NumberFormat;

public class Lesson5 {

public static void main(String[] args) {

double x = 2;

double y = 3;

double result = x / y; // kết quả 0.6666666(6)

System.out.println(Math.ceil(result)); // kq = 1

System.out.println(Math.floor(result)); // kq = 0

System.out.println(Math.round(result)); // kq = 1

// sử dụng DecimalFormat

// NumberFormat là kiểu cha của DecimalFormat

NumberFormat numberFormat = new DecimalFormat("###.#########");

System.out.println(numberFormat.format(result));

// sử dụng printf và chuỗi chuyển đổi định dạng:

System.out.printf("%10.9f", result);

}

}

**Bài 2.6. Tìm hiểu lớp String**

**Nội dung bài học**

1. [Bản chất của String](https://braniumacademy.net/courses/java/lessons/java-string/#s1)
2. [Phép nối chuỗi](https://braniumacademy.net/courses/java/lessons/java-string/#s2)
3. [String template Java 21+](https://braniumacademy.net/courses/java/lessons/java-string/#s8)
4. [Các kí tự đặc biệt](https://braniumacademy.net/courses/java/lessons/java-string/#s3)
5. [Phép cộng String với các giá trị số](https://braniumacademy.net/courses/java/lessons/java-string/#s4)
6. [Các phương thức của lớp String](https://braniumacademy.net/courses/java/lessons/java-string/#s5)
7. [Nhập dữ liệu từ bàn phím](https://braniumacademy.net/courses/java/lessons/java-string/#s6)
8. [Bài tập thực hành](https://braniumacademy.net/courses/java/lessons/java-string/#s7)

**Bản chất của String**

* Kiểu String dùng để lưu trữ chuỗi các kí tự trong ngôn ngữ lập trình Java. Chuỗi kí tự là tập các kí tự được sắp đặt theo một trật tự nhất định nhằm biểu đạt một ý nghĩa nào đó và đặt trong cặp nháy kép “”.
* Ví dụ: String message = “I Love You”;
* String là một kiểu bất biến (immutable) trong ngôn ngữ lập trình Java.
* Về bản chất nó là một final class.
* Kiểu bất biến là một kiểu mà giá trị của nó kể từ sau khi được gán thì không thể thay đổi.
* Khi tạo biến kiểu String thì đó là biến kiểu đối tượng(object) không phải biến của kiểu nguyên thủy. Do đó ta có thể thực hiện các thao tác gọi phương thức trên đối tượng của kiểu String.
* Ví dụ:

public class Lesson26 {

public static void main(String[] args) {

String fullName = "Lê Nguyễn Bảo Ngọc";

String upperCaseName = fullName.toUpperCase(); // viết hoa toàn bộ chuỗi kí tự

var lowerCaseName = fullName.toLowerCase(); // viết thường toàn bộ chuỗi kí tự

// hiển thị kết quả lên màn hình

System.out.println("Họ và tên: " + fullName);

System.out.println("Họ và tên viết HOA: " + upperCaseName);

System.out.println("Họ và tên viết thường: " + lowerCaseName);

}

}

Họ và tên: Lê Nguyễn Bảo Ngọc

Họ và tên viết HOA: LÊ NGUYỄN BẢO NGỌC

Họ và tên viết thường: lê nguyễn bảo ngọc

**Phép nối chuỗi**

* Để nối chuỗi trong Java ta dùng toán tử +. Khi nối chuỗi thì chuỗi được nối, tức sau dấu + sẽ cắm vào cuối chuỗi trước đó.
* Chuỗi kết quả sau khi nối là một đối tượng String mới được tạo ra với nội dung chứa cả 2 chuỗi được nối.
* Tính đến phiên bản Java 21 ta có ít nhất 4 cách nối chuỗi kí tự:
  + Sử dụng toán tử +.
  + Sử dụng phương thức concat().
  + Sử dụng string template. Chi tiết về string template ta tìm hiểu ở mục kế tiếp.
  + Sử dụng StringBuilder/StringBuffer. Chi tiết ta tìm hiểu ở một bài học gần cuối chương này.
* Ví dụ:

public class Lesson26 {

public static void main(String[] args) {

var first = "Ngọc";

var last = "Lê";

var middle = "Nguyễn Bảo";

// cách 1: dùng toán tử +

var fullName = last + " " + middle + " " + first;

System.out.println(fullName);

// cách 2: sử dụng string template

fullName = STR."\{last} \{middle} \{first}";

System.out.println(fullName);

// cách 3: sử dụng phương thức nối chuỗi

fullName = last.concat(" ").concat(middle).concat(" ").concat(first);

System.out.println(fullName);

}

}

Lê Nguyễn Bảo Ngọc

Lê Nguyễn Bảo Ngọc

Lê Nguyễn Bảo Ngọc

**String template Java 21+**

* Từ phiên bản **Java 21** trở về sau ta có thể sử dụng String template để nhúng các biến vào trong String. Các phiên bản Java từ 20 trở về trước KHÔNG HỖ TRỢ cú pháp này.
* Để kích hoạt string template trên Java 21(bản preview) và nếu các phiên bản Java cao hơn vẫn chưa hỗ trợ chính thức thì ta làm như sau:
  + B1: gõ code chứa string template.
  + B2: trỏ chuột lên dòng code báo lỗi của string template và chọn mục đóng khung như trong hình:

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, hàng

Mô tả được tạo tự động

* Nếu các phiên bản Java cao hơn 21 đã hỗ trợ chính thức string template thì ta chỉ việc sử dụng trực tiếp mà không cần thiết lập language level preview.
* Cú pháp của string template: **“\{expression}”**. Trong đó expression có thể là các biến, các biểu thức, lời gọi phương thức, giá trị cụ thể…
* Lưu ý rằng biểu thức \{…} đặt trong giá trị của kiểu String.
* Ví dụ đoạn chương trình trên có thể viết lại sử dụng String template như sau:

public class Lesson26 {

public static void main(String[] args) {

String fullName = "Lê Nguyễn Bảo Ngọc";

String upperCaseName = fullName.toUpperCase(); // viết hoa toàn bộ chuỗi kí tự

var lowerCaseName = fullName.toLowerCase(); // viết thường toàn bộ chuỗi kí tự

// hiển thị kết quả lên màn hình, sử dụng string template

System.out.println(STR."Họ và tên: \{fullName}");

System.out.println(STR."Họ và tên viết HOA: \{upperCaseName}");

System.out.println(STR."Họ và tên viết thường: \{lowerCaseName}");

}

}

Họ và tên: Lê Nguyễn Bảo Ngọc

Họ và tên viết HOA: LÊ NGUYỄN BẢO NGỌC

Họ và tên viết thường: lê nguyễn bảo ngọc

**Các kí tự đặc biệt**

* Trong ví dụ trên ta thấy rằng, để hiển thị ra dấu nháy kép “ thì ta phải cho thêm dấu \ trước “ thành \”. Đó là ví dụ về các kí tự đặc biệt trong Java. Nội dung của String trong Java quy ước nằm trong cặp “” nên nếu bạn muốn in ra “ mà không cung cấp thêm \ ở trước thì sẽ bị lỗi.
* Sau đây là một số kí tự đặc biệt khác trong ngôn ngữ Java:

|  |  |
| --- | --- |
| **Kí tự đặc biệt** | **Mô tả chức năng** |
| \’ | Đại diện cho kí tự ‘ |
| \” | Đại diện cho kí tự “ |
| \\ | Đại diện cho kí tự \ |
| \n | Xuống dòng mới |
| \t | Tạo một tab ngang thường tương đương với 4 kí tự |

* Ví dụ minh họa sử dụng các kí tự đặc biệt trên:

public class Lesson26 {

public static void main(String[] args) {

// một số kí tự đặc biệt

var someMessage = "Xin chào bạn 'Hương'";

System.out.println(someMessage);

someMessage = "Xin chào bạn \"Hương\"";

System.out.println(someMessage);

// \n

var message = "Where\nare\nyou\nnow?";

System.out.println(message);

var tab = "\t";

// \t

message = STR."Welcome\{tab}to\{tab}Branium\{tab}Academy!";

System.out.println(message);

// \\

var path = "C:\\Users\\trieu\\Downloads\\IT\_Report\_2023\_Vie.pdf";

System.out.println(path);

}

}

Xin chào bạn 'Hương'

Xin chào bạn "Hương"

Where

are

you

now?

Welcome to Branium Academy!

C:\Users\trieu\Downloads\IT\_Report\_2023\_Vie.pdf

**Phép cộng String với các giá trị số**

* Khi thực hiện phép cộng + với String ta phải hết sức lưu ý. Bởi vì mặc định chương trình Java coi rằng nếu có 1 toán hạng của phép + là String thì đó là phép nối chuỗi. Không phải phép cộng hai số.
* Khi ta cộng số với String, kết quả nhận được là String. Do đó ví dụ sau cho kết quả không như ta mong muốn:

int numberA = 200;

int numberB = 500;

String result = "numberA + numberB = " + numberA + numberB;

// Mong muốn: numberA + numberB = 700

System.out.println(result);

// Thực tế: numberA + numberB = 200500

* Để có được kết quả chính xác ta sử dụng cặp ngoặc tròn ( ) gói cụm biểu thức cần ưu tiên thực hiện lại. Ví dụ sau đã sửa lại:

int numberA = 200;

int numberB = 500;

String result = "numberA + numberB = " + (numberA + numberB);

// Mong muốn: numberA + numberB = 700

System.out.println(result);

// Thực tế: numberA + numberB = 700

**Các phương thức của lớp String**

* Lưu ý rằng vị trí của kí tự trong chuỗi, mảng của Java luôn bắt đầu từ 0. Do đó phần tử đầu tiên của chuỗi kí tự là vị trí 0, phần tử thứ hai bắt đầu từ vị trí 1… Phần tử cuối cùng của chuỗi có chỉ số là N – 1 với N là độ dài chuỗi lấy từ phương thức length().
* Sau đây là một số phương thức thường dùng của lớp String và ý nghĩa của nó:

|  |  |
| --- | --- |
| **Phương thức** | **Ý nghĩa sử dụng** |
| **length()** | Cho biết độ dài của chuỗi tính theo số kí tự. Luôn >= 0 |
| **toUpperCase()** | Viết hoa toàn bộ chuỗi |
| **toLowerCase()** | Viết thường toàn bộ chuỗi |
| **indexOf()** | Tìm vị trí đầu tiên của một kí tự/một chuỗi con trong chuỗi kí tự |
| **lastIndexOf()** | Tìm vị trí cuối cùng của một kí tự/chuỗi con trong chuỗi gốc |
| **charAt(index)** | Lấy ra kí tự tại vị trí index. Giá trị index phải >= 0 và < length() của chuỗi đang xét |
| **compareTo(other)** | So sánh sự tương đương có phân biệt chữ hoa, chữ thường về mặt giá trị của hai chuỗi. Kết quả trả về giá trị số âm, 0 hoặc dương tùy vào mức độ tương quan của hai chuỗi. |
| **compareToIgnoreCase(other)** | So sánh hai chuỗi về mặt giá trị. Không phân biệt chữ hoa chữ thường. Kết quả trả về giá trị âm nếu chuỗi hiện thời đứng trước other. Bằng 0 nếu hai chuỗi cùng giá trị. Dương nếu chuỗi hiện thời đứng sau chuỗi other. |
| **trim()** | Loại bỏ kí tự khoảng trắng ở đầu và cuối chuỗi. Khoảng trắng là các kí tự dấu cách, dấu tab, xuống dòng. |
| **replace(old, newStr)** | Thay thế kí chuỗi kí tự old bằng kí tự/chuỗi kí tự newStr. Thực hiện thay thế toàn bộ chuỗi old bằng newStr xuất hiện trong chuỗi. |
| **replaceAll(regex, newStr)** | Thay thế tất cả các chuỗi con thỏa mãn regex trong chuỗi gốc bằng newStr. |
| **split(regex)** | Tách chuỗi và trả về mảng String sau khi tách. regex là mẫu kí tự so khớp cần để xác định điều kiện tách chuỗi. Ví dụ regex =”\\s+” chỉ định tách chuỗi tại vị trí có 1 hoặc nhiều dấu cách. Chi tiết về regex và các quy tắc so khớp bạn tìm hiểu trong chương**Regular Expression**. |
| **substring(startIndex)** | Trích xuất chuỗi con của chuỗi gốc từ vị trí startIndex đến hết chuỗi gốc. |
| **substring(start, end)** | Trích xuất chuỗi con của chuỗi gốc từ vị trí start đến trước vị trí end. |
| **isEmpty()** | Kiểm tra xem chuỗi hiện thời có rỗng hay không. Kết quả nhận được là true nếu độ dài chuỗi bằng 0 và ngược lại kết quả là false. |
| **isBlank()** | Kiểm tra xem chuỗi hiện thời có rỗng hoặc chỉ chứa các kí tự khoảng trắng hay không. Kí tự khoảng trắng là các kí tự như dấu cách, dấu tab, xuống dòng. Như vậy một chuỗi là blank chưa chắc empty. |

* Để biết thêm các phương thức khác bạn để con trỏ chuột sau tên đối tượng String. Nhấn phím . sẽ hiện ra lúc này bạn chọn phương thức và ấn **Ctrl, click chuột trái vào tên phương thức** để xem mô tả chức năng chi tiết.
* Ví dụ 1:

public class Lesson26 {

public static void main(String[] args) {

var fullName = "lê nguyễn hoàng long";

System.out.println(STR."Độ dài của fullName: \{fullName.length()}");

System.out.println(STR."Viết hoa: \{fullName.toUpperCase()}");

System.out.println(STR."Viết thường: \{fullName.toLowerCase()}");

System.out.println(STR."Vị trí đầu tiên của kí tự 'o': \{fullName.indexOf('o')}");

System.out.println(STR."Vị trí cuối cùng của kí tự 'g': \{fullName.lastIndexOf('g')}");

System.out.println(STR."Kí tự tại vị trí 5: \{fullName.charAt(5)}");

var newFullName = fullName.replace('l', 'L');

System.out.println(STR."Họ tên sau khi thay thế: \{newFullName}");

var message = " Phong Vân ";

System.out.println(STR."Sau khi loại bỏ khoảng trắng thừa: \"\{message.trim()}\"");

var words = fullName.split("\\s+");

System.out.println(STR."Số từ trong tên: \{words.length}");

}

}

Độ dài của fullName: 32

Viết hoa: LÊ NGUYỄN HOÀNG LONG

Viết thường: lê nguyễn hoàng long

Vị trí đầu tiên của kí tự 'o': 20

Vị trí cuối cùng của kí tự 'g': 31

Kí tự tại vị trí 5:

Họ tên sau khi thay thế: Lê nguyễn hoàng Long

Sau khi loại bỏ khoảng trắng thừa: "Phong Vân"

Số từ trong tên: 4

* Ví dụ 2: so sánh hai chuỗi kí tự với nhau.

String nam = "Nam";

String other = "nam";

System.out.println(nam.compareTo(other));

System.out.println(nam.compareToIgnoreCase(other));

// kết quả:

-32

0

**Nhập dữ liệu từ bàn phím**

* Để nhập dữ liệu cho chuỗi kí tự từ bàn phím. Ta dùng phương thức sau:
  + next(): Đọc vào từng từ. Việc đọc sẽ dừng khi gặp khoảng trắng như dấu cách, dấu tab, hoặc ấn phím enter.
  + nextLine(): Đọc vào cả dòng. Việc đọc sẽ dừng khi bạn ấn phím enter.
* **Chú ý: khi đọc một số, một từ hay một kí tự mà sau đó ta đọc cả dòng, cần phải đọc bỏ kí tự thừa của lần đọc trước đó với nextLine(). Tức là phải nextLine() hai lần.**
* Ví dụ:

Scanner input = new Scanner(System.in);

String name;

System.out.println("Hey, what your name?");

name = input.nextLine();

System.out.println("Hello " + name + "!");

**Bài 2.7. Kiểu tự suy luận**

**Nội dung bài học**

1. [Khái niệm](https://braniumacademy.net/courses/java/lessons/java-type-inference/#s1)
2. [Cú pháp](https://braniumacademy.net/courses/java/lessons/java-type-inference/#s2)
3. [Sử dụng trong trường hợp nào?](https://braniumacademy.net/courses/java/lessons/java-type-inference/#s3)
4. [Ví dụ minh họa](https://braniumacademy.net/courses/java/lessons/java-type-inference/#s4)

**Khái niệm**

* Kiểu tự suy luận là kiểu dữ liệu của một biến được chương trình Java tự động xác định dựa trên giá trị mà ta gán cho biến.
* Kiểu tự suy luận trong Java được khai báo với keyword *var*.
* Ví dụ:

var name = "Thanh Pham"; // kiểu của name là String

var x = 100; // kiểu của x là int

var f = 3.125f; // kiểu của f là float

var d = 5.687; // kiểu của d là double

var other = true; // kiểu của other là boolean

**Cú pháp**

* Cú pháp tổng quát của khai báo: ***var name = value;***
* Trong đó:
  + var là keyword
  + Name là tên biến, đặt theo quy tắc đặt tên biến
  + Trong khai báo bắt buộc phải có thao tác khởi tạo giá trị
  + Value là giá trị gán cho biến. Chương trình sẽ dựa vào giá trị này để xác định kiểu cho biến.
* Các giới hạn:
  + Chỉ khai báo duy nhất 1 biến với 1 keyword *var*.
  + Bắt buộc phải khởi tạo giá trị cho biến ngay sau khi ta khai báo biến với keyword var.
  + Giá trị khởi tạo bắt buộc phải khác null. Nếu bạn gán giá trị null, chương trình sẽ báo lỗi vì giá trị null không đủ thông tin để xác định kiểu của biến.
  + Không được chứa tham chiếu đến chính nó trong khi khởi tạo giá trị.
  + Không được khai báo biến kiểu mảng với var.
  + Không được gán cú pháp khởi tạo mảng cho biến của var.
  + Không sử dụng var như là kiểu trả về của phương thức(học sau).
  + Không sử dụng var làm kiểu của tham số(học sau).
* Sau khi khai báo biến với var thì kiểu của biến không thể thay đổi. Nhưng giá trị của biến có thể thay đổi.
* Ví dụ:

var scanner = new Scanner(System.in);

var age = scanner.nextInt();

var fullName = scanner.nextLine();

var address = scanner.nextLine();

var gpa = scanner.nextFloat();

var pi = scanner.nextDouble();

var email = "abc@xmail.com"; // ok

var temperature = 15.25f; // ok

* Làm như sau sẽ gây lỗi:

var arr[] = new int[10]; // error: khai báo kiểu mảng với var

var y = (y = 99); // error: chứa cú pháp tự tham chiếu trong ()

var something = {1, 2, 3}; // không hợp lệ vì chứa cú pháp khởi tạo mảng

var b = null; // không thể xác định được kiểu dữ liệu từ giá trị null

* Để kiểm tra kiểu của một biến sau khi tạo biến với var, ta dùng cú pháp:

System.out.println(((Object)name).getClass().getName());

* Trong đó name là tên biến cần kiểm tra.
* Kết quả sẽ cho chúng ta tên lớp bao của kiểu tương ứng nếu kiểu của name là nguyên thủy. Và tên kiểu của biến name nếu kiểu nó không phải kiểu nguyên thủy.
* Ta có thể sử dụng keyword var để khai báo hằng số cục bộ:

final var MAX\_AGE = 125; // hằng số kiểu int

final var MAX\_LENGTH = 50L; // hằng số kiểu long

final var MAX\_GPA = 4.0; // hằng số kiểu double

**Sử dụng trong trường hợp nào?**

* Ta thường sử dụng keyword *var* để khai báo các biến cục bộ.
* Biến cục bộ là biến khai báo trong một khối code ví dụ trong một phương thức, vòng lặp, cấu trúc ra quyết định, biểu thức lambda.
* Có thể dùng để thay thế các kiểu tường minh trong cú pháp khai báo biến mà có khởi tạo.
* Khi kiểu dữ liệu của biến không được biết trước.
* Ví dụ sử dụng trong vòng lặp:

/\*\*

\* Tính tổng các số nguyên tố trong mảng.

\*

\* @param numbers mảng chứa các giá trị số cần tính tổng.

\* @return giá trị tổng thỏa mãn.

\*/

private static long sumPrimeNumber(int[] numbers) {

var sum = 0;

for (var number : numbers) { // sử dụng kiểu tự suy luận var

if (isPrime(number)) {

sum += number;

}

}

return sum;

}

* Ví dụ sử dụng trong phương thức:

/\*\*

\* Phương thức kiểm tra số cho trước có phải là số nguyn tố hay không.

\* Cách thức kiểm tra: nếu trong đoạn [2, [căn n]] mà có ước số thì n không

\* phải là số nguyên tố.

\*

\* @param number số cần kiểm tra tính nguyên tố.

\* @return true nếu number là số nguyên tố và false trong trường hợp ngược lại.

\*/

private static boolean isPrime(int number) {

if (number < 2) { // số nhỏ hơn 2 KHÔNG bao giờ là số nguyên tố

return false;

}

// xét biên là phần nguyên căn bậc hai của n

var bound = (int) Math.sqrt(number);

// nếu n có ước số trong đoạn từ 2 -> bound

for (var i = 2; i <= bound; i++) {

if (number % i == 0) // thì n không phải số nguyên tố

return false;

}

return true; // nếu ko return ở trên thì n là nguyên tố

}

**Ví dụ minh họa**

import java.util.Scanner;

public class Lesson27 {

public static void main(String[] args) {

var name = "Thanh Pham"; // kiểu của name là String

var x = 100; // kiểu của x là int

var f = 3.125f; // kiểu của f là float

var d = 5.687; // kiểu của d là double

var other = true; // kiểu của other là boolean

var input = new Scanner(System.in); // ok

// var a; // không hợp lệ vì thiếu cú pháp khởi tạo

// var b = null; // không hợp lệ vì không thể xác định kiểu với giá trị null

// var p = 3.14, q = 20; // không hợp lệ vì khai báo nhiều hơn 1 biến

// var arr[] = new int[10]; // không hợp lệ vì khai báo kiểu mảng với var

// var y = (y = 99); // Không hợp lệ vì chứa cú pháp tự tham chiếu trong ()

// var something = {1, 2, 3}; // không hợp lệ vì chứa cú pháp khởi tạo mảng

System.out.println(((Object)input).getClass().getName());

}

}

**Bài 2.8. Tìm hiểu lớp Math**

**Nội dung bài học**

1. [Mục đích sử dụng](https://braniumacademy.net/courses/java/lessons/java-math-class/#s1)
2. [Cú pháp](https://braniumacademy.net/courses/java/lessons/java-math-class/#s2)
3. [Các phương thức thường dùng và mô tả](https://braniumacademy.net/courses/java/lessons/java-math-class/#s3)
4. [Ví dụ minh họa](https://braniumacademy.net/courses/java/lessons/java-math-class/#s4)
5. [Bài tập thực hành](https://braniumacademy.net/courses/java/lessons/java-math-class/#s5)

**Mục đích sử dụng**

* Lớp Math chứa các hằng số và các công thức tính toán toán học đã được xây dựng sẵn.
* Ta sử dụng các hằng số và công thức tính toán này khi muốn tính toán toán học.
* Ví dụ tính trị tuyệt đối, căn bậc hai, bậc ba, tính sin, cos…

**Cú pháp**

* Cú pháp tổng quát để sử dụng lớp Math:
  + Với các hằng số: Math.*tên\_thành\_phần*
  + Với các phương thức: Math.*phương\_thức(các tham số)*
* Ví dụ:

System.out.println(Math.E); // hiển thị số E

var r = 12.25; // bán kính đường tròn

double area = r \* r \* Math.PI; // sử dụng Math.PI

int x = Math.abs(-200); // lấy trị tuyệt đối của -200

**Các phương thức thường dùng và mô tả**

* Danh sách các phương thức thường dùng của lớp Math và mô tả:

|  |  |
| --- | --- |
| **Phương thức** | **Mô tả** |
| **abs(v)** | Lấy trị tuyệt đối của v. Kiểu của v là các kiểu số trong Java. |
| **sqrt(v)** | Tính căn bậc hai của v. Giá trị của v phải là số dương. |
| **cbrt(v)** | Tính căn bậc ba của v. Kiểu của v là các kiểu số. |
| **sin(x)** | Tính sin của x. Giá trị của x ở hệ radian. |
| **cos(x)** | Tính cos của x. Giá trị của x ở hệ radian. |
| **tan(x)** | Tính tan của x. Giá trị của x ở hệ radian. |
| **asin(a)** | Tính góc x từ giá trị a = sin(x). Kết quả ở hệ radian. |
| **ceil(x)** | Làm tròn giá trị số x lên số nguyên gần nhất. Kết quả ở dạng double. |
| **floor(x)** | Làm tròn giá trị số x xuống số nguyên gần nhất. Kết quả ở dạng double. |
| **round(x)** | Làm tròn tự động giá trị số x đến số nguyên gần nhất. Kết quả ở dạng double. |
| **log(x)** | Tính logrit cơ số tự nhiên(e) của x. |
| **exp(x)** | Tính e^x. Với e = Math.E |
| **log10(x)** | Tính logarit cơ số 10 của x. |
| **min(a, b)** | Tìm giá trị min trong hai số a, b. |
| **max(a, b)** | Tìm giá trị max trong hai số a, b. |
| **pow(a, b)** | Tính a^b. |
| **random()** | Sinh số ngẫu nhiên trong đoạn từ 0.0 đến < 1.0 |
| **toRadians(x)** | Chuyển x từ độ sang radian |
| **toDegrees(x)** | Chuyển x từ radian sang độ |

**Ví dụ minh họa**

public class Lesson28 {

public static void main(String[] args) {

int x = -200;

int newX = Math.abs(x);

System.out.println(STR."newX = \{newX}");

float degree = 90;

System.out.println(Math.toRadians(degree));

System.out.println(Math.toDegrees(Math.PI / 2));

System.out.println(STR."sin(PI / 3) = \{Math.sin(Math.PI / 3)}");

System.out.println(STR."cos(PI / 3) = \{Math.cos(Math.PI / 3)}");

System.out.println(STR."tan(PI / 3) = \{Math.tan(Math.PI / 3)}");

var radius = 125.66;

var perimeter = 2 \* Math.PI \* radius;

var area = Math.PI \* Math.pow(radius, 2);

System.out.println(STR."Chu vi đường tròn bán kính \{radius} = \{perimeter}");

System.out.println(STR."Diện tích đường tròn bán kính \{radius} = \{area}");

var a = 200;

var b = 500;

System.out.println(STR."min(a, b) = \{Math.min(a, b)}");

System.out.println(STR."max(a, b) = \{Math.max(a, b)}");

}

}

* Kết quả:

newX = 200

1.5707963267948966

90.0

sin(PI / 3) = 0.8660254037844386

cos(PI / 3) = 0.5000000000000001

tan(PI / 3) = 1.7320508075688767

Chu vi đường tròn bán kính 125.66 = 789.5450657001868

Diện tích đường tròn bán kính 125.66 = 49607.116477942734

min(a, b) = 200

max(a, b) = 500

**Bài 2.9. Các định dạng dữ liệu thường gặp trong Java**

**Nội dung bài học**

1. [Các định dạng thường gặp](https://braniumacademy.net/courses/java/lessons/java-format-specifiers/#s1)
2. [Ví dụ minh họa](https://braniumacademy.net/courses/java/lessons/java-format-specifiers/#s2)

**Các định dạng thường gặp**

* Các định dạng này thường gặp khi ta in dữ liệu ra màn hình, ghi file… và áp dụng cho các phương thức như printf(), String.format().
* Cú pháp sử dụng: printf(“format”, valueToPrint); trong đó lưu ý rằng format là định dạng luôn để trong nháy kép “”.
* Sau đây là các định dạng và ý nghĩa:
  + “%d”: in ra các số nguyên.
  + “%,d”: in ra các số nguyên, phân tách phần nghìn, triệu, tỉ.. bằng dấu phẩy.
  + “%10d”: dành 10 vị trí ứng với 10 kí tự cho việc in ra 1 số nguyên, căn lề phải.
  + “%-10d”: dành 10 vị trí ứng với 10 kí tự để in ra giá trị số nguyên, căn lề trái. Bạn có thể thay 10 bởi giá trị số nguyên bất kỳ.
  + “%+d”: thêm dấu vào trước số nguyên cần in ra. Ví dụ +100, -200.
  + “%o”: in ra số nguyên n ở hệ cơ số 8. Các giá trị biểu diễn chỉ gồm từ 0-7.
  + “%x”: in ra số nguyên n ở hệ cơ số 16. Các giá trị biểu diễn chỉ gồm các kí tự 0-9, a,b,c,d,e,f và là các chữ cái thường.
  + “%X”: tương tự %x nhưng các kí tự chữ cái là chữ hoa ABCDEF.
  + “%h” hoặc “%H”: in ra giá trị mã băm tương ứng với giá trị đối số ở dạng hexa, biểu diễn bởi chữ cái thường như %x hoặc chữ cái hoa ở %X.
  + “%c”: in ra một kí tự.
  + “%f”: định dạng số thực.
  + “%10f”: dành tối thiểu 10 vị trí để in ra giá trị số thực n. Căn lề phải.
  + “%-10f”: dành tối thiểu 10 vị trí để in ra giá trị số thực n. Căn lề trái.
  + “%-10.2f”: dành tối thiểu 10 vị trí để in ra giá trị số thực n, làm tròn 2 chữ số phần thập phân. Căn lề trái.
  + “%e” và “%E”: định dạng số thực số mũ dạng khoa học. Ví dụ 9.31e-30.
  + “%b” hoặc “%B” định dạng giá trị kiểu boolean.
  + “%%” chèn kí tự % vào kết quả.
  + “%n” in xuống dòng.
  + “%s”: định dạng của chuỗi kí tự string.
  + “%25s”: dành ra tối thiểu 25 kí tự in ra chuỗi string, căn lề phải.
  + “%-25s”: dành ra tối thiểu 25 kí tự in ra chuỗi string, căn lề trái.
* Một số định dạng thời gian:
  + “%ty”: lấy phần hai số cuối của năm. Ví dụ 2024 -> 24.
  + “%tY”: lấy đầy đủ phần năm 4 số, ví dụ 2025.
  + “%tb”: lấy tên tháng viết tắt, ví dụ December -> Dec.
  + “%tB”: lấy tên tháng đầy đủ trong tiếng Anh: December.
  + “%tm”: tên tháng dạng số. 1-12.
  + “%td”: lấy ngày của tháng, ví dụ các ngày từ 1-31 của tháng tương ứng.
  + “%tD”: lấy ngày tháng năm dưới dạng 19/01/25.
  + “%tH”: lấy giờ dạng 24 giờ. Trả về giá trị 0-23.
  + “%tM”: lấy phút của thời gian hiện tại. Giá trị trả về 0-59.
  + “%tS”: lấy giây của thời gian hiện tại. Giá trị 0-59.
  + “%tL”: lấy mili giây của thời gian hiện tại. Giá trị từ 0-999.
* Trong tất cả các định dạng trên, bạn có thể thay thế các con số trong đó thành những con số bạn mong muốn sao cho phù hợp.
* Trên đây chỉ là một số định dạng cơ bản, ngoài ra còn rất nhiều các định dạng khác bạn có thể tham khảo thêm[tại đây](https://docs.oracle.com/javase/8/docs/api/java/util/Formatter.html).

**Ví dụ minh họa**

* Ví dụ 1: định dạng các số nguyên.

public class Lesson29Ex1 {

public static void main(String[] args) {

var number = 9999999999L;

// in ra số nguyên:

System.out.println("Giá trị gốc: ");

var result = String.format("%d", number);

System.out.println(result);

System.out.printf("%d%n", number);

// dành 10 vị trí in ra số nguyên number

System.out.printf("%10d%n", number);

System.out.printf("%-10d%n", number); // căn lề trái

// in ra có dấu

System.out.printf("%+d%n", number);

// in ra có phân tách phần nghìn

System.out.printf("%,+-15d%n", number);

// in ra số nguyên ở dạng biểu diễn cơ số 8, 16, mã băm

System.out.println("Chuyển đổi sang hệ cơ số 8, 16, hash code");

System.out.printf("Hệ 8: %o%n", number);

System.out.printf("Hệ 16: %x%n", number);

System.out.printf("Hệ 16: %X%n", number); // viết hoa

System.out.printf("Hash code: %h%n", number);

System.out.printf("Hash code: %H%n", number); // mã băm cơ số 16 viết hoa

}

}

Giá trị gốc:

9999999999

9999999999

9999999999

9999999999

+9999999999

+9,999,999,999

Chuyển đổi sang hệ cơ số 8, 16, hash code

Hệ 8: 112402761777

Hệ 16: 2540be3ff

Hệ 16: 2540BE3FF

Hash code: 540be3fd

Hash code: 540BE3FD

* Ví dụ 2: định dạng số thực.

public class Lesson29Ex2 {

public static void main(String[] args) {

var number = 3.1415926;

// in ra giá trị gốc:

System.out.printf("Giá trị gốc: %f\n", number);

// làm tròn số và căn lề

System.out.printf("Làm tròn đến 2 chữ số: %10.2f\n", number);

System.out.printf("Làm tròn đến 3 chữ số: %-10.3f\n", number);

// sử dụng biểu diễn khoa học cho số thực:

System.out.printf("%e%n", number);

var result = String.format("%E", number);

System.out.println(result);

}

}

Giá trị gốc: 3.141593

Làm tròn đến 2 chữ số: 3.14

Làm tròn đến 3 chữ số: 3.142

3.141593e+00

3.141593E+00

* Ví dụ 3: định dạng chuỗi kí tự.

public class Lesson29Ex3 {

public static void main(String[] args) {

var fullName = "Họ và tên";

var address = "Địa chỉ";

var gpa = "Điểm Gpa";

var result = String.format("%-35s%-30s%-15s", fullName, address, gpa);

System.out.println(result);

System.out.printf("%-35s%-30s%-15.2f%n", "Lê Công Tuấn", "Hà Nội", 3.66);

System.out.printf("%-35s%-30s%-15.2f%n", "Hoàng Văn Nam", "Đà Nẵng", 3.24);

System.out.printf("%-35s%-30s%-15.2f%n", "Nguyễn Ngọc Mai", "Tp.Hồ Chí Minh", 3.66);

}

}

Họ và tên Địa chỉ Điểm Gpa

Lê Công Tuấn Hà Nội 3.66

Hoàng Văn Nam Đà Nẵng 3.24

Nguyễn Ngọc Mai Tp.Hồ Chí Minh 3.66

* Ví dụ 4: định dạng thời gian.

import java.time.LocalDateTime;

public class Lesson29Ex4 {

public static void main(String[] args) {

var currentTime = LocalDateTime.now();

System.out.printf("Năm: %ty%n", currentTime);

System.out.printf("Năm: %tY\n", currentTime);

System.out.printf("Tháng: %tb\n", currentTime);

System.out.printf("Tháng: %tB\n", currentTime);

System.out.printf("Ngày: %td\n", currentTime);

System.out.printf("Ngày: %tD\n", currentTime);

System.out.printf("Giờ: %tH\n", currentTime);

System.out.printf("Phút: %tM\n", currentTime);

System.out.printf("Giây: %tS\n", currentTime);

System.out.printf("Mili giây: %tL\n", currentTime);

System.out.printf("%td %tm %tY %tH:%tM:%tS.%tL",

currentTime, currentTime, currentTime,

currentTime, currentTime, currentTime,

currentTime);

}

}

Năm: 23

Năm: 2023

Tháng: Dec

Tháng: December

Ngày: 17

Ngày: 12/17/23

Giờ: 20

Phút: 47

Giây: 49

Mili giây: 975

17 12 2023 20:47:49.975

**Bài 2.10. StringBuilder và StringBuffer**

**Nội dung bài học**

1. [Giới thiệu về StringBuilder](https://braniumacademy.net/courses/java/lessons/java-stringbuilder-vs-stringbuffer/#s1)
2. [Các phương thức thường gặp](https://braniumacademy.net/courses/java/lessons/java-stringbuilder-vs-stringbuffer/#s2)
3. [Giới thiệu về StringBuffer](https://braniumacademy.net/courses/java/lessons/java-stringbuilder-vs-stringbuffer/#s3)
4. [Các phương thức thường dùng](https://braniumacademy.net/courses/java/lessons/java-stringbuilder-vs-stringbuffer/#s4)

**Giới thiệu về StringBuilder**

* Lớp chứa một chuỗi các kí tự có thể sửa đổi được(mutable). Lớp này cung cấp một API tương thích với StringBuffer nhưng không đảm bảo tính đồng bộ hóa.
* Lớp StringBuilder được thiết kế để sử dụng trong trường hợp đơn luồng. Khi có thể, ta nên sử dụng lớp này thay vì StringBuffer vì tốc độ của StringBuilder sẽ nhanh hơn trong hầu hết các triển khai.
* Các phương thức chính của StringBuilder xoay quanh việc chèn thêm phần tử và nối phần tử vào cuối. Các phương thức tương ứng của chúng là insert() và append() được nạp chồng để có thể hoạt động với bất kì kiểu dữ liệu nào.
* Cả hai phương thức trên đều chuyển đổi một cách hiệu quả dữ liệu được cung cấp sang string sau đó chèn hoặc thêm vào nội tại của đối tượng StringBuilder.
* Mỗi StringBuilder có một khả năng chứa(capacity) nhất định. Chừng nào các kí tự chứa trong StringBuilder chưa vượt quá giới hạn này thì giá trị capacity chưa cần cấp phát để mở rộng.
* Nếu bộ đệm nội tại bị tràn, nó sẽ tự động được nới ra với kích thước lớn hơn.
* Khuyến nghị chỉ sử dụng StringBuilder trong trường hợp xử lý đơn luồng, với trường hợp đa luồng ta sử dụng StringBuffer.
* Sử dụng StringBuilder khi ta muốn sửa đổi chuỗi kí tự: thêm, sửa xóa nội dung của chuỗi kí tự. Khi thực hiện, các thao tác này diễn ra trên cùng đối tượng StringBuilder.

**Các phương thức thường gặp**

* Đầu tiên là các phương thức khởi tạo: dùng để tạo đối tượng StringBuilder với các thông số khác nhau:
  + **StringBuilder()**: tạo lập một đối tượng StringBuilder mới không chứa bất kì kí tự nào trong nội tại của nó. Mặc định phương thức này tạo một bộ đệm với kích thước 16 kí tự.
  + **StringBuilder(CharSequence seq)**: tạo đối tượng StringBuilder với bộ đệm chứa các kí tự như trong tham số kiểu CharSequence truyền vào.
  + **StringBuilder(int capacity)**: tạo đối tượng StringBuilder rỗng với khả năng chứa capacity kí tự.
  + **StringBuilder(String str)**: tạo một đối tượng StringBuilder với bộ đệm nội tại chứa nội dung của string cho trước.
* Sau đây là các phương thức thường gặp và mô tả:
  + **StringBuilder append(type param)**: nối nội dung string đại diện cho giá trị chứa trong param vào cuối bộ đệm chứa các kí tự của đối tượng StringBuilder hiện thời. Trong đó type có thể là các kiểu nguyên thủy, các lớp bao của kiểu nguyên thủy, mảng char[], CharSequence, Object, String, StringBuffer.
  + **StringBuilder append(char[] str, int offset, int len)**: thêm string đại diện của một mảng con của mảng str vào bộ đệm nội tại của đối tượng StringBuilder. Chuỗi con gồm len kí tự lấy trong str tính từ vị trí offset. Kích thước bộ đệm nội tại trong StringBuilder sẽ tăng lên một lượng bằng len.
  + **StringBuilder append(CharSequence s, int start, int end)**: nối một chuỗi con của s tính từ vị trí start đến trước end vào bộ đệm nội tại của đối tượng StringBuilder. Kích thước bộ đệm của StringBuilder tăng lên một lượng bằng end – start.
  + **int capacity()**: trả về khả năng chứa hiện tại của đối tượng StringBuilder.
  + **char charAt(int index)**: trả về kí tự tại vị trí index trong bộ đệm nội tại của StringBuilder.
  + **StringBuilder delete(int start, int end)**: xóa các kí tự từ vị trí start đến trước vị trí end trong bộ đệm nội tại của StringBuilder.
  + **StringBuilder deleteCharAt(int index)**: xóa kí tự tại vị trí chỉ định trong bộ đệm của StringBuilder.
  + **int indexOf(String str)**: trả về vị trí kí tự đầu tiên của chuỗi con đầu tiên trong StringBuilder có giá trị trùng với giá trị cho trong tham số str. Tức là vị trí của chuỗi con đầu tiên trong nội tại StringBuilder trùng với str. Không tìm thấy thì return -1.
  + **int indexOf(String str, int fromIndex)**: trả về vị trí kí tự đầu tiên của chuỗi con đầu tiên trong StringBuilder tính từ vị trí fromIndex có giá trị trùng với giá trị cho trong tham số str. Tức là vị trí của chuỗi con đầu tiên trong nội tại StringBuilder trùng với str. Không tìm thấy return -1.
  + **StringBuilder insert(int offset, type param)**: chèn nội dung trong tham số param vào nội tại StringBuilder tại vị trí offset. type có thể là bất kì kiểu dữ liệu nào trong số các kiểu nguyên thủy, các lớp bao của kiểu nguyên thủy, mảng char[], CharSequence, Object, String, StringBuffer.
  + **StringBuilder insert(int offset, char[] str, int offset, int len)**: chèn len kí tự từ vị trí offset trong str vào bộ đệm nội tại của đối tượng StringBuilder.
  + **StringBuilder insert(int offset, CharSequence seq, int start, int end)**: chèn các kí tự từ vị trí start đến trước end trong seq vào bộ đệm nội tại của đối tượng StringBuilder.
  + **int lastIndexOf(String str)**: trả về vị trí của kí tự đầu tiên của chuỗi con cuối cùng trong nội tại StringBuilder có giá trị trùng với giá trị tham số str. Nếu không tìm thấy return -1.
  + **int lastIndexOf(String str, int fromIndex)**: trả về vị trí của kí tự đầu tiên của chuỗi con cuối cùng trong nội tại StringBuilder tính từ vị trí fromIndex có giá trị trùng với giá trị tham số str. Nếu không tìm thấy return -1.
  + **int length()**: trả về số lượng kí tự của bộ đệm nội tại trong đối tượng StringBuilder. Giá trị luôn >= 0.
  + **StringBuilder replace(int start, int end, String str)**: thay thế các kí tự từ vị trí start đến trước end bằng nội dung trong str.
  + **StringBuilder reverse()**: đảo ngược nội dung các kí tự trong bộ đệm nội tại của StringBuilder.
  + **void setCharAt(int index, char ch)**: cập nhật giá trị kí tự tại vị trí index thành kí tự cho trong tham số ch.
  + **void setLength(int newLength)**: thiết lập độ dài mới cho các kí tự trong bộ đệm nội tại của đối tượng StringBuilder. newLength phải đảm bảo >= 0.
  + **CharSequence subSequence(int start, int end)**: trả về một chuỗi kí tự trích ra từ vị trí start đến trước end trong bộ đệm nội tại của StringBuilder.
  + **String subString(int start)**: trả về chuỗi con trong bộ đệm của StringBuilder từ vị trí start đến hết bộ đệm của đối tượng StringBuilder.
  + **String subString(int start, int end)**: trả về chuỗi con của các kí tự trong bộ đệm của StringBuilder từ vị trí start đến trước vị trí end.
  + **String toString()**: trả về một đối tượng String đại diện cho nội dung các kí tự lưu trong bộ đệm của StringBuilder.
  + **void trimToSize()**: giảm khả năng chứa kí tự của StringBuilder xuống. Cách thức hoạt động: nếu khả năng chứa  > số kí tự thực tế hiện có, nó tự động cắt giảm capacity cho phù hợp.
* **Ví dụ 1**: tạo họ và tên đầy đủ từ các thành phần đơn lẻ họ – đệm – tên.

import java.util.Scanner;

public class Lesson210 {

public static void main(String[] args) {

var scanner = new Scanner(System.in);

System.out.print("Họ: ");

var lastName = scanner.nextLine();

System.out.print("Đệm: ");

var midName = scanner.nextLine();

System.out.print("Tên: ");

var firstName = scanner.nextLine();

// tạo họ tên đầy đủ với StringBuilder

var nameBuilder = new StringBuilder();

nameBuilder.append(lastName)

.append(" ")

.append(midName)

.append(" ")

.append(firstName);

var fullName = nameBuilder.toString();

System.out.println(STR."Họ và tên: \{fullName}");

System.out.println(STR."Họ và tên viết Hoa: \{fullName.toUpperCase()}");

}

}

Họ: Nguyễn

Đệm: Trần Khánh

Tên: Vân

Họ và tên: Nguyễn Trần Khánh Vân

Họ và tên viết Hoa: NGUYỄN TRẦN KHÁNH VÂN

* Ví dụ 2: viết hoa chữ cái đầu từ của họ và tên, các kí tự khác viết thường.

import java.util.Scanner;

public class Lesson210 {

public static void main(String[] args) {

var scanner = new Scanner(System.in);

System.out.print("Họ: ");

var lastName = scanner.nextLine().trim();

System.out.print("Đệm: ");

var midName = scanner.nextLine().trim();

System.out.print("Tên: ");

var firstName = scanner.nextLine().trim();

// tạo họ tên đầy đủ với StringBuilder

var nameBuilder = new StringBuilder();

nameBuilder.append(lastName.toLowerCase())

.append(" ")

.append(midName.toLowerCase())

.append(" ")

.append(firstName.toLowerCase());

// viết hoa chữ cái đầu tiên

nameBuilder.setCharAt(0, Character.toUpperCase(nameBuilder.charAt(0)));

var previousIndex = 0;

while (true) {

var index = nameBuilder.indexOf(" ", previousIndex);

if (index == -1) {

break;

}

var character = nameBuilder.charAt(index + 1);

nameBuilder.setCharAt(index + 1, Character.toUpperCase(character));

previousIndex = index + 1;

}

System.out.println(STR."Họ và tên: \{nameBuilder.toString()}");

}

}

**Họ: nGuYỄn**

**Đệm: tRần kHánH**

**Tên: vÂN**

Họ và tên: Nguyễn Trần Khánh Vân

**Giới thiệu về StringBuffer**

* Lớp StringBuffer là một lớp an toàn luồng, có thể sửa đổi của một chuỗi các kí tự. Nó cơ bản giống như String nhưng có thể sửa đổi nội dung các kí tự lưu trong bộ đệm nội tại.
* Ta thường sử dụng lớp này để thực hiện việc sửa đổi chuỗi kí tự trong trường hợp đa luồng. Các phương thức được đồng bộ khi cần do đó tất cả các thao tác trên bất kì phiên bản luồng nào cũng sẽ hoạt động như thể chúng xuất hiện theo một thứ tự nhất quán với các lời gọi phương thức được thực hiện bởi từng luồng độc lập có liên quan.
* Giống như StringBuilder, StringBuffer thể hiện các tính năng chính của nó thông qua các thao tác chèn và nối chuỗi kí tự.

**Các phương thức thường dùng**

* Các phương thức của StringBuffer tương tự như StringBuilder. Ta thay các kiểu trả về là StringBuilder thành StringBuffer là xong.
* Sau đây là ví dụ sử dụng StringBuffer trong đa luồng:

public class Lesson210 {

public static void main(String[] args) {

var buffer = new StringBuffer();

var task1 = new MyRunnable(buffer, 700, "Hello ");

var task2 = new MyRunnable(buffer, 1500, "World!");

var thread1 = new Thread(task1);

var thread2 = new Thread(task2);

thread2.start();

thread1.start();

try {

thread2.join();

thread1.join();

} catch (InterruptedException e) {

System.out.println(e.getMessage());

}

System.out.println(STR."Kết quả: \{buffer}");

}

}

/\*\*

\* Lớp triển khai xử lý dữ liệu trong một luồng.

\*/

class MyRunnable implements Runnable {

private final StringBuffer buffer;

private final String message;

private final int duration;

public MyRunnable(StringBuffer buffer, int sleepDuration, String message) {

this.buffer = buffer;

this.message = message;

this.duration = sleepDuration;

}

@Override

public void run() {

synchronized (buffer) {

buffer.append(message);

System.out.println(buffer);

try {

Thread.sleep(duration);

buffer.append(message);

System.out.println(buffer);

} catch (InterruptedException e) {

throw new RuntimeException(e);

}

}

}

}

World!

World!World!

World!World!Hello

World!World!Hello Hello

Kết quả: World!World!Hello Hello

# Bài 3.1. Cấu trúc ra quyết định if-else

### Nội dung bài học

1. [Mục đích sử dụng](https://braniumacademy.net/courses/java/lessons/java-if-else/#s1)
2. [Cú pháp tổng quát](https://braniumacademy.net/courses/java/lessons/java-if-else/#s2)
3. [Sơ đồ khối](https://braniumacademy.net/courses/java/lessons/java-if-else/#s3)
4. [Ví dụ minh họa](https://braniumacademy.net/courses/java/lessons/java-if-else/#s4)
5. [Bài tập thực hành](https://braniumacademy.net/courses/java/lessons/java-if-else/#s5)

### Mục đích sử dụng

* Cấu trúc if hoặc if-else hoặc if-else-if dùng để đưa ra quyết định có thực hiện chương trình hay không.
* Nó còn có tên là cấu trúc rẽ nhánh, câu lệnh ra quyết định.
* Ví dụ vấn đề: nếu điểm trung bình >= 3.20 thì sinh viên A sẽ được học bổng. Như vậy ta sẽ có một điều kiện là điểm trung bình >= 3.20 để xét xem sinh viên A có đạt học bổng không. Nếu có thì ta quyết định trao học bổng cho sinh viên A. Ngược lại sinh viên A sẽ không được học bổng. Làm sao để chương trình biết mà thực hiện thay chúng ta?
* Khi nào thì sử dụng cấu trúc nào?
  + Nếu muốn chương trình chỉ thực hiện khi điều kiện thỏa mãn. Ta sử dụng cấu trúc if.
  + Nếu muốn khi điều kiện thỏa, thực hiện phương án A. Nếu không, thực hiện phương án B, ta dùng cấu trúc if-else.
  + Nếu có nhiều hơn 2 phương án để thực hiện và các phương án mang tính loại trừ nhau, dùng if-else-if.
* Loại trừ nhau có nghĩa là các điều kiện có sự ràng buộc với nhau, không là cái này thì có thể là cái kia.
  + Ví dụ: phân loại điểm chữ chỉ có thể là A, B, C, D, F. Xét từ cao xuống thấp, nếu không phải điểm A thì chỉ có thể là B trở xuống, nếu không phải A, B thì chỉ có thể là C trở xuống. Nếu không phải là A, B, C thì chỉ có thể là D trở xuống… Vậy với một mức điểm ta có nhiều khả năng và chỉ 1 trong số các khả năng là chính xác.
  + Ví dụ 2: tiền giấy hiện hành chỉ có các mệnh giá 1k, 2k, 5k, 10k, 20k, 50k, 100k, 200k, 500k. Vậy nếu người khác đưa cho ta một tờ tiền thật, tờ tiền đó chỉ có thể là một trong các mệnh giá đã liệt kê ở trên. Khi xét từ cao xuống thấp: nếu ko phải 500k thì chỉ có thể là 200k hoặc nhỏ hơn, nếu ko phải 200k thì chỉ có thể là 100k hoặc nhỏ hơn.
  + Như vậy ta thấy có mối liên hệ ràng buộc giữa các điều kiện và điều kiện đứng trước loại trừ khỏi các điều kiện đứng sau.

### Cú pháp tổng quát

#### Cú pháp tổng quát cấu trúc if:

**if (condition) {**

**// các câu lệnh cần thực hiện**

**}**

* Trong đó:
  + Luôn bắt đầu cấu trúc này với từ khóa if
  + Theo sau if là cặp ngoặc tròn **( )** và trong đó chứa điều kiện
  + *condiion*là các điều kiện cần thỏa mãn để phần thân if được thực hiện. Điều kiện của if luôn là giá trị hoặc biểu thức có tính đúng hoặc sai.
  + Phần thân if bao bởi cặp ngoặc **{ }** chứa các câu lệnh cần thực hiện nếu điều kiện thỏa mãn
  + Cấu trúc if kết thúc bởi dấu **}**
* Ví dụ 1:

import java.util.Scanner;

public class Lesson31 {

public static void main(String[] args) {

var scanner = new Scanner(System.in);

System.out.println(“Nhập số nguyên n: “);

var number = scanner.nextInt();

// kiểm tra xem number có phải số chẵn không

if (number % 2 == 0) {

System.out.println(STR.”\{number} là số chẵn.”);

}

// số lẻ

if (number % 2 == 1) {

System.out.println(STR.”\{number} là số lẻ.”);

}

// số âm

if (number < 0) {

System.out.println(STR.”\{number} là số âm.”);

}

// số dương

if (number > 0) {

System.out.println(STR.”\{number} là số dương.”);

}

System.out.println(“Nhập số nguyên k: “);

var k = scanner.nextInt();

// số chia hết cho k

if (number % k == 0) {

System.out.println(STR.”\{number} chia hết cho \{k}.”);

}

}

}

Nhập số nguyên n:

100

100 là số chẵn.

100 là số dương.

Nhập số nguyên k:

5

100 chia hết cho 5.

* Ví dụ 2:

public class Lesson9 {

public static void main(String[] args) {

float avgGrade = 3.75f;

if (avgGrade >= 3.2f) { // nếu điểm TB >= 3.2

// kết luận A đạt học bổng

System.out.println(“Sinh viên A đạt học bổng.”);

}

}

}

#### Cú pháp tổng quát cấu trúc if-else:

**if(condition) {**

**// làm gì đó nếu điều kiện thỏa mãn**

**} else {**

**// làm gì đó nếu điều kiện không thỏa**

**}**

* Trong đó các thành phần giống với if và thêm:
  + else là một từ khóa đi liền sau dấu ngoặc } của if trước nó.
  + Mỗi if chỉ có nhiều nhất 1 else.
  + Mỗi else chỉ thuộc về duy nhất 1 if liền trước nó.
  + Nếu condition không xảy ra thì nội dung khối else sẽ được thực hiện
* Ví dụ 1:

import java.util.Scanner;

public class Lesson31 {

public static void main(String[] args) {

var scanner = new Scanner(System.in);

System.out.println(“Nhập số nguyên n: “);

var number = scanner.nextInt();

// kiểm tra xem number có phải số chẵn không

if (number % 2 == 0) {

System.out.println(STR.”\{number} là số chẵn.”);

} else {

System.out.println(STR.”\{number} là số lẻ.”);

}

// số âm

if (number < 0) {

System.out.println(STR.”\{number} là số âm.”);

}

// số dương

if (number > 0) {

System.out.println(STR.”\{number} là số dương.”);

}

System.out.println(“Nhập số nguyên k: “);

var k = scanner.nextInt();

// số chia hết cho k

if (number % k == 0) {

System.out.println(STR.”\{number} chia hết cho \{k}.”);

} else {

System.out.println(STR.”\{number} KHÔNG chia hết cho \{k}.”);

}

}

}

Nhập số nguyên n:

-99

-99 là số lẻ.

-99 là số âm.

Nhập số nguyên k:

3

-99 chia hết cho 3.

* Ví dụ 2: Nếu sinh viên đạt điểm gpa >= 3.20 thì được học bổng. Nếu không thì được giấy chứng nhận. Điểm gpa là điểm trung bình tích lũy thường ở hệ 4.

Public class Lesson31 {

public static void main(String[] args) {

float gpa = 3.75f;

if (gpa >= 3.2f) { // nếu điểm TB >= 3.2

// kết luận A đạt học bổng

System.out.println(“Sinh viên A đạt học bổng.”);

} else {

// tặng cho sinh viên A cái giấy chứng nhận

System.out.println(“A đạt giấy chứng nhận.”);

}

}

}

#### Cú pháp tổng quát cấu trúc if-else-if:

**if(điều\_kiện1) {**

**// làm gì đó nếu điều\_kiện1 thỏa**

**} else if(điều\_kiện2) {**

**// làm gì đó nếu điều\_kiện2 thỏa**

**} else if(điều\_kiện3) {**

**// làm gì đó nếu điều\_kiện3 thỏa**

**} else {**

**// làm gì đó nếu tất cả điều kiện không thỏa**

**}**

* Trong đó giống cấu trúc if-else và thêm:
  + Cấu trúc else if nối liền sau cấu trúc if trước đó. Nếu không phải là else if mà chỉ là if thì nó trở về cấu trúc if đơn đầu tiên ở trên.
  + Cấu trúc else cuối cùng là tùy chọn, tức có thể có hoặc không tùy bài toán.
  + Nếu điều\_kiện1 thỏa, nội dung thân if đầu tiên được thực hiện.
  + Nếu không, biểu thức điều\_kiện2 sẽ được đánh giá. Nếu thỏa(tức true) thì khối if thứ hai sẽ được thực hiện.
  + Nếu không, biểu thức điều\_kiện3 sẽ được đánh giá. Nếu thỏa, khối if thứ 3 sẽ được thực hiện.
  + Tương tự đánh giá được thực hiện cho biểu thức if thứ n.
  + Sau cùng, nếu mọi điều kiện trên không thỏa. Khối code của vế else ở cuối(nếu có) sẽ được thực hiện.
* Ví dụ 1: kiểm tra số nguyên: một số nguyên chỉ có thể thuộc 3 loại là âm, dương hoặc bằng 0.

// kiểm tra số âm

if (number < 0) {

System.out.println(STR.”\{number} là số âm.”);

} else if (number > 0) { // số dương

System.out.println(STR.”\{number} là số dương.”);

} else { // không dấu

System.out.println(STR.”\{number} là số không âm, không dương.”);

}

* Ví dụ 2: Giả sử có 3 mức học bổng là Khá, Giỏi và Xuất sắc. Điều kiện điểm gpa của 3 mức lần lượt là:
  + Khá: >= 3.0 và < 3.2: thưởng 3.5 triệu đ.
  + Giỏi: >= 3.2 và < 3.6: thưởng 5.0 triệu đ.
  + Xuất sắc: >= 3.6 đến 4.0: thưởng 10 triệu đ.
  + Nếu không đạt mức nào thì thông báo không được học bổng.
  + Nếu điểm gpa > 4.0 hoặc < 0 thì điểm không hợp lệ.

import java.util.Scanner;

public class Lesson31 {

public static void main(String[] args) {

var scanner = new Scanner(System.in);

System.out.println(“Nhập điểm gpa của sinh viên: “);

var gpa = scanner.nextFloat();

if (gpa > 4.0f || gpa < 0) {

System.out.println(“Điểm gpa không hợp lệ!”);

} else if (gpa >= 3.6f) {

System.out.println(“Sinh viên đặt học bổng loại xuất sắc, trị giá 10.000.000đ”);

} else if (gpa >= 3.2f) {

System.out.println(“Sinh viên đặt học bổng loại giỏi, trị giá 5.000.000đ”);

} else if (gpa >= 3.0f) {

System.out.println(“Sinh viên đặt học bổng loại khá, trị giá 3.500.000đ”);

} else {

System.out.println(“Sinh viên không đạt học bổng.”);

}

}

}

Nhập điểm gpa của sinh viên:

0.5

Sinh viên không đạt học bổng.

Nhập điểm gpa của sinh viên:

3.78

Sinh viên đặt học bổng loại xuất sắc, trị giá 10.000.000đ

Nhập điểm gpa của sinh viên:

3.56

Sinh viên đặt học bổng loại giỏi, trị giá 5.000.000đ

Nhập điểm gpa của sinh viên:

3.15

Sinh viên đặt học bổng loại giỏi, trị giá 3.500.000đ

Nhập điểm gpa của sinh viên:

4.75

Điểm gpa không hợp lệ!

### Sơ đồ khối

* Các khối chức năng cơ bản:
  + **Hình thoi**: khối kiểm tra điều kiện.
  + **Hình bình hành**: khối nhập xuất dữ liệu.
  + **Hình chữ nhật**: thực hiện lệnh: gán, tính toán…
  + **Hình oval:** bắt đầu, kết thúc chương trình.
  + **Hình tròn nhỏ**: một điểm nào đó trong chương trình.
  + **Mũi tên**: hướng thực hiện các lệnh chương trình kế tiếp.

Ảnh có chứa văn bản, ảnh chụp màn hình, biểu đồ, thiết kế

Mô tả được tạo tự động

* Sơ đồ khối của cấu trúc if:

Ảnh có chứa văn bản, Phông chữ, ảnh chụp màn hình, hàng

Mô tả được tạo tự động

* Sơ đồ khối của cấu trúc if-else:

Ảnh có chứa văn bản, ảnh chụp màn hình, biểu đồ, hàng

Mô tả được tạo tự động

* Sơ đồ khối của cấu trúc if-else-if:

Ảnh có chứa văn bản, biểu đồ, ảnh chụp màn hình, hàng

Mô tả được tạo tự động

* Trong sơ đồ trên thể hiện rõ rằng trong khi xét các khối if thành phần nếu thấy điều kiện của if nào thỏa mãn thì phần thân của khối if đó sẽ được thực thi.
* Sau khi thực thi xong thì sẽ chuyển tới các đoạn lệnh bên dưới khối if-else-if. Không tiếp tục kiểm tra các if khác trong chuỗi lệnh if-else-if khi đã thấy một if được thỏa mãn.
* Sau cùng nếu không có bất kì điều kiện của khối if thành phần nào thỏa mãn, vế else của if sau cùng sẽ được thực thi sau đó chuyển tới các lệnh ngay phía bên dưới đó.
* Nếu không có vế else thì chương trình sẽ đi tới khối lệnh ngay phía dưới chuỗi if-else-if để tiếp tục thực hiện các lệnh còn lại.

### Ví dụ minh họa

* Yêu cầu: Giả sử có 3 mức học bổng là Khá, Giỏi và Xuất sắc. Điều kiện của 3 mức lần lượt là:
  + Khá: >= 3.0 và < 3.2: thưởng 3.5 triệu đ.
  + Giỏi: >= 3.2 và < 3.6: thưởng 5.0 triệu đ.
  + Xuất sắc: >= 3.6 đến 4.0: thưởng 10 triệu đ.
  + Nếu không đạt mức nào thì thông báo không được học bổng.
* Hãy vẽ sơ đồ khối cho chức năng kiểm tra xem sinh viên với điểm gpa cho trước có đạt học bổng không.
* Phân tích khả năng có thể xảy ra của điểm gpa do nó được nhập tùy ý từ bàn phím:
  + TH1: nếu gpa > 4.0  hoặc gpa < 0.0 thì điểm gpa này không hợp lệ. Theo hệ quy chiếu 4, điểm gpa chỉ từ 0.0 đến 4.0.
  + TH2: điểm gpa thuộc loại xuất sắc.
  + TH3: điểm gpa thuộc loại giỏi.
  + TH4: điểm gpa thuộc loại khá.
  + TH5: điểm gpa không đạt học bổng, tức dưới mức khá 3.0.
* Sơ đồ khối phần xét duyệt học bổng dựa trên điểm gpa của bài này như sau:

Ảnh có chứa văn bản, ảnh chụp màn hình, biểu đồ, hàng

Mô tả được tạo tự động

* Code mẫu:

import java.text.DecimalFormat;

import java.util.Scanner;

public class Lesson31 {

public static void main(String[] args) {

var scanner = new Scanner(System.in);

System.out.println(“Họ và tên sinh viên: “);

var fullName = scanner.nextLine();

System.out.println(“Nhập điểm gpa: “);

var gpa = scanner.nextFloat();

// bắt đầu khởi tạo và xét duyệt

var level = “”; // loại học bổng

int scholarship; // số tiền được thưởng tính theo vnđ

if (gpa > 4.0f || gpa < 0) {

level = “Gpa không hợp lệ”;

scholarship = 0;

} else if (gpa >= 3.6f) { // điều kiện học bổng xuất sắc

level = “Xuất sắc”;

scholarship = 10\_000\_000;

} else if (gpa >= 3.2f) { // điều kiện học bổng giỏi

level = “Giỏi”;

scholarship = 5\_000\_000;

} else if (gpa >= 3.0f) { // điều kiện học bổng khá

level = “Khá”;

scholarship = 3\_500\_000;

} else { // không đạt

level = “Không đạt”;

scholarship = 0;

}

// kết luận

if (scholarship > 0) {

var numberFormat = new DecimalFormat(“#,###”);

System.out.println(STR.”Chúc mừng sinh viên \”\{fullName}\” đã đạt học bổng!”);

System.out.println(STR.”Cấp độ học bổng: \{level}.”);

System.out.println(STR.”Mức thưởng: \{numberFormat.format(scholarship)}đ”);

} else {

System.out.println(STR.”Rất tiếc sinh viên \”\{fullName}\” không đạt học bổng.”);

System.out.println(STR.”Lý do: \{level}”);

}

}

}

Họ và tên sinh viên:

Nguyễn Thị Ngọc Mai

Nhập điểm gpa:

3.87

Chúc mừng sinh viên “Nguyễn Thị Ngọc Mai” đã đạt học bổng!

Cấp độ học bổng: Xuất sắc.

Mức thưởng: 10,000,000đ

Họ và tên sinh viên:

Lê Bá Nam

Nhập điểm gpa:

3.16

Chúc mừng sinh viên “Lê Bá Nam” đã đạt học bổng!

Cấp độ học bổng: Khá.

Mức thưởng: 3,500,000đ

Họ và tên sinh viên:

Hoàng Ngọc Ánh

Nhập điểm gpa:

3.54

Chúc mừng sinh viên “Hoàng Ngọc Ánh” đã đạt học bổng!

Cấp độ học bổng: Giỏi.

Mức thưởng: 5,000,000đ

Họ và tên sinh viên:

Trần Khánh Tùng

Nhập điểm gpa:

2.99

Rất tiếc sinh viên “Trần Khánh Tùng” không đạt học bổng.

Lý do: Không đạt

Họ và tên sinh viên:

Ma Thị Hải

Nhập điểm gpa:

5.6

Rất tiếc sinh viên “Ma Thị Hải” không đạt học bổng.

Lý do: Gpa không hợp lệ

Họ và tên sinh viên:

Lưu Viết Sơn

Nhập điểm gpa:

-3.66

Rất tiếc sinh viên “Lưu Viết Sơn” không đạt học bổng.

Lý do: Gpa không hợp lệ

**Bài 3.2. Cấu trúc ra quyết định switch**

**Nội dung bài học**

1. [Mục đích sử dụng](https://braniumacademy.net/courses/java/lessons/java-switch/#s1)
2. [Cú pháp tổng quát](https://braniumacademy.net/courses/java/lessons/java-switch/#s2)
3. [Switch expression](https://braniumacademy.net/courses/java/lessons/java-switch/#s6)
4. [Sơ đồ khối](https://braniumacademy.net/courses/java/lessons/java-switch/#s3)
5. [Ví dụ minh họa](https://braniumacademy.net/courses/java/lessons/java-switch/#s4)
6. [Bài tập thực hành](https://braniumacademy.net/courses/java/lessons/java-switch/#s5)

**Mục đích sử dụng**

* Sử dụng cấu trúc switch khi một biểu thức nào đó có nhiều hơn 2 trường hợp cần đánh giá. Các biểu thức của cấu trúc này thường là các hằng số, các kiểu nguyên hoặc String.
* Như vậy:
  + Nếu một biểu thức chỉ có hoặc đúng hoặc sai ta dùng cấu trúc if
  + Nếu một biểu thức cần đối chiếu với nhiều giá trị khác nhau, ta dùng cấu trúc switch

**Cú pháp tổng quát**

* Sau đây là cú pháp tổng quát của cấu trúc switch:

switch (biểu\_thức) {

case giá\_trị1:

// làm gì đó nếu trường hợp này xảy ra

break;

case giá\_trị2:

// làm gì đó nếu trường hợp này xảy ra

break;

case giá\_trị3:

// làm gì đó nếu trường hợp này xảy ra

break;

...

default:

// làm gì đó nếu tất cả các case phía trên không xảy ra

}

* Trong đó:
  + Cấu trúc bắt đầu với từ khóa switch.
  + Kế tiếp là cặp ngoặc tròn ( ).
  + Bên trong ngoặc tròn là biểu thức có tính chất nguyên, ví dụ biến kiểu nguyên(byte, short, char,int) hoặc lớp bao của chúng, hằng số, enum, String…
  + không dùng kiểu long hoặc Long
  + Phần thân switch ở phía trong cặp ngoặc { }.
  + Mỗi case tương đương với 1 vế if.
  + Đi sau case là dấu cách, sau đó đến giá trị cần đối sánh và cuối cùng là dấu hai chấm. Giá trị cần đối sánh phải cùng kiểu với kiểu của biểu thức trong switch.
  + Nội dung cần thực hiện của mỗi case sẽ viết sau dấu hai chấm :
  + Kết thúc mỗi case là từ khóa break. Nhiệm vụ của break là thoát khỏi cấu trúc switch khi thực hiện xong nhiệm vụ. Nếu bạn quên break, các case liền dưới của case hiện tại sẽ được thực hiện cho dù case đó không thỏa mãn. Việc thực hiện khối switch sẽ chỉ dừng khi gặp break hoặc đến khi kết thúc khối switch.
  + Tại mỗi thời điểm thường thì chỉ có một case của switch được thực hiện.
  + Nếu sau khi tất cả các case được đánh giá mà không có case nào được thực hiện, khối default sẽ được thực hiện. Kết thúc khối default không cần có break nhưng bạn có thể thêm vào. default luôn ở sau cùng cấu trúc switch.

**Switch expression**

* Từ phiên bản Java 12 ta có thể sử dụng biểu thức switch ở chế độ preview. Sau đó nó được hỗ trợ chính thức trong Java từ phiên bản Java 14.
* Cú pháp tổng quát của switch expression:

switch (variable) {

case value1 -> {

// làm gì đó nếu trường hợp này xảy ra

// yield value; trả về giá trị của case này.

}

case value2 -> {

// làm gì đó nếu trường hợp này xảy ra

// yield value; trả về giá trị của case này.

}

case value3 -> {

// làm gì đó nếu trường hợp này xảy ra

// yield value; trả về giá trị của case này.

}

...

default -> // làm gì đó nếu trường hợp này xảy ra

}

* Từ cấu trúc trên ta có thể gán giá trị của biểu thức switch cho một biến nào đó:

var someVariable = switch (variable) {

case value1 -> {

// làm gì đó nếu trường hợp này xảy ra

// yield value; trả về giá trị của case này.

}

case value2 -> {

// làm gì đó nếu trường hợp này xảy ra

// yield value; trả về giá trị của case này.

}

case value3 -> {

// làm gì đó nếu trường hợp này xảy ra

// yield value; trả về giá trị của case này.

}

...

default -> // làm gì đó nếu trường hợp này xảy ra

};

* Về cơ bản thì biểu thức switch giống với câu lệnh switch truyền thống tuy nhiên có một số khác biệt như sau:
  + Ta có thể trả về giá trị từ biểu thức switch còn câu lệnh switch thì KHÔNG.
  + Với các câu lệnh đơn ta viết các lệnh cần thực thi của mỗi case sau dấu mũi tên -> thay vì dấu hai chấm.
  + Với các câu lệnh phức tạp, ta để đoạn code cần thực thi trong khối lệnh bao bởi ngoặc { } sau mũi tên.
  + Để trả về giá trị từ các case trong biểu thức switch ta sử dụng keyword yield.
  + Ta không cần sử dụng break để kết thúc mỗi case của biểu thức switch.
  + Ta có thể đánh giá nhiều giá trị trong mỗi case với biểu thức switch.
  + Nếu giá trị đánh giá là enum, ta không cần tới nhãn default trong biểu thức switch.
  + Nếu giá trị đánh giá không phải enum, ta phải cung cấp nhãn default trong biểu thức switch.
* Ví dụ:

import java.util.Scanner;

public class Lesson32 {

public static void main(String[] args) {

// sử dụng switch expression

var scanner = new Scanner(System.in);

System.out.println("Nhập hai số a, b cách nhau một vài khoảng trắng: ");

var a = scanner.nextDouble();

var b = scanner.nextDouble();

System.out.println("===========> Các lựa chọn <===========");

System.out.println("1. Cộng hai số.");

System.out.println("2. Trừ a cho b.");

System.out.println("3. Nhân hai số.");

System.out.println("4. Chia a cho b.");

System.out.println("0. Thoát chương trình.");

System.out.println("Xin mời bạn lựa chọn: ");

var choice = scanner.nextInt();

switch (choice) {

case 0 -> System.out.println("==> Chương trình kết thúc.");

case 1 -> {

var sum = a + b;

System.out.println(STR."\{a} + \{b} = \{sum}");

}

case 2 -> {

var diff = a - b;

System.out.println(STR."\{a} - \{b} = \{diff}");

}

case 3 -> {

var prod = a \* b;

System.out.println(STR."\{a} \* \{b} = \{prod}");

}

case 4 -> {

var quot = a / b;

System.out.println(STR."\{a} / \{b} = \{quot}");

}

default -> System.out.println("==> Sai chức năng. Vui lòng chọn lại!");

}

}

}

Nhập hai số a, b cách nhau một vài khoảng trắng:

2.69 36.54

===========> Các lựa chọn <===========

1. Cộng hai số.

2. Trừ a cho b.

3. Nhân hai số.

4. Chia a cho b.

0. Thoát chương trình.

Xin mời bạn lựa chọn:

3

2.69 \* 36.54 = 98.2926

* Các ví dụ phần dưới đây tiếp tục sử dụng các tính năng của biểu thức switch.

**Sơ đồ khối**

* Sau đây là sơ đồ khối tổng quát của cấu trúc switch:

Ảnh có chứa văn bản, ảnh chụp màn hình, biểu đồ, hàng

Mô tả được tạo tự động

**Ví dụ minh họa**

* Ví dụ 1: Một mùa của năm chỉ có thể là xuân, hạ, thu và đông. Yêu cầu nhập vào tên mùa tiếng Anh và đưa ra màn hình tên mùa tương ứng bằng tiếng Việt.
  + Các khả năng có thể của input:
    - input = “spring” => output là “Mùa Xuân”.
    - input = “summer” => output là “Mùa Hạ”.
    - input = “fall” => output là “Mùa Thu”.
    - input = “winter” => output là “Mùa Đông”.
    - input là các giá trị khác => output sẽ là “INVALID”.
  + Sơ đồ khối xử lý các trường hợp kể trên:

Ảnh có chứa văn bản, ảnh chụp màn hình, biểu đồ, hàng

Mô tả được tạo tự động

* Code mẫu triển khai biểu thức switch truyền thống:

import java.util.Scanner;

public class Lesson32 {

public static void main(String[] args) {

var scanner = new Scanner(System.in);

System.out.println("Nhập tên mùa trong tiếng Anh: ");

var season = scanner.nextLine().trim().toLowerCase();

String currentSeason;

switch (season) {

case "spring":

currentSeason = "Mùa Xuân";

break;

case "summer":

currentSeason = "Mùa Hạ";

break;

case "fall":

case "autumn":

currentSeason = "Mùa Thu";

break;

case "winter":

currentSeason = "Mùa Đông";

break;

default:

currentSeason = "INVALID";

}

System.out.println(STR."Tiếng Anh: \{season} => Tiếng Việt: \{currentSeason}");

}

}

Nhập tên mùa trong tiếng Anh:

Summer

**Tiếng Anh: summer => Tiếng Việt: Mùa Hạ**

**===**

Nhập tên mùa trong tiếng Anh:

winter

**Tiếng Anh: winter => Tiếng Việt: Mùa Đông**

**===**

Nhập tên mùa trong tiếng Anh:

season

Tiếng Anh: season => Tiếng Việt: INVALID

* Code mẫu triển khai sử dụng phiên bản biểu thức switch:

import java.util.Scanner;

public class Lesson32 {

public static void main(String[] args) {

var scanner = new Scanner(System.in);

System.out.println("Nhập tên mùa trong tiếng Anh: ");

var season = scanner.nextLine().trim().toLowerCase();

String currentSeason = switch (season) {

case "spring" -> "Mùa Xuân";

case "summer" -> "Mùa Hạ";

case "fall", "autumn" -> "Mùa Thu";

case "winter" -> "Mùa Đông";

default -> "INVALID";

};

System.out.println(STR."Tiếng Anh: \{season} => Tiếng Việt: \{currentSeason}");

}

}

Nhập tên mùa trong tiếng Anh:

fall

Tiếng Anh: fall => Tiếng Việt: Mùa Thu

* Lưu ý rằng đây chỉ là 1 cách làm. Với mỗi vấn đề đưa ra bạn có nhiều cách làm khác nhau miễn là cách làm đó phù hợp và đưa ra được đáp án.
* Ví dụ 2: Viết chương trình cho phép người dùng nhập vào 2 số nguyên và các lựa chọn từ 0-4 và thực hiện các chức năng tương ứng như sau:
  + 0: thoát chương trình
  + 1: cộng hai số
  + 2: trừ hai số
  + 3: nhân hai số
  + 4: chia hai số
* Sau đây là sơ đồ khối phần xử lý logic chính của bài tập trên:

[Ảnh có chứa biểu đồ, văn bản, Kế hoạch, Bản vẽ kỹ thuật

Mô tả được tạo tự động](https://braniumacademy.net/wp-content/uploads/2020/08/lesson-10-example.png)

* Code mẫu:

import java.util.Scanner;

public class Lesson32 {

public static void main(String[] args) {

// sử dụng switch statement

var scanner = new Scanner(System.in);

System.out.println("Nhập hai số a, b cách nhau một vài khoảng trắng: ");

var a = scanner.nextDouble();

var b = scanner.nextDouble();

System.out.println("===========> Các lựa chọn <===========");

System.out.println("1. Cộng hai số.");

System.out.println("2. Trừ a cho b.");

System.out.println("3. Nhân hai số.");

System.out.println("4. Chia a cho b.");

System.out.println("0. Thoát chương trình.");

System.out.println("Xin mời bạn lựa chọn: ");

var choice = scanner.nextInt();

switch (choice) {

case 0:

System.out.println("==> Chương trình kết thúc.");

break;

case 1:

var sum = a + b;

System.out.println(STR."\{a} + \{b} = \{sum}");

break;

case 2:

var diff = a - b;

System.out.println(STR."\{a} - \{b} = \{diff}");

break;

case 3:

var prod = a \* b;

System.out.println(STR."\{a} \* \{b} = \{prod}");

break;

case 4:

var quot = a / b;

System.out.println(STR."\{a} / \{b} = \{quot}");

break;

default:

System.out.println("==> Sai chức năng. Vui lòng chọn lại!");

}

}

}

Nhập hai số a, b cách nhau một vài khoảng trắng:

1.25 3.66

===========> Các lựa chọn <===========

1. Cộng hai số.

2. Trừ a cho b.

3. Nhân hai số.

4. Chia a cho b.

0. Thoát chương trình.

Xin mời bạn lựa chọn:

1

**1.25 + 3.66 = 4.91**

**===**

Nhập hai số a, b cách nhau một vài khoảng trắng:

1.25 3.66

===========> Các lựa chọn <===========

1. Cộng hai số.

2. Trừ a cho b.

3. Nhân hai số.

4. Chia a cho b.

0. Thoát chương trình.

Xin mời bạn lựa chọn:

4

**1.25 / 3.66 = 0.34153005464480873**

**===**

1.25 3.66

===========> Các lựa chọn <===========

1. Cộng hai số.

2. Trừ a cho b.

3. Nhân hai số.

4. Chia a cho b.

0. Thoát chương trình.

Xin mời bạn lựa chọn:

9

==> Sai chức năng. Vui lòng chọn lại!

* Code mẫu sử dụng switch expression:

import java.util.Scanner;

public class Lesson32 {

public static void main(String[] args) {

// sử dụng switch expression

var scanner = new Scanner(System.in);

System.out.println("Nhập hai số a, b cách nhau một vài khoảng trắng: ");

var a = scanner.nextDouble();

var b = scanner.nextDouble();

System.out.println("===========> Các lựa chọn <===========");

System.out.println("1. Cộng hai số.");

System.out.println("2. Trừ a cho b.");

System.out.println("3. Nhân hai số.");

System.out.println("4. Chia a cho b.");

System.out.println("0. Thoát chương trình.");

System.out.println("Xin mời bạn lựa chọn: ");

var choice = scanner.nextInt();

var result = switch (choice) {

case 0 -> "==> Chương trình kết thúc.";

case 1 -> {

var sum = a + b;

yield STR."\{a} + \{b} = \{sum}";

}

case 2 -> {

var diff = a - b;

yield STR."\{a} - \{b} = \{diff}";

}

case 3 -> {

var prod = a \* b;

yield STR."\{a} \* \{b} = \{prod}";

}

case 4 -> {

var quot = a / b;

yield STR."\{a} / \{b} = \{quot}";

}

default -> "==> Sai chức năng. Vui lòng chọn lại!";

};

System.out.println(result);

}

}

Nhập hai số a, b cách nhau một vài khoảng trắng:

1.25 3.66

===========> Các lựa chọn <===========

1. Cộng hai số.

2. Trừ a cho b.

3. Nhân hai số.

4. Chia a cho b.

0. Thoát chương trình.

Xin mời bạn lựa chọn:

6

==> Sai chức năng. Vui lòng chọn lại!

**Bài 3.3. Toán tử ba ngôi**

**Nội dung bài học**

1. [Cấu trúc tổng quát](https://braniumacademy.net/courses/java/lessons/java-ternary-operator/#s1)
2. [Mục đích sử dụng](https://braniumacademy.net/courses/java/lessons/java-ternary-operator/#s2)
3. [Ví dụ minh họa](https://braniumacademy.net/courses/java/lessons/java-ternary-operator/#s3)
4. [Bài tập thực hành](https://braniumacademy.net/courses/java/lessons/java-ternary-operator/#s4)

**Cấu trúc tổng quát**

* Toán tử ba ngôi, hay còn có tên là conditional operator, ternary operator.
* Sau đây là cấu trúc tổng quát của toán tử ba ngôi:

**variable = (expression) ? value1: value2;**

* Trong đó:
  + *variable*là biến nào đó cần gán giá trị.
  + *expression*là một biểu thức nào đó có tính đúng sai. Từ việc xét tính chất đúng sai của biểu thức này sẽ quyết định giá trị được gán cho biến.
  + Nếu giá trị trong expression là true thì *value1*sẽ được gán cho biến variable.
  + Ngược lại, *value2*sẽ được gán cho biến variable.
* Khi sử dụng cú pháp trên để in ra màn hình hoặc làm giá trị trả về từ một phương thức, thì ta bỏ cụm cú pháp từ dấu bằng trở về trước.

**Mục đích sử dụng**

* Mục đích sử dụng của toán tử ba ngôi đó là làm cho câu lệnh điều kiện ngắn gọn lại trên 1 dòng.
* Thường áp dụng khi các điều kiện đơn giản hoặc làm giá trị trả về của hàm tác vụ đơn giản.
* Có thể sử dụng thay thế cho cấu trúc if-else.
* Tuy nhiên việc sử dụng cấu trúc if-else hoặc toán tử ba ngôi là do bạn quyết định. Không bắt buộc.

**Ví dụ minh họa**

* Ví dụ sau sẽ minh họa cú pháp sử dụng toán tử ba ngôi và cấu trúc if-else tương ứng.
* Dùng if-else:

int n = 200;

var result = "";

if(n % 2 == 0) {

result = "n là số chẵn";

} else {

result = "n là số lẻ";

}

System.out.println(result);

* **Ví dụ 1**: đoạn code trên khi dùng toán tử ba ngôi:

int n = 200;

var result = (n % 2 == 0) ? "n là số chẵn" : "n là số lẻ";

System.out.println(result);

* Ta cũng có thể lồng luôn vào phương thức println() như sau:

int n = 200;

System.out.println((n % 2 == 0) ? "n là số chẵn" : "n là số lẻ");

* Bạn đã thấy sự bá đạo của toán tử ba ngôi chưa?
* **Ví dụ 2:** kiểm tra và trả về giá trị n chẵn hay lẻ

public static boolean isEven(int n) {

return (n % 2 == 0) ? true : false;

}

* Tương đương với:

public static boolean isEven(int n) {

return (n % 2 == 0);

}

* Mở rộng: ta có thể sử dụng toán tử ba ngôi thay cho if-else-if.
* **Ví dụ 3**: kiểm tra số đặc biệt. Số đặc biệt là số nguyên chia hết cho 2 hoặc 3 hoặc 5.

import java.util.Scanner;

public class Lesson33 {

public static void main(String[] args) {

var scanner = new Scanner(System.in);

var number = scanner.nextInt();

var result = isSpecialNumber(number);

System.out.println(STR."\{number} is special number? \{result}");

}

private static boolean isSpecialNumber(int number) {

return (number % 2 == 0) ? true

: (number % 3 == 0) ? true

: number % 5 == 0 ? true : false;

}

}

* Phuong thức isSpecial() trên tương đương với:

public static boolean isSpecialNumber(int n) {

if(n % 2 == 0){

return true;

} else if(n % 3 == 0) {

return true;

} else if(n % 5 == 0) {

return true;

}

return false;

}

* **Ví dụ 4:** xét mức học bổng nhận được theo điểm gpa hệ 4.
  + Nếu gpa >= 3.6 thì được học bổng loại khá, trị giá 10 củ.
  + Nếu gpa >= 3.2 thì được học bổng loại giỏi trị giá 5 củ.
  + Nếu gpa >= 3.0 thì được học bổng trị giá 3.5 củ.
* Code triển khai:

import java.text.DecimalFormat;

import java.util.Scanner;

public class Lesson33 {

public static void main(String[] args) {

var scanner = new Scanner(System.in);

System.out.print("Your gpa: ");

var gpa = scanner.nextFloat();

var scholarship = (gpa < 3.0f || gpa > 4.0f) ? 0

: (gpa >= 3.6f) ? 10\_000\_000

: (gpa >= 3.2f) ? 5\_000\_000

: 3\_500\_000;

var numberFormat = new DecimalFormat("#,###");

var result = numberFormat.format(scholarship);

System.out.println(STR."Scholarship amount = \{result}đ");

}

}

Your gpa: 3.69

**Scholarship amount = 10,000,000đ**

**=====**

Your gpa: 3.33

**Scholarship amount = 5,000,000đ**

**=====**

Your gpa: 3.14

**Scholarship amount = 3,500,000đ**

**=====**

Your gpa: 2.35

**Scholarship amount = 0đ**

**=====**

Your gpa: 4.63

Scholarship amount = 0đ

**Bài 3.5. Vòng lặp for**

**Nội dung bài học**

1. [Mục đích sử dụng](https://braniumacademy.net/courses/java/lessons/java-for/#s1)
2. [Cú pháp tổng quát](https://braniumacademy.net/courses/java/lessons/java-for/#s2)
3. [Sơ đồ khối tổng quát](https://braniumacademy.net/courses/java/lessons/java-for/#s3)
4. [Ví dụ minh họa](https://braniumacademy.net/courses/java/lessons/java-for/#s4)
5. [Bài tập thực hành](https://braniumacademy.net/courses/java/lessons/java-for/#s5)

Video lý thuyết

Video minh họa

**Mục đích sử dụng**

* Cấu trúc lặp hay vòng lặp được sử dụng để lặp lại một đoạn chương trình nào đó. Hãy tưởng tượng, bạn muốn thực hiện 1 chức năng nào đó 99 lần. Bạn sẽ làm như nào? Copy đoạn code của chức năng đó và pate lại 99 lần? Có vẻ khả thi. Nhưng điều gì xảy ra nếu bạn phải copy pate 1000 lần, 10000 lần? Quả là tốn thời gian và tẻ nhạt đúng không nào. Rất may mắn, ngôn ngữ Java đã có sẵn cấu trúc lặp hỗ trợ ta. Bạn chỉ cần viết 1 lần chạy tùy ý.
* Có 2 loại vòng lặp:
  + Vòng lặp có biến kiểm soát lặp: for
  + Vòng lặp không có biến kiểm soát lặp: while, do-while
* Vòng lặp có biến kiểm soát lặp thường được sử dụng trong trường hợp biết trước số lần lặp.
* Vòng lặp không có biến kiểm soát lặp thường được sử dụng khi không biết trước số lần lặp. Để kiểm soát vòng lặp loại này ta dựa vào giá trị tính toán được/kết quả nhận được trong quá trình thực hiện lặp.

**Cú pháp tổng quát**

Sau đây là cú pháp tổng quát của vòng lặp for:

**for(khởi\_tạo; điều\_kiện; bước\_nhảy) {**

**// nội dung cần lặp**

**}**

* Trong đó:
  + Vòng lặp for bắt đầu với keyword for.
  + Theo sau for là cặp ngoặc tròn **( )**.
  + Trong ngoặc tròn gồm 3 thành phần phân tách nhau bằng dấu ; dùng để kiểm soát vòng lặp.
  + Phần *khởi\_tạo* được thực hiện trước nhất và duy nhất 1 lần ở lần lặp đầu. Nhiệm vụ của phần này là khởi tạo biến kiểm soát lặp. Biến kiểm soát lặp thường đặt là i, j, k.
  + Phần *điều\_kiện* là nơi kiểm tra xem điều kiện lặp có thỏa mãn(true) hay không. Nếu điều kiện lặp là true, thân vòng lặp sẽ được thực hiện. Ngược lại vòng lặp kết thúc.
  + Phần *bước\_nhảy* là nơi thay đổi giá trị biến kiểm soát lặp. Việc thay đổi có thể là tăng hoặc giảm tùy ngữ cảnh cụ thể. Thường thì phần bước nhảy sẽ thay đổi theo hướng làm cho biến kiểm soát lặp tiến đến giá trị mà tại đó điều kiện lặp không còn thỏa mãn. Như vậy vòng lặp sẽ tiến đến điểm kết thúc.
  + Phần thân vòng lặp trong cặp **{ }** chứa các câu lệnh cần thực hiện lặp.
* Nếu điều kiện lặp luôn đúng, vòng lặp sẽ lặp vô hạn. Do đó khi sử dụng vòng lặp cần hết sức chú ý.
* Nguyên nhân dẫn đến lặp vô hạn có thể là do giá trị ở biểu thức điều kiện luôn đúng. Hoặc phần bước nhảy không làm thay đổi biến kiểm soát lặp theo hướng tiến đến điểm kết thúc vòng lặp.
* Thứ tự thực hiện vòng lặp được mô tả trong phần kế tiếp.

**Sơ đồ khối tổng quát**

[Ảnh có chứa văn bản, biểu đồ, ảnh chụp màn hình, hàng

Mô tả được tạo tự động](https://braniumacademy.net/wp-content/uploads/2020/08/for-prototype.png)

* Trong sơ đồ ta thấy các đặc trưng của vòng lặp for được thể hiện rõ ràng:
  + Việc khởi tạo được thực hiện trước tiên và duy nhất
  + Nếu điều kiện lặp thỏa, thân vòng lặp mới được thực hiện
  + Tiếp đó mới đến lượt thực hiện bước nhảy
  + Sau đó tiến đến kiểm tra điều kiện lặp và tiếp tục lặp lại theo thứ tự mũi tên.
  + Nếu điều kiện lặp không thỏa, dù ở bất kì lần lặp nào, vòng lặp sẽ kết thúc.

**Ví dụ minh họa**

*Ví dụ 1: đếm số chữ cái trong một chuỗi kí tự nhập vào từ keyboard. chữ cái là các kí tự a-z và A-Z. Không tính khoảng trắng, kí tự đặc biệt.*

Sơ đồ khối của ví dụ này:

[Ảnh có chứa biểu đồ, văn bản, Bản vẽ kỹ thuật, Kế hoạch

Mô tả được tạo tự động](https://braniumacademy.net/wp-content/uploads/2020/08/for-example1.png)

* Code mẫu:

import java.util.Scanner;

public class Lesson13 {

public static void main(String[] args) {

var input = new Scanner(System.in);

System.out.println("Nhập vào một chuỗi kí tự");

var str = input.nextLine(); // đọc vào chuỗi kí tự

var count = 0; // khởi tạo biến đếm

// thực hiện việc kiểm tra

for (int i = 0; i < str.length(); i++) {

if(Character.isAlphabetic(str.charAt(i))) {

count++;

}

}

// hiện kết quả:

System.out.println("Số kí tự chữ cái trong chuỗi: " + count);

}

}

* Kết quả:

Nhập vào một chuỗi kí tự

xin chao abc123\*\*-\*

Số kí tự chữ cái trong chuỗi: 10

*Ví dụ 2: nhập một giá trị nguyên dương n. Tính tổng 1 + 1/2 + 1/3 + … + 1/n.*

* Sơ đồ khối tổng quát:

[Ảnh có chứa biểu đồ, văn bản, Bản vẽ kỹ thuật, Kế hoạch

Mô tả được tạo tự động](https://braniumacademy.net/wp-content/uploads/2020/08/for-example1.png)

* Code mẫu:

import java.util.Scanner;

public class Lesson13 {

public static void main(String[] args) {

var input = new Scanner(System.in);

System.out.println("Nhập vào số nguyên dương n: ");

var n = input.nextInt(); // đọc vào số n

var sum = 0.0; // khởi tạo biến tổng

// thực hiện việc cộng tổng

for (int i = 1; i <= n; i++) {

sum += 1.0/i; // nhớ ép kiểu sang double

}

// hiện kết quả:

System.out.println("Tổng sum = " + sum);

}

}

* Kết quả:

Nhập vào số nguyên dương n:

100

Tổng sum = 5.187377517639621

**Bài 3.6. Vòng lặp while, do-while**

**Nội dung bài học**

1. [Mục đích sử dụng](https://braniumacademy.net/courses/java/lessons/java-while/#s1)
2. [Cú pháp tổng quát](https://braniumacademy.net/courses/java/lessons/java-while/#s2)
3. [Sơ đồ khối tổng quát](https://braniumacademy.net/courses/java/lessons/java-while/#s3)
4. [Ví dụ minh họa](https://braniumacademy.net/courses/java/lessons/java-while/#s4)
5. [Bài tập thực hành](https://braniumacademy.net/courses/java/lessons/java-while/#s5)

Video lý thuyết

Video minh họa

**Mục đích sử dụng**

* Mục đích chung của vòng lặp là để thực hiện một nhóm các câu lệnh hay chức năng có tính chất lặp đi lặp lại.
* Vòng lặp while và do-while thuộc nhóm vòng lặp không có biến kiểm soát lặp. Do đó việc quyết định xem có lặp tiếp hay không sẽ dựa vào kết quả có được trong từng lần lặp.
* Vòng lặp while và do-while sử dụng khi cần lặp mà không biết chính xác số lần lặp.
* Vòng lặp do-while đặc biệt được sử dụng khi muốn rằng phần thân vòng lặp luôn đảm bảo được thực hiện ít nhất một lần cho dù điều kiện lặp có xảy ra hay không.

**Cú pháp tổng quát**

* Sau đây là cú pháp tổng quát của vòng lặp while:

**while(điều\_kiện) {**

**// nội dung cần lặp**

**}**

* Trong đó:
  + Cấu trúc lặp bắt đầu với keyword while
  + Theo sau đó là cặp ngoặc (), phía trong () là điều kiện lặp
  + Phần điều\_kiện là điều kiện lặp. Có thể gồm 1 hoặc nhiều điều kiện kết hợp lại. Điều kiện lặp chỉ có thể đúng hoặc sai.
  + Phần thân vòng lặp nằm trong cặp ngoặc {} và sẽ được thực hiện khi điều kiện lặp là true.
  + Nếu điều kiện lặp luôn đúng, vòng lặp sẽ lặp vô hạn.
  + Việc quyết định có lặp tiếp hay không phụ thuộc kết quả được kiểm soát ở trong thân vòng lặp.
* Sau đây là cú pháp tổng quát của vòng lặp do-while:

**do {**

**// nội dung cần lặp**

**} while(điều\_kiện);**

* Trong đó:
  + Vòng lặp bắt đầu với keyword do.
  + Tiếp đó là cặp ngoặc {} chứa thân vòng lặp bên trong.
  + Nội dung thân vòng lặp chứa các câu lệnh cần lặp và luôn được đảm bảo thực hiện ít nhất 1 lần.
  + Sau khi kết thúc thân vòng lặp sẽ đến phần while(điều\_kiện);
  + Điều kiện lặp có thể là điều kiện đơn hoặc điều kiện kết hợp và phải có tính đúng sai.
  + Nếu điều kiện lặp true, thân vòng lặp sẽ được thực hiện lặp lại.
  + Kết thúc cấu trúc lặp do-while là dấu chấm phẩy ;
  + Nếu điều kiện lặp luôn đúng, vòng lặp sẽ lặp vô hạn.

**Sơ đồ khối tổng quát**

* Sau đây là sơ đồ khối của hai loại vòng lặp trên:

[Ảnh có chứa văn bản, biểu đồ, hàng, ảnh chụp màn hình

Mô tả được tạo tự động](https://braniumacademy.net/wp-content/uploads/2020/08/while-vs-do-while.png)

**Ví dụ minh họa**

***Ví dụ 1:****minh họa sử dụng vòng lặp while kiểm tra n có phải số đẹp không. Số đẹp là số mà đọc ngược đọc xuôi vẫn cho cùng kết quả. Ví dụ: 123321.*

* Sơ đồ khối của thuật toán kiểm tra số đẹp:

[Ảnh có chứa văn bản, biểu đồ, hàng

Mô tả được tạo tự động](https://braniumacademy.net/wp-content/uploads/2020/08/so-dep.png)

* Code mẫu:

import java.util.Scanner;

public class Lesson14 {

public static void main(String[] args) {

var input = new Scanner(System.in);

System.out.println("Nhập vào số dương n: ");

var n = input.nextInt();

// lưu giá trị của n ra biến khác để tránh làm mất giá trị gốc

var m = n;

// tiến hành đảo số

var rev = 0;

while (m > 0) {

rev = rev \* 10 + m % 10;

m /= 10; // loại bỏ phần đơn vị đã lấy của m

}

// có kết quả đảo, tiến hành kiểm tra

if (n == rev) {

System.out.println("n là số đẹp");

} else {

System.out.println("n không phải số đẹp");

}

}

}

* Kết quả:

Nhập vào số dương n:

123

n không phải số đẹp

-----------------------------

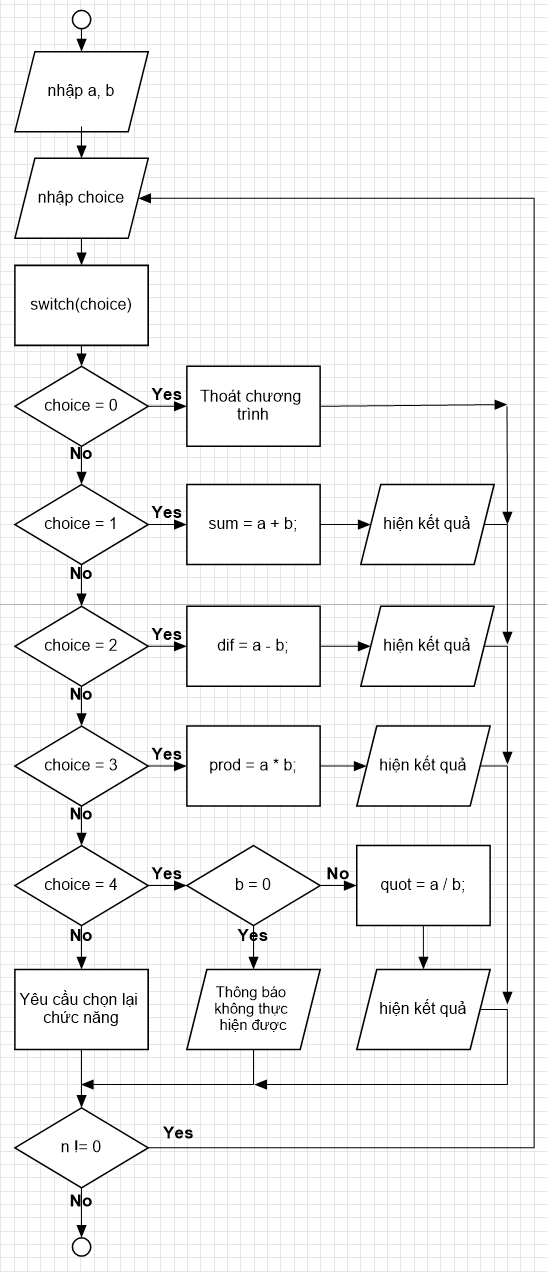
Nhập vào số dương n:

2552

n là số đẹp

***Ví dụ 2:****minh họa sử dụng vòng lặp do-while để thực hiện menu lựa chọn các phép tính toán cộng trừ nhân chia.*

* Sơ đồ khối chương trình:

[](https://braniumacademy.net/wp-content/uploads/2020/08/do-while-ex1-2.png)

* Code mẫu:

import java.util.Scanner;

public class Lesson14\_2 {

public static void main(String[] args) {

var input = new Scanner(System.in);

System.out.println("Nhập vào hai số thực a, b: ");

var a = input.nextDouble();

var b = input.nextDouble();

var choice = 0; // khởi tạo biến lựa chọn

do {

System.out.println("======== MENU ========");

System.out.println("1. Cộng hai số");

System.out.println("2. Trừ hai số");

System.out.println("3. Nhân hai số");

System.out.println("4. Chia hai số");

System.out.println("0. Thoát chương trình");

System.out.println("Xin mời chọn: ");

choice = input.nextInt(); // nhập lựa chọn

// tiến hành đánh giá và thực hiện chương trình

switch (choice) {

case 0:

System.out.println("Xin chào và hẹn gặp lại!");

break;

case 1: // nếu giá trị của choice = 1

var sum = a + b; // tính tổng hai số

System.out.println(a + " + " + b + " = " + sum);

break;

case 2: // nếu giá trị của choice = 2

var dif = a - b;

System.out.println(a + " - " + b + " = " + dif);

break;

case 3: // nếu giá trị của choice = 3

var prod = a \* b;

System.out.println(a + " \* " + b + " = " + prod);

break;

case 4: // nếu giá trị của choice = 4

if (b != 0) {

var quot = a / b;

System.out.println(a + " / " + b + " = " + quot);

} else {

System.out.println("Mẫu số = 0. Không thực hiện được phép chia.");

}

break;

default:

System.out.println("Sai chức năng. Vui lòng chọn lại.");

break;

}

} while (choice != 0); // đk lặp là choice != 0

}

}

* Kết quả:

Nhập vào hai số thực a, b:

65 21

======== MENU ========

1. Cộng hai số

2. Trừ hai số

3. Nhân hai số

4. Chia hai số

0. Thoát chương trình

Xin mời chọn:

1

65.0 + 21.0 = 86.0

======== MENU ========

1. Cộng hai số

2. Trừ hai số

3. Nhân hai số

4. Chia hai số

0. Thoát chương trình

Xin mời chọn:

**Bài 3.7. Vòng lặp lồng nhau**

**Nội dung bài học**

1. [Mục đích sử dụng](https://braniumacademy.net/courses/java/lessons/java-nested-loop/#s1)
2. [Cú pháp tổng quát](https://braniumacademy.net/courses/java/lessons/java-nested-loop/#s2)
3. [Ví dụ minh họa](https://braniumacademy.net/courses/java/lessons/java-nested-loop/#s3)
4. [Bài tập thực hành](https://braniumacademy.net/courses/java/lessons/java-nested-loop/#s4)

**Mục đích sử dụng**

* Vòng lặp lồng nhau là vòng lặp chứa trong vòng lặp khác. Các vòng lặp không nhất thiết phải cùng kiểu.
* Phổ biến nhất ta thấy là for lồng for.
* Mục đích chủ yếu của vòng lặp lồng nhau là để thực hiện các thao tác với mảng, các thao tác tìm kiếm, sắp xếp, thao tác trong hệ tọa độ n chiều…
* Khuyến nghị không nên lồng quá nhiều vòng lặp vào nhau vì sẽ làm tăng độ phức tạp, làm cho chương trình chạy chậm, giảm hiệu năng và trải nghiệm người dùng.
* Chi tiết về độ phức tạp thuật toán sẽ trình bày cụ thể trong Khóa học cấu trúc dữ liệu và giải thuật. Còn bây giờ các bạn hiểu đơn giản là:
  + Nếu một vòng lặp đơn thì độ phức tạ thuật toán là O(n)
  + Nếu hai vòng lặp lồng nhau thì độ phức tạp sẽ là O(n^2)
  + Ba vòng lặp lồng nhau liên tiếp thì độ phức tạp là O(n^3).

**Cú pháp tổng quát**

* Cú pháp vòng lặp lồng nhau tổng quát như sau:

Vòng lặp cấp 1 {

// nội dung vòng lặp cấp 1

Vòng lặp cấp 2 {

// nội dung vòng lặp cấp 2

Vòng lặp cấp 3 {

// nội dung vòng lặp cấp 3

}

}

}

* Trong đó:
  + Mỗi vòng lặp được đánh dấu lần lượt là vòng lặp cấp I, ví dụ cấp 1, cấp 2.
  + Các vòng lặp lồng nhau không nhất thiết phải cùng loại.
  + Ngoài việc chứa vòng lặp khác thì thân các vòng lặp còn có thể chứa các câu lệnh bình thường khác.
  + Các cấu trúc ra quyết định cũng có thể lồng nhau.
* Ví dụ:

for (int i = 0; i < m; i++) {

for (int j = 0; j < n; j++) {

// làm gì đó

}

}

**Ví dụ minh họa**

* **Ví dụ 1**: Viết chương trình vẽ hình chữ nhật đặc bằng các dấu \* kích thước m x n ra màn hình.

import java.util.Scanner;

public class Lesson15 {

public static void main(String[] args) {

var input = new Scanner(System.in);

System.out.println("Nhập chiều cao và chiều rộng của hình chữ nhật: ");

int m = input.nextInt();

int n = input.nextInt();

// tiến hành vẽ

for (int i = 1; i <= m; i++) {

for (int j = 1; j <= n; j++) {

System.out.print(" \* "); // in ra n dấu \*

}

System.out.println(); // in xuống dòng m lần

}

}

}

Nhập chiều cao và chiều rộng của hình chữ nhật:

5 6

\* \* \* \* \* \*

\* \* \* \* \* \*

\* \* \* \* \* \*

\* \* \* \* \* \*

\* \* \* \* \* \*

* **Ví dụ 2**: Viết chương trình vẽ hình chữ nhật rỗng bằng các dấu \* kích thước m x n ra màn hình.

import java.util.Scanner;

public class Lesson15B {

public static void main(String[] args) {

var input = new Scanner(System.in);

System.out.println("Nhập chiều cao và chiều rộng của hình chữ nhật: ");

int m = input.nextInt();

int n = input.nextInt();

// tiến hành vẽ

for (int i = 1; i <= m; i++) {

for (int j = 1; j <= n; j++) {

// chú ý bỏ ln ở println đi.

if(i == 1 || j == 1 || i == m || j == n) {

System.out.print(" \* "); // in ra n dấu \*

} else {

System.out.print(" "); // in ra 3 dấu cách

}

}

System.out.println(); // in xuống dòng m lần

}

}

}

Nhập chiều cao và chiều rộng của hình chữ nhật:

5 6

\* \* \* \* \* \*

\* \*

\* \*

\* \*

\* \* \* \* \* \*

* **Ví dụ 3**: Vẽ tam giác vuông góc trái dưới chiều cao h ra màn hình.

import java.util.Scanner;

public class Lesson15C {

public static void main(String[] args) {

var input = new Scanner(System.in);

System.out.println("Nhập chiều cao của tam giác: ");

int h = input.nextInt();

// tiến hành vẽ

for (int i = 1; i <= h; i++) {

for (int j = 1; j <= i; j++) {

System.out.print(" \* "); // in ra n dấu \*

}

System.out.println(); // in xuống dòng h lần

}

}

}

Nhập chiều cao của tam giác:

5

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

* **Ví dụ 4**: vẽ hình chữ nhật có lõi là các dấu – và viền tăng theo chiều kim đồng hồ từ giá trị x có 2 chữ số nào đó ví dụ:

Ảnh có chứa văn bản, Phông chữ, ảnh chụp màn hình, số

Mô tả được tạo tự động

* Trong ví dụ trên ta thấy rằng khi tiến hành vẽ từ trái qua phải, từ trên xuống dưới thì:
  + Hàng đầu tiên tăng từ giá trị khởi đầu start đến giá trị start + width – 1.
  + Cột phải cùng(trừ hàng đầu) tăng từ giá trị start + width đến start + width + height – 2.
  + Hàng cuối cùng giảm từ giá trị start + 2\*width + height – 3.
  + Cột đầu tiên giảm từ giá trị start + 2\*(width + height) – 5.
* Từ đó ta gọi counter1, counter2, counter3, counter4 ứng với giá trị cần hiển thị trên mỗi cạnh vừa xét, ta có triển khai code như sau:

import java.util.Scanner;

public class Lesson374 {

public static void main(String[] args) {

var input = new Scanner(System.in);

System.out.println("Giá trị bắt đầu: ");

int start = input.nextInt();

System.out.println("Chiều cao: ");

int width = input.nextInt();

System.out.println("Chiều rộng: ");

int height = input.nextInt();

int counter1 = start;

int counter2 = start + width;

int counter3 = start + 2 \* width + height - 3;

int counter4 = start + 2 \* (width + height) - 5;

for (int i = 1; i <= height; i++) {

for (int j = 1; j <= width; j++) {

if (i == 1) {

System.out.printf("%3d", counter1++);

} else if (j == width) {

System.out.printf("%3d", counter2++);

} else if (i == height) {

System.out.printf("%3d", counter3--);

} else if (j == 1) {

System.out.printf("%3d", counter4--);

} else {

System.out.printf("%3s", "-");

}

}

System.out.println();

}

}

}

Giá trị bắt đầu:

11

Chiều cao:

12

Chiều rộng:

12

11 12 13 14 15 16 17 18 19 20 21 22

54 - - - - - - - - - - 23

53 - - - - - - - - - - 24

52 - - - - - - - - - - 25

51 - - - - - - - - - - 26

50 - - - - - - - - - - 27

49 - - - - - - - - - - 28

48 - - - - - - - - - - 29

47 - - - - - - - - - - 30

46 - - - - - - - - - - 31

45 - - - - - - - - - - 32

44 43 42 41 40 39 38 37 36 35 34 33

**Bài 3.8. Vòng lặp vô hạn, lệnh break, continue**

**Nội dung bài học**

1. [Vòng lặp vô hạn](https://braniumacademy.net/courses/java/lessons/java-break-continue/#s1)
2. [Lệnh break](https://braniumacademy.net/courses/java/lessons/java-break-continue/#s2)
3. [Lệnh continue](https://braniumacademy.net/courses/java/lessons/java-break-continue/#s3)
4. [Ví dụ minh họa](https://braniumacademy.net/courses/java/lessons/java-break-continue/#s4)
5. [Bài tập thực hành](https://braniumacademy.net/courses/java/lessons/java-break-continue/#s5)

**Vòng lặp vô hạn**

* Vòng lặp mà lặp vô tận, không có điểm dừng gọi là vòng lặp vô hạn
* Vòng lặp vô hạn xảy ra khi điều kiện lặp luôn đúng
* Vòng lặp vô hạn có cả hai mặt lợi và hại. Tuy vậy khi biết ứng dụng vào từng trường hợp cụ thể thì bạn sẽ tận dụng được những lợi thế của vòng lặp này.
  + Mặt lợi là nó cho phép ta chạy chương trình đến vô thời hạn. Điều này sẽ cho ta có cơ hội giám sát được thời điểm kết thúc của vòng lặp dựa trên một dữ kiện nào đó.
  + Mặt hại là nếu cứ chạy vô thời hạn mà không dừng, thì sẽ chiếm CPU, RAM, các thành phần phần cứng liên quan nào đó. Dẫn đến không có đủ tài nguyên để phục vụ các chương trình khác.
* Các vòng lặp vô hạn thường sử dụng kèm với lệnh break để kết thúc lặp khi điều kiện nào đó xảy đến.
* Loại vòng lặp này thường được sử dụng để kiểm soát dữ liệu đầu vào hoặc chờ đợi một sự kiện nào đó.
* **Ví dụ:** Nhập các giá trị vào từ bàn phím, việc nhập sẽ kết thúc nếu nhập giá trị 0. Trong trường hợp này vòng lặp sẽ là vòng lặp vô hạn vì chưa biết khi nào mới kết thúc. Nhưng biết điều kiện để kết thúc là khi người dùng ấn phím 0.

var input = new Scanner(System.in);

var n = 0;

while (true) {

System.out.println("Nhập 0 để kết thúc. Nhập khác 0 để tiếp tục");

n = input.nextInt();

// ... làm gì đó ở đây

if(n == 0) { // nếu n == 0

break; // thoát khỏi vòng lặp

}

}

**Lệnh break**

* Khi nào cần chủ động thoát khỏi một cấu trúc, ví dụ cấu trúc lặp, cấu trúc switch, ta dùng break.
* Lệnh break sẽ cho phép ta kết thúc sớm và bỏ qua các lần lặp hoặc đánh giá kế tiếp vì lúc này ta đã đạt được mục đích. Do việc lặp tiếp hoặc đánh giá tiếp là dư thừa, không cần thiết.
* Lệnh break sẽ thoát khỏi cấu trúc lặp hoặc switch gần nhất chứa nó. Không thoát hoàn toàn khỏi chương trình.
* Khi gặp lệnh break, các lệnh phía dưới break(nếu có) và trong cùng khối với break sẽ không được thực hiện nữa.
* **Ví dụ 1**:

import java.util.Scanner;

public class Lesson16 {

public static void main(String[] args) {

var input = new Scanner(System.in);

var n = 0;

while (true) {

System.out.println("Nhập 0 để kết thúc. Nhập khác 0 để tiếp tục");

n = input.nextInt();

// ... làm gì đó ở đây

if(n == 0) { // nếu n == 0

break; // thoát khỏi vòng lặp

}

}

}

}

* Trong ví dụ trên lệnh break chỉ thoát việc thực thi khỏi vòng lặp while. Không làm kết thúc chương trình. Sau khi thoát vòng lặp, các lệnh phía sau thân vòng lặp sẽ lần lượt được thực hiện theo logic của mã nguồn.
* **Ví dụ 2**:

var choice = input.nextInt(); // nhập lựa chọn

// tiến hành đánh giá và thực hiện chương trình

switch (choice) {

case 0:

System.out.println("Xin chào và hẹn gặp lại!");

break;

case 1: // nếu giá trị của choice = 1

var sum = a + b; // tính tổng hai số

System.out.println(a + " + " + b + " = " + sum);

break;

case 2: // nếu giá trị của choice = 2

var dif = a - b;

System.out.println(a + " - " + b + " = " + dif);

break;

case 3: // nếu giá trị của choice = 3

var prod = a \* b;

System.out.println(a + " \* " + b + " = " + prod);

break;

case 4: // nếu giá trị của choice = 4

if(b != 0) {

var quot = a / b;

System.out.println(a + " / " + b + " = " + quot);

} else {

System.out.println("Mẫu số = 0. Không thực hiện được phép chia.");

}

break;

default:

System.out.println("Sai chức năng. Vui lòng chọn lại.");

break;

}

* Mỗi case trong cấu trúc switch trên đều chứa break ở cuối. Sở dĩ break ở cuối là sau khi thực hiện xong case tương ứng thì sẽ thoát logic khỏi cấu trúc switch. Các case khác sẽ không được đánh giá hay xem xét tiếp nữa vì ta đã tìm đc thứ mong muốn ở case vừa thực hiện rồi. Nếu break đặt đầu mỗi case thì các lệnh trong case đó sẽ không được thực hiện vì logic chương trình đã kết thúc tại thời điểm nó gặp break.
* **Ví dụ 3:** Tìm giá trị đầu tiên trong đoạn [a, b] chia hết cho k != 0.

var input = new Scanner(System.in);

System.out.println("Nhập số nguyên a < b và giá trị k: ");

var a = input.nextInt();

var b = input.nextInt();

var k = input.nextInt();

if(k != 0) {

for (int i = a; i <= b; i++) {

if(i % k == 0) {

System.out.println("giá trị đầu tiên thỏa: " + i);

break; // kết thúc vòng lặp

}

}

// làm gì đó tiếp ở đây

} else {

System.out.println("Nhập k != 0.");

}

* Trong ví dụ trên, nếu ta tìm thấy một giá trị i nào đó trong đoạn [a, b] chia hết cho k != 0, ta thoát khỏi vòng lặp. Vì lúc này ta đã tìm đc lời giải. Vấn đề đã được xử lý. Logic chương trình chỉ thoát khỏi vòng lặp nếu gặp break. Chương trình sau đó sẽ tiếp tục thực hiện các lệnh phía dưới vòng lặp(nếu có).

**Lệnh continue**

* Khác với break, continue không làm chương trình thoát khỏi một khối nào khi gặp nó. Thực tế continue cho phép logic chương trình bỏ qua các lệnh ở phía dưới và cùng nằm trong thân vòng lặp với continue.
* Continue thường áp dụng kết hợp với các cấu trúc điều khiển như ra quyết định hoặc vòng lặp.
* **Ví dụ 4**: Liệt kê các giá trị trong đoạn [a, b] chia hết cho k!=0.

if(k != 0) {

for (int i = a; i <= b; i++) {

if(i % k != 0) { // nếu i không chia hết cho k

continue; // bỏ qua các lệnh phía dưới

}

// ngược lại, tức i mà chia hết cho k, thì thực hiện:

System.out.println(i + " ");

}

// làm gì đó tiếp ở đây

} else {

System.out.println("Nhập k != 0.");

}

**Ví dụ minh họa**

* Ví dụ 3 đầy đủ:

import java.util.Scanner;

public class Lesson16 {

public static void main(String[] args) {

var input = new Scanner(System.in);

System.out.println("Nhập số nguyên a < b và giá trị k: ");

var a = input.nextInt();

var b = input.nextInt();

var k = input.nextInt();

if(k != 0) {

for (int i = a; i <= b; i++) {

if(i % k == 0) {

System.out.println("giá trị đầu tiên thỏa: " + i);

break; // kết thúc vòng lặp

}

}

// làm gì đó tiếp ở đây

} else {

System.out.println("Nhập k != 0.");

}

}

}

Nhập số nguyên a < b và giá trị k:

50 120 6

giá trị đầu tiên thỏa: 54

* Ví dụ 4 đầy đủ:

import java.util.Scanner;

public class Lesson16 {

public static void main(String[] args) {

var input = new Scanner(System.in);

System.out.println("Nhập số nguyên a < b và giá trị k: ");

var a = input.nextInt();

var b = input.nextInt();

var k = input.nextInt();

if(k != 0) {

for (int i = a; i <= b; i++) {

if(i % k != 0) { // nếu i không chia hết cho k

continue; // bỏ qua các lệnh phía dưới

}

// ngược lại, tức i mà chia hết cho k, thì thực hiện:

System.out.println(i + " ");

}

// làm gì đó tiếp ở đây

} else {

System.out.println("Nhập k != 0.");

}

}

}

Nhập số nguyên a < b và giá trị k:

3 50 7

7

14

21

28

35

42

49

# Bài 4.1. Tổng quan về phương thức

### Nội dung bài học

1. [Định nghĩa](https://braniumacademy.net/courses/java/lessons/java-method-overview/#s1)
2. [Cú pháp tổng quát](https://braniumacademy.net/courses/java/lessons/java-method-overview/#s2)
3. [Kiểu trả về của phương thức](https://braniumacademy.net/courses/java/lessons/java-method-overview/#s3)
4. [Ví dụ minh họa](https://braniumacademy.net/courses/java/lessons/java-method-overview/#s4)
5. [Bài tập thực hành](https://braniumacademy.net/courses/java/lessons/java-method-overview/#s5)

Video lý thuyết

Video minh họa

### Định nghĩa

* Phương thức hiểu đơn giản là tập hợp của các câu lệnh cùng thực hiện trọn vẹn một chức năng được đặt dưới 1 cái tên chung.
* Mỗi phương thức chỉ thực hiện một nhiệm vụ. Hay còn gọi tác vụ, task. Do đó phương thức thường dùng thể hiện hành động của một đối tượng nào đó. Hoặc dùng để mô tả việc thực hiện một nhiệm vụ cụ thể.
* Không nhồi nhét nhiều hơn 1 nhiệm vụ cho một phương thức. Không nên tạo những phương thức quá dài, ví dụ trên 40 dòng code.
* Ví dụ 1: phương thức main, thứ đã quen thuộc ngay từ đầu.

public class Lesson18 {

public static void main(String[] args) {

System.out.println("Hello World!");

}

}

* Ví dụ 2: phương thức kiểm tra số nguyên tố:

public static boolean isPrime(int n) {

if (n < 2) {

return false;

}

for (int i = 2; i <= Math.sqrt(n); i++) {

if (n % i == 0) {

return false;

}

}

return true;

}

### Cú pháp tổng quát

Cú pháp liên quan đến phương thức bao gồm cú pháp định nghĩa phương thức và cú pháp gọi/sử dụng phương thức:

##### Định nghĩa phương thức:

**access non-access type name(params) {**

**// statements**

**}**

* Trong đó:
  + Phần access là access modifier, hiểu là cấp độ truy cập hay khả năng phương thức được nhìn thấy từ bên ngoài lớp. Các access modifier sẽ được trình bày cụ thể trong phần chương kế tiếp nói về OOP. Để cho đơn giản hiện tại ta cho phần này là public: ở đâu cũng nhìn thấy và sử dụng được.
  + Phần non-access là phần có thể có hoặc không. None-access modifier có nhiệm vụ làm rõ tính chất của một phương thức. Chúng ta cũng sẽ tìm hiểu dần trong các bài sau. Hiện tại bạn cứ mặc định phần này là static. Ý nghĩa là cho phép truy cập không cần đối tượng của lớp(vì đối tượng ta chưa học).
  + Phần type là kiểu của phương thức. Có thể là bất kì kiểu hợp lệ nào trong ngôn ngữ Java như các kiểu nguyên thủy, void, String, các struct, enum, class, interface… Mỗi phương thức chỉ có thể có 1 kiểu.
  + Phần name là tên phương thức. Tên của phương thức là các hành động, nhiệm vụ nên nó thường là động từ hoặc cụm động từ. Trừ tên các phương thức đặc thù ví dụ như của lớp Math là ngoại lệ. Tên phương thức bắt đầu với chữ cái thường. Nếu tên có nhiều từ thì các từ viết liền và viết hoa chữ cái đầu từ. Tên phương thức đặt sao cho nó thể hiện hành động mà phương thức đảm nhiệm.
  + Sau tên phương thức là cặp ngoặc tròn. Đây là dấu hiệu nhận biết phương thức. Trong ngoặc tròn chứa danh sách các tham số.
  + Phần params là danh sách tham số, làm nhiệm vụ nhận dữ liệu truyền vào khi thực hiện lời gọi phương thức. Phần tham số có thể có 0, 1 hoặc nhiều tham số. Nếu có nhiều tham số thì sẽ phân tách nhau bởi dấu phẩy. Tham số là các biến hoặc hằng số có tên, kiểu cụ thể.
  + Phần thân phương thức bao bởi cặp {} chứa các câu lệnh cùng thực hiện chức năng mà phương thức đảm nhiệm.
  + Nếu phương thức trả về giá trị, kiểu của phương thức khác void và trong thân phương thức có lệnh return giá\_trị; Nếu phương thức không trả về thì kiểu của phương thức là void.
* Ví dụ:

// phương thức trả về kiểu boolean, nhận 1 tham số

public static boolean isPrime(int n) {

if (n < 2) {

return false; // trả về false

}

for (int i = 2; i <= Math.sqrt(n); i++) {

if (n % i == 0) {

return false; // trả về false

}

}

return true; // trả về true

}

// phương thức không trả về, không có tham số

public static void move() {

System.out.println("Tôi đang bò trên mặt đất");

}

### Gọi phương thức để sử dụng

* Cú pháp:

name (params);

// nếu muốn lưu lại giá trị trả về từ phương thức:

variable = name (params);

* Trong đó:
  + Lời gọi phương thức cần cung cấp đầy đủ tên và danh sách, số lượng, thứ tự đối số.
  + Phần name là tên phương thức cần gọi.
  + Đi sau tên là cặp ngoặc tròn kết thúc bởi dấu chấm phẩy ;
  + Trong ngoặc tròn là danh sách, thứ tự đối số.
  + Khi định nghĩa phương thức thì thứ trong ngoặc tròn sau tên hàm là tham số. Tham số là khai báo biến.
  + Khi gọi phương thức thì thứ trong ngoặc tròn của lời gọi là đối số. Đối số có thể là biến, hằng số, đối tượng hoặc giá trị cụ thể nào đó.
  + Số lượng đối số phải tương ứng với số lượng tham số.
* Sau đây là 1 số lời gọi phương thức:

public static void main(String[] args) {

// gọi phương thức khởi tạo của lớp scanner

var input = new Scanner(System.in); // gán giá trị có đc cho input

// gọi phương thức đọc vào giá trị số nguyên:

int i = input.nextInt(); // gán giá trị nhận đc cho i

// gọi phương thức hiển thị dữ liệu

System.out.println("Hello World!"); // chỉ hiển thị k gán cho biến nào

// gọi phương thức kiểm tra số ng.tố, gán giá trị cho biến prime

boolean prime = isPrime(500); // truyền vào đối số là 500

// gọi phương thức move:

move(); // không truyền gì vào

}

### Kiểu trả về của phương thức

Bây giờ ta sẽ làm rõ khái niệm trả về và không trả về.

* Một phương thức gọi là trả về nếu nó return một giá trị nào đó. Ví dụ phương thức nextInt() trả về giá trị int. Hoặc phương thức isPrime(500) trả về giá trị boolean. Trong thân phương thức có cú pháp return giá\_trị;
* Phương thức trả về có kiểu khác void. Ví dụ: int, float, boolean, String…
* Sử dụng phương thức trả về khi ta muốn đánh giá một tiêu chí nào đó; hoặc giá trị tính toán/có được từ phương thức đó sẽ là đầu vào hoặc được sử dụng bởi các thành phần thức khác của chương trình.
* Về bản chất phương thức trả về là phương thức gửi cho nơi đã gọi nó kết quả sau khi thực hiện nhiệm vụ. Như vậy ta biết được nhiệm vụ đó ra sao. Giống như đi thi cần biết kết quả thi để còn tính xem có phải học lại hay không vậy.
* Ví dụ như việc đọc vào một giá trị, tính tổng các chữ số, kiểm tra một số có phải số đẹp hay không…
* Phương thức không trả về là các phương thức có kiểu void. Phương thức không trả về có thể sử dụng return nhưng không có giá trị theo sau return.
* Sử dụng phương thức không trả về khi mà nhiệm vụ cần thực hiện không cần báo kết quả cho ai cả hoặc kết quả nhận được khi thực hiện nhiệm vụ là không quan trọng, không được sử dụng vào mục đích gì nữa.
* Ví dụ như phương thức hiển thị kết quả, hiện thông báo…

Tên phương thức kiểu boolean thường bắt đầu với is…

### Ví dụ minh họa

import java.util.Scanner;

public class Lesson18 {

public static void main(String[] args) {

// gọi phương thức khởi tạo của lớp scanner

var input = new Scanner(System.in); // gán giá trị có đc cho input

// gọi phương thức đọc vào giá trị số nguyên:

int i = input.nextInt(); // gán giá trị nhận đc cho i

// gọi phương thức hiển thị dữ liệu

System.out.println("Hello World!"); // chỉ hiển thị k gán cho biến nào

// gọi phương thức kiểm tra số ng.tố, gán giá trị cho biến prime

boolean prime = isPrime(500); // truyền vào đối số là 500

// gọi phương thức move:

move(); // không truyền gì vào

// tính tổng hai số:

int sum = add(100, 200); // gán cho biến sum

// kiểm tra một số có phải số đẹp:

boolean isBeautiNum = isRevert(1223221L); // gán cho biến isBeautiNum

}

// phương thức tính tổng

public static int add(int a, int b) {

return a + b;

}

// phương thức kiểm tra số đẹp

public static boolean isRevert(long n) {

long m = n;

long rev = 0;

while (m > 0) {

rev = rev \* 10 + m % 10;

m /= 10;

}

return m == rev;

}

// phương thức trả về kiểu boolean, nhận 1 tham số

public static boolean isPrime(int n) {

if (n < 2) {

return false; // trả về false

}

for (int i = 2; i <= Math.sqrt(n); i++) {

if (n % i == 0) {

return false; // trả về false

}

}

return true; // trả về true

}

public static void move() {

System.out.println("Tôi đang bò trên mặt đất");

}

}

200

Hello World!

Tôi đang bò trên mặt đất

**Bài 4.2. Nạp chồng phương thức**

**Nội dung bài học**

1. [Định nghĩa](https://braniumacademy.net/courses/java/lessons/java-overloading/#s1)
2. [Mục đích sử dụng](https://braniumacademy.net/courses/java/lessons/java-overloading/#s2)
3. [Yêu cầu của nạp chồng](https://braniumacademy.net/courses/java/lessons/java-overloading/#s3)
4. [Ví dụ minh họa](https://braniumacademy.net/courses/java/lessons/java-overloading/#s4)
5. [Bài tập thực hành](https://braniumacademy.net/courses/java/lessons/java-overloading/#s5)

**Khái niệm**

* Nạp chồng phương thức là kĩ thuật cho phép nhiều phương thức có cùng tên nhưng khác tham số.
* Nạp chồng phương thức không được chấp nhận nếu hai hay nhiều phương thức chỉ khác nhau kiểu trả về.
* Ví dụ về phương thức nạp chồng:

[Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, hàng

Mô tả được tạo tự động](https://braniumacademy.net/wp-content/uploads/2020/08/v%C3%AD-d%E1%BB%A5-overloading2.png)[Ảnh có chứa văn bản, Phông chữ, số, phần mềm

Mô tả được tạo tự động](https://braniumacademy.net/wp-content/uploads/2020/08/v%C3%AD-d%E1%BB%A5-overloading1.png)

* Giải thích: các phương thức max cùng thực hiện chức năng tìm giá trị lớn nhất trong hai số. Tuy nhiên chúng có kiểu tham số khác nhau: int-int, long-long, double-double… và kiểu trả về tương ứng kiểu của tham số. Điều này giúp cho chương trình dễ dàng nhận các tham số ở các kiểu khác nhau. Do vậy có thể tìm được giá trị lớn nhất của các kiểu số thay vì chỉ 1 kiểu số.
* Tương tự như thế, phương thức abs() cũng nhận vào các tham số khác nhau và thực hiện lấy trị tuyệt đối cho các kiểu số khác nhau.

**Mục đích sử dụng**

* Nạp chồng phương thức giúp cho nhiều phương thức thực hiện cùng một chức năng có thể chia sẻ với nhau một cái tên chung.
* Việc này làm cho tên của phương thức được thống nhất và có thể áp dụng với nhiều kiểu dữ liệu khác nhau.
* Thay vì mỗi phương thức lại phải đặt một cái tên riêng gây khó khăn và không nhất quán.

**Yêu cầu của nạp chồng**

* Các phương thức nạp chồng phải cùng tên, thực hiện chức năng giống nhau.
* Các phương thức nạp chồng phải khác nhau về số lượng, thứ tự và kiểu của tham số.
* Các phương thức nạp chồng có thể khác nhau kiểu trả về.
* Các phương thức nạp chồng không thể chỉ khác nhau mỗi kiểu trả về, còn các thành phần khác giống nhau.
* Các phương thức nạp chồng thường nằm trong 1 lớp hoặc trong 1 chuỗi kế thừa(học sau).

**Ví dụ minh họa**

Sau đây là ví dụ minh họa nạp chồng phương thức:

* Hai phương thức nạp chồng sau không hợp lệ vì chỉ khác nhau kiểu trả về

// tính và trả về tổng hai số nguyên int

public static void add(int a, int b) { // kiểu void

var sum = a + b;

System.out.println(sum);

}

// tính và trả về tổng hai số nguyên int

public static int add(int a, int b) { // kiểu int

var sum = a + b;

return sum;

}

* Các phương thức nạp chồng sau là hợp lệ:

// tính và trả về tổng hai số nguyên int

public static int add(int a, int b) { // kiểu int

return a + b;

}

// tính và trả về tổng hai số thực float

public static float add(float a, float b) { // kiểu float

return a + b;

}

// tính và trả về tổng hai số thực double

public static double add(double a, double b) { // kiểu double

return a + b;

}

// tính và trả về tổng hai số nguyên long

public static long add(long a, long b) { // kiểu long

return a + b;

}

// tính và trả về tổng hai số 3 nguyên int, int, long

public static long add(int a, int b, long c) { // kiểu long

return a + b + c;

}

# Bài 4.3. Các package

### Nội dung bài học

1. [Tổng quan về package](https://braniumacademy.net/courses/java/lessons/java-packages/#s1)
2. [Các quy ước với package](https://braniumacademy.net/courses/java/lessons/java-packages/#s2)
3. [Các package có sẵn trong Java](https://braniumacademy.net/courses/java/lessons/java-packages/#s3)
4. [Cú pháp static import](https://braniumacademy.net/courses/java/lessons/java-packages/#s4)
5. [Xử lý xung đột tên](https://braniumacademy.net/courses/java/lessons/java-packages/#s5)

### Tổng quan về package

* Package hay gói trong Java là kĩ thuật để đóng gói một nhóm các lớp, interface, các gói con có liên quan lại với nhau.
* Gói trong Java dùng trong một số mục đích sau:
  + Tránh xung đột tên. Thông thường trong cùng một gói thì tên lớp, interface, enum là duy nhất. Nếu muốn có các tên giống nhau thì ta phải chia gói. Mỗi tên trong một gói khác nhau.
  + Làm cho quá trình tìm kiếm, sử dụng các lớp, interface, enum… trở nên đơn giản hơn. Vì các thứ liên quan đã được gom lại một chỗ.
  + Cho phép ta điều khiển khả năng truy cập và sử dụng các thành phần của lớp. Kết hợp với các access modifier như protected, và mặc định cấp độ truy cấp gói(bỏ trống). Các thành phần protected có thể được truy cập bởi các lớp cùng gói và các lớp con của nó. Còn các thành phần mặc định(không có access modifier) sẽ chỉ được truy cập ở bởi lớp chứa nó và các lớp trong cùng một gói.
  + Góp phần thể hiện tính đóng gói dữ liệu trong Java.
* Như vậy ta chỉ cần gom nhóm các lớp có liên quan lại với nhau và bỏ chúng vào một gói phù hợp. Ở nơi sử dụng ta chỉ cần import class/interface/enum… từ nơi mà nó tồn tại và sử dụng như bình thường.
* Có thể hình dung một gói là một thư mục, trong đó có thể chứa các lớp, interface, enum… liên quan đến một chức năng, modul nào đó.
* Chúng ta có thể tái sử dụng lại các lớp đã có trong các gói trước đó một cách tùy ý.

### Các quy ước với package

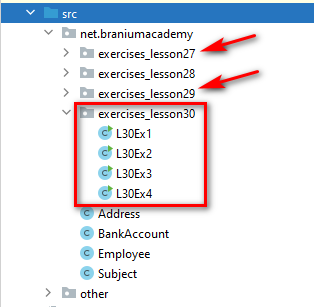
* Tên của package luôn viết thường hoàn toàn để tránh nhầm lẫn với tên các lớp, interface.
* Tên package thường dùng tên miền ngược của trang web nếu như đó là một tổ chức nào đó. Ví dụ tên pakage là: net.braniumacademy.core hoặc net.braniumacademy.example. Với package đích là core, example.
* Trong ngôn ngữ Java, các gói của ngôn ngữ mặc định bắt đầu với java. hoặc javax. như java.lang, java.util…
* Với các project nhỏ, có thể không cần có tên package hoặc tên package là đơn giản. Tên của package phải là duy nhất.
* Không dùng các keyword hay các gói gốc đã có trong project hay thư viện Java để đặt tên cho gói của bạn. Nếu tên gói là các keyword hoặc bắt đầu bởi số hoặc có dấu – phân tách thì thay bằng gạch dưới.

#### Ví dụ:

* example.\_123name
* example
* example.my\_package
* net.braniumacademy.java.lesson4\_3

#### Để thêm một lớp hay interface vào một gói:

* Ta tạo lớp/interface trong gói đã có hoặc đặt dòng package tên\_package; ở đầu file .java và chuyển file .java vào thư mục cùng tên với package.
* Ta thường tạo lớp mới có access modifier là public hoặc thêm lớp cần thêm vào file .java đã có.
* Để có hiệu lực ta phải compile lại project.
* Gói con(subpackage) là các gói ở trong một gói nào đó. Các gói này cũng là các gói độc lập và muốn sử dụng phải import chúng một cách tường minh.
* Ví dụ sau mô tả hình ảnh các gói, gói con và các lớp trong gói:

[](https://braniumacademy.net/wp-content/uploads/2020/08/vi-du-goi-trong-Java.png)

* Sau đây là nội dung của một file trong gói **chapter4**:

package net.braniumacademy.lesson.chapter4; // khai báo file java thuộc gói nào

import net.braniumacademy.lesson.other.Student; // nạp file java từ một gói khác

public class Lesson43 {

public static void main(String[] args) {

Student student = new Student("Lê Công Nhật Minh", 3.26f);

System.out.println(STR."Họ tên: \{student.getFullName()}");

System.out.println(STR."Điểm gpa: \{student.getGpa()}");

}

}

### Các package đã có sẵn trong Java

* Các gói trong Java chia làm hai loại:
  + Các gói có sẵn trong thư viện Java.
  + Và các gói do người dùng tự định nghĩa ra.
* Một số gói có sẵn thường dùng được liệt kê và mô tả như sau:
  + java.lang: chứa các lớp nền tảng của ngôn ngữ lập trình Java. Ví dụ như các lớp định nghĩa các kiểu nguyên thủy trong Java, lớp Math… gói này mặc định được tự động import.
  + java.io: chứa các lớp hỗ trợ các hành động thao tác vào ra dữ liệu.
  + java.util: chứa các lớp tiện ích thực thi các cấu trúc dữ liệu như LinkedList, Dictionary… các lớp hỗ trợ về quản lý thời gian như Date…
  + java.aplet: chứa các lớp để tạo Applet.
  + java.awt: chứa các lớp hỗ trợ tạo giao diện đồ họa như các nút, menu, bảng…
  + java.net: chứa các lớp hỗ trợ các thao tác với networking.
* Các gói do người dùng tự định nghĩa: các nhà phát triển có thể tự định nghĩa các gói theo nhu cầu của bản thân:
  + Đầu tiên tạo gói trong project với tên tuân thủ quy ước tên package đã nói ở trên.
  + Sau đó thêm các lớp/interface/enum… mới hoặc đã tồn tại vào package này.
  + Phần này sẽ được ví dụ minh họa chi tiết trong phần thực hành.
* Để sử dụng gói ta import lớp cần dùng vào. Có hai cách import:
  + Import tường minh: chỉ rõ tên gói và tên lớp cần dùng(khuyến nghị).
  + Import chung chung: chỉ rõ tên gói nhưng không chỉ rõ tên lớp. Thay vào đó dùng dấu \*. Đây là cách import cũ và khuyến nghị không nên sử dụng. (Ví dụ xem ở phần xử lý xung đột tên).
* Ví dụ:

package net.braniumacademy.lesson.chapter4;

//import java.text.DecimalFormat;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.Scanner;

public class Lesson43 {

public static void main(String[] args) throws ParseException {

Scanner scanner = new Scanner(System.in);

System.out.println("Ngày sinh, ví dụ: 10/10/2009: ");

var dateFormat = "dd/MM/yyyy";

SimpleDateFormat simpleDateFormat = new SimpleDateFormat(dateFormat);

var dateStr = scanner.nextLine();

Date birthday = simpleDateFormat.parse(dateStr);

System.out.println(simpleDateFormat.format(birthday));

java.sql.Date otherDate = new java.sql.Date(birthday.getTime());

// 12/10/2009

// float salary = scanner.nextFloat();

// DecimalFormat format = new DecimalFormat("#,###.##");

// System.out.println(STR."Lương: \{format.format(salary)}");

}

}

### Static import

* **Static import** là một tính năng của Java cho phép các thành phần như các trường, các phương thức static được nạp vào chương trình và sử dụng không cần phải cung cấp tên đầy đủ của lớp nơi thành phần đó được định nghĩa.
* Đôi khi ta sử dụng static import để làm cho câu lệnh Java trở nên ngắn gọn hơn.
* Ví dụ trong package chapter4 có lớp Calculator như sau:

package net.braniumacademy.lesson.chapter4;

public class Calculator {

public static double add(double a, double b, double c) {

return a + b + c;

}

}

* Ta muốn sử dụng phương thức add của lớp trên, ta có hai cách: sử dụng lời gọi thông thường hoặc dùng static import để khai báo nơi định nghĩa phương thức sau đó chỉ cần gọi tới tên phương thức đó và sử dụng.
* Sau đây là hai phiên bản code cho cùng kết quả:

// không sử dụng static import

package net.braniumacademy.lesson.chapter4;

public class Lesson43 {

public static void main(String[] args) {

var angle = 90;

var angleRadian = Math.toRadians(angle);

var sinX = Math.sin(angleRadian);

var sum = Calculator.add(18.98, 45.2, 35.36);

System.out.println(sum);

}

}

package net.braniumacademy.lesson.chapter4;

// sử dụng static import

import static java.lang.Math.toRadians;

import static java.lang.Math.sin;

import static net.braniumacademy.lesson.chapter4.Calculator.add;

public class Lesson43 {

public static void main(String[] args) {

var angle = 90;

var angleRadian = toRadians(angle);

var sinX = sin(angleRadian);

var sum = add(18.98,45.2,35.36);

System.out.println(sum);

}

}

### Xử lý xung đột tên

* Thường chúng ta chỉ quan tâm đến package khi xảy ra xung đột về tên. Ví dụ ta định nghĩa nhiều lớp cùng tên trong nhiều gói khác nhau. Khi đó ta phải chỉ rõ lớp cần dùng là thuộc gói nào.
* Một ví dụ về lớp của hệ thống là lớp Date. Trong cả hai gói java.util và java.sql đều có lớp này. Do đó ta phải chỉ rõ chúng thuộc gói nào tại nơi sử dụng.
* Ví dụ:

package other;

import java.sql.\*;

import java.util.\*;

public class Example {

public static void main(String[] args) {

Date today; // error, Date thuộc gói nào??

}

}

* Để xử lý vấn đề này ta sẽ import tường minh lớp Date cần dùng:

package other;

import java.sql.\*;

import java.util.Date;

public class Example {

public static void main(String[] args) {

Date today; // ok, Date của gói java.util.

}

}

* Nếu cần dùng cả hai ta có thể chỉ rõ tên và lớp như sau:

package other;

public class Example {

public static void main(String[] args) {

java.util.Date today = new java.util.Date();

java.sql.Date sqlDate = new java.sql.Date(today.getTime());

//...

}

}

* Ví dụ trong video:
  + File Calculator.java cùng package: cho ở phần trên.
  + File Calculator.java của package chapter3:

package net.braniumacademy.lesson.chapter3;

public class Calculator {

public static int add(int... numbers) {

int sum = 0;

for (var item : numbers) {

sum += item;

}

return sum;

}

}

* Code mẫu:

package net.braniumacademy.lesson.chapter4;

import java.util.Date;

import net.braniumacademy.lesson.chapter3.Calculator;

public class Lesson43 {

public static void main(String[] args) {

Date now = new Date();

java.sql.Date sqlDate = new java.sql.Date(now.getTime());

Calculator.add(1,2,3);

net.braniumacademy.lesson.chapter4.Calculator.add(1,2,3);

}

}

* Lưu ý quan trọng:
  + Mỗi lớp đều nằm trong một package nào đó.
  + Nếu tên package không được chỉ rõ trong file .java thì lớp trong file đó tự động được bỏ vào một gói mặc định không có tên.
  + Các class/interface/enum trong một file đều thuộc cùng một gói. Một gói có thể chứa nhiều file.
  + Nếu tên package đã được chỉ định, file đó phải nằm trong thư mục của gói. Tên gói và tên thư mục phải trùng khớp nhau.
  + Ta có thể sử dụng các class/interface/enum có access modifier là public từ các gói khác.

**Bài 4.4. Biểu thức switch**

**Nội dung bài học**

1. [Tổng quan](https://braniumacademy.net/courses/java/lessons/java-switch-expression/#s1)
2. [Các đặc trưng](https://braniumacademy.net/courses/java/lessons/java-switch-expression/#s2)
3. [Ví dụ minh họa](https://braniumacademy.net/courses/java/lessons/java-switch-expression/#s3)
4. [Bài tập thực hành](https://braniumacademy.net/courses/java/lessons/java-switch-expression/#s4)

**Tổng quan**

* Được giới thiệu ở phiên bản Java 12 và được định nghĩa lại ở phiên bản Java 13.
* Đây là một tính năng đang ở chế độ preview và có thể sẽ được tích hợp ở các phiên bản Java mới về sau.
* Cho phép cấu trúc switch được sử dụng như một biểu thức. Tức là có thể đem cả biểu thức đi đánh giá như một giá trị trị cụ thể.
* Có thể coi biểu thức switch là rút gọn của cấu trúc switch giống như toán tử ba ngôi là rút gọn của if-else vậy.
* Ta có thể xác định cụ thể giá trị của từng nhãn qua việc sử dụng từ khóa yield hoặc biểu thức mũi tên.

**Các đặc trưng**

* Có thể dùng nhiều nhãn trong một case.
* Biểu thức switch được đánh giá như một giá trị. Sau đó nó có thể được gán hoặc truyền vào các biểu thức lớn hơn.
* Nếu kiểu mục tiêu đã được xác định rõ ràng, tất cả các nhánh(case) đều phải có giá trị trả về ở kiểu mục tiêu đó.
* Nếu kiểu mục tiêu chưa xác định thì kiểu cha tương thích với hầu hết các kiểu trong biểu thức switch sẽ được sử dụng.
* Sử dụng yield để trả về một giá trị từ các case của biểu thức switch.
* Không còn cần sử dụng break.
* Có thể sử dụng mũi tên thay cho yield: tránh trôi thực thi lệnh xuống các nhãn khác, cho phép các câu lệnh biểu diễn trên 1 dòng hoặc chứa khối bao bởi cặp ngoặc {}.
* Trường hợp biến là kiểu enum thì không bắt buộc có default.
* Nếu biến đầu vào của biểu thức switch có kiểu khác enum thì phải có nhãn default.

**Ví dụ minh họa**

* **Ví dụ**: Trả lời câu hỏi trắc nghiệm, sử dụng cấu trúc switch truyền thống.

var question = "Logo hình quả táo cắn dở là của hãng công nghệ nào?";

var ans = "A. Apple.\nB. Samsung.\nC. Huawei.\nD. Oppo.\n";

System.out.println(question);

System.out.println(ans);

System.out.println("Câu trả lời của bạn là? ");

var input = new Scanner(System.in);

char choice = input.next().charAt(0);

String result = "";

switch (choice) {

case 'a':

case 'A':

result = "Chính xác!";

break;

case 'b':

case 'B':

case 'c':

case 'C':

case 'd':

case 'D':

result = "Sai mất rồi bạn ơi!";

break;

default:

result = "Vui lòng chọn các đáp án: A, B, C hoặc D";

break;

}

System.out.println(result);

* Sử dụng switch expression hay biểu thức switch:

var question = "Logo hình quả táo cắn dở là của hãng công nghệ nào?";

var ans = "A. Apple.\nB. Samsung.\nC. Huawei.\nD. Oppo.\n";

System.out.println(question);

System.out.println(ans);

System.out.println("Câu trả lời của bạn là?");

var input = new Scanner(System.in);

char choice = input.next().charAt(0);

String result = switch (choice) {

case 'a', 'A' -> "Chính xác!";

case 'b', 'B', 'c', 'C', 'd', 'D' -> "Sai mất rồi bạn ơi!";

default -> "Vui lòng chọn các đáp án: A, B, C hoặc D";

};

System.out.println(result);

Logo hình quả táo cắn dở là của hãng công nghệ nào?

A. Apple.

B. Samsung.

C. Huawei.

D. Oppo.

Câu trả lời của bạn là?

a

Chính xác!

# Bài 4.5. Phương thức đệ quy

### Nội dung bài học

1. [Định nghĩa](https://braniumacademy.net/courses/java/lessons/java-recursion/#s1)
2. [Đặc điểm](https://braniumacademy.net/courses/java/lessons/java-recursion/#s2)
3. [Ý nghĩa sử dụng](https://braniumacademy.net/courses/java/lessons/java-recursion/#s3)
4. [Ví dụ minh họa](https://braniumacademy.net/courses/java/lessons/java-recursion/#s4)
5. [Bài tập thực hành](https://braniumacademy.net/courses/java/lessons/java-recursion/#s5)

### Định nghĩa

* Đệ quy là chủ đề phức tạp sẽ được trình bày cụ thể trong môn học cấu trúc dữ liệu và giải thuật. Ở đây ta chỉ xem xét đệ quy ở mức độ cơ bản nhất.
* Một phương thức đệ quy là phương thức gọi lại chính nó một cách trực tiếp hoặc gián tiếp thông qua các phương thức khác
* Mở rộng ra, một cách giải quyết gọi là đệ quy nếu trong các bước thực hiện để giải quyết vấn đề lại chứa chính bản thân nó.
* Ví dụ:

// phương thức đệ quy tính n!

public static long factorial(int n) {

if (n <= 1) { // trường hợp cơ sở

return 1; // trả về giá trị đơn giản nhất của bài toán

} else { // thực hiện lời gọi đệ quy

return n \* factorial(n - 1);

}

}

### Đặc điểm

* Đệ quy thường áp dụng với các phương thức. Phương thức đệ quy thông thường chỉ biết cách giải quyết vấn đề ở trường hợp cơ bản nhất.
* Trường hợp này còn có tên gọi là trường hợp cơ sở, điểm dừng.
* Với các lời gọi trùng với trường hợp cơ sở thì phương thức đệ quy sẽ trả về giá trị của trường hợp cơ sở tương ứng.
* Khi phương thức đệ quy được gọi với trường hợp phức tạp hơn, phương thức đệ quy thường chia vấn đề thành hai phần: phần thứ nhất là phần mà phương thức này biết cách giải quyết; phần thứ hai phương thức này không biết cách giải quyết.
* Ở phần thứ hai này vấn đề cần xử lý có vẻ giống hệt vấn đề gốc ban đầu nhưng ở quy mô nhỏ hơn. Và để xử lý vấn đề thì phương thức đệ quy thực hiện lời gọi đến chính nó.
* Hành động gọi lại chính nó được gọi là lời gọi đệ quy, bước đệ quy.
* Phương thức chứa lời gọi đệ quy(thường đến chính nó) là phương thức đệ quy
* Khi thực hiện lời gọi đệ quy, chương trình sẽ phải chờ đến khi tất cả các bước đệ quy trong lời gọi trả về kết quả thì nó mới có kết quả để trả về cho nơi đã gọi nó.
* Để đảm bảo phương thức đệ quy có thể kết thúc được, sau mỗi lời gọi đệ quy vấn đề phải có xu hướng thu nhỏ lại, tiến dần đến trường hợp cơ sở, điểm dừng.
* Đệ quy cũng có thể lặp vô hạn nếu vấn đề không nhỏ hơn sau mỗi bước đệ quy.

### Ý nghĩa sử dụng

* Thường dùng phương thức đệ quy giải quyết vấn đề mang tính chất truy hồi.
* Vấn đề nào đó có thể giải quyết bằng đệ quy thì có thể giải quyết bằng vòng lặp.
* Cách giải quyết vấn đề bằng đệ quy ngắn gọn, tường minh hơn sử dụng vòng lặp
* Mỗi cách giải quyết vấn đề luôn có ưu điểm và nhược điểm riêng. Trong đó nhược điểm của đệ quy là tốn tài nguyên CPU, bộ nhớ và thời gian thực hiện.
* Lưu ý luôn đảm bảo vấn đề sẽ trở nên nhỏ hơn sau mỗi lời gọi đệ quy nếu không thì vấn đề sẽ không thể giải quyết được bằng đệ quy.

### Ví dụ minh họa

#### Ví dụ 1: Tính n!

* Ví dụ tính 5! = 5 \* 4 \* 3 \* 2 \* 1 = 120.
* Định nghĩa đệ quy của n! là n! = n \* (n – 1)!, cụ thể:
  + 5! = 5 \* 4 \* 3 \* 2 \* 1
  + 5! = 5 \* (4 \* 3 \* 2 \* 1)
  + 5! = 5 \* (4!)
* Suy ra n! = n\*(n-1)!
* Phân tích chi tiết:

[Ảnh có chứa văn bản, ảnh chụp màn hình, biểu đồ, Hình chữ nhật

Mô tả được tạo tự động](https://braniumacademy.net/wp-content/uploads/2020/08/v%C3%AD-d%E1%BB%A5-%C4%91%E1%BB%87-quy.png)

import java.util.Scanner;

public class Example {

public static void main(String[] args) {

var input = new Scanner(System.in);

System.out.println("Nhập n không quá 23: ");

var n = input.nextInt();

if(n < 0) {

System.out.println("Không thể tính giai thừa của số âm");

} else {

var result = factorial(n);

System.out.println(result);

}

}

// phương thức đệ quy tính n!

public static long factorial(int n) {

if (n <= 1) { // trường hợp cơ sở

return 1; // trả về giá trị đơn giản nhất của bài toán

} else { // thực hiện lời gọi đệ quy

return n \* factorial(n - 1);

}

}

}

* Kết quả

Nhập n không quá 23:

20

2432902008176640000

#### Ví dụ 2:

Tìm số fibonacci thứ n. Số Fibonacci được định nghĩa như sau: Số fibonacci là số cho theo công thức sau:  , từ n >= 2 thì  =  + .

* Phân tích: như vậy công thức tính số Fibonacci có thể đệ quy vì nó truy hồi dựa vào kết quả đã biết: Fn = fn-1 + fn-2. Điểm dừng là n = 0 hoặc n = 1.
* Code mẫu:

import java.util.Scanner;

public class Example {

public static void main(String[] args) {

var input = new Scanner(System.in);

System.out.println("Nhập n >= 0 không quá 90: ");

var n = input.nextInt();

if (n < 0) {

System.out.println("Không thể tính Fn với n âm");

} else {

var result = fibonacci(n);

System.out.println("F" + n + " = " + result);

}

}

// tìm số fibonacci dùng vòng lặp

public static long fiboLoop(int n) {

long f0 = 0;

long f1 = 1;

long fn = n;

for (int i = 2; i <= n; i++) {

fn = f0 + f1;

f0 = f1;

f1 = fn;

}

return fn;

}

}

* Kết quả:

Nhập n >= 0 không quá 94:

40

F40 = 102334155

# Bài 5.1. Mảng 1 chiều và vòng lặp foreach

### Nội dung bài học

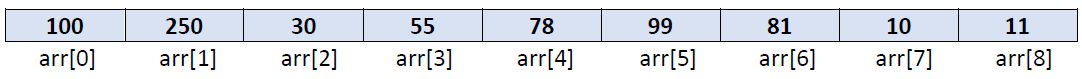
1. [Định nghĩa](https://braniumacademy.net/courses/java/lessons/java-one-dimension-array/#s1)
2. [Cú pháp khai báo tổng quát](https://braniumacademy.net/courses/java/lessons/java-one-dimension-array/#s2)
3. [Khởi tạo mảng](https://braniumacademy.net/courses/java/lessons/java-one-dimension-array/#s3)
4. [Truy xuất giá trị trong mảng](https://braniumacademy.net/courses/java/lessons/java-one-dimension-array/#s4)
5. [Vòng lặp foreach](https://braniumacademy.net/courses/java/lessons/java-one-dimension-array/#s5)
6. [Ví dụ minh họa](https://braniumacademy.net/courses/java/lessons/java-one-dimension-array/#s6)
7. [Bài tập thực hành](https://braniumacademy.net/courses/java/lessons/java-one-dimension-array/#s7)

Video lý thuyết

Video minh họa

### Định nghĩa

* Mảng là cấu trúc dữ liệu dùng để lưu trữ tập hợp của các giá trị cùng kiểu.
* Các giá trị đơn lẻ trong mảng được gọi là phần tử của mảng, nói gọn là phần tử hoặc phần tử mảng.
* Mỗi phần tử mảng được truy xuất, tức gán hoặc đọc giá trị ra thông qua một giá trị nguyên gọi là chỉ số mảng.
* Chỉ số mảng là số nguyên. Chỉ số phần tử đầu tiên trong mảng là 0. Chỉ số phần tử cuối trong mảng là n-1 với n là số phần tử tối đa có thể chứa của mảng.
* Như vậy:
  + Kiểu mảng là kiểu tập hợp, ta có thể dùng mảng để lưu một tập hợp các giá trị cùng kiểu nào đó
  + Thay vì dùng n biến lưu n giá trị khác nhau của cùng 1 kiểu nào đó ta dùng 1 biến mảng lưu trữ n phần tử
* Ví dụ: mảng dùng để lưu trữ danh sách sinh viên, danh sách môn học, danh sách các tỉ phú thế giới, danh sách người yêu cũ…
* Hình ảnh về mảng kiểu int tên arr. Dòng trên là từng giá trị của phần tử mảng, dòng dưới là cú pháp truy cập phần tử tương ứng:

[](https://braniumacademy.net/wp-content/uploads/2020/08/example-array.png)

### Cú pháp khai báo tổng quát

* Sau đây là cú pháp tổng quát của khai báo mảng:

**type[] name;**

* Trong đó:
  + type là kiểu dữ liệu của mảng, thường gọi tắt là kiểu của mảng hay kiểu mảng. Kiểu của mảng là các kiểu hợp lệ trong ngôn ngữ lập trình Java như các kiểu nguyên thủy: int, float boolean; các kiểu tham chiếu: String, các class, interface… như StringBuilder, StringBuffer, Student, BankAccount, ContentProvider, Comparator, Comparable…
  + Sau kiểu của mảng ta dùng cặp ngoặc vuông [] đi liền. Đây là cú pháp để nhận biết một biến kiểu mảng với một biến thông thường. Ta có thể để [] sau tên mảng tuy nhiên khuyến nghị đặt [] ngay sau kiểu của mảng.
  + Sau cặp móc vuông là 1 dấu cách, tiếp đó đến phần name. Đây chính là tên mảng. Quy ước đặt tên như đặt tên biến nhưng ở dạng số nhiều.
  + Kết thúc cú pháp khai báo mảng luôn có dấu chấm phẩy ;.
* Ví dụ:

int[] myNumbers; // khai báo biến mảng kiểu int

float[] avgGrades; // khai báo mảng kiểu float

String[] brands; // khai báo mảng kiểu String

String[] students; // khai báo mảng lưu danh sách các sinh viên

String[] friends; // khai báo mảng lưu danh sách những người bạn

### Khởi tạo mảng

Mảng khai báo như trên chưa sử dụng được do chúng ta chỉ mới khai báo, chưa khởi tạo giá trị cho nó. Mọi mảng trước khi đem sử dụng đều phải được khởi tạo hoặc gán giá trị.

* Cú pháp khởi tạo mảng:

type[] name = new type[num\_of\_elements];

type[] name = new type[]{elements};

type[] name ={elements};

* Trong đó giống cú pháp khai báo mảng và cộng thêm:
  + Dấu bằng = và vế bên phải dấu = là tổ hợp new …
  + Phần num\_of\_elements là số lượng phần tử của mảng mà ta muốn lưu trữ. Nhớ đặt trong ngoặc vuông. Số phần tử tối đa luôn là số nguyên dương.
  + Cách thứ nhất sẽ mặc định tạo ra một mảng kiểu type có num\_of\_elements phần tử. **Tất cả các phần tử này mặc định sẽ là 0 nếu là các kiểu số, false nếu là boolean và null nếu là các kiểu tham chiếu như String, các class, interface khác.**
  + Cách thứ hai tạo ra một mảng kiểu type có số phần tử xác định qua số lượng phần tử được khởi tạo trong cặp { }.
  + Cách thứ ba là rút gọn của cách thứ 2, lúc này số lượng phần tử liệt kê trong cặp ngoặc {} sẽ là số lượng phần tử của mảng.
  + Các phần tử trong phần elements là các giá trị cùng kiểu với kiểu của mảng và phân tách nhau bởi dấu phẩy.
  + Kết thúc cú pháp khởi tạo luôn có dấu chấm phẩy(;).
* Ví dụ:

int[] myNumbers = new int[10] {1, 2, 3, 5}; // error

float[] avgGrades = {1.25f, 3.24, 5.66f, 7.49}; // error

String[] brands = new String[]{"Apple", "Samsung", "Huawei"}; // ok

String[] students = {"Loan", "Hoa", "Nhung", "Phương", "Thanh"}; // ok

### Truy xuất giá trị trong mảng

* Truy xuất là việc gán giá trị cho các phần tử mảng hoặc lấy giá trị của các phần tử mảng ra để sử dụng vào mục đích nào đó.
* Để sử dụng giá trị trong mảng, tức đọc giá trị ra để tính toán, thao tác; hoặc gán, thay đổi giá trị một phần tử nào đó, ta sử dụng tên mảng và chỉ số mảng.
* Cú pháp tổng quát gán giá trị cho 1 phần tử là: **name[index] = value;**
* Cú pháp gán giá trị của một phần tử mảng cho một biến cùng kiểu: **variable = arrName[index];**
  + Trong đó name hoặc arrName là tên mảng
  + value, variable phải cùng kiểu với kiểu của mảng. Vế phải phép gán = có thể là giá trị cụ thể: “abc”, 647,… hoặc các biến, biểu thức, hằng số, lời gọi phương thức.
  + index là chỉ số mảng luôn nằm trong đoạn [0, n-1] với n là số lượng phần tử của mảng.
* Cú pháp lấy số lượng phần tử hiện có của mảng là: **name.length**
* Ta có thể gán hai biến mảng cùng kiểu cho nhau. Khi đó chúng cùng tham chiếu tới một mảng. Lý do là kiểu mảng là kiểu đối tượng.
* Ví dụ:

String[] brands = new String[]{"Apple", "Samsung", "Huawei", "Oppo"};

int size = brands.length; // lấy số phần tử mảng

System.out.println("Số phần tử của mảng là: " + size);

// gán giá trị cho một phần tử mảng:

brands[2] = "Xiaomi";

System.out.println("Phần tử tại chỉ số 2 của mảng là: " + brands[2]);

// gán mảng cho một biến mảng khác cùng kiểu

String[] other = brands; // gán mảng brands cho mảng khác cùng kiểu String

System.out.println(brands[0]); // hiển thị giá trị phần tử đầu tiên trong brands

System.out.println(other[0]); // hiển thị giá trị phần tử đầu tiên trong other

// kết quả cùng là chuỗi kí tự "Apple"

// ví dụ gán giá trị mảng kiểu int:

int[] integerNumbers = new int[100];

integerNumbers[10] = 500;

// ví dụ gán giá trị mảng các sinh viên Student:

// phần này minh họa mở rộng

Student[] students = new Student[20];

students[2] = new Student();

students[3] = student[0]l;

students[1] = createStudent("SV100", "Lê Thanh Trúc");

* Kết quả:

Số phần tử của mảng là: 4

Phần tử tại chỉ số 2 của mảng là: Xiaomi

Apple

Apple

// ....

#### Copy mảng

* Nếu muốn copy nội dung từ một mảng sang 1 mảng khác ta dùng Arrays.copyOf(source, new\_size);
* Trong đó:
  + Tham số thứ nhất source là mảng nguồn cần copy.
  + Tham số thứ hai new\_size là kích thước mảng đích.
  + Phương thức trên trả về mảng mới cùng kiểu mảng gốc trong tham số thứ nhất.
* Ví dụ:

String[] brands = new String[]{"Apple", "Samsung", "Huawei", "Oppo"};

var copyArr = Arrays.copyOf(brands, 2);

System.out.println(copyArr[1]); // kết quả in ra: Samsung

#### Truy xuất mảng

* Truy xuất tất cả hoặc một đoạn phần tử trong mảng, ta thường sử dụng kết hợp với vòng lặp for:

String[] brands = new String[]{"Apple", "Samsung", "Huawei", "Oppo"};

// duyệt toàn bộ mảng

for (int i = 0; i < brands.length; i++) {

System.out.println(brands[i]);

}

Apple

Samsung

Huawei

Oppo

* Lưu ý rằng nếu chỉ số của phần tử mảng mà nằm ngoài đoạn [0, length-1] với length là độ dài hay số phần tử mảng. Thì bạn sẽ nhận được một ngoại lệ:

public class Bai22 {

public static void main(String[] args) {

String[] brands = new String[]{"Apple", "Samsung", "Huawei", "Oppo"};

// duyệt toàn bộ mảng

for (int i = 0; i <= brands.length; i++) {

System.out.println(brands[i]); // dòng xuất hiện ngoại lệ

}

}

}

Apple

Samsung

Huawei

Oppo

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 4 out of bounds for length 4

at Other.main(Other.java:6)

* Do đó luôn đảm bảo rằng sử dụng chỉ số mảng đúng trong đoạn [0, length-1].

### Vòng lặp foreach

* Khi sử dụng vòng lặp for, ta có thể gặp vấn đề đó là chỉ số phần tử vượt biên của mảng. Điều này làm chương trình bị crash và kết thúc không mong muốn.
* Để giải quyết vấn đề trên, ta sử dụng vòng lặp foreach.
* Đọc là for-each.
* Vòng lặp này sẽ lần lượt duyệt từ đầu đến hết tập hợp hiện có.
* Cú pháp của vòng lặp foreach là:

**for (type element : collection) {**

**// do something**

**}**

* Trong đó:
  + type là kiểu của biến, cùng kiểu với kiểu mảng cần duyệt, có thể sử dụng var.
  + element là tên biến nào đó ngắn gọn dùng để đại diện cho từng phần tử của mảng.
  + Có dấu hai chấm phân tách ở giữa element và collection.
  + Phần collection là tên một tập hợp nào đó như tên mảng chẳng hạn.
  + Trong phần thân của foreach là các câu lệnh cần thực hiện nào đó.
* Ta sử dụng foreach khi duyệt mảng là chủ yếu. Còn nếu muốn thao tác sâu hơn như thay đổi giá trị phần tử, duyệt một đoạn thay vì toàn bộ mảng,… ta dùng vòng for thường.
* Để sử dụng foreach trong IntelliJ bạn gõ foreach rồi ấn phím Tab.
* Ví dụ:

String[] brands = new String[]{"Apple", "Samsung", "Huawei", "Oppo"};

// duyệt toàn bộ mảng

for (int i = 0; i < brands.length; i++) {

System.out.println(brands[i]); // hiển thị p.tử thứ i

}

// tương đương:

for (var item : brands) { // sử dụng kiểu tự suy luận var

System.out.println(item);

}

// hoặc:

for (String item : brands) { // sử dụng kiểu tường minh: String

System.out.println(item);

}

* Kết quả:

Apple

Samsung

Huawei

Oppo

* Foreach không làm thay đổi giá trị của phần tử mảng:

String[] brands = new String[]{"Apple", "Samsung", "Huawei", "Oppo"};

// duyệt toàn bộ mảng

for (var item : brands) { // sử dụng kiểu tự suy luận var

item = item + item.toString();

}

// duyệt lại xem có thay đổi gì không

for (String item : brands) { // sử dụng kiểu tường minh: String

System.out.println(item);

}

Apple

Samsung

Huawei

Oppo

* Vòng lặp thường thì có:

String[] brands = new String[]{"Apple", "Samsung", "Huawei", "Oppo"};

// duyệt toàn bộ mảng

for (int i = 0; i < brands.length; i++) {

brands[i] = brands[i] + brands[i];

}

// duyệt lại xem có thay đổi gì không

for (String item : brands) { // sử dụng kiểu tường minh: String

System.out.println(item);

}

AppleApple

SamsungSamsung

HuaweiHuawei

OppoOppo

### Ví dụ minh họa

**Ví dụ 1:** Nhập vào danh sách n > 0 người bạn của bạn, lưu vào mảng. Tìm người tên dài nhất và hiển thị ra màn hình.

import java.util.Scanner;

public class Vidu1 {

public static void main(String[] args) {

var input = new Scanner(System.in);

System.out.println("Nhập số người bạn của bạn: ");

var size = input.nextInt();

var friends = getFriends(size, input);

var maxLength = findMaxLength(friends);

// hiện kết quả:

showFriendList(friends);

showResult(friends, maxLength);

}

// phương thức hiển thị danh sách những người bạn

private static void showFriendList(String[] friends) {

System.out.println("Danh sách những người bạn: ");

for (var friend : friends) {

System.out.print(friend + " ");

}

}

// phương thức hiển thị danh sách những người có tên dài nhất

private static void showResult(String[] friends, int maxLength) {

System.out.println("\nDanh sách những người có tên dài nhất là: ");

for (var friend : friends) {

if (friend.length() == maxLength) {

System.out.print(friend + " ");

}

}

}

// phương thức tìm giá trị độ dài lớn nhất của tên

private static int findMaxLength(String[] friends) {

var max = 0;

for (var friend : friends) {

if (friend.length() > max) { // tìm tên có độ dài max

max = friend.length();

}

}

return max;

}

// phương thức nhập vào một danh sách những người bạn và trả về ds này

private static String[] getFriends(int size, Scanner input) {

var friends = new String[size];

for (int i = 0; i < size; i++) {

System.out.println("Nhập vào tên người bạn thứ " + (i + 1) + ": ");

friends[i] = input.next(); // chỉ nhập tên

}

return friends;

}

}

Nhập số người bạn của bạn:

6

Nhập vào tên người bạn thứ 1:

Nam

Nhập vào tên người bạn thứ 2:

Nhân

Nhập vào tên người bạn thứ 3:

Hoa

Nhập vào tên người bạn thứ 4:

Nhung

Nhập vào tên người bạn thứ 5:

Phương

Nhập vào tên người bạn thứ 6:

Linh

Danh sách những người bạn:

Nam Nhân Hoa Nhung Phương Linh

Danh sách những người có tên dài nhất là:

Phương

Process finished with exit code 0

* **Ví dụ 2:** Cho mảng nguyên n > 0 phần tử nhập vào từ bàn phím. Liệt kê các số nguyên tố trong mảng này.

import java.util.Scanner;

public class Vidu2 {

public static void main(String[] args) {

var input = new Scanner(System.in);

System.out.println("Nhập số phần tử mảng > 0: ");

var n = input.nextInt();

if (n > 0) {

// tạo mảng chứa dữ liệu nhập vào:

var numbers = getNumbers(n, input);

// hiển thị kết quả:

showRessult(numbers);

} else {

System.out.println("Vui lòng nhập số phần tử mảng > 0");

}

}

// phương thức hiển thị kết quả

private static void showRessult(int[] numbers) {

System.out.println("Các số nguyên tố trong mảng: ");

for (int i = 0; i < numbers.length; i++) {

if (isPrime(numbers[i])) { // nếu số tại vị trí i là số ng.tố

System.out.print(numbers[i] + " "); // in nó ra

}

}

}

// phương thức nhập vào danh sách các số nguyên cần kiểm tra

private static int[] getNumbers(int n, Scanner input) {

var numbers = new int[n];

for (int i = 0; i < n; i++) {

System.out.print("Nhập phần tử thứ " + (i + 1) + ": ");

numbers[i] = input.nextInt();

}

return numbers;

}

// phương thức kiểm tra n có phải số nguyên tố hay không

private static boolean isPrime(int n) {

if (n < 2) {

return false;

}

for (int i = 2; i <= Math.sqrt(n); i++) {

if (n % i == 0) {

return false;

}

}

return true;

}

}

Nhập số phần tử mảng > 0:

10

Nhập phần tử thứ 1: 1

Nhập phần tử thứ 2: 2

Nhập phần tử thứ 3: 5

Nhập phần tử thứ 4: 4

Nhập phần tử thứ 5: 8

Nhập phần tử thứ 6: 11

Nhập phần tử thứ 7: 7

Nhập phần tử thứ 8: 9

Nhập phần tử thứ 9: 21

Nhập phần tử thứ 10: 101

Các số nguyên tố trong mảng:

2 5 11 7 101

Process finished with exit code 0

**Bài 5.3. Sắp xếp các phần tử mảng**

**Nội dung bài học**

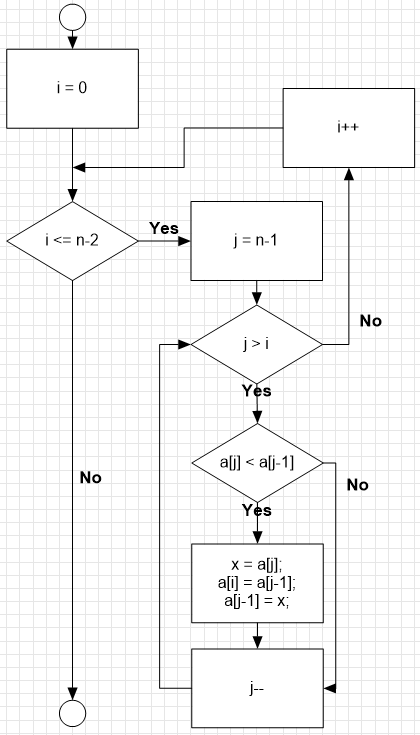
1. [Mục đích của sắp xếp](https://braniumacademy.net/courses/java/lessons/java-array-sorting/#s1)
2. [Sắp xếp thủ công](https://braniumacademy.net/courses/java/lessons/java-array-sorting/#s2)
3. [Sắp xếp quicksort](https://braniumacademy.net/courses/java/lessons/java-array-sorting/#s3)
4. [Bài tập thực hành](https://braniumacademy.net/courses/java/lessons/java-array-sorting/#s4)

**Mục đích của sắp xếp**

* Sắp xếp làm cho các phần tử của mảng được sắp đặt theo một thứ tự nào đó nhằm thuận lợi cho quá trình thực hiện các thao tác khác.
* Ví dụ sắp xếp danh sách sinh viên theo điểm thi giảm dần
* Sắp xếp danh sách các tỉ phú trên thế giới theo tổng tài sản giảm dần
* Sắp xếp danh sách thí sinh theo tên abc

**Sắp xếp thủ công**

* Phần này ta sẽ tự viết thuật toán sắp xếp các phần tử của mảng. Thuật toán ta sử dụng ở đây là thuật toán sắp xếp nổi bọt.
* Các bước thực hiện của thuật toán gồm:
  + Cho i chạy từ 0 đến n – 2
  + Cho j chạy từ n – 1 đến i + 1
  + Nếu phần tử thứ j nhỏ hơn phần tử thứ j – 1:
    - Đổi chỗ hai phần tử này
  + Thuật toán sẽ đẩy dần các phần tử có giá trị nhỏ hơn về phía trước. Giống như hình ảnh bọt khí(phần tử nhỏ hơn) nhẹ hơn nước sẽ nổi lên mặt nước, phần tử nặng hơn sẽ chìm xuống(đưa về cuối mảng).
  + Sơ đồ khối của thuật toán:

[](https://braniumacademy.net/wp-content/uploads/2020/08/buble-sort.png)

* Thực thi thuật toán bằng chương trình:

import java.util.Arrays;

public class BubbleSort {

public static void main(String[] args) {

// mảng gốc ban đầu

int[] numbers = {9, 8, 5, 2, 3, 1, 0, 4, 7, 8, 9, 100, 7};

System.out.println("Mảng gốc ban đầu: ");

showArray(numbers);

// sắp xếp toàn bộ theo thứ tự tăng dần:

Arrays.sort(numbers);

System.out.println("Mảng sau khi sắp xếp: ");

showArray(numbers);

}

/\*\*

\* phương thức hiển thị các phần tử của mảng nguyên

\*

\* @param numbers mảng chứa dữ liệu cần hiển thị

\*/

private static void showArray(int[] numbers) {

System.out.println("Các phần tử của mảng là: ");

for (int i = 0; i < numbers.length; i++) {

System.out.print(numbers[i] + " ");

}

System.out.println(); // in xuống dòng

}

}

Mảng gốc ban đầu:

Các phần tử của mảng là:

9 8 5 2 3 1 0 4 7 8 9 100 7

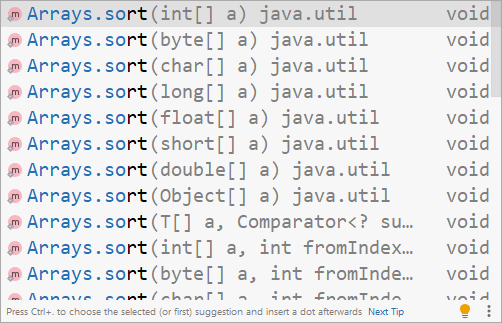
Mảng sau khi sắp xếp:

Các phần tử của mảng là:

0 1 2 3 4 5 7 7 8 8 9 9 100

**Sắp xếp quicksort**

* Lớp Arrays trong Java đã có sẵn một tập phương thức sắp xếp quicksort cho ta sử dụng:

[](https://braniumacademy.net/wp-content/uploads/2020/08/quick-sort.png)

* Bạn có thể sử dụng một trong các phương thức overload này để sắp xếp mảng. Ví dụ:

import java.util.Arrays;

public class BubbleSort {

public static void main(String[] args) {

// mảng gốc ban đầu

int[] numbers = {9, 8, 5, 2, 3, 1, 0, 4, 7, 8, 9, 100, 7};

System.out.println("Mảng gốc ban đầu: ");

showArray(numbers);

// sắp xếp toàn bộ theo thứ tự tăng dần:

Arrays.sort(numbers, 0, 12);

System.out.println("Mảng sau khi sắp xếp: ");

showArray(numbers);

}

/\*\*

\* phương thức hiển thị các phần tử của mảng nguyên

\*

\* @param numbers mảng chứa dữ liệu cần hiển thị

\*/

private static void showArray(int[] numbers) {

System.out.println("Các phần tử của mảng là: ");

for (int i = 0; i < numbers.length; i++) {

System.out.print(numbers[i] + " ");

}

System.out.println(); // in xuống dòng

}

}

Mảng gốc ban đầu:

Các phần tử của mảng là:

9 8 5 2 3 1 0 4 7 8 9 100 7

Mảng sau khi sắp xếp:

Các phần tử của mảng là:

0 1 2 3 4 5 7 7 8 8 9 9 100

* Sắp xếp một đoạn trong mảng: tham số thứ hai là vị trí bắt đầu, tham số thứ ba là vị trí kết thúc sắp xếp.

import java.util.Arrays;

public class BubbleSort {

public static void main(String[] args) {

// mảng gốc ban đầu

int[] numbers = {9, 8, 5, 2, 3, 1, 0, 4, 7, 8, 9, 100, 7};

System.out.println("Mảng gốc ban đầu: ");

showArray(numbers);

// sắp xếp các phần tử từ index 2 đến 8 theo thứ tự tăng dần:

Arrays.sort(numbers, 2, 8);

System.out.println("Mảng sau khi sắp xếp: ");

showArray(numbers);

}

/\*\*

\* phương thức hiển thị các phần tử của mảng nguyên

\*

\* @param numbers mảng chứa dữ liệu cần hiển thị

\*/

private static void showArray(int[] numbers) {

System.out.println("Các phần tử của mảng là: ");

for (int i = 0; i < numbers.length; i++) {

System.out.print(numbers[i] + " ");

}

System.out.println(); // in xuống dòng

}

}

Mảng gốc ban đầu:

Các phần tử của mảng là:

9 8 5 2 3 1 0 4 7 8 9 100 7

Mảng sau khi sắp xếp:

Các phần tử của mảng là:

9 8 0 1 2 3 4 5 7 8 9 100 7

**Bài 5.4. Tìm kiếm trong mảng**

**Nội dung bài học**

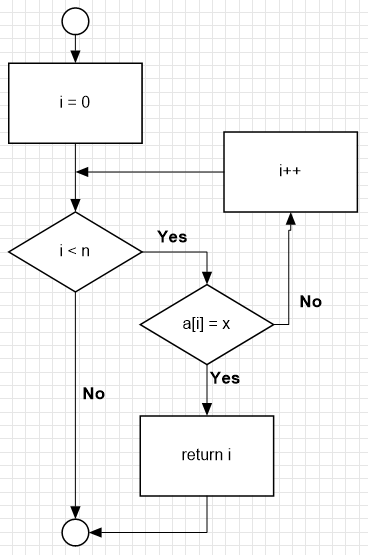
1. [Mục đích của tìm kiếm](https://braniumacademy.net/courses/java/lessons/java-array-searching/#s1)
2. [Tìm kiếm tuyến tính](https://braniumacademy.net/courses/java/lessons/java-array-searching/#s2)
3. [Tìm kiếm nhị phân](https://braniumacademy.net/courses/java/lessons/java-array-searching/#s3)
4. [Bài tập thực hành](https://braniumacademy.net/courses/java/lessons/java-array-searching/#s4)

**Mục đích của tìm kiếm**

* Mục đích của tìm kiếm trong mảng là nhằm tìm ra các phần tử thỏa mãn tiêu chí nào đó.
* Ví dụ ta cần tìm ra các sinh viên trong lớp có tên bắt đầu bằng chữ H
* Ta cần tìm kiếm xem những sinh viên nào có xếp hạng loại giỏi trong một danh sách lớp
* Ta cần tìm kiếm xem dịch bệnh nào gây tổn thất cho nhân loại nhiều nhất…

**Tìm kiếm tuyến tính**

* Hình thức đơn giản nhất của tìm kiếm là thuật toán tìm kiếm tuyến tính.
* Giả sử n là số phần tử của mảng, i là một biến chạy, x là giá trị cần tìm.
* Cụ thể như sau:
  + Cho biến i chạy từ 0 đến n
  + Nếu phần tử thứ i bằng x, trả về i
  + Sau khi tìm kiếm mặc định trả về -1
* Sơ đồ khối của chương trình:

[](https://braniumacademy.net/wp-content/uploads/2020/08/linear-search.png)

* Thực thi chương trình:

import java.util.Scanner;

public class LinearSearch {

public static void main(String[] args) {

int[] numbers = {9, 8, 5, 2, 3, 1, 0, 4, 7, 8, 9, 100, 7};

System.out.println("Mảng gốc ban đầu: ");

showArray(numbers);

System.out.println("Nhập giá trị cần tìm: ");

var x = new Scanner(System.in).nextInt();

int index = linearSearch(numbers, x);

var result = index == -1 ? "Không tồn tại x"

: "Tìm thấy x tại vị trí " + index;

System.out.println(result);

}

/\*\*

\* phương thức sử dụng để tìm kiếm tuyến tính xem x có

\* xuất hiện trong mảng numbers hay không

\*

\* @param numbers là mảng gốc cần tìm

\* @param x là biến chứa giá trị cần tìm

\* @return giá trị khác -1 nếu tìm thấy x và ngược lại

\*/

private static int linearSearch(int[] numbers, int x) {

for (int i = 0; i < numbers.length; i++) {

if (numbers[i] == x) { // nếu tìm thấy

return i; // trả về chỉ số i

}

}

return -1; // mặc định không tìm thấy

}

/\*\*

\* phương thức hiển thị các phần tử của mảng nguyên

\*

\* @param numbers mảng chứa dữ liệu cần hiển thị

\*/

private static void showArray(int[] numbers) {

System.out.println("Các phần tử của mảng là: ");

for (int i = 0; i < numbers.length; i++) {

System.out.print(numbers[i] + " ");

}

System.out.println(); // in xuống dòng

}

}

Mảng gốc ban đầu:

Các phần tử của mảng là:

9 8 5 2 3 1 0 4 7 8 9 100 7

Nhập giá trị cần tìm:

7

Tìm thấy x tại vị trí 8

Mảng gốc ban đầu:

Các phần tử của mảng là:

9 8 5 2 3 1 0 4 7 8 9 100 7

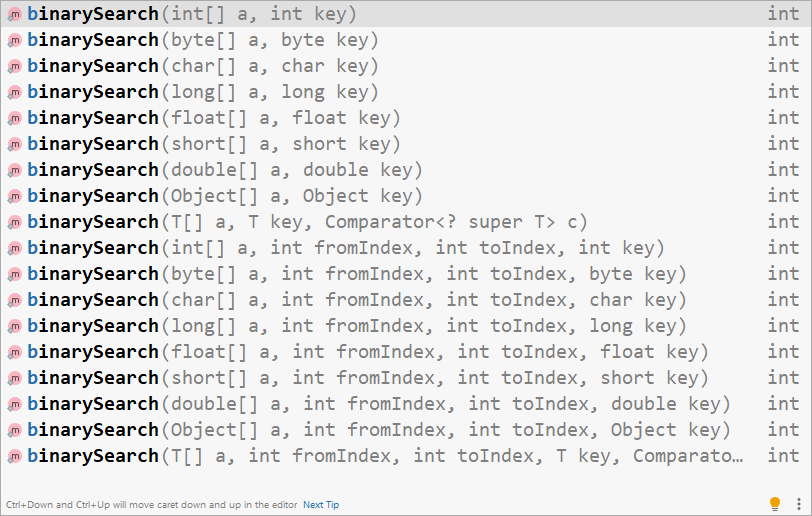
Nhập giá trị cần tìm:

1001

Không tồn tại x

**Tìm kiếm nhị phân**

* Trong thư viện Java đã có sẵn phương thức tìm kiếm nhị phân. Do đó chúng ta chỉ việc gọi phương thức và sử dụng.
* Để đảm bảo việc tìm kiếm được chính xác, bạn phải sắp xếp mảng theo thứ tự tăng dần của giá trị trước khi tìm kiếm.
* Phương thức tìm kiếm nhị phân sẽ trả về giá trị kiểu int. Nếu tìm thấy thì chỉ số phần tử sẽ được trả về. Ngược lại phương thức trả về giá trị âm.
* Các biến thể của thuật toán tìm kiếm nhị phân là:

[](https://braniumacademy.net/wp-content/uploads/2020/08/binary-search.png)

* **Ví dụ 1**: tìm trong toàn mảng.

import java.util.Arrays;

import java.util.Scanner;

public class BinarySearch {

public static void main(String[] args) {

int[] numbers = {9, 8, 5, 2, 3, 1, 0, 4, 7, 8, 9, 100, 7};

// tiến hành sắp xếp bằng quick sort

Arrays.sort(numbers);

showArray(numbers);

System.out.println("Nhập giá trị cần tìm: ");

var x = new Scanner(System.in).nextInt();

int index = Arrays.binarySearch(numbers, x);

var result = index < 0 ? "Không tồn tại x"

: "Tìm thấy x tại vị trí " + index;

System.out.println(result);

}

/\*\*

\* phương thức hiển thị các phần tử của mảng nguyên

\*

\* @param numbers mảng chứa dữ liệu cần hiển thị

\*/

private static void showArray(int[] numbers) {

System.out.println("Các phần tử của mảng là: ");

for (int i = 0; i < numbers.length; i++) {

System.out.print(numbers[i] + " ");

}

System.out.println(); // in xuống dòng

}

}

Các phần tử của mảng là:

0 1 2 3 4 5 7 7 8 8 9 9 100

Nhập giá trị cần tìm:

200

Không tồn tại x

* **Ví dụ 2**: Tìm trong đoạn fromIndex đến toIndex.

import java.util.Arrays;

import java.util.Scanner;

public class BinarySearch {

public static void main(String[] args) {

int[] numbers = {9, 8, 5, 2, 3, 1, 0, 4, 7, 8, 9, 100, 7};

// tiến hành sắp xếp bằng quick sort

Arrays.sort(numbers);

showArray(numbers);

System.out.println("Nhập giá trị cần tìm: ");

var x = new Scanner(System.in).nextInt();

int index = Arrays.binarySearch(numbers, 0, 8, x);

var result = index < 0 ? "Không tồn tại x"

: "Tìm thấy x tại vị trí " + index;

System.out.println(result);

}

/\*\*

\* phương thức hiển thị các phần tử của mảng nguyên

\*

\* @param numbers mảng chứa dữ liệu cần hiển thị

\*/

private static void showArray(int[] numbers) {

System.out.println("Các phần tử của mảng là: ");

for (int i = 0; i < numbers.length; i++) {

System.out.print(numbers[i] + " ");

}

System.out.println(); // in xuống dòng

}

}

Các phần tử của mảng là:

0 1 2 3 4 5 7 7 8 8 9 9 100

Nhập giá trị cần tìm:

9

Không tồn tại x

**Bài 5.5. Mảng nhiều chiều**

**Nội dung bài học**

1. [Mục đích sử dụng](https://braniumacademy.net/courses/java/lessons/java-two-dimension-array/#s1)
2. [Cú pháp khai báo tổng quát](https://braniumacademy.net/courses/java/lessons/java-two-dimension-array/#s2)
3. [Khởi tạo mảng](https://braniumacademy.net/courses/java/lessons/java-two-dimension-array/#s3)
4. [Truy xuất giá trị trong mảng](https://braniumacademy.net/courses/java/lessons/java-two-dimension-array/#s4)
5. [Mảng zigzag](https://braniumacademy.net/courses/java/lessons/java-two-dimension-array/#s5)
6. [Ví dụ minh họa](https://braniumacademy.net/courses/java/lessons/java-two-dimension-array/#s6)
7. [Bài tập thực hành](https://braniumacademy.net/courses/java/lessons/java-two-dimension-array/#s7)

Video lý thuyết

Video minh họa

**Mục đích sử dụng**

* Mục đích sử dụng của mảng nhiều chiều là để lưu trữ các dữ liệu dạng bảng.
* Phổ biến nhất của mảng nhiều chiều là mảng 2 chiều.
* Ví dụ lưu trữ hình ảnh, bảng tính xcel, bàn cờ, tọa độ hai chiều, thông tin bảng điểm, thông tin nhân viên…

**Cú pháp khai báo tổng quát**

* Cú pháp khai báo mảng hai chiều tổng quát là: **type[][] name;**
* Trong đó:
  + Phần type là kiểu của mảng. Có thể là bất kì kiểu dữ liệu hợp lệ nào trong Java. Ví dụ như int, float, boolean, String…
  + Sau phần kiểu mảng là hai cặp móc vuông [][] đi liền nhau. Đây là dấu hiệu nhận biết mảng hai chiều.
  + Sau đó là dấu cách và phần name. Name là tên mảng. Đặt theo quy tắc đặt tên mảng 1 chiều.
  + Kết thúc khai báo là dấu chấm phẩy.
* Mọi mảng đều phải được cấp phát hoặc khởi tạo trước khi có thể sử dụng.
* **Ví dụ**:

int[][] table; // lưu trữ một bảng các số nguyên

double[][] ballances; // lưu trữ số dư của từng tháng tương ứng

int[][] matrix; // lưu trữ một ma trận

float[][] coordinates; // mảng lưu trữ tọa độ 2 chiều

**Khởi tạo mảng**

* Ta có thể khởi tạo mảng theo nhiều cách khác nhau:

// cách 1:

type[][] name = new type[row][col];

// cách 2:

type[][] name = new type[][] {

{v[0, 0], v[0, 1], ..., v[0, col-1]},

...,

{v[row-1, 0], v[row-1, 1], ..., v[row-1, col-1]}

};

// cách 3:

type[][] name = { {v[0, 0], v[0, 1], ..., v[0, col-1]},

...,

{v[row-1, 0], v[row-1, 1], ..., v[row-1, col-1]}

};

* Trong đó:
  + Nếu khởi tạo theo cách 1, tất cả các phần tử của mảng sẽ được khởi tạo giá trị mặc định của kiểu. Các kiểu số là 0, kiểu boolean là false, kiểu tham chiếu là null. Tổng số phần tử trong mảng là row x col.
  + Nếu khởi tạo theo cách 2 hoặc 3 thì thường ta sẽ biết trước tập dữ liệu. Và tập dữ liệu tương đối nhỏ. Số cặp {} bên trong {} sẽ là số hàng của mảng. Số phần tử trong mỗi cặp {} bên trong là số phần tử trên từng hàng tương ứng, gọi là số cột.
  + Cách 3 là cú pháp rút gọn của cách 2. Tổng số phần tử khởi tạo là tổng số phần tử của mảng.
* Mảng sau khi cấp phát hoặc khởi tạo là không thể thay đổi số lượng phần tử.

**Truy xuất giá trị trong mảng**

* Để truy xuất một phần tử mảng, ta dùng tên mảng và chỉ số hàng, cột. Cú pháp là: name[row\_index][col\_index];
* Lưu ý rằng chỉ số hàng, cột theo quy tắc của mảng 1 chiều. Luôn bắt đầu từ 0 và kết thúc ở row-1 và col-1 tương ứng cho hàng, cột.
* Số hàng, số cột luôn là số nguyên dương.
* Nếu truy cập phần tử ngoài mảng thì sẽ bị ngoại lệ như mảng 1 chiều.
* **Ví dụ**:

int[][] matrix = new int[][]{

{1, 2, 3},

{4, 5, 6},

{7, 8, 9}

};

System.out.println(matrix[0][0]); // lấy giá trị phần tử đầu tiên

System.out.println(matrix[2][2]); // lấy giá trị phần tử cuối

matrix[0][2] = 100; // thay đổi giá trị hàng 0 cột 2

matrix[3][3] = 200; // xảy ra ngoại lệ vì chỉ số hàng cột chỉ đến 2, 2

* Để truy xuất toàn mảng hoặc một phần của mảng, ta thường dùng vòng lặp for lồng nhau.
* Cú pháp lấy số hàng của mảng hai chiều: **name.length**
* Cú pháp lấy số phần tử trên hàng i: **name[i].length**
* **Ví dụ**:

int[][] matrix = new int[][]{

{1, 2, 3},

{4, 5, 6},

{7, 8, 9}

};

// truy xuất từng phần tử mảng, trong đó matrix.length là số hàng

// matrix[i].length là số cột trên từng hàng

for (int i = 0; i < matrix.length; i++) {

for (int j = 0; j < matrix[i].length; j++) {

System.out.print(matrix[i][j] + " ");

}

System.out.println(); // in xuống dòng

}

* **Kết quả**:

1 2 3

4 5 6

7 8 9

* Nếu dùng foreach thì phải duyệt toàn bộ mảng. Cú pháp sẽ khác đôi chút với foreach ở mảng 1 chiều.
* Ví dụ:

int[][] matrix = new int[][]{

{1, 2, 3},

{4, 5, 6},

{7, 8, 9}

};

for (var row : matrix) { // duyệt từng hàng trong ma trận

for (var element : row) { // duyệt từng phần tử trên hàng

System.out.print(element + " ");

}

System.out.println(); // in xuống dòng

}

* Kết quả:

1 2 3

4 5 6

7 8 9

* Cú pháp tương tự áp dụng cho các mảng hai chiều khác.

**Mảng zigzag**

* Mảng từ đầu tới giờ ta xét là mảng có số phần tử trên từng hàng là như nhau.
* Trong Java cho phép tạo mảng hai chiều mà số cột trên từng hàng là khác nhau. Tức là số phần tử trên từng hàng khác nhau.
* Thực chất Java chỉ có mảng 1 chiều. Mảng nhiều chiều là mảng được tạo ra từ mảng của mảng. Tức là mỗi hàng trong mảng hai chiều có bản chất là một mảng 1 chiều.
* Để tạo mảng zigzag ta thực hiện qua hai bước:
  + Cấp phát số hàng
  + Cấp phát số phần tử trên từng hàng(số cột của từng hàng)
* Ví dụ:

type[][] name = new type[row][]; // cấp phát số hàng

for (int i = 0; i < row; i++) { // cấp phát số p.tử

name[i] = new type[col\_i]; // cho từng hàng

}

* Sau đó sử dụng mảng như bình thường:

// giả sử tạo mảng có số phần tử trên từng hàng tăng dần

int[][] triangle = new int[4][];

for (int i = 0; i < triangle.length; i++) {

triangle[i] = new int[i + 1];

}

// gán các giá trị cho mảng:

for (int i = 0; i < triangle.length; i++) {

for (int j = 0; j < triangle[i].length; j++) {

triangle[i][j] = i + j;

}

}

// hiện giá trị từng phần tử mảng zigzag

for (var row : triangle) { // duyệt từng hàng trong ma trận

for (var element : row) { // duyệt từng phần tử trên hàng

System.out.print(element + " ");

}

System.out.println(); // in xuống dòng

}

* Kết quả:

0

1 2

2 3 4

3 4 5 6

**Ví dụ minh họa**

* Ví dụ lưu trữ hình nhật đặc kích thướng m x n bằng các dấu \* sau đó hiển thị ra màn hình.

import java.util.Scanner;

public class Example {

public static void main(String[] args) {

var input = new Scanner(System.in);

System.out.println("Nhập vào chiều cao, chiều rộng hình CN: ");

var height = input.nextInt();

var width = input.nextInt();

// cấp phát mảng với số hàng và số cột cho trước

String[][] rectangle = new String[height][width];

// gán các giá trị cho mảng:

fillRect(rectangle);

// vẽ hình chữ nhật ra màn hình

drawRect(rectangle);

}

/\*\*

\* phương thức vẽ các dấu \* và xuống dòng ra màn hình

\*

\* @param rect mảng chứa các phần tử cần vẽ ra

\*/

private static void drawRect(String[][] rect) {

for (int i = 0; i < rect.length; i++) {

for (int j = 0; j < rect[i].length; j++) {

System.out.print(rect[i][j]);

}

System.out.println(); // in xuống dòng

}

}

/\*\*

\* phương thức gán giá trị cho từng phần tử của mảng

\* hai chiều lưu trữ hình chữ nhật bằng các dấu \*

\*

\* @param rect mảng chứa giá trị các dấu \*

\*/

private static void fillRect(String[][] rect) {

for (int i = 0; i < rect.length; i++) {

for (int j = 0; j < rect[i].length; j++) {

rect[i][j] = " \* ";

}

}

}

}

Nhập vào chiều cao, chiều rộng hình CN:

4 5

\* \* \* \* \*

\* \* \* \* \*

\* \* \* \* \*

\* \* \* \* \*

**Bài 5.6. Tham số biến thiên**

**Nội dung bài học**

1. [Giới thiệu](https://braniumacademy.net/courses/java/lessons/java-varagrs/#s1)
2. [Tiền truyện](https://braniumacademy.net/courses/java/lessons/java-varagrs/#s2)
3. [Sử dụng tham số biến thiên](https://braniumacademy.net/courses/java/lessons/java-varagrs/#s3)
4. [Quy tắc sử dụng tham số biến thiên](https://braniumacademy.net/courses/java/lessons/java-varagrs/#s4)
5. [Nguy cơ khi sử dụng tham số biến thiên](https://braniumacademy.net/courses/java/lessons/java-varagrs/#s5)
6. [Bài tập thực hành](https://braniumacademy.net/courses/java/lessons/java-varagrs/#s6)

**Giới thiệu**

* Tham số biến thiên varagrs được hỗ trợ từ phiên bản Java 5, cung cấp một cách ngắn gọn cho phép một phương thức nhận số lượng tham số tùy ý của cùng một kiểu dữ liệu.
* Gọi là tham số biến thiên vì số lượng tham số truyền vào có thể tùy ý thay đổi.
* Nội dung này ta sẽ tìm hiểu cách sử dụng, nguy cơ cũng như cách khắc phục các nguy cơ có thể gặp phải khi sử dụng tham số biến thiên.

**Tiền truyện**

* Trước Java 5, khi ta cần truyền số lượng đối số tùy ý ta phải để các đối số đó trong một mảng rồi truyền nó vào phương thức hoặc ta phải nạp chồng nhiều lần một phương thức cần sử dụng.
* Bạn hãy xem ví dụ sau để rõ hơn điều này:

package net.braniumacademy.lesson.chapter5;

// Cách 1: dùng một mảng chứa giá trị các đối số cần truyền vào

// phương thức, sau đó truyền mảng vào làm đối số trong lời gọi

// phương thức.

public class Lesson56 {

public static void main(String[] args) {

// tạo mảng:

var numbers = new int[]{1, 2, 5, 6, 9, 8, 7, 4, 1, 2, 0, 3, 6};

// truyền mảng vào làm đối số trong lời gọi phương thức

var sum = add(numbers);

System.out.println(STR."Tổng các phần tử trong mảng: \{sum}");

}

// phương thức tính tổng các số nguyên, nó nhận vào một mảng các số nguyên

public static int add(int[] numbers) {

var sum = 0;

for (var item : numbers) {

sum += item;

}

return sum;

}

}

// kết quả: Tổng các phần tử trong mảng: 54

package net.braniumacademy.lesson.chapter5;

public class Lesson56 {

public static void main(String[] args) {

System.out.println(STR."Tổng 1 số: \{add(255)}");

System.out.println(STR."Tổng 2 số: \{add(255, 354)}");

System.out.println(STR."Tổng 3 số: \{add(255, 354, 788)}");

System.out.println(STR."Tổng 4 số: \{add(255, 354, 788, 725)}");

}

/\*\*

\* Phương thức tính tổng 1 số nguyên

\*

\* @param number số nguyên cần tính tổng

\* @return tổng tính được

\*/

public static int add(int number) {

return number;

}

/\*\*

\* Phương thức tính tổng 2 số nguyên

\*

\* @param number1 số nguyên cần tính tổng

\* @param number2 số nguyên cần tính tổng

\* @return tổng tính được

\*/

public static int add(int number1, int number2) {

return number1 + number2;

}

/\*\*

\* Phương thức tính tổng 3 số nguyên

\*

\* @param number1 số nguyên cần tính tổng

\* @param number2 số nguyên cần tính tổng

\* @param number3 số nguyên cần tính tổng

\* @return tổng tính được

\*/

public static int add(int number1, int number2, int number3) {

return number1 + number2 + number3;

}

/\*\*

\* Phương thức tính tổng 4 số nguyên

\*

\* @param number1 số nguyên cần tính tổng

\* @param number2 số nguyên cần tính tổng

\* @param number3 số nguyên cần tính tổng

\* @param number4 số nguyên cần tính tổng

\* @return tổng tính được

\*/

public static int add(int number1, int number2, int number3, int number4) {

return number1 + number2 + number3 + number4;

}

}

/\*\*

\* Tổng 1 số: 255

\* Tổng 2 số: 609

\* Tổng 3 số: 1397

\* Tổng 4 số: 2122

\*/

* Làm sao để có thể tạo một phương thức duy nhất mà nhận vào số lượng các đối số tùy ý?

**Sử dụng tham số biến thiên**

* Để trả lời câu hỏi ở cuối phần trên, ta sử dụng một cú pháp mới trong Java gọi là tham số biến thiên.
* Với cú pháp này ta có thể truyền vào số lượng các đối số tùy ý của cùng kiểu dữ liệu.
* Về bản chất trong nội tại của nó sử dụng một mảng để lưu các tham số nhận được.
* Ta biểu diễn nó bởi dấu ba chấm sau kiểu dữ liệu. Tên tham số biến thiên đặt theo quy tắc đặt tên cho biến mảng.
* Ví dụ:

package net.braniumacademy.lesson.chapter5;

public class Lesson56 {

public static void main(String[] args) {

// ta có thể gọi phương thức add với số lượng các đối số tùy ý

// yêu cầu các đối số đó cùng kiểu là ok

System.out.println(STR."Tổng 1 số: \{add(255)}");

System.out.println(STR."Tổng 2 số: \{add(255, 354)}");

System.out.println(STR."Tổng 3 số: \{add(255, 354, 788)}");

System.out.println(STR."Tổng 4 số: \{add(255, 354, 788, 725)}");

}

/\*\*

\* Phương thức tính tổng các số nguyên

\*

\* @param numbers tham số biến thiên chứa các phần tử cần tính tổng

\* @return tổng tính được

\*/

public static int add(int... numbers) {

var sum = 0;

for (var number : numbers) {

sum += number;

}

return sum;

}

}

/\*\*

\* Tổng 1 số: 255

\* Tổng 2 số: 609

\* Tổng 3 số: 1397

\* Tổng 4 số: 2122

\*/

* Thật bá đạo phải không các bạn? chỉ với một cú pháp ngắn gọn ta đã thay thế được rất nhiều dòng code nhàm chán lặp đi lặp lại khi nạp chồng việc tính tổng của các số nguyên.
* Tương tự, bạn có thể sử dụng tham số biến thiên tại bất kì đâu nhận vào số lượng đối số tùy ý của cùng một kiểu dữ liệu hợp lệ trong Java.
* Thêm một ví dụ nữa cho bạn thấy nó vi diệu:

package net.braniumacademy.lesson.chapter5;

public class Lesson56 {

public static void main(String[] args) {

var line1 = createLine("Hi", "there!");

var line2 = createLine("Today", "is", "a", "good", "day", "to", "fight");

var line3 = createLine("This", "is", "Java", "5", "feature");

System.out.println(STR."Dòng 1: \"\{line1}\"");

System.out.println(STR."Dòng 2: \"\{line2}\"");

System.out.println(STR."Dòng 3: \"\{line3}\"");

}

/\*\*

\* Phương thức nối các từ lại với nhau thành câu hoàn chỉnh.

\*

\* @param words chứa các từ cần nối

\* @return câu hoàn chỉnh sau khi nối

\*/

public static String createLine(String... words) {

var line = new StringBuilder();

for (var word : words) {

if (!word.isBlank()) {

line.append(word).append(' ');

}

}

if (!line.isEmpty()) {

line.deleteCharAt(line.lastIndexOf(" "));

}

return line.toString();

}

}

/\*\*

\* Dòng 1: "Hi there!"

\* Dòng 2: "Today is a good day to fight"

\* Dòng 3: "This is Java 5 feature"

\*/

**Quy tắc sử dụng tham số biến thiên**

* Qua các ví dụ ở phần trên, ta thấy rằng cách sử dụng tham số biến thiên tương tự sử dụng mảng 1 chiều.
* Sau đây là một số quy tắc cần tuân thủ khi sử dụng tham số biến thiên:
  + Mỗi phương thức chỉ được phép có duy nhất 1 tham số biến thiên.
  + Nếu phương thức có nhiều tham số, trong đó bao gồm cả tham số thông thường và tham số biến thiên thì thằng tham số biến thiên phải là tham số cuối cùng trong danh sách tham số của phương thức.
* Ví dụ sau sẽ minh họa cho bạn rõ hơn những điều kể trên:

package net.braniumacademy.lesson.chapter5;

public class Lesson56 {

public static void main(String[] args) {

var line1 = createLine(" ", "Hi", "there!");

// ta có thể truyền mảng vào làm đối số tương ứng cho tham số biến thiên:

var line2 = createLine("-", new String[] { "Today", "is", "a", "good", "day", "to", "fight"});

var line3 = createLine("\_", "This", "is", "Java", "5", "feature");

System.out.println(STR."Dòng 1: \"\{line1}\"");

System.out.println(STR."Dòng 1: \"\{line2}\"");

System.out.println(STR."Dòng 1: \"\{line3}\"");

}

/\*\*

\* Phương thức nối các từ lại với nhau thành câu hoàn chỉnh.

\* Để ý rằng phương thức này có hai tham số, tham số biến thiên nằm cuối.

\*

\* @param delimiter dấu phân tách giữa các từ(dấu cách, phẩy, hỏi, chấm...)

\* @param words chứa các từ cần nối

\* @return câu hoàn chỉnh sau khi nối

\*/

public static String createLine(String delimiter, String... words) {

var line = new StringBuilder();

for (var word : words) {

if (!word.isBlank()) {

line.append(word).append(delimiter);

}

}

if (!line.isEmpty()) {

line.deleteCharAt(line.lastIndexOf(delimiter));

}

return line.toString();

}

}

/\*\*

\* Dòng 1: "Hi there!"

\* Dòng 1: "Today-is-a-good-day-to-fight"

\* Dòng 1: "This\_is\_Java\_5\_feature"

\*/

* Bây giờ nếu ta đảo thứ tự tham số ta sẽ nhận được một thông báo lỗi như sau:

/\*\*

\* Phương thức nối các từ lại với nhau thành câu hoàn chỉnh.

\* Để ý rằng phương thức này có hai tham số, tham số biến thiên nằm cuối.

\*

\* @param words chứa các từ cần nối

\* @param delimiter dấu phân tách giữa các từ(dấu cách, phẩy, hỏi, chấm...)

\* @return câu hoàn chỉnh sau khi nối

\*/

// error message: Vararg parameter must be the last in the list

public static String createLine(String... words, String delimiter) {

var line = new StringBuilder();

for (var word : words) {

if (!word.isBlank()) {

line.append(word).append(delimiter);

}

}

if (!line.isEmpty()) {

line.deleteCharAt(line.lastIndexOf(delimiter));

}

return line.toString();

}

* Ví dụ trên cho thấy rằng nếu có tham số biến thiên trong khai báo phương thức, nó phải là tham số cuối cùng của phương thức đó.
* Ví dụ sau đây sẽ cho ta kiểm chứng quy tắc đầu tiên đã nêu ở trên:

// phương thức có nhiều tham số biến thiên là KHÔNG hợp lệ.

// error: Vararg parameter must be the last in the list

public static int doSomething(int... args1, String... args2, float... args3) {

// do something here...

}

**Nguy cơ khi sử dụng tham số biến thiên**

* Khi sử dụng tham số biến thiên với kiểu generic ta có thể sẽ gặp phải vấn đề gọi là heap polution. Nó xảy ra khi một biến của kiểu tham số hóa trỏ tới một đối tượng không cùng kiểu tham số hóa.
* Thông thường, trình biên dịch sẽ phát hiện và cảnh báo vấn đề này cho ta.
* Ví dụ:

package net.braniumacademy.lesson.chapter5;

import java.util.Arrays;

import java.util.Collections;

import java.util.List;

public class Lesson56 {

public static void main(String[] args) {

String first = getFirstElement(Arrays.asList("Monday", "Tuesday"), Collections.emptyList());

System.out.println(first);

}

public static String getFirstElement(List<String>... strings) {

List<Double> doubleList = Collections.singletonList(10.25);

List<Integer> integerList= Collections.singletonList(100);

Object[] objects = strings;

objects[0] = doubleList; // heap pollution

objects[1] = integerList; // heap pollution

return strings[0].getFirst();

}

}

// java.lang.ClassCastException: class java.lang.Double cannot be cast to class java.lang.String

// (java.lang.Double and java.lang.String are in module java.base of loader 'bootstrap')

* Khi trỏ chuột lên tên phương thức ta sẽ nhận được thông báo như sau:

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, màn hình

Mô tả được tạo tự động

* Để tránh bị vấn đề heap pollution, ta có 2 cách:
  + Một là không lưu bất kì thứ gì khác trong mảng ngầm định(tham số biến thiên), ở ví dụ trên ta lưu một list giá trị double và một list giá trị integer vào strings thông qua biến objects.
  + Hai là không cho tham chiếu tới tham số biến thiên truyền ra ngoài phương thức.
* Sau đây là ví dụ trả về tham số biến thiên từ phương thức:

public static <T> T[] toArray(T... args) {

return args; // trả về tham số tham chiếu

}

* Ví dụ trên vi phạm nguyên tắc thứ hai nêu ngay trên nên ta sẽ gặp vấn đề heap polution. Để tránh vấn đề này khi sử dụng tham số biến thiên ta chỉ sử dụng nó trong nội bộ phương thức như các ví dụ ở các phần trên. Không truyền tham số biến thiên ra ngoài hay lưu các kiểu khác kiểu được chỉ định ban đầu của tham số biến thiên.

# Bài 6.1. Nhập môn lớp và đối tượng

### Nội dung bài học

1. [Khái niệm](https://braniumacademy.net/courses/java/lessons/java-class-objects/#s1)
2. [Mục đích sử dụng](https://braniumacademy.net/courses/java/lessons/java-class-objects/#s2)
3. [Cú pháp tổng quát](https://braniumacademy.net/courses/java/lessons/java-class-objects/#s3)
4. [Ví dụ minh họa](https://braniumacademy.net/courses/java/lessons/java-class-objects/#s4)
5. [Bài tập thực hành](https://braniumacademy.net/courses/java/lessons/java-class-objects/#s5)

### Khái niệm

* Trong thế giới thực ta dễ dàng nhận thấy rất nhiều những thực thể hay đối tượng có các đặc điểm giống hệt nhau. Ví dụ như những chiếc iPhone 15 Pro Max, iPhone 15 Max, những chiếc xe Tesla model X, những chiếc Macbook Pro 16′ phiên bản 2025, những chiếc laptop ASUS dòng XYZ nào đó… Tất cả các sản phẩm cùng phiên bản đó được tạo ra từ cùng một bản thiết kế. Do đó có các chức năng, đặc điểm và giá thành tương tự nhau.
* Một đối tượng thường có các đặc điểm nhận biết đặc trưng và hành động cụ thể đi kèm.
* Từ ý tưởng này, người ta cũng muốn tạo ra một chuẩn chung, bản mẫu chung để mô tả thông tin các đối tượng giống nhau trong ngôn ngữ lập trình. Để làm được điều này, ta sử dụng keyword class.
* **Một lớp là một bản mô tả cụ thể(có thể coi như bản vẽ, mẫu thiết kế, mô hình hóa) của một đối tượng trong ngôn ngữ lập trình.**
* **Hoặc: một lớp là một bản mẫu thiết kế chi tiết mà từ đó các đối tượng riêng lẻ của nó được tạo ra.**
* Java là ngôn ngữ lập trình hướng đối tượng, đặc trưng của nó là tất cả các thành phần được xây dựng trên các lớp.
* Các đối tượng có thể là đối tượng hữu hình, cụ thể: cái xe máy, ô tô, smartphone, con người, con vật, cây cối, vật dụng… hoặc cũng có thể là các đối tượng trừu tượng vô hình: đơn hàng, giỏ hàng, tài khoản facebook, tài khoản tiktok,…
* Bất cứ thứ gì mà ta có thể dùng danh từ để đặt tên cho nó thì đều có thể đem nó vào làm đối tượng trong ngôn ngữ lập trình.

### Mục đích sử dụng

* Vậy, mục đích của class là để mô hình hóa một đối tượng từ thế giới thực vào thế giới của ngôn ngữ lập trình để máy tính hiểu được.
* Một lớp thường có hai thành phần chính: các thuộc tính và các phương thức.
* Ngoài ra nó có thể chỉ có các phương thức, chỉ chứa thuộc tính hoặc có thêm các enum, các lớp, interface khác.
* Các đặc điểm của đối tượng sẽ được mô tả thông qua các thuộc tính của lớp.
* Các hành động của đối tượng sẽ được mô tả thông qua các phương thức.

### Cú pháp tổng quát

* Sau đây là cú pháp tổng quát của một định nghĩa lớp trong Java:

**access non-access class name {**

**// properties**

**// constructors**

**// methods**

**// others**

**}**

* Trong đó:
  + Phần access là access modifier, tức khả năng nhìn thấy của một thành phần so với thế giới bên ngoài lớp. Các access modifier có thể là public, protected, private. Mỗi lớp chỉ có 1 access modifier và nó chỉ xuất hiện 1 lần. Mỗi file chương trình Java chỉ có 1 và chỉ 1 public class. Ta sẽ lần lượt tìm hiểu kĩ hơn qua từng bài học.
  + non-acess là phần bổ trợ làm rõ nghĩa cho một lớp. Phần này có thể có hoặc không. non-access có thể là final, abstract, static, strictfp. Ta sẽ lần lượt tìm hiểu cụ thể trong các bài học. Bây giờ để cho đơn giản ta sẽ mặc định phần non-access là bỏ trống.
  + Tiếp theo là keyword class. Đây là keyword bắt buộc để khai báo và nhận biết lớp trong Java.
  + Phần name là tên của một lớp. Tên lớp là danh từ, cụm danh từ số ít mô tả đối tượng mà nó thể hiện. Tên lớp luôn bắt đầu bằng chữ cái hoa và nếu tên có nhiều từ, các từ viết liền nhau và viết hoa chữ cái đầu từ.
  + Trong cặp **{ }** chứa các thành phần mô tả và bổ trợ nhằm làm rõ đối tượng cần mô hình hóa. Phần này có thể có nhiều loại thành phần cấu thành như trên hoặc cũng có thể chỉ có 1 thành phần bất kì trong số đó.
  + Phần properties là các thuộc tính, còn gọi là các trường(fields) là phần mô tả đặc điểm đặc trưng của đối tượng. Đây là thành phần dữ liệu của lớp. Giả định từ giờ đến khi học bài tính đóng gói dữ liệu ta sẽ coi rằng các thuộc tính là public.
  + Phần constructors là các phương thức khởi tạo dùng để tạo đối tượng với những thông tin ban đầu từ lớp.
  + Phần methods là các phương thức khác constructors. Dùng để thể hiện các hành động, nhiệm vụ, hành vi của đối tượng mà lớp mô tả.
  + Phần others có thể là các class, enum, interface. Để cho đơn giản thì phần này sẽ được đề cập chi tiết trong các bài học về sau.

#### Ví dụ một số hình ảnh về lớp:

/\*\*

\* lớp Student để mô tả thông tin sinh viên

\*/

public class Student {

// các thuộc tính:

public String studentId; // mã sinh viên

public String fullName; // họ tên đầy đủ

public float gpa; // điểm TB

public String email; // địa chỉ email

public String address; // địa chỉ cư trú

// constructors:

/\*\*

\* phương thức khởi tạo mặc định

\*/

public Student() {

}

/\*\*

\* phương thức khởi tạo 1 tham số

\*

\* @param name chứa tên sinh viên

\*/

public Student(String name) {

fullName = name;

}

/\*\*

\* phương thức khởi tạo có hai tham số

\*

\* @param name chứa tên sinh viên

\* @param id chứa mã sinh viên

\*/

public Student(String name, String id) {

studentId = id;

fullName = name;

}

// các phương thức khác:

/\*\*

\* phương thức thể hiện hành động làm bài tập của SV

\* @param subject chứa tên môn học sinh viên cần làm bài tập

\*/

public void doHomework(String subject) {

// sinh viên đang làm bài tập...

}

/\*\*

\* phương thức thể hiện sinh viên làm bài thi môn nào đó

\* @param subject chứa tên môn học cần làm bài thi

\*/

public void doExam(String subject) {

// sinh viên làm bài thi...

}

}

/\*\*

\* lớp mô tả thông tin về con mèo

\*/

public class Cat {

// thuộc tính

public String name;

public int age;

public float weight;

public String color;

public String favoriteFood;

// constructor:

public Cat(String name) {

this.name = name;

}

// các phương thức khác:

/\*\*

\* phương thức mô tả con mèo nói chuyện với nhau như nào

\*/

public void speak() {

// mèo nói chuyện...

}

/\*\*

\* phương thức mô tả con mèo di chuyển như nào

\*/

public void move() {

// mèo đang bay qua nóc nhà...

}

/\*\*

\* phương thức mô tả mèo ăn thức ăn nào đó ra sao

\*

\* @param food món ăn nào đó như cá luộc chẳng hạn

\*/

public void eat(String food) {

// mèo ăn món food ngấu nghiến...

}

}

* Lớp mô tả thông tin các mặt hàng:

/\*\*

\* Lớp mô tả thông tin các mặt hàng trong siêu thị

\*/

class Item {

private String id;

private String name;

private float price;

private Date expiry;

public Item() {

}

public Item(String id, String name, float price, Date expiry) {

this.id = id;

this.name = name;

this.price = price;

this.expiry = expiry;

}

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public String getName() {

return name;

}

// ...

}

* Lớp mô tả thông tin mặt hàng được chọn đưa vào giỏ hàng:

/\*\*

\* Lớp mô tả các mặt hàng được người mua chọn đưa vào giỏ hàng

\*/

class SelectedItem extends Item {

private float salePercent;

private int quantity;

private float salePrice;

// ...

}

* Lớp mô tả thông tin và hành động của giỏ hàng:

/\*\*

\* Lớp mô tả thông tin và hành động của giỏ hàng

\*/

class Cart {

private String id; // mã giỏ hàng

private ArrayList<SelectedItem> items; // các sản phầm trong giỏ hàng

private int totalItem; // tổng số lượng mặt hàng

private float totalPrice; // tổng tiền

private float discountAmount; // số tiền giảm giá

private float subTotal; // tổng phụ(tiền phải thanh toán)

private String status; // trạng thái của giỏ hàng

// constructor 1 tham số

public Cart(String id) {

this.id = id;

}

// constructor nhiều tham số

public Cart(String id, ArrayList<SelectedItem> items, int totalItem,

float totalPrice, float discountAmount, float subTotal) {

this.id = id;

this.items = items;

this.totalItem = totalItem;

this.totalPrice = totalPrice;

this.discountAmount = discountAmount;

this.subTotal = subTotal;

}

public void addToCart(SelectedItem item) {

// thêm mặt hàng vào giỏ

}

public void remoteItem(SelectedItem item) {

// xóa mặt hàng khỏi giỏ hàng

}

public void checkout() {

// tiến hành thanh toán

}

public void exportBill() {

// xuất hóa đơn

}

// ...

}

#### Cú pháp tạo và sử dụng đối tượng

* Để sử tạo ra đối tượng ta dùng cú pháp: **type variable = new type(params);** với type là tên của lớp muốn sử dụng. params là danh sách tham số(có hoặc không).
* Ví dụ:

Cat tom = new Cat("Tom");

Student linh = new Student("Nguyễn Thùy Linh");

Student nhan = new Student("B25DCCN111", "Hoàng Trọng Nhân");

// ta có thể sử dụng kiểu tự suy luận:

var student = new Student("B25DCCN102", "Nguyễn Ngọc Mai");

* Để truy cập đến một thành phần nào đó của lớp, ta sử dụng tên đối tượng và toán tử dấu chấm sau đó là tên thành phần cần truy cập: **objectName.componentName**.
* Trong đó objectName là tên đối tượng cần truy cập, componentName là tên thành phần public nào đó.
* Ví dụ:

Cat tom = new Cat("Tom");

Student linh = new Student("Nguyễn Thùy Linh");

var nhan = new Student("B25DCCN111", "Hoàng Trọng Nhân");

// hiển thị tên của mèo tom:

System.out.println("Hello " + tom.name);

// lấy mã sinh viên:

System.out.println("Mã sinh viên: " + nhan.studentId);

// cho sinh viên Linh đi thi môn Java:

linh.doExam("Lập trình Java OOP");

### Ví dụ minh họa

* **Ví dụ 1: thông tin sinh viên cơ bản.**
* Sau đây là ví dụ đầy đủ về lớp Student.java:

package net.braniumacademy.lesson.chapter6;

/\*\*

\* Lớp Student để mô tả thông tin và hành động của sinh viên.

\*/

public class Student {

public String studentId; // mã sinh viên

public String fullName; // họ tên đầy đủ

public float gpa; // điểm TB

public String email; // địa chỉ email

public String address; // địa chỉ cư trú

/\*\*

\* Phương thức khởi tạo mặc định của lớp.

\*/

public Student() {

}

/\*\*

\* Phương thức khởi tạo 2 tham số.

\*

\* @param studentId mã sinh viên cho trước

\* @param fullName họ và tên sinh viên cho trước

\*/

public Student(String studentId, String fullName) {

this.studentId = studentId;

this.fullName = fullName;

}

/\*\*

\* Phương thức khởi tạo đối tượng sinh viên với các thông tin cho trước.

\*

\* @param studentId mã sinh viên

\* @param fullName họ và tên

\* @param gpa điểm trung bình tích lũy

\* @param email email

\* @param address địa chỉ

\*/

public Student(String studentId, String fullName,

float gpa, String email, String address) {

this.studentId = studentId;

this.fullName = fullName;

this.gpa = gpa;

this.email = email;

this.address = address;

}

/\*\*

\* Phương thức thể hiện hành động làm bài tập về nhà của sinh viên.

\*

\* @param subject môn học cần làm bài tập.

\*/

public void doHomework(String subject) {

System.out.println(STR."Sinh viên \{fullName} đang làm bài tập môn \{subject}.");

}

/\*\*

\* Phương thức thể hiện hành động làm bài kiểm tra của sinh viên.

\*

\* @param subject môn học cần làm bài kiểm tra.

\*/

public void doExam(String subject) {

System.out.println(STR."Sinh viên \{fullName} đang làm bài kiểm tra môn \{subject}.");

}

/\*\*

\* Phương thức chuyển thông tin sinh viên thành chuỗi văn bản mô tả tương ứng.

\*

\* @return một String mô tả thông tin sinh viên

\*/

@Override

public String toString() {

return STR."Student{studentId='\{studentId}\{'\''}, fullName='\{fullName}\{'\''}, gpa=\{gpa}, " +

STR."email='\{email}\{'\''}, address='\{address}\{'\''}\{'}'}";

}

}

* Lớp Test.java:

package net.braniumacademy.lesson.chapter6;

public class Lesson61 {

public static void main(String[] args) {

// tạo một mảng các sinh viên

Student[] students = new Student[10];

// liên kết từng phần tử với đối tượng sinh viên mong muốn:

students[0] = new Student("SV1002", "Nguyễn Thúy Lan");

students[1] = new Student("SV1003", "Nguyễn Hải Nam",

3.26f, "namdeptrai@xmail.com", "Hà Nội");

students[2] = new Student("SV1005", "Nguyễn Trần Khánh Nga",

3.26f, "ngacute@xmail.com", "Hà Nội");

// tạo một đối tượng sinh viên đơn lẻ

var loan = new Student("SV1001", "Nguyễn Mai Loan");

// truy cập vào các hành động và thuộc tính

loan.doExam("Java OOP");

loan.doHomework("DSA");

System.out.println(STR."Mã SV: \{loan.studentId}");

System.out.println(STR."Họ và tên: \{loan.fullName}");

System.out.println(STR."Địa chỉ: \{loan.address}");

// gọi phương thức hiển thị danh sách sinh viên

showStudentInfo(students);

}

private static void showStudentInfo(Student[] students) {

System.out.println("===> Danh sách sinh viên trong mảng: ");

for (var student : students) {

System.out.println(student); // gọi tới phương thức toString() của lớp Student

}

}

}

* Kết quả:

Sinh viên Nguyễn Mai Loan đang làm bài kiểm tra môn Java OOP.

Sinh viên Nguyễn Mai Loan đang làm bài tập môn DSA.

Mã SV: SV1001

Họ và tên: Nguyễn Mai Loan

Địa chỉ: null

===> Danh sách sinh viên trong mảng:

Student{studentId='SV1002', fullName='Nguyễn Thúy Lan', gpa=0.0, email='null', address='null'}

Student{studentId='SV1003', fullName='Nguyễn Hải Nam', gpa=3.26, email='namdeptrai@xmail.com', address='Hà Nội'}

Student{studentId='SV1005', fullName='Nguyễn Trần Khánh Nga', gpa=3.26, email='ngacute@xmail.com', address='Hà Nội'}

null

null

null

null

null

null

null

* **Ví dụ 2: thông tin mặt hàng cơ bản.**
  + File Item.java:

/\*\*

\* Lớp mô tả thông tin các mặt hàng trong siêu thị

\*/

class Item {

private String id; // mã mặt hàng

private String name; // tên mặt hàng

private float price; // giá gốc

private Date expiry; // hạn sử dụng

private String brand; // hãng sản xuất

private String productType; // dòng sản phẩm, ví dụ: thường, pro, ultra...

public Item() {

}

public Item(String id, String name, float price,

Date expiry, String brand, String productType) {

this.id = id;

this.name = name;

this.price = price;

this.expiry = expiry;

this.brand = brand;

this.productType = productType;

}

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public float getPrice() {

return price;

}

public void setPrice(float price) {

this.price = price;

}

public Date getExpiry() {

return expiry;

}

public void setExpiry(Date expiry) {

this.expiry = expiry;

}

public String getBrand() {

return brand;

}

public void setBrand(String brand) {

this.brand = brand;

}

public String getProductType() {

return productType;

}

public void setProductType(String productType) {

this.productType = productType;

}

}

* + File Test.java:

package net.braniumacademy.lesson.chapter6;

import java.text.SimpleDateFormat;

import java.util.Date;

public class Lesson61 {

public static void main(String[] args) {

Item iphone16 = new Item("ip16", "iPhone 16", 26560000,

null, "Apple", "Normal");

Item iphone16Pro = new Item("ip16pro", "iPhone 16 Pro", 28860000,

null, "Apple", "Pro");

// hiện thông tin sản phẩm

Lesson61 lesson61 = new Lesson61();

lesson61.showHeader();

lesson61.showInfo(iphone16);

lesson61.showInfo(iphone16Pro);

}

private void showInfo(Item item) {

System.out.printf("%-12s%-25s%-12.2f%-15s%-20s%-30s\n",

item.getId(), item.getName(), item.getPrice(),

expiryDateString(item.getExpiry()), item.getBrand(),

item.getProductType());

}

private void showHeader() {

System.out.println("========= Thông tin sản phẩm ==========");

System.out.printf("%-12s%-25s%-12s%-15s%-20s%-30s\n",

"Mã SP", "Tên sản phẩm", "Giá", "Hạn sử dụng", "Hãng SX", "Dòng SP");

}

private String expiryDateString(Date date) {

if (date == null) {

return "null";

}

var simpleDateFormat = new SimpleDateFormat("dd/MM/yyyy");

return simpleDateFormat.format(date);

}

}

========= Thông tin sản phẩm ==========

Mã SP Tên sản phẩm Giá Hạn sử dụng Hãng SX Dòng SP

ip16 iPhone 16 26560000.00 null Apple Normal

ip16pro iPhone 16 Pro 28860000.00 null Apple Pro

**Bài 6.2. Các phương thức khởi tạo**

**Nội dung bài học**

1. [Mục đích sử dụng](https://braniumacademy.net/courses/java/lessons/java-constructor/#s1)
2. [Cú pháp tổng quát](https://braniumacademy.net/courses/java/lessons/java-constructor/#s2)
3. [Nhắc lại về constructor không tham số](https://braniumacademy.net/courses/java/lessons/java-constructor/#s3)
4. [Các đặc trưng của phương thức khởi tạo](https://braniumacademy.net/courses/java/lessons/java-constructor/#s4)
5. [Sử dụng keyword this](https://braniumacademy.net/courses/java/lessons/java-constructor/#s5)
6. [Cấm một lớp tạo đối tượng](https://braniumacademy.net/courses/java/lessons/java-constructor/#s6)
7. [Bài tập thực hành](https://braniumacademy.net/courses/java/lessons/java-constructor/#s7)

**Mục đích sử dụng**

* Mục đích sử dụng của phương thức khởi tạo là để tạo ra đối tượng của lớp với những dữ liệu ban đầu cho trước.
* Mọi lớp muốn tạo đối tượng đều phải có phương thức khởi tạo với access modifier khác private.
* Chúng ta không sử dụng phương thức khởi tạo với các mục đích khác.

**Cú pháp tổng quát**

* Sau đây là cú pháp tổng quát của phương thức khởi tạo:

**access name(params) {**

**// các câu lệnh cần thực hiện**

**}**

* Trong đó:
  + Phần *access*là access modifier, có thể là public, protected, private hoặc để trống. Ý nghĩa của access modifier là giới hạn việc cho phép phương thức khởi tạo được gọi từ bên ngoài lớp ở mức độ nào. Nếu access là public có nghĩa là mọi nơi đều có thể nhìn thấy và gọi phương thức khởi tạo để tạo đối tượng từ lớp. Nếu là private thì chỉ có thể gọi phương thức khởi tạo từ bên trong lớp. Riêng protected sẽ nói sau.
  + Phần *name*là tên phương thức khởi tạo. Bắt buộc tên của phương thức khởi tạo phải giống chính xác như tên lớp định nghĩa nó. Nếu không sẽ không có phương thức khởi tạo hợp lệ.
  + Tiếp sau đó là cặp ngoặc tròn chứa danh sách tham số *params*bên trong. Giống như phương thức thông thường, phương thức khởi tạo cũng có thể có 0, 1 hoặc nhiều tham số. Nhằm cung cấp các giá trị ban đầu cho đối tượng khi ta tạo ra nó.
  + Phần thân phương thức nằm trong cặp { } và chỉ chứa các câu lệnh thực hiện việc khởi tạo, gán hoặc gọi tới các phương thức cần thiết khác khi tạo lập đối tượng.
* Ví dụ:

package net.braniumacademy.lesson.chapter6;

/\*\*

\* Lớp mô tả thông tin về chiếc ô tô.

\*/

public class Car {

private String carId;

private String name;

private String type;

private String brand;

private int year;

private float weight;

public Car() {

}

public Car(String name) {

this.name = name;

}

public Car(String name, String brand) {

this.name = name;

this.brand = brand;

}

public Car(String name, String type, String brand, int year) {

this.name = name;

this.type = type;

this.brand = brand;

this.year = year;

}

public Car(String carId, String name, String type,

String brand, int year, float weight) {

this.carId = carId;

this.name = name;

this.type = type;

this.brand = brand;

this.year = year;

this.weight = weight;

}

}

* Tại nơi sử dụng, ta tạo đối tượng của Car thông qua các constructor:

package net.braniumacademy.lesson.chapter6;

public class Lesson62 {

public static void main(String[] args) {

// tạo lập các đối tượng của lớp Car

Car car = new Car();

Car car1 = new Car ("Vios", "Toyota");

Car car2 = new Car ("Vios", "G", "Toyota", 2030);

Car car3 = new Car ("toy-vios", "Vios", "G", "Toyota", 2030, 1050.0f);

}

}

**Nhắc lại về constructor không tham số**

* Mỗi lớp Java khi được tạo ra mặc định sẽ có một phương thức khởi tạo mặc định không tham số đi liền với nó.
* Phương thức này không được thể hiện trong định nghĩa lớp.
* Do đó nếu bạn khai báo một lớp Car như sau:

public class Car {

public String name;

public int year;

public float weight;

}

* Thì sau đó bạn hoàn toàn có thể tạo mới đối tượng của lớp Car: Car myCar = new Car(). Hiển nhiên bạn không thấy phương thức khởi tạo Car() nào trong định nghĩa trên. Đó là tác dụng của default constructor.
* Khi bạn sửa đổi lớp trên thành như sau:

public class Car {

public String name;

public int year;

public float weight;

// phương thức khởi tạo 1 tham số

public Car(String name) {

this.name = name;

}

// phương thức khởi tạo 3 tham số

public Car(String name, int year, float weight) {

this.name = name;

this.year = year;

this.weight = weight;

}

}

* Sau đó bạn thực hiện tạo đối tượng mới của Car:

Car car = new Car(); // lỗi

Car car = new Car("Testla Model W"); // ok

Car car = new Car("Mazda CX8", 2025, 2050); // ok

* Bạn nhận ra là không thể tạo đối tượng với default constructor nữa. Tại sao? Lí do là vì khi bạn nạp chồng các constructor với bộ n > 1 tham số, default constructor sẽ bị**vô hiệu hóa**. Do đó muốn sử dụng default constructor hay constructor không có tham số bạn **phải định nghĩa tường minh nó trong lớp** của bạn.
* Ta sửa lại lớp Car như sau:

public class Car {

public String name;

public int year;

public float weight;

// phương thức khởi tạo không tham số

public Car() {

// không cần làm gì cả

}

// phương thức khởi tạo 1 tham số

public Car(String name) {

this.name = name;

}

// phương thức khởi tạo 3 tham số

public Car(String name, int year, float weight) {

this.name = name;

this.year = year;

this.weight = weight;

}

}

* Lúc này bạn có thể tạo đối tượng của lớp Car tùy thích với cả 3 constructor đã khai báo.
* Nhấn mạnh rằng TA CHỈ TẠO ĐƯỢC ĐỐI TƯỢNG CỦA LỚP VỚI CÁC CONSTRUCTOR ĐÃ ĐƯỢC KHAI BÁO/ĐỊNH NGHĨA TƯỜNG MINH TRONG LỚP đang muốn sử dụng:

Car car = new Car(); // ok vì đã khai báo constructor Car() không tham số

Car car = new Car("Testla Model Y"); // ok vì đã khai báo constructor Car(String name)

Car car = new Car("Mazda CX8", 2025, 2050); // ok vì đã khai báo constructor 3 tham số

// Car(String name, int year, float weight)

**Các đặc trưng của phương thức khởi tạo**

* Phần access của phương thức khởi tạo chỉ có thể chứa tối đa 01 keyword. Nếu có sự xuất hiện từ 2 keyword trở lên kể cả việc lặp lại keyword trước đó bạn sẽ bị báo lỗi.
* Phương thức khởi tạo có thể được nạp chồng. Không được phép tồn tại hai phương thức khởi tạo cùng có bộ tham số y hệt nhau.
* Phương thức khởi tạo không có kiểu trả về.
* Không thực hiện gọi đệ quy tới chính nó.
* Phương thức khởi tạo không thể là final, abstract, static, native, strictfp hay synchronized.
* Nếu một lớp không được khai báo phương thức khởi tạo tường minh nào. Java sẽ tự động tạo cho lớp đó một phương thức khởi tạo ngầm định.
* Phương thức khởi tạo mặc định có cùng access modifier, cùng tên với tên lớp, không chứa tham số, không có thực thi nào cả.
* Nếu có một phương thức khởi tạo có tham số được định nghĩa tường minh thì phương thức khởi tạo mặc định sẽ bị ẩn đi. Muốn sử dụng ta phải khai báo tường minh phương thức này ra.
* Các thuộc tính của lớp nếu không được khởi tạo thì nó sẽ nhận giá trị mặc định của kiểu mà nó đang mang. Các kiểu số giá trị mặc định là 0, kiểu boolean là false, kiểu tham chiếu là null.
* Ví dụ:

public class Car {

public String name;

}

// tương đương với:

public class Car {

public String name;

// phương thức khởi tạo mặc định

public Car() {

super();

}

}

// do đó ta có thể thực hiện việc tạo đối tượng của lớp Car:

Car myCar = new Car(); // ok

**Sử dụng keyword this**

* Trong ví dụ trên ta đã thấy sự xuất hiện của từ khóa this. Về bản chất this chính là đối tượng của lớp, nó đang thực hiện các hành động được đề cập tại thời điểm hiện tại.
* Và sau đây là các trường hợp sử dụng phổ biến của this:
  + Sử dụng this để chỉ rõ thuộc tính của lớp nếu như tham số của phương thức trùng với tên của thuộc tính. Để đơn giản hóa vấn đề.

package net.braniumacademy.lesson.chapter6;

public class Cat {

private String name;

private float weight;

private int age;

private String favoriteFood;

// sử dụng this khi tham số trùng tên với tên thuộc tính.

// lý do: trong một phương thức, biến cục bộ có mức ưu tiên cao hơn

// biến class(thuộc tính của lớp)

public Cat(String name, float weight, int age, String favoriteFood) {

this.name = name; // bên có this. là thuộc tính

this.weight = weight; // bên không có this là tham số

this.age = age;

this.favoriteFood = favoriteFood;

}

}

* + Sử dụng this để gọi tới phương thức khởi tạo khác trong cùng một lớp. Mục đích tái sử dụng lại code đã có.

package net.braniumacademy.lesson.chapter6;

public class Cat {

private String name;

private float weight;

private int age;

private String favoriteFood;

public Cat() {

}

public Cat(String name) {

this.name = name;

}

public Cat(String name, int age) {

this(name); // gọi constructor 1 tham số

this.age = age;

}

public Cat(String name, float weight, int age) {

this(name, age); // gọi constructor 2 tham số

this.weight = weight;

}

public Cat(String name, float weight, int age, String favoriteFood) {

this(name, weight, age); // gọi constructor 3 tham số

this.favoriteFood = favoriteFood;

}

}

* + Sử dụng this để đại diện cho đối tượng hiện thời của lớp.
* Lưu ý this không áp dụng cho các biến cục bộ, các thành phần static.
* Với lời gọi tới phương thức khởi tạo khác qua this thì lời gọi này phải là lệnh đầu tiên trong phương thức khởi tạo.
* Ví dụ sử dụng this:

package net.braniumacademy.lesson.chapter6;

/\*\*

\* Lớp mô tả thông tin về chiếc ô tô.

\*/

public class Car {

private String carId;

private String name;

private String type;

private String brand;

private int year;

private float weight;

private boolean isActive;

private char typeName;

public Car() {

}

public Car(String name) {

this.name = name;

}

public Car(String name, String brand) {

this(name);

this.brand = brand;

System.out.println(this);

}

public Car(String name, String type, String brand, int year) {

this(name, brand);

this.type = type;

this.year = year;

}

public Car(String carId, String name, String type, String brand, int year, float weight) {

this(name, type, brand, year);

this.carId = carId;

this.weight = weight;

}

public void startEngine() {

// khởi động xe

System.out.println(this);

}

}

* Tại nơi sử dụng:

package net.braniumacademy.lesson.chapter6;

public class Lesson62 {

public static void main(String[] args) {

// tạo lập các đối tượng của lớp Car

Car car = new Car();

Car car2 = new Car("Camry", "Toyota");

car2.startEngine();

System.out.println(car2);

}

}

net.braniumacademy.lesson.chapter6.Car@723279cf

net.braniumacademy.lesson.chapter6.Car@723279cf

net.braniumacademy.lesson.chapter6.Car@723279cf

// cùng mã hash code do đó các đối tượng đang thực hiện lời gọi là 1

* Ví dụ sau là các trường hợp bị lỗi:

public class Car {

public String name;

public int year;

public float weight;

// phương thức khởi tạo không tham số

public Car() {

}

// phương thức khởi tạo 1 tham số

public Car(String name) {

this.name = name;

}

public Car(String name, int year) {

this.year = year; // year ở this.year là thuộc tính

// year ở vế phải dấu = là tham số

this(name); // error! lời gọi này phải là lời gọi đầu tiên

}

// phương thức khởi tạo 3 tham số

public Car(String name, int year, float weight) {

this(name, year, weight); // gọi đệ quy chính nó -> error!

this.weight = weight; // gán giá trị trong tham số weight cho

// thuộc tính weight

}

}

**Cấm 1 lớp tạo đối tượng**

* Để ngăn chặn việc tạo đối tượng từ bên ngoài lớp ta sử dụng private package access hoặc **private constructor**.
* private package constructor sẽ ngăn cản việc tạo đối tượng từ bên ngoài một gói(package).
* Còn private constructor sẽ ngăn cản việc tạo đối tượng từ bên ngoài lớp.
* Điều này rất hữu dụng nếu bạn muốn quản lý việc tạo đối tượng của lớp.
* Ví dụ khi kết nối ứng dụng đến cơ sở dữ liệu. Bạn muốn kiểm soát đối tượng quản lý việc truy xuất DB và giới hạn nó về 1 đối tượng duy nhất. Lúc này giải pháp của bạn là sử dụng **singleton parttern**. Nguyên lý của phương pháp này là tạo ra một đối tượng duy nhất của lớp để dùng chung và constructor của lớp sẽ để cấp độ truy cập là private.
* Ví dụ sau minh họa sử dụng private constructor:

public class Car {

public String name;

public int year;

public float weight;

// phương thức khởi tạo private

private Car() {

}

public Car createNewCar() {

return new Car(); // ok vì đang thao tác trong cùng lớp Car

}

}

// lớp Test.java

public class Test {

public static void main(String[] args) {

Car myCar = new Car(); // error!

}

}

* Ví dụ sau sẽ minh họa private package access constructor:

package net.braniumacademy.lesson.chapter6;

public class Car {

private String name;

private String brand;

private int year;

private float weight;

// phương thức khởi tạo default package access

Car() {

}

Car(String name, String brand, int year, float weight) {

this.name = name;

this.brand = brand;

this.year = year;

this.weight = weight;

}

}

package net.braniumacademy.lesson.other; // package other

import net.braniumacademy.lesson.chapter6.Car; // package chapter6

public class Test {

public static void main(String[] args) {

// truy cập từ một package khác

Car car = new Car(); // error

Car otherCar = new Car("Camry", "Toyota", 2025, 1520); // error

}

}

**Bài 6.3. Tính đóng gói dữ liệu**

**Nội dung bài học**

1. [Mục đích sử dụng](https://braniumacademy.net/courses/java/lessons/java-encapsulation/#s1)
2. [Các phương thức getter/setter](https://braniumacademy.net/courses/java/lessons/java-encapsulation/#s2)
3. [Các phương thức private](https://braniumacademy.net/courses/java/lessons/java-encapsulation/#s3)
4. [Ví dụ minh họa](https://braniumacademy.net/courses/java/lessons/java-encapsulation/#s4)
5. [Bài tập thực hành](https://braniumacademy.net/courses/java/lessons/java-encapsulation/#s5)

**Mục đích sử dụng**

* Tính đóng gói dữ liệu là 1 trong 4 tính chất của ngôn ngữ lập trình Java. Các tính chất còn lại là: tính kế thừa, tính đa hình và tính trừu tượng
* Tính đóng gói dữ liệu là kĩ thuật gói gọn các dữ liệu và các chức năng vận hành trên dữ liệu đó vào trong một đơn vị đơn nhất(1 lớp).
* Với tính đóng gói dữ liệu, dữ liệu của một lớp sẽ được bảo vệ khỏi thế giới bên ngoài lớp. Chỉ những thành phần bên trong lớp đó mới nhìn thấy và sử dụng được các dữ liệu này.
* Nói cách khác, với tính đóng gói dữ liệu, dữ liệu của lớp nào chỉ lớp đó được phép trực tiếp truy xuất. Các lớp khác muốn sử dụng hoặc thao tác với các dữ liệu này phải thông qua các phương thức public mà lớp cung cấp.
* Biểu hiện của tính đóng gói dữ liệu gồm:
  + Các thuộc tính(dữ liệu) sẽ có access modifier là private.
  + Có các getter/setter public để thao tác với các thuộc tính private kể trên.
* Ví dụ:

package net.braniumacademy.lesson.chapter6;

/\*\*

\* Lớp Student để mô tả thông tin và hành động của sinh viên.

\*/

public class Student {

private String studentId; // mã sinh viên

private String fullName; // họ tên đầy đủ

private float gpa; // điểm TB

private String email; // địa chỉ email

private String address; // địa chỉ cư trú

/\*\*

\* Phương thức khởi tạo không tham số.

\*/

public Student() {

}

/\*\*

\* Phương thức khởi tạo 2 tham số.

\*

\* @param studentId mã sinh viên cho trước

\* @param fullName họ và tên sinh viên cho trước

\*/

public Student(String studentId, String fullName) {

this.studentId = studentId;

this.fullName = fullName;

}

/\*\*

\* Phương thức khởi tạo đối tượng sinh viên với các thông tin cho trước.

\*

\* @param studentId mã sinh viên

\* @param fullName họ và tên

\* @param gpa điểm trung bình tích lũy

\* @param email email

\* @param address địa chỉ

\*/

public Student(String studentId, String fullName,

float gpa, String email, String address) {

this.studentId = studentId;

this.fullName = fullName;

this.gpa = gpa;

this.email = email;

this.address = address;

}

public String getStudentId() {

return studentId;

}

public void setStudentId(String studentId) {

this.studentId = studentId;

}

/\*\*

\* Phương thức thể hiện hành động làm bài tập về nhà của sinh viên.

\*

\* @param subject môn học cần làm bài tập.

\*/

public void doHomework(String subject) {

System.out.println(STR."Sinh viên \{fullName} đang làm bài tập môn \{subject}.");

}

/\*\*

\* Phương thức thể hiện hành động làm bài kiểm tra của sinh viên.

\*

\* @param subject môn học cần làm bài kiểm tra.

\*/

public void doExam(String subject) {

System.out.println(STR."Sinh viên \{fullName} đang làm bài kiểm tra môn \{subject}.");

}

/\*\*

\* Phương thức chuyển thông tin sinh viên thành chuỗi văn bản mô tả tương ứng.

\*

\* @return một String mô tả thông tin sinh viên

\*/

@Override

public String toString() {

return STR."Student{studentId='\{studentId}\{'\''}, fullName='\{fullName}\{'\''}, gpa=\{gpa}, " +

STR."email='\{email}\{'\''}, address='\{address}\{'\''}\{'}'}";

}

}

* Lớp sử dụng thông tin sinh viên:

package net.braniumacademy.lesson.chapter6;

public class Lesson63 {

public static void main(String[] args) {

Student student = new Student();

System.out.println("Trước khi cập nhật: ");

System.out.println(STR."Mã SV: \{student.getStudentId()}");

student.setStudentId("SV1009");

System.out.println("Sau khi cập nhật: ");

System.out.println(STR."Mã SV: \{student.getStudentId()}");

}

}

**Các getter/setter**

* Để đọc dữ liệu của các thuộc tính tương ứng ra ta dùng các phương thức getX() gọi chung là các getter methods.
* Kiểu dữ liệu của getter luôn cùng kiểu với kiểu của thuộc tính ta cần đọc ra.
* Tên của getter luôn có dạng: getX() trong đó X là tên thuộc tính cần đọc. Ví dụ thuộc tính studentId sẽ có getter tương ứng là getStudentId().
* Getter không nhận bất kì tham số nào.
* Để thay đổi, cập nhật dữ liệu của đối tượng ta sử dụng các phương thức setX() gọi chung là các setter methods.
* Setter thường có kiểu void với định dạng setX(value) với x là tên thuộc tính cần thay đổi giá trị.
* Phương thức setter luôn nhận vào một tham số cùng kiểu với kiểu của thuộc tính private ta cần sửa đổi.
* Trong setter có thể thực hiện các hành động kiểm tra, so khớp trước khi gán giá trị để đảm bảo dữ liệu luôn hợp lệ.
* Một thuộc tính có thể có đầy đủ cả getter/setter hoặc cũng có thể chỉ có getter hoặc chỉ có setter.
* Ví dụ:

package net.braniumacademy.lesson.chapter6;

/\*\*

\* Lớp Student để mô tả thông tin và hành động của sinh viên.

\*/

public class Student {

private String studentId; // mã sinh viên

private String fullName; // họ tên đầy đủ

private float gpa; // điểm TB

private String email; // địa chỉ email

private String address; // địa chỉ cư trú

/\*\*

\* Phương thức khởi tạo không tham số.

\*/

public Student() {

}

/\*\*

\* Phương thức khởi tạo 2 tham số.

\*

\* @param studentId mã sinh viên cho trước

\* @param fullName họ và tên sinh viên cho trước

\*/

public Student(String studentId, String fullName) {

setStudentId(studentId);

setFullName(fullName);

}

/\*\*

\* Phương thức khởi tạo đối tượng sinh viên với các thông tin cho trước.

\*

\* @param studentId mã sinh viên

\* @param fullName họ và tên

\* @param gpa điểm trung bình tích lũy

\* @param email email

\* @param address địa chỉ

\*/

public Student(String studentId, String fullName,

float gpa, String email, String address) {

this(studentId, fullName);

setGpa(gpa);

setEmail(email);

setAddress(address);

}

// tạo các getter/setter

// Alt Ins -> Getter / Setter -> select properties -> Enter

public String getStudentId() {

return studentId;

}

public void setStudentId(String studentId) {

this.studentId = studentId;

}

public String getFullName() {

return fullName;

}

public void setFullName(String fullName) {

this.fullName = fullName;

}

public float getGpa() {

return gpa;

}

public void setGpa(float gpa) {

if (gpa < 0 || gpa > 4) {

System.out.println("==> Gpa không hợp lệ.");

this.gpa = 0;

} else {

this.gpa = gpa;

}

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public String getAddress() {

return address;

}

public void setAddress(String address) {

this.address = address;

}

/\*\*

\* Phương thức thể hiện hành động làm bài tập về nhà của sinh viên.

\*

\* @param subject môn học cần làm bài tập.

\*/

public void doHomework(String subject) {

System.out.println(STR."Sinh viên \{fullName} đang làm bài tập môn \{subject}.");

}

/\*\*

\* Phương thức thể hiện hành động làm bài kiểm tra của sinh viên.

\*

\* @param subject môn học cần làm bài kiểm tra.

\*/

public void doExam(String subject) {

System.out.println(STR."Sinh viên \{fullName} đang làm bài kiểm tra môn \{subject}.");

}

/\*\*

\* Phương thức chuyển thông tin sinh viên thành chuỗi văn bản mô tả tương ứng.

\*

\* @return một String mô tả thông tin sinh viên

\*/

@Override

public String toString() {

return STR."Student{studentId='\{studentId}\{'\''}, fullName='\{fullName}\{'\''}, gpa=\{gpa}, " +

STR."email='\{email}\{'\''}, address='\{address}\{'\''}\{'}'}";

}

}

* Tại nơi sử dụng:

package net.braniumacademy.lesson.chapter6;

public class Lesson63 {

public static void main(String[] args) {

Student student = new Student();

System.out.println("Trước khi sửa đổi: ");

System.out.println(STR."Mã sinh viên: \{student.getStudentId()}");

student.setStudentId("SV1005");

student.setGpa(3.25f);

System.out.println("Sau khi sửa đổi: ");

System.out.println(STR."Mã sinh viên: \{student.getStudentId()}");

System.out.println(STR."Gpa: \{student.getGpa()}");

}

}

Trước khi sửa đổi:

Mã sinh viên: null

Sau khi sửa đổi:

Mã sinh viên: SV1005

Gpa: 3.25

**Các phương thức private**

* Tương tự như các thuộc tính private, phương thức private cũng chỉ có thể được sử dụng trực tiếp ở bên trong một lớp.
* Khi ta loại bỏ các private medthod sẽ không ảnh hưởng gì đến code bên ngoài. Vì đơn giản không có bất kì lời gọi nào tới phương thức private xuất phát từ phía bên ngoài lớp hiện thời.
* Mục đích của các private methods là để thực hiện các xử lý nội bộ nào đó trong một lớp. Thường dùng để tạo các helper method(phương thức phụ trợ) cục bộ của lớp.
* Nếu một nhiệm vụ nào đó có nhiều vấn đề nhỏ hơn cần xử lý thì ta có thể chia các vấn đề đó thành các private method để xử lý.
* Khi sinh code trong công cụ IntelliJ ta thấy rằng các phương thức sinh ra mặc định là private. Nếu muốn cho phép các phương thức này được sử dụng bởi các thành phần bên ngoài lớp, thì ta chỉ cần đổi private thành một trong các access modifier khác tùy mục đích: public, protected hoặc không có access modifier.
* Mẫu kiểm tra họ tên có hợp lệ không, áp dụng cho cả tiếng Việt có dấu. Chi tiết về regular expression bạn xem trong chương biểu thức chính quy:

.\*[^a-zắằẳẵặăấầẩẫậâáàãảạđếềểễệêéèẻẽẹíìỉĩịốồổỗộôớờởỡợơóòõỏọứừửữựưúùủũụýỳỷỹỵ\s]+.\*

* Ví dụ sử dụng phương thức private để kiểm tra tính hợp lệ của điểm gpa, họ và tên:

public void setFullName(String fullName) {

if (isFullNameValid(fullName)) {

this.fullName = fullName;

} else {

System.out.println("==> Họ và tên không hợp lệ.");

}

}

/\*\*

\* Phương thức hỗ trợ kiểm tra nội bộ thông tin họ tên sinh viên

\* có hợp lệ không.

\* @param fullName họ và tên cần kiểm tra

\* @return true nếu họ và tên hợp lệ và false nếu ngược lại

\*/

private boolean isFullNameValid(String fullName) {

if (fullName.trim().isEmpty()) {

return false;

}

var str = fullName.trim().toLowerCase();

return !str.matches(".\*[^a-zắằẳẵặăấầẩẫậâáàãảạđếềểễệêéèẻẽẹíìỉĩịốồổỗộôớờởỡợơóòõỏọứừửữựưúùủũụýỳỷỹỵ\\s]+.\*");

}

public void setGpa(float gpa) {

if (!isValidGpa(gpa)) {

System.out.println("==> Gpa không hợp lệ.");

this.gpa = 0;

} else {

this.gpa = gpa;

}

}

/\*\*

\* Phương thức kiểm tra điểm gpa có hợp lệ không. Ở đây chỉ xét điểm hệ 4.

\* @param gpa điểm cần kiểm tra

\* @return true nếu gpa hợp lệ ngược lại trả về false

\*/

private boolean isValidGpa(float gpa) {

return !(gpa < 0 || gpa > 4);

}

**Ví dụ minh họa**

* Sau đây là code mẫu đầy đủ của lớp Student.java:

package net.braniumacademy.lesson.chapter6;

/\*\*

\* Lớp Student để mô tả thông tin và hành động của sinh viên.

\*/

public class Student {

private String studentId; // mã sinh viên

private String fullName; // họ tên đầy đủ

private float gpa; // điểm TB

private String email; // địa chỉ email

private String address; // địa chỉ cư trú

/\*\*

\* Phương thức khởi tạo không tham số.

\*/

public Student() {

}

/\*\*

\* Phương thức khởi tạo 2 tham số.

\*

\* @param studentId mã sinh viên cho trước

\* @param fullName họ và tên sinh viên cho trước

\*/

public Student(String studentId, String fullName) {

setStudentId(studentId);

setFullName(fullName);

}

/\*\*

\* Phương thức khởi tạo đối tượng sinh viên với các thông tin cho trước.

\*

\* @param studentId mã sinh viên

\* @param fullName họ và tên

\* @param gpa điểm trung bình tích lũy

\* @param email email

\* @param address địa chỉ

\*/

public Student(String studentId, String fullName,

float gpa, String email, String address) {

this(studentId, fullName);

setGpa(gpa);

setEmail(email);

setAddress(address);

}

// tạo các getter/setter

// Alt Ins -> Getter / Setter -> select properties -> Enter

public String getStudentId() {

return studentId;

}

public void setStudentId(String studentId) {

this.studentId = studentId;

}

public String getFullName() {

return fullName;

}

public void setFullName(String fullName) {

if (isFullNameValid(fullName)) {

this.fullName = fullName;

} else {

System.out.println("==> Họ và tên không hợp lệ.");

}

}

/\*\*

\* Phương thức hỗ trợ kiểm tra nội bộ thông tin họ tên sinh viên

\* có hợp lệ không.

\* @param fullName họ và tên cần kiểm tra

\* @return true nếu họ và tên hợp lệ và false nếu ngược lại

\*/

private boolean isFullNameValid(String fullName) {

if (fullName.trim().isEmpty()) {

return false;

}

var str = fullName.trim().toLowerCase();

return !str.matches(".\*[^a-zắằẳẵặăấầẩẫậâáàãảạđếềểễệêéèẻẽẹíìỉĩịốồổỗộôớờởỡợơóòõỏọứừửữựưúùủũụýỳỷỹỵ\\s]+.\*");

}

public float getGpa() {

return gpa;

}

public void setGpa(float gpa) {

if (!isValidGpa(gpa)) {

System.out.println("==> Gpa không hợp lệ.");

this.gpa = 0;

} else {

this.gpa = gpa;

}

}

/\*\*

\* Phương thức kiểm tra điểm gpa có hợp lệ không. Ở đây chỉ xét điểm hệ 4.

\* @param gpa điểm cần kiểm tra

\* @return true nếu gpa hợp lệ ngược lại trả về false

\*/

private boolean isValidGpa(float gpa) {

return !(gpa < 0 || gpa > 4);

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public String getAddress() {

return address;

}

public void setAddress(String address) {

this.address = address;

}

/\*\*

\* Phương thức thể hiện hành động làm bài tập về nhà của sinh viên.

\*

\* @param subject môn học cần làm bài tập.

\*/

public void doHomework(String subject) {

System.out.println(STR."Sinh viên \{fullName} đang làm bài tập môn \{subject}.");

}

/\*\*

\* Phương thức thể hiện hành động làm bài kiểm tra của sinh viên.

\*

\* @param subject môn học cần làm bài kiểm tra.

\*/

public void doExam(String subject) {

System.out.println(STR."Sinh viên \{fullName} đang làm bài kiểm tra môn \{subject}.");

}

/\*\*

\* Phương thức chuyển thông tin sinh viên thành chuỗi văn bản mô tả tương ứng.

\*

\* @return một String mô tả thông tin sinh viên

\*/

@Override

public String toString() {

return STR."Student{studentId='\{studentId}\{'\''}, fullName='\{fullName}\{'\''}, gpa=\{gpa}, " +

STR."email='\{email}\{'\''}, address='\{address}\{'\''}\{'}'}";

}

}

* Code của lớp Lesson63.java:

package net.braniumacademy.lesson.chapter6;

public class Lesson63 {

public static void main(String[] args) {

Student student = new Student();

System.out.println("Trước khi sửa đổi: ");

System.out.println(STR."Mã sinh viên: \{student.getStudentId()}");

// cập nhật thông tin sinh viên:

student.setStudentId("SV1005");

student.setGpa(3.25f);

student.setFullName("Lê Nguyễn Bá Nam");

System.out.println("Sau khi sửa đổi: ");

System.out.println(STR."Mã sinh viên: \{student.getStudentId()}");

System.out.println(STR."Họ và tên: \{student.getFullName()}");

System.out.println(STR."Gpa: \{student.getGpa()}");

}

}

* Kết quả:

Trước khi sửa đổi:

Mã sinh viên: null

Sau khi sửa đổi:

Mã sinh viên: SV1005

Họ và tên: Lê Nguyễn Bá Nam

Gpa: 3.25

========================

Trước khi sửa đổi:

Mã sinh viên: null

==> Gpa không hợp lệ.

==> Họ và tên không hợp lệ.

Sau khi sửa đổi:

Mã sinh viên: SV1005

Họ và tên: null

Gpa: 0.0

**Bài 6.4. Kiểu enum**

**Nội dung bài học**

1. [Mục đích sử dụng](https://braniumacademy.net/courses/java/lessons/java-enum/#s1)
2. [Cú pháp tổng quát](https://braniumacademy.net/courses/java/lessons/java-enum/#s2)
3. [Các thao tác với enum](https://braniumacademy.net/courses/java/lessons/java-enum/#s3)
4. [Lấy hằng số enum khi biết giá trị](https://braniumacademy.net/courses/java/lessons/java-enum/#s6)
5. [Ví dụ minh họa](https://braniumacademy.net/courses/java/lessons/java-enum/#s4)
6. [Bài tập thực hành](https://braniumacademy.net/courses/java/lessons/java-enum/#s5)

**Mục đích sử dụng**

* Đôi khi một biến chỉ nên chứa những giá trị đã được giới hạn trong một tập xác định nào đó. Enum giúp ta tạo ra tập hợp của các giá trị như vậy. Cụ thể hơn, enum giúp ta tạo ra tập hợp của các giá trị là hằng số.
* Enum là một lớp đặc biệt dùng để tập hợp và vận hành dựa trên các thành phần hằng số.
* Ví dụ 1: một tuần chỉ có 7 ngày từ thứ 2 đến chủ nhật. Điều này là không đổi, ta có thể dùng enum để tạo ta tập hợp của ngày trong tuần cho thuận tiện sử dụng.
* Ví dụ 2: kích thước quần áo có các size siêu nhỏ, nhỏ, vừa, lớn và siêu lớn. Ta hoàn toàn có thể dùng enum để lưu trữ các hằng số trên để tiện sử dụng.
* Ví dụ 3: một năm có 12 tháng từ January tới December, ta có thể dùng enum để biểu diễn các tháng trong năm.
* Ví dụ 4: học lực có 5 mức gồm không đạt, trung bình, khá, giỏi, xuất sắc. Ta có thể sử dụng enum biểu diễn các mức học lực này.
* Một trong những ưu điểm của enum so với việc sử dụng các giá trị thông thường là tránh được việc bị gán nhầm giá trị vì tên các thành phần hằng số của nó rất tường minh và trong mỗi tập hợp thì mỗi hằng số chỉ xuất hiện duy nhất 1 lần.

**Cú pháp tổng quát**

**access enum name {**

**// các hằng số**

**// thuộc tính**

**// phương thức khởi tạo**

**// phương thức khác**

**}**

Trong đó:

* Phần access là access modifier – cấp độ truy cập. Có thể là public, protected, private hoặc bỏ trống. Ý nghĩa sử dụng của access modifier như áp dụng với lớp đã biết.
* Mặc định enum là final nên ta không thêm keyword này vào. Ngoài ra không sử dụng từ khóa abstract trong khai báo enum.
* Enum bắt buộc phải có từ khóa enum trong khai báo.
* Phần name là tên của enum. Do tính chất tập hợp của enum nên ta đặt tên enum sao cho biểu đạt được ý nghĩa của tập hợp ở số ít. Ví dụ:
  + Size với tập hợp kích thước quần áo.
  + Season với tập hợp mùa của 1 năm.
  + Rank với học lực, các mức giải thưởng chẳng hạn.
* Phần kế tiếp là danh sách các hằng số. Hằng số sẽ đặt tên theo quy ước hằng số đã biết. Chúng nên liên quan đến ý nghĩa chung của tập hợp mà enum đang mô tả.
* Các hằng số phân biệt nhau bằng dấu phẩy. Các hằng số thành phần trong enum không được phép trùng nhau nhưng có thể trùng giá trị.
* Các thuộc tính đặt theo quy ước tính đóng gói dữ liệu. Thường dùng các thuộc tính mô tả giá trị hoặc biểu hiện của các hằng số.
* Tiếp theo là phần phương thức khởi tạo. Phương thức khởi tạo mặc định của enum là private. Nếu ta khai báo phương thức khởi tạo mà không xác định access modifier cho nó thì nó sẽ có access modifier là private.
* Phần phương thức có thể chứa các phương thức cung cấp các hành động nào đó liên quan đến tập hợp đang mô tả. Ví dụ trả về giá trị tương ứng từng hằng số trong enum.
* Ví dụ:

package net.braniumacademy.lesson.chapter6;

import java.util.Scanner;

public class Lesson64 {

public static void main(String[] args) {

Season season;

var scanner = new Scanner(System.in);

System.out.println("Tên mùa: ");

var seasonName = scanner.nextLine();

season = Season.valueOf(seasonName);

System.out.println(STR."Hằng số enum tương ứng: \{season}");

System.out.println(STR."Tên mùa trong tiếng Việt: \{season.getSeasonName()}");

}

}

enum Season {

SPRING("MÙA XUÂN"),

SUMMER("MÙA HẠ"),

FALL("MÙA THU"),

WINTER("MÙA ĐÔNG");

private final String seasonName;

Season(String name) {

seasonName = name;

}

public String getSeasonName() {

return seasonName;

}

}

**Các thao tác với enum**

* Có thể dùng phép so sánh tương == đương để so sánh giá trị của các hằng số trong enum.
* Dùng phương thức values() để lấy ra mảng các hằng số theo thứ tự khai báo trong enum hiện thời.
* Dùng valueOf(name) để lấy ra hằng số trong enum với tên tương ứng.
* Dùng tên enum.tên\_hằng để truy cập đến hằng số nào đó trong enum. Ví dụ: Season.SPRING.

**Lấy hằng số enum**

* Khi biết giá trị của một hằng số thành phần trong enum ta hoàn toàn có thể lấy được hằng số enum tương ứng bằng cách sử dụng cú pháp

enum\_class\_name.valueOf(constant\_name)

* Trong đó:
  + enum\_class\_name là tên lớp enum.
  + constant\_name là tên hằng số enum thành phần. Giá trị tên phải chính xác và phân biệt chữ hoa, chữ thường.
* Ví dụ:

package net.braniumacademy.lesson.chapter6;

import java.util.Scanner;

public class Lesson64 {

public enum Season {

// các hằng số

SPRING("MÙA XUÂN"),

SUMMER("MÙA HẠ"),

FALL("MÙA THU"),

WINTER("MÙA ĐÔNG");

// phương thức khởi tạo

Season(String name) {

seasonName = name;

}

// tên mùa trong tiếng Việt

private final String seasonName;

// phương thức trả về tên mùa trong tiếng việt

public String getSeasonName() {

return seasonName;

}

}

public static void main(String[] args) {

Season season;

Scanner scanner = new Scanner(System.in);

System.out.println("Tên mùa: ");

var seasonName = scanner.nextLine();

season = Season.valueOf(seasonName);

System.out.println(STR."Hằng số enum tương ứng: \{season}");

System.out.println(STR."Tên mùa trong tiếng Việt: \{season.getSeasonName()}");

}

}

Tên mùa:

SPRING

Hằng số enum tương ứng: SPRING

Tên mùa trong tiếng Việt: MÙA XUÂN

**Ví dụ minh họa**

* Ví dụ 1: minh họa sử dụng enum mô tả mùa trong năm.

public class ExampleEnum1 {

// enum mô tả tập các mùa trong năm

enum Season {

// các hằng số

SPRING("MÙA XUÂN"), SUMMER("MÙA HẠ"),

FALL("MÙA THU"), WINTER("MÙA ĐÔNG");

// phương thức khởi tạo

Season(String name) {

seasonName = name;

}

// tên mùa trong tiếng Việt

private String seasonName;

// phương thức trả về tên mùa trong tiếng việt

public String getSeasonName() {

return seasonName;

}

}

public static void main(String[] args) {

Season winter = Season.WINTER;

System.out.println("Tên mùa trong tiếng Anh: " + winter);

System.out.println("Tên mùa trong tiếng Việt: " + winter.getSeasonName());

System.out.println(Season.valueOf("WINTER"));

}

}

* Kết quả:

Tên mùa trong tiếng Anh: WINTER

Tên mùa trong tiếng Việt: MÙA ĐÔNG

WINTER

* Ví dụ 2: dùng enum mô tả kích thước quần áo.

public class ExampleEnum2 {

// enum mô tả size quần áo

public enum Size{

EXTRA\_SMALL, SMALL, MEDIUM, LARGE, EXTRA\_LARGE

}

public static void main(String[] args) {

// minh họa sử dụng

Size mySize = Size.EXTRA\_LARGE;

Size myBrotherSize = Size.LARGE;

if(myBrotherSize == mySize) {

System.out.println("We can share our clothes with each other.");

} else {

System.out.println("We cannot share our clothes with each other.");

}

}

}

* Kết quả:

We cannot share our clothes with each other.

# Bài 6.6. Các thành phần static của lớp

### Nội dung bài học

1. [Các trường static](https://braniumacademy.net/courses/java/lessons/java-static-components/#s1)
2. [Các hằng số static](https://braniumacademy.net/courses/java/lessons/java-static-components/#s2)

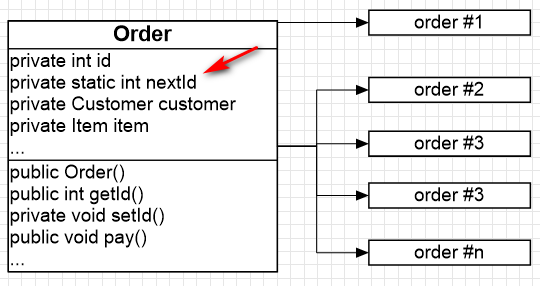
[Các phương thức static](https://braniumacademy.net/courses/java/lessons/java-static-components/#s3)

1. [Phương thức main](https://braniumacademy.net/courses/java/lessons/java-static-components/#s4)
2. [Bài tập thực hành](https://braniumacademy.net/courses/java/lessons/java-static-components/#s5)

Kết luận đầu tiên khi nói đến thành phần static của lớp là: mọi thành phần static là thành phần thuộc sở hữu lớp. Và chúng vận hành không phụ thuộc vào đối tượng nào của lớp. Tức là có thể dùng chung, chia sẻ chung cho mọi đối tượng của cùng lớp hoặc lớp khác tùy vào access modifier.

### Các trường static

* Các trường static là các thuộc tính của lớp nhưng được khai báo với keyword static.
* Các trường static là thành phần dùng chung cho toàn bộ các đối tượng của lớp. Nếu access modifier là public thì trường static này có thể dùng chung cho cả các đối tượng của lớp khác.
* Khi ta tạo ra một đối tượng từ lớp với keyword new, đối tượng đó sẽ có một bản copy riêng các thuộc tính không phải static. Nhưng lại dùng chung các thành phần static với lớp tạo ra nó. Đó là lý do tại sao khi ta thay đổi giá trị thuộc tính riêng của đối tượng A thì không ảnh hưởng gì đến thuộc tính này của đối tượng B.
* Ví dụ áp dụng: giả sử ta cần xây dựng hệ thống thương mại điện tử. Ở trong đó có lớp đơn hàng. Mã đơn hàng của ta là một số nguyên tự động tăng mỗi khi có đơn hàng mới được đặt. Lúc này ta sẽ thiết kế ra một trường nextId static dùng chung cho các đối tượng đơn hàng của lớp đơn hàng(Order).

[](https://braniumacademy.net/wp-content/uploads/2020/09/vi-du-ve-thanh-phan-static-1.png)Lớp(trái) và đối tượng(phải)

* Cụ thể:

// lớp mô tả thông tin đơn hàng

public class Order {

private int id;

private static int nextId = 100; // bắt đầu mã đơn hàng từ 100

private Customer customer;

private Item item;

private int quantity;

private float price;

//...

public Order() {

//...

setId();

}

public int getId() {

return id;

}

// việc thiết lập mã đơn hàng không cần thiết

// phải public ra ngoài nên ta để là private

private void setId() {

this.id = nextId;

nextId++;

}

}

// lớp mô tả thông tin mặt hàng

class Item{

private int id;

private String name;

//...

}

// lớp mô tả thông tin của khách hàng

class Customer{

private String id;

private String name;

//...

}

* Mỗi khi khách hàng ấn nút mua hàng thì đơn hàng sẽ được tạo ở đâu đó trong hệ thống. Phần code sau minh họa việc tạo đơn hàng:

public static void main(String[] args) {

Order order1 = new Order(); // tạo đơn hàng thứ nhất

Order order2 = new Order(); // tạo đơn hàng thứ hai

// .. tạo đơn hàng thứ n...

System.out.println("order1 id = " + order1.getId());

System.out.println("order2 id = " + order2.getId());

}

* Kết quả hiển thị ra màn hình là:

order1 id = 100

order2 id = 101

### Các hằng số static

* Các trường static tương đối hiếm được sử dụng nhưng các hằng số static thì lại thường xuyên được sử dụng. Điển hình nhất là các hằng số PI, E của lớp Math. Chúng ta có thể sử dụng chúng bất kì lúc nào nếu muốn và khi sử dụng ta không cần phải có đối tượng, chỉ cần tên lớp Math.PI hay Math.E là xong.
* Nhắc lại rằng hằng số static trên không vận hành bởi đối tượng của lớp Math. Ta chỉ cần cung cấp tên lớp chứa nó và tên hằng là sử dụng được. Tương tự cho các hằng static bạn tự tạo ra.
* Ví dụ khác là System.out. Đây là một hằng số ta dùng rất nhiều từ đầu khóa học đến giờ. Bản chất nó chính là một hằng số:

public static final PrintStream out = null;

* Do hằng số là thành phần sử dụng chung nên ta có thể để public nếu muốn cho các lớp khác cùng sử dụng

### Các phương thức static

* Phương thức static là phương thức cũng không vận hành dựa trên đối tượng.
* Nếu sử dụng trong nội bộ lớp, ta chỉ cần tên lớp và danh sách đối số tương ứng.
* Nếu sử dụng bên ngoài lớp, ta cung cấp thêm tên lớp ở đầu làm tham chiếu.
* Các thành phần static nói chung không sử dụng được với từ khóa this vì static không thuộc sở hữu của đối tượng nào cả.
* Phương thức static chỉ dùng để thao tác với các thành phần static. Không dùng để thao tác với các thuộc tính hay các phương thức non-static.
* Ví dụ:

public class StaticExample {

public static void main(String[] args) {

doSomeStatic(); // ok

doSomething(); // error

}

// phương thức vận hành không cần object

public static void doSomeStatic() {

System.out.println("Bên trong doSomeStatic()");

}

// phương thức vận hành trên object

public void doSomething(){

System.out.println("Bên trong doSomething()");

}

}

* Ta sử dụng phương thức static trong hai trường hợp:
  + Khi phương thức không cần truy cập đến thuộc tính của đối tượng vì tất cả những gì cần thiết đã được cung cấp qua các đối số của lời gọi phương thức
  + Khi 1 phương thức chỉ cần truy cập đến các trường static của lớp

### Phương thức main

* Để ý rằng phương thức main là một static method. Ta dùng phương thức main để khởi chạy chương trình.
* Phương thức main không vận hành dựa vào đối tượng. Thực tế rằng khi chương trình bắt đầu chạy, không có bất kì đối tượng nào cả. Lúc này chương trình vào main, khởi chạy và các đối tượng cần thiết lần lượt được tạo ra.
* Mỗi một lớp đều có thể có cho riêng mình một phương thức main. Điều này rất hữu ích cho việc thực hiện unit testing.
* Ví dụ hình ảnh của phương thức main:

public static void main(String[] args) {

Order order1 = new Order(); // tạo đơn hàng thứ nhất

Order order2 = new Order(); // tạo đơn hàng thứ hai

// .. tạo đơn hàng thứ n...

System.out.println("order1 id = " + order1.getId());

System.out.println("order2 id = " + order2.getId());

}

# Bài 6.7. Lớp ArrayList

### Nội dung bài học

1. [Bản chất ArrayList](https://braniumacademy.net/courses/java/lessons/java-arraylist/#s1)
2. [Tạo mới ArrayList](https://braniumacademy.net/courses/java/lessons/java-arraylist/#s2)
3. [Thêm phần tử vào ArrayList](https://braniumacademy.net/courses/java/lessons/java-arraylist/#s3)
4. [Thay đổi giá trị của phần tử trong ArrayList](https://braniumacademy.net/courses/java/lessons/java-arraylist/#s4)
5. [Loại bỏ phần tử khỏi ArrayList](https://braniumacademy.net/courses/java/lessons/java-arraylist/#s5)
6. [Duyệt ArrayList](https://braniumacademy.net/courses/java/lessons/java-arraylist/#s6)
7. [Một số phương thức khác](https://braniumacademy.net/courses/java/lessons/java-arraylist/#s7)
8. [Bài tập thực hành](https://braniumacademy.net/courses/java/lessons/java-arraylist/#s8)

### Bản chất ArrayList

* Khi ta muốn lưu trữ một tập hợp các phần tử cùng kiểu, ta sử dụng mảng. Tuy nhiên nhược điểm của mảng là số phần tử của nó cố định. Do đó không thể mở rộng thêm hoặc giảm kích thước của nó đi kể từ sau khi mảng được cấp phát hoặc khởi tạo.
* Để kế thừa các tính chất của mảng và mang đến những tính năng mềm dẻo và ưu việt hơn, ta dùng ArrayList.
* ArrayList là một phần của collection framework trong Java. ArrayList nằm trong gói java.util. ArrayList cung cấp cho ta khả năng lưu trữ mảng động trong Java.
* ArrayList có thể chậm hơn mảng thông thường nhưng nó có các tính năng hữu dụng giúp ta quản lý các phần tử dễ dàng.
* ArrayList dùng để lưu trữ các đối tượng cùng kiểu, thay cho mảng.
* Kích thước của ArrayList tự động tăng lên khi ta thêm các phần tử mới vào. Và tự động giảm đi khi ta loại bỏ các phần tử khỏi ArrayList.
* ArrayList chỉ dùng để lưu trữ các đối tượng cùng kiểu của các kiểu tham chiếu. Không dùng để lưu trữ các giá trị kiểu nguyên thủy. Nếu muốn lưu giá trị kiểu nguyên thủy ta sẽ sử dụng lớp bao của chúng.
* Ta có thể sử dụng chỉ số mảng để truy cập các phần tử trong ArrayList. Tư tưởng sử dụng chỉ số phần tử tương tự như trong mảng.

### Tạo mới ArrayList

* Để tạo mới một ArrayList, ta có thể dùng 1 trong 3 phương thức khởi tạo sau:
  + **ArrayList():** Tạo một đối tượng ArrayList rỗng với khả năng chứa 10 phần tử.
  + **ArrayList(Collection c):** Tạo một đối tượng ArrayList với khởi đầu là một tập các phần tử trong một collection c nào đó. Nếu tham số là null ta sẽ nhận NullPointerException.
  + **ArrayList(int capacity):** Tạo một đối tượng ArrayList rỗng với khả năng lưu trữ khởi đầu bằng số phần tử trong tham số.
* Ví dụ:

ArrayList<Order> orders = new ArrayList<>();

ArrayList<Order> orders1 = new ArrayList<>(40);

ArrayList<Order> orders2 = new ArrayList<>(orders);

// hoặc sử dụng kiểu tự suy luận var:

var pets = new ArrayList<Pet>();

### Thêm phần tử vào ArrayList

* Để thêm phần tử vào ArrayList ta dùng phương thức add:
  + **add(object):** Thêm mới phần tử object cùng kiểu với kiểu của ArrayList vào cuối danh sách.
  + **add(index, object):**Thêm mới một đối tượng object cùng kiểu với kiểu của ArrayList vào vị trí chỉ số index trong danh sách. Các phần tử phía sau chỉ số đó sẽ tự động được lùi về sau 1 vị trí. Lưu ý rằng index phải nằm trong đoạn [0, size-1] với size là kích thước hay số phần tử thực tế hiện thời của ArrayList.
  + **addAll(Collection c):** Thêm toàn bộ các phần tử của collection c vào cuối danh sách hiện thời.
  + **addAll(index, Collection c):** Chèn toàn bộ các phần tử của collection c vào vị trí index của danh sách hiện thời. Lưu ý index phải hợp lệ, tức thuộc khoảng [0, size-1].
* Ví dụ:

package net.braniumacademy.lesson.chapter6;

public class Pet {

private String name; // tên

private float age; // tuổi

private float weight; // cân nặng

public Pet() {

name = "No Name";

age = 0.0f;

weight = 0.0f;

}

public Pet(String name, int age, float weight) {

setName(name);

setAge(age);

setWeight(weight);

}

public String getName() {

return name.toUpperCase();

}

public void setName(String name) {

this.name = name;

}

public float getAge() {

return age;

}

public void setAge(float age) {

this.age = age;

}

public float getWeight() {

return weight;

}

public void setWeight(float weight) {

this.weight = weight;

}

}

package net.braniumacademy.lesson.chapter6;

import java.util.ArrayList;

public class Lesson67 {

public static void main(String[] args) {

var pets = new ArrayList<Pet>();

var pet1 = new Pet("Tom Cat", 3, 3.5f);

var pet2 = new Pet("Jerry Mouse", 3, 0.5f);

var pet3 = new Pet("Bob Dog", 5, 15.5f);

// thêm vào cuối array list

pets.add(pet1);

pets.add(pet2);

// thêm vào vị trí được chỉ định

pets.add(0, pet3);

pets.add(1, new Pet("Terrot Bird", 2, 0.3f));

// [0, pets.size() - 1]

var otherPets = new ArrayList<Pet>();

// thêm cả collection vào cuối array list

otherPets.addAll(pets);

}

}

* Khi ta tạo mới một đối tượng với từ khóa new. Vùng nhớ được cấp phát cho đối tượng sẽ nằm trong bộ nhớ heap.

Quá trình mở rộng ArrayList khi đã đầy và nếu chúng ta tiếp tục thêm phần tử mới vào có thể được mô tả như sau:

* Tạo một ArrayList tại một vùng nhớ lớn hơn trong bộ nhớ heap(ví dụ gấp 1.5 lần kích thước cũ).
* Copy các phần tử hiện thời trong danh sách cũ sang phân vùng nhớ mới này.
* Thêm phần tử mới vào danh sách mới tạo.
* Xóa bỏ vùng nhớ cũ.

### Thay đổi giá trị của phần tử trong ArrayList

* Để thay đổi một phần tử nào đó trong ArrayList khi phần tử đó đã tồn tại, ta sử dụng phương thức set():
  + **set(index, newObject):** Thay thế phần tử tại vị trí index trong ArrayList hiện thời bằng phần tử newObject. Trong đó newObject phải đảm bảo cùng kiểu với kiểu của ArrayList. Lưu ý rằng index phải nằm trong đoạn [0, size-1] với size là kích thước hay số phần tử thực tế hiện thời của ArrayList.
* Ví dụ:

package net.braniumacademy.lesson.chapter6;

import java.util.ArrayList;

public class Lesson67 {

public static void main(String[] args) {

var pets = new ArrayList<Pet>();

var pet1 = new Pet("Tom Cat", 3, 3.5f);

var pet2 = new Pet("Jerry Mouse", 3, 0.5f);

var pet3 = new Pet("Bob Dog", 5, 15.5f);

// thêm vào cuối array list

pets.add(pet1);

pets.add(pet2);

// thêm vào vị trí được chỉ định

pets.add(pet3);

pets.add(1, new Pet("Terrot Bird", 2, 0.3f));

// show arraylist elements:

System.out.println("Trước khi thay đổi:");

for (int i = 0; i < pets.size(); i++) {

Pet pet = pets.get(i);

pet.showInfo();

}

// thay thế object tại vị trí index = 1

pets.set(1, new Pet("Donald Duck", 5, 2.5f));

System.out.println("Sau khi thay thế pet tại vị trí index = 1: ");

for (int i = 0; i < pets.size(); i++) {

Pet pet = pets.get(i);

pet.showInfo();

}

}

}

Trước khi thay đổi:

Tên pet: Tom Cat

Tuổi pet: 3.0

Cân nặng pet: 3.5kg

==========================

Tên pet: Terrot Bird

Tuổi pet: 2.0

Cân nặng pet: 0.3kg

==========================

Tên pet: Jerry Mouse

Tuổi pet: 3.0

Cân nặng pet: 0.5kg

==========================

Tên pet: Bob Dog

Tuổi pet: 5.0

Cân nặng pet: 15.5kg

==========================

Sau khi thay thế pet tại vị trí index = 1:

Tên pet: Tom Cat

Tuổi pet: 3.0

Cân nặng pet: 3.5kg

==========================

Tên pet: Donald Duck

Tuổi pet: 5.0

Cân nặng pet: 2.5kg

==========================

Tên pet: Jerry Mouse

Tuổi pet: 3.0

Cân nặng pet: 0.5kg

==========================

Tên pet: Bob Dog

Tuổi pet: 5.0

Cân nặng pet: 15.5kg

==========================

### Loại bỏ phần tử khỏi ArrayList

* Để loại bỏ phần tử nào đó khỏi ArrayList ta sử dụng các phương thức sau:
  + **remove(object):** Xóa phần tử object khỏi danh sách. Nếu phần tử này xuất hiện nhiều lần trong danh sách thì xóa phần tử đầu tiên tìm được.
  + **remove(index):** Xóa bỏ phần tử tại chỉ số index khỏi danh sách hiện thời. Lưu ý index phải nằm trong đoạn [0, size-1] nếu không bạn sẽ gặp ngoại lệ.
  + **clear():** Xóa bỏ toàn bộ các phần tử hiện có trong ArrayList.
  + **removeAll(Collection c):** Xóa bỏ toàn bộ các phần tử chứa trong collection cho trong tham số.
  + **removeRange(fromIndex, toIndex)(Đã lỗi thời):** Xóa bỏ các phần tử trong danh sách mà chỉ số của các phần tử này nằm trong khoảng [fromIndex, toIndex). Lưu ý các index phải hợp lệ.
  + **removeFirst()**: xóa bỏ phần tử đầu array list.
  + **removeLast()**: xóa bỏ phần tử cuối array list.
* Ví dụ:

package net.braniumacademy.lesson.chapter6;

import java.util.ArrayList;

public class Lesson67 {

public static void main(String[] args) {

var pets = new ArrayList<Pet>();

var others = new ArrayList<Pet>();

var pet1 = new Pet("Tom Cat", 3, 3.5f);

var pet2 = new Pet("Jerry Mouse", 3, 0.5f);

var pet3 = new Pet("Bob Dog", 5, 15.5f);

// thêm vào cuối array list

pets.add(pet1);

pets.add(pet2);

pets.add(pet3);

others.addAll(pets);

pets.add(1, new Pet("Terrot Bird", 2, 0.3f));

// show arraylist elements:

System.out.println("Trước khi xóa:");

for (int i = 0; i < pets.size(); i++) {

Pet pet = pets.get(i);

pet.showInfo();

}

// xoá phần tử trong pets mà cũng xuất hiện trong others

pets.removeAll(others);

System.out.println("Sau khi xóa phần tử ở vị trí index = 1: ");

for (int i = 0; i < pets.size(); i++) {

Pet pet = pets.get(i);

pet.showInfo();

}

}

}

Trước khi xóa:

Tên pet: Tom Cat

Tuổi pet: 3.0

Cân nặng pet: 3.5kg

==========================

Tên pet: Terrot Bird

Tuổi pet: 2.0

Cân nặng pet: 0.3kg

==========================

Tên pet: Jerry Mouse

Tuổi pet: 3.0

Cân nặng pet: 0.5kg

==========================

Tên pet: Bob Dog

Tuổi pet: 5.0

Cân nặng pet: 15.5kg

==========================

Sau khi xóa phần tử chung:

Tên pet: Terrot Bird

Tuổi pet: 2.0

Cân nặng pet: 0.3kg

==========================

### Duyệt ArrayList

* Để duyệt ArrayList, ta có thể sử dụng vòng lặp for thường hoặc vòng lặp for-each:

package net.braniumacademy.lesson.chapter6;

import java.util.ArrayList;

public class Lesson67 {

public static void main(String[] args) {

var pets = new ArrayList<Pet>();

var pet1 = new Pet("Tom Cat", 3, 3.5f);

var pet2 = new Pet("Jerry Mouse", 3, 0.5f);

var pet3 = new Pet("Bob Dog", 5, 15.5f);

// thêm vào cuối array list

pets.add(pet1);

pets.add(pet2);

// thêm vào vị trí được chỉ định

pets.add(0, pet3);

pets.add(1, new Pet("Terrot Bird", 2, 0.3f));

// show arraylist elements:

// cách 1: dùng for thường

System.out.println("Cách 1: dùng for thường:");

for (int i = 0; i < pets.size(); i++) {

Pet pet = pets.get(i);

pet.showInfo();

}

// dùng for-each

System.out.println("Cách 2: dùng for-each:");

for (var pet : pets) {

pet.setName(STR."\{pet.getName()}Haha");

}

for(var pet : pets) {

pet.showInfo();

}

}

}

Cách 1: dùng for thường:

Tên pet: Bob Dog

Tuổi pet: 5.0

Cân nặng pet: 15.5kg

==========================

Tên pet: Terrot Bird

Tuổi pet: 2.0

Cân nặng pet: 0.3kg

==========================

Tên pet: Tom Cat

Tuổi pet: 3.0

Cân nặng pet: 3.5kg

==========================

Tên pet: Jerry Mouse

Tuổi pet: 3.0

Cân nặng pet: 0.5kg

==========================

Cách 2: dùng for-each:

Tên pet: BOB DOGHaha

Tuổi pet: 5.0

Cân nặng pet: 15.5kg

==========================

Tên pet: TERROT BIRDHaha

Tuổi pet: 2.0

Cân nặng pet: 0.3kg

==========================

Tên pet: TOM CATHaha

Tuổi pet: 3.0

Cân nặng pet: 3.5kg

==========================

Tên pet: JERRY MOUSEHaha

Tuổi pet: 3.0

Cân nặng pet: 0.5kg

==========================

### Một số phương thức khác

* Một số phương thức khác của lớp ArrayList:
  + **size():** Trả về số lượng phần tử hiện thời trong danh sách. Tương đương với length của mảng.
  + **indexOf(object):** Trả về chỉ số của lần xuất hiện đầu tiên của phần tử object trong danh sách. Nếu không tồn tại object này thì trả về -1.
  + **lastIndexOf(object):** Trả về chỉ số của lần xuất hiện cuối cùng của phần tử object trong danh sách. Nếu không tồn tại object này thì trả về -1.
  + **isEmpty():** Trả về true nếu danh sách rỗng và false trong trường hợp ngược lại.
  + **clone():** Trả về một bản sao của danh sách hiện thời.
  + **contains(object)**: kiểm tra xem một đối tượng có tồn tại trong array list không. Trả về true nếu tồn tại và false nếu ngược lại.
  + **ensureCapacity(minCapacity):** Tăng khả năng chứa của danh sách hiện thời lên tối thiểu minCapacity phần tử. Sử dụng khi muốn chắc chắn rằng danh sách hiện thời có thể chứa ít nhất minCapacity phần tử.
  + **subList(fromIndex, toIndex):** Trả về một danh sách các phần tử của ArrayList hiện thời có chỉ số trong khoảng [fromIndex, toIndex).
  + **toArray():** Trả về một mảng các phần tử trong danh sách theo thứ tự xuất hiện từ đầu đến cuối.
  + **toArray(T[] arr):** Trả về một mảng các phần tử trong danh sách ở kiểu T[] theo thứ tự xuất hiện từ đầu đến cuối.
* Ví dụ:

package net.braniumacademy.lesson.chapter6;

import java.util.ArrayList;

public class Lesson67 {

public static void main(String[] args) {

var pets = new ArrayList<Pet>();

var pet1 = new Pet("Tom Cat", 3, 3.5f);

var pet2 = new Pet("Jerry Mouse", 3, 0.5f);

var pet3 = new Pet("Bob Dog", 5, 15.5f);

// thêm vào cuối array list

pets.add(pet1);

pets.add(pet2);

pets.add(pet3);

System.out.println(STR."Vị trí đầu tiên xuất hiện pet1: \{pets.indexOf(pet1)}");

System.out.println(STR."Vị trí đầu tiên xuất hiện pet1: \{pets.lastIndexOf(pet1)}");

System.out.println(STR."Số lượng phần tử hiện có trong pets: \{pets.size()}");

System.out.println(STR."pets rỗng?: \{pets.isEmpty()}");

System.out.println(STR."pet2 có tồn tại trong pets?: \{pets.contains(pet2)}");

}

}

Vị trí đầu tiên xuất hiện pet1: 0

Vị trí đầu tiên xuất hiện pet1: 0

Số lượng phần tử hiện có trong pets: 3

pets rỗng?: false

pet2 có tồn tại trong pets?: true

# Bài 6.8. Inner class và nested class

### Nội dung bài học

1. [Nested class](https://braniumacademy.net/courses/java/lessons/java-inner-class/#s1)
2. [Static nested class](https://braniumacademy.net/courses/java/lessons/java-inner-class/#s2)
3. [Inner class](https://braniumacademy.net/courses/java/lessons/java-inner-class/#s3)
4. [Local class](https://braniumacademy.net/courses/java/lessons/java-inner-class/#s4)
5. [Anonymous class](https://braniumacademy.net/courses/java/lessons/java-inner-class/#s5)

### Nested class

* Ngôn ngữ lập trình Java cho phép bạn định nghĩa một lớp bên trong một lớp khác. Những lớp như vậy được gọi là nested class(lớp lồng trong lớp).
* Lớp chứa một hoặc nhiều lớp khác bên trong nó thì ta gọi là outer class hay lớp ngoài.
* Hình ảnh của nested class:

**public class OuterClass {**

**//...**

**class NestedClass {**

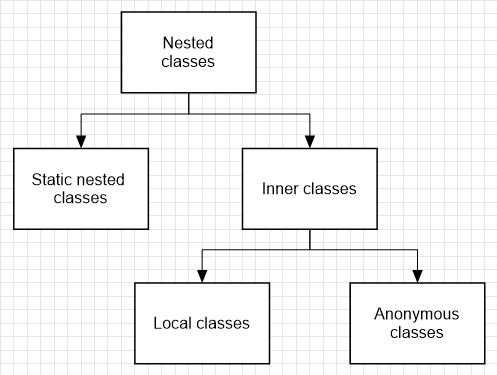
**//...**

**}**

**//...**

**}**

* Trong đó NestedClass lại chia làm 2 loại: static và non-static như sau.

[](https://braniumacademy.net/wp-content/uploads/2020/08/nested-classes.png)

Trong đó:

* Static nested classes là các lớp nested có chứa keyword static trong khai báo lớp.
* Các nested classes không chứa keyword static trong khai báo gọi là các non-static nested class hay các inner class.
* Một nested class là thành phần của lớp chứa nó. Đây là thành phần thứ tư trong cú pháp định nghĩa lớp tổng quát ta đã học.
* Các lớp inner có thể truy cập vào tất cả các thành phần của lớp bên ngoài chứa nó. Kể cả thành phần private.
* Các lớp static nested không truy cập đến các thành phần khác của lớp chứa nó(OuterClass).
* Các nested class có thể khai báo với keyword chỉ định mức độ truy cập public, protected, private hoặc để trống.
* Với OuterClass thì access modifier chỉ có thể là public hoặc để trống(private package access).
* Ví dụ về inner class và outer class:

package net.braniumacademy.chapter6.lesson8;

import java.util.ArrayList;

/\*\*

\* Lớp test các đối tượng Student

\*/

public class Lesson68 {

public static void main(String[] args) {

ArrayList<Student> students = new ArrayList<>();

createFakeData(students);

// Trước khi sắp xếp

System.out.println("==> Trước khi sắp xếp: ");

showStudents(students);

// sau khi sắp xếp

sort(students);

System.out.println("==> Sau khi sắp xếp: ");

showStudents(students);

}

// phương thức tạo dữ liệu giả định về các sinh viên

private static void createFakeData(ArrayList<Student> students) {

students.add(new Student("ST1001", "Nguyễn Ngọc Mơ", "CNTT", 3.56f));

students.add(new Student("ST1002", "Nguyễn Ngọc Mai", "CNTT", 3.40f));

students.add(new Student("ST1003", "Nguyễn Ngọc My", "CNTT", 3.69f));

students.add(new Student("ST1004", "Nguyễn Ngọc Khánh", "CNTT", 3.34f));

students.add(new Student("ST1005", "Nguyễn Ngọc Mun", "CNTT", 3.15f));

students.add(new Student("ST1006", "Lý Ngọc Mơ", "CNTT", 3.56f));

students.add(new Student("ST1007", "Mai Ngọc Mơ", "CNTT", 3.56f));

}

// sắp xếp theo điểm gpa giảm dần, tên tăng dần, họ tăng dần

private static void sort(ArrayList<Student> students) {

for (int i = 0; i < students.size(); i++) {

for (int j = students.size() - 1; j > i; j--) {

if (shouldSwap(students.get(j - 1), students.get(j))) {

var student = students.get(j);

students.set(j, students.get(j - 1));

students.set(j - 1, student);

}

}

}

}

// xét điểm gpa, tên, họ xem có cần tráo đổi hai sinh viên trong danh sách không

private static boolean shouldSwap(Student student, Student other) {

if (student.getGpa() < other.getGpa()) { // nếu điểm sv đứng trước nhỏ hơn -> tráo đổi

return true;

} else if (student.getGpa() == other.getGpa() // điểm bằng nhau -> xét tên

&& student.getFirstName().compareTo(other.getFirstName()) > 0) {

return true; // tên đứng trước lớn hơn(theo bảng chữ cái) -> tráo đổi vị trí

} else return student.getGpa() == other.getGpa() // tên và điểm trùng nhau -> xét họ

&& student.getFirstName().compareTo(other.getFirstName()) == 0

&& student.getLastName().compareTo(other.getLastName()) > 0;

}

// phương thức hiển thị danh sách sinh viên lên màn hình dạng bảng gồm các hàng, cột

private static void showStudents(ArrayList<Student> students) {

System.out.println("========================== DANH SÁCH SINH VIÊN ==========================");

System.out.printf("%-10s%-10s%-15s%-10s%-20s%-10s\n",

"Mã SV", "Họ", "Đệm", "Tên", "Chuyên ngành", "Điểm gpa");

for (var student : students) {

System.out.printf("%-10s%-10s%-15s%-10s%-20s%-10.2f\n",

student.getStudentId(), student.getLastName(), student.getMidName(),

student.getFirstName(), student.getMajor(), student.getGpa());

}

}

}

/\*\*

\* Lớp mô tả thông tin sinh viên

\*/

class Student {

private String studentId; // mã sinh viên

private final FullName fullName; // họ và tên

private String major; // chuyên ngành

private float gpa; // điểm gpa

public Student() {

fullName = new FullName("", "", "");

}

public Student(String studentId, String fullName, String major, float gpa) {

this();

this.setStudentId(studentId);

this.setFullName(fullName);

this.setMajor(major);

this.setGpa(gpa);

}

public String getStudentId() {

return studentId;

}

public void setStudentId(String studentId) {

this.studentId = studentId;

}

public String getFullName() {

return STR."\{fullName.lastName} \{fullName.midName} \{fullName.firstName}";

}

public String getLastName() {

return fullName.lastName;

}

public String getFirstName() {

return fullName.firstName;

}

public String getMidName() {

return fullName.midName;

}

public void setFullName(String fullName) {

var words = fullName.split("\\s+");

if (words.length == 1) {

this.fullName.firstName = words[0];

} else if (words.length == 2) {

this.fullName.firstName = words[1];

this.fullName.lastName = words[0];

} else if (words.length > 2) {

var midNameBuilder = new StringBuilder();

for (int i = 1; i < words.length - 1; i++) {

midNameBuilder.append(words[i]).append(" ");

}

this.fullName.firstName = words[words.length - 1];

this.fullName.midName = midNameBuilder.toString().trim();

this.fullName.lastName = words[0];

}

}

public String getMajor() {

return major;

}

public void setMajor(String major) {

this.major = major;

}

public float getGpa() {

return gpa;

}

public void setGpa(float gpa) {

this.gpa = gpa;

}

/\*\*

\* Lớp static nested class mô tả thông tin về họ và tên chi tiết.

\* Sử dụng lớp này với ngữ cảnh static vì nó không có nhu cầu truy cập

\* vào các thành phần của outer class chứa nó. Ta để lớp này private vì

\* chỉ được sử dụng nội bộ trực tiếp trong Student.

\*/

private static class FullName {

String firstName; // tên

String lastName; // họ

String midName; // đệm

FullName(String firstName, String lastName, String midName) {

this.firstName = firstName;

this.lastName = lastName;

this.midName = midName;

}

}

}

==> Trước khi sắp xếp:

========================== DANH SÁCH SINH VIÊN ==========================

Mã SV Họ Đệm Tên Chuyên ngành Điểm gpa

ST1001 Nguyễn Ngọc Mơ CNTT 3.56

ST1002 Nguyễn Ngọc Mai CNTT 3.40

ST1003 Nguyễn Ngọc My CNTT 3.69

ST1004 Nguyễn Ngọc Khánh CNTT 3.34

ST1005 Nguyễn Ngọc Mun CNTT 3.15

ST1006 Lý Ngọc Mơ CNTT 3.56

ST1007 Mai Ngọc Mơ CNTT 3.56

==> Sau khi sắp xếp:

========================== DANH SÁCH SINH VIÊN ==========================

Mã SV Họ Đệm Tên Chuyên ngành Điểm gpa

ST1003 Nguyễn Ngọc My CNTT 3.69

ST1006 Lý Ngọc Mơ CNTT 3.56

ST1007 Mai Ngọc Mơ CNTT 3.56

ST1001 Nguyễn Ngọc Mơ CNTT 3.56

ST1002 Nguyễn Ngọc Mai CNTT 3.40

ST1004 Nguyễn Ngọc Khánh CNTT 3.34

ST1005 Nguyễn Ngọc Mun CNTT 3.15

### Lý do sử dụng nested class

* Nó làm tăng tính đóng gói: Hãy xem xét hai lớp A và B, trong đó B cần quyền truy cập vào các thành phần của A. Bằng cách ẩn lớp B trong lớp A, các thành viên của A có thể được khai báo là private và B có thể truy cập chúng. Ngoài ra, bản thân B có thể bị che giấu khỏi thế giới bên ngoài.
* Đó là một cách hợp lý để gom nhóm các lớp chỉ được sử dụng ở một nơi nào đó. Nếu một lớp chỉ hữu ích cho một lớp khác, thì hợp lý là nhúng nó vào lớp đó và giữ hai lớp đó lại với nhau. Việc lồng các “lớp trợ giúp” như vậy làm cho gói của chúng được sắp xếp hợp lý hơn.
* Code dễ đọc, dễ bảo trì hơn. Việc lồng các lớp nhỏ vào bên trong các lớp có sẵn sẽ cho phép ta để các lớp này ngay gần nơi nó được sử dụng.
* Trong ví dụ trên, ta thấy rằng việc phân tách dữ liệu về họ và tên rạch ròi giúp ta nhanh chóng lấy ra được các mẩu dữ liệu cần thiết để sắp xếp theo tiêu chí gpa giảm dần, tên tăng dần, họ tăng dần. Khi ta hiển thị dữ liệu cũng dễ dàng phân tách từng thành phần họ-đệm-tên.

### Static nested class

* Giống như các phương thức và thuộc tính, static nested class được liên kết với lớp chứa nó.
* Do là một thành phần static của lớp nên static nested class không thể trực tiếp truy cập tới các thuộc tính hay phương thức không phải static của lớp chứa nó.
* Nếu muốn truy cập và sử dụng các thành phần này thì phải thông qua các đối tượng của lớp chứa nó.
* Sử dụng static nested class khi ta muốn gói các lớp có liên quan lại trong 1 lớp và lớp inner không có nhu cầu sử dụng trực tiếp các thành phần của lớp outer.
* Để truy cập đến lớp static nested ta dùng tên lớp chứa và tên lớp cần truy cập:

**OuterClass.StaticNestedClass**

* Ví dụ sau khai báo lớp static nested và sử dụng nó để tạo đối tượng:

/\*\*

\* Lớp mô tả thông tin sinh viên

\*/

class Student {

private String studentId; // mã sinh viên

private final FullName fullName; // họ và tên

private String major; // chuyên ngành

private float gpa; // điểm gpa

public Student() {

fullName = new FullName("", "", ""); // tạo đối tượng của static nested class

}

public Student(String studentId, String fullName, String major, float gpa) {

this(); // gọi đến constructor không tham số

this.setStudentId(studentId);

this.setFullName(fullName);

this.setMajor(major);

this.setGpa(gpa);

}

// ...

public String getFullName() {

return STR."\{fullName.lastName} \{fullName.midName} \{fullName.firstName}";

}

public String getLastName() {

return fullName.lastName;

}

public String getFirstName() {

return fullName.firstName;

}

public String getMidName() {

return fullName.midName;

}

public void setFullName(String fullName) {

var words = fullName.split("\\s+");

if (words.length == 1) {

this.fullName.firstName = words[0];

} else if (words.length == 2) {

this.fullName.firstName = words[1];

this.fullName.lastName = words[0];

} else if (words.length > 2) {

var midNameBuilder = new StringBuilder();

for (int i = 1; i < words.length - 1; i++) {

midNameBuilder.append(words[i]).append(" ");

}

this.fullName.firstName = words[words.length - 1];

this.fullName.midName = midNameBuilder.toString().trim();

this.fullName.lastName = words[0];

}

}

// ...

/\*\*

\* Lớp static nested class mô tả thông tin về họ và tên chi tiết.

\* Sử dụng lớp này với ngữ cảnh static vì nó không có nhu cầu truy cập

\* vào các thành phần của outer class chứa nó. Ta để lớp này private vì

\* chỉ được sử dụng nội bộ trực tiếp trong Student.

\*/

private static class FullName {

String firstName; // tên

String lastName; // họ

String midName; // đệm

FullName(String firstName, String lastName, String midName) {

this.firstName = firstName;

this.lastName = lastName;

this.midName = midName;

}

}

}

### Inner class

* Một inner class là một nested class mà không có chứa từ khóa static trong khai báo lớp.
* Là một thành phần của lớp chứa nó nên inner class có thể truy cập trực tiếp đến các thuộc tính và phương thức của lớp chứa nó.
* Trong inner class không thể định nghĩa bất kì thành phần static nào nếu không sẽ bị lỗi. Lý do là bởi vì inner class được liên kết với đối tượng của lớp bên ngoài chứa nó.
* Đối tượng của inner class tồn tại trong đối tượng của lớp chứa nó.
* Để tạo ra đối tượng của lớp inner, ta cần tạo đối tượng của lớp chứa nó. Sau đó mới tạo đối tượng của inner class.
* Cú pháp như sau:

OuterClass.InnerClass innerObject = outerObject.new InnerClass();

* Ví dụ sau sử dụng inner class thay vì static nested class:

package net.braniumacademy.chapter6.lesson8;

import java.util.ArrayList;

/\*\*

\* Lớp test các đối tượng Student

\*/

public class Lesson68 {

public static void main(String[] args) {

ArrayList<Student> students = new ArrayList<>();

createFakeData(students);

// Trước khi sắp xếp

System.out.println("==> Trước khi sắp xếp: ");

showStudents(students);

// sau khi sắp xếp

sort(students);

System.out.println("==> Sau khi sắp xếp: ");

showStudents(students);

}

// phương thức tạo dữ liệu giả định về các sinh viên

private static void createFakeData(ArrayList<Student> students) {

students.add(new Student("ST1001", "Nguyễn Ngọc Mơ", "CNTT", 3.56f));

students.add(new Student("ST1002", "Nguyễn Ngọc Mai", "CNTT", 3.40f));

students.add(new Student("ST1003", "Nguyễn Ngọc My", "CNTT", 3.69f));

students.add(new Student("ST1004", "Nguyễn Ngọc Khánh", "CNTT", 3.34f));

students.add(new Student("ST1005", "Nguyễn Ngọc Mun", "CNTT", 3.15f));

students.add(new Student("ST1006", "Lý Ngọc Mơ", "CNTT", 3.56f));

students.add(new Student("ST1007", "Mai Ngọc Mơ", "CNTT", 3.56f));

}

// sắp xếp theo điểm gpa giảm dần, tên tăng dần, họ tăng dần

private static void sort(ArrayList<Student> students) {

for (int i = 0; i < students.size(); i++) {

for (int j = students.size() - 1; j > i; j--) {

if (shouldSwap(students.get(j - 1), students.get(j))) {

var student = students.get(j);

students.set(j, students.get(j - 1));

students.set(j - 1, student);

}

}

}

}

// xét điểm gpa, tên, họ xem có cần tráo đổi hai sinh viên trong danh sách không

private static boolean shouldSwap(Student student, Student other) {

if (student.getGpa() < other.getGpa()) { // nếu điểm sv đứng trước nhỏ hơn -> tráo đổi

return true;

} else if (student.getGpa() == other.getGpa() // điểm bằng nhau -> xét tên

&& student.getFullNameObject().getFirstName()

.compareTo(other.getFullNameObject().getFirstName()) > 0) {

return true; // tên đứng trước lớn hơn(theo bảng chữ cái) -> tráo đổi vị trí

} else return student.getGpa() == other.getGpa() // tên và điểm trùng nhau -> xét họ

&& student.getFullNameObject().getFirstName()

.compareTo(other.getFullNameObject().getFirstName()) == 0

&& student.getFullNameObject().getLastName()

.compareTo(other.getFullNameObject().getLastName()) > 0;

}

// phương thức hiển thị danh sách sinh viên lên màn hình dạng bảng gồm các hàng, cột

private static void showStudents(ArrayList<Student> students) {

System.out.println("========================== DANH SÁCH SINH VIÊN ==========================");

System.out.printf("%-10s%-10s%-15s%-10s%-20s%-10s\n",

"Mã SV", "Họ", "Đệm", "Tên", "Chuyên ngành", "Điểm gpa");

for (var student : students) {

System.out.printf("%-10s%-10s%-15s%-10s%-20s%-10.2f\n",

student.getStudentId(), student.getFullNameObject().getLastName(),

student.getFullNameObject().getMidName(), student.getFullNameObject().getFirstName(),

student.getMajor(), student.getGpa());

}

}

}

/\*\*

\* Lớp mô tả thông tin sinh viên

\*/

class Student {

private String studentId; // mã sinh viên

private String fullName; // họ và tên

private String major; // chuyên ngành

private float gpa; // điểm gpa

private final FullName fullNameObj;

public Student() {

fullNameObj = new FullName();

}

public Student(String studentId, String fullName, String major, float gpa) {

this();

this.setStudentId(studentId);

this.setFullName(fullName);

this.setMajor(major);

this.setGpa(gpa);

}

public String getStudentId() {

return studentId;

}

public void setStudentId(String studentId) {

this.studentId = studentId;

}

public void setFullName(String fullName) {

this.fullName = fullName;

}

public String getFullName() {

return this.fullName;

}

public String getMajor() {

return major;

}

public void setMajor(String major) {

this.major = major;

}

public float getGpa() {

return gpa;

}

public void setGpa(float gpa) {

this.gpa = gpa;

}

public FullName getFullNameObject() {

return fullNameObj;

}

/\*\*

\* Lớp inner class mô tả thông tin về họ và tên chi tiết.

\* Sử dụng lớp này với ngữ cảnh inner class vì nó cần truy cập vào thuộc

\* tính của lớp outer chứa nó.

\*/

public class FullName {

String firstName; // tên

String lastName; // họ

String midName; // đệm

FullName() {

initName();

}

private void initName() {

var words = fullName.split("\\s+");

if (words.length == 1) {

this.firstName = words[0];

} else if (words.length == 2) {

this.firstName = words[1];

this.lastName = words[0];

} else if (words.length > 2) {

var midNameBuilder = new StringBuilder();

for (int i = 1; i < words.length - 1; i++) {

midNameBuilder.append(words[i]).append(" ");

}

this.firstName = words[words.length - 1];

this.midName = midNameBuilder.toString().trim();

this.lastName = words[0];

}

}

public String getFirstName() {

return firstName;

}

public String getLastName() {

return lastName;

}

public String getMidName() {

return midName;

}

}

}

==> Trước khi sắp xếp:

========================== DANH SÁCH SINH VIÊN ==========================

Mã SV Họ Đệm Tên Chuyên ngành Điểm gpa

ST1001 Nguyễn Ngọc Mơ CNTT 3.56

ST1002 Nguyễn Ngọc Mai CNTT 3.40

ST1003 Nguyễn Ngọc My CNTT 3.69

ST1004 Nguyễn Ngọc Khánh CNTT 3.34

ST1005 Nguyễn Ngọc Mun CNTT 3.15

ST1006 Lý Ngọc Mơ CNTT 3.56

ST1007 Mai Ngọc Mơ CNTT 3.56

==> Sau khi sắp xếp:

========================== DANH SÁCH SINH VIÊN ==========================

Mã SV Họ Đệm Tên Chuyên ngành Điểm gpa

ST1003 Nguyễn Ngọc My CNTT 3.69

ST1006 Lý Ngọc Mơ CNTT 3.56

ST1007 Mai Ngọc Mơ CNTT 3.56

ST1001 Nguyễn Ngọc Mơ CNTT 3.56

ST1002 Nguyễn Ngọc Mai CNTT 3.40

ST1004 Nguyễn Ngọc Khánh CNTT 3.34

ST1005 Nguyễn Ngọc Mun CNTT 3.15

* Ở đây ta sử dụng inner class vì ta cần truy cập tới fullName của lớp Student để tách full name thành họ-đệm-tên tiện quản lý về sau.
* Nếu ta muốn tạo đối tượng FullName độc lập ta cần thực hiện như sau:

Student student = new Student();

Student.FullName fullName = student.new FullName();

### Local class

* Local class – lớp cục bộ là nested class mà không phải thành phần của bất kì lớp nào và nó có tên rõ ràng.
* Tất cả các lớp local đều là inner class.
* Các khai báo của lớp cục bộ thường đặt trong một khối, trong một phương thức nào đó.
* Lớp cục bộ không chứa access modifier public, private, protected hay non-access static.
* Sử dụng lớp cục bộ khi bạn cần tạo một kiểu mới có tên kiểu cụ thể, muốn tạo nhiều thể hiện của lớp, truy cập phương thức khởi tạo của nó.
* Ví dụ hình ảnh về local class:

package net.braniumacademy.chapter6.lesson8;

public class TestAnonymous {

public static void main(String[] args) {

// ví dụ về lớp cục bộ nằm trong phương thức main():

class MyGreeting implements Greeting {

@Override

public void sayHello(String name) {

System.out.println(STR."Hello \{name}!");

}

@Override

public void sayGoodbye(String name) {

System.out.println(STR."Goodbye \{name}!");

}

}

// tạo đối tượng và sử dụng đối tượng này:

var greeting = new MyGreeting();

greeting.sayHello("Mai");

greeting.sayGoodbye("Khang");

}

}

// interface mô tả hành vi chào hỏi

interface Greeting {

void sayHello(String name);

void sayGoodbye(String name);

}

Hello Mai!

Goodbye Khang!

* Có hai loại biến thể của inner class là local classes và anonymous classes. Nội dung kế tiếp ta sẽ điểm qua về anonymous class.

### Anonymous class

* Anonymous class – lớp vô danh cho phép bạn tạo những đoạn code ngắn gọn không cần tên lớp.
* Anonymous class cho bạn khả năng tạo lớp và tạo đối tượng tại cùng một thời điểm.
* Anonymous class giống local class nhưng không có tên, thường dùng để triển khai một interface nào đó.
* Sử dụng anonymous khi bạn cần sử dụng local class chỉ 1 lần, tức không cần sử dụng lại về sau.
* Ta sử dụng lớp vô danh khi cần sắp xếp, tìm kiếm, triển khai nhanh đối tượng của một interface nào đó.
* Cụ thể hơn về anonymous class ta sẽ gặp lại chi tiết hơn trong phần các bài học liên quan tính trừu tượng.
* Sau đây là ví dụ về lớp vô danh:

package net.braniumacademy.chapter6.lesson8;

public class TestAnonymous {

public static void main(String[] args) {

// ví dụ về lớp vô danh:

Greeting greeting = new Greeting() {

@Override

public void sayHello(String name) {

System.out.println(STR."Hello \{name}!");

}

@Override

public void sayGoodbye(String name) {

System.out.println(STR."Goodbye \{name}!");

}

};

greeting.sayHello("Mai");

greeting.sayGoodbye("Khang");

}

}

// interface mô tả hành vi chào hỏi

interface Greeting {

void sayHello(String name);

void sayGoodbye(String name);

}

Hello Mai!

Goodbye Khang!

* Ví dụ khác sử dụng lớp vô danh để sắp xếp danh sách sinh viên theo tên a-z: chi tiết về inteface Comparator bạn xem trong chương 9.

students.sort(new Comparator<Student>() {

@Override

public int compare(Student student, Student other) {

return student.getFullNameObj()

.getFirstName()

.compareTo(other.getFullNameObj().getFirstName());

}

});

// tương đương với:

students.sort((student, other) -> student.getFullNameObj()

.getFirstName()

.compareTo(other.getFullNameObj().getFirstName()));

// tương đương với:

students.sort(Comparator.comparing(student -> student.getFullNameObj().getFirstName()));

# Bài 6.9. Mối quan hệ giữa các lớp Java

### Nội dung bài học

1. [Giới thiệu](https://braniumacademy.net/courses/java/lessons/java-class-uml-relationship/#s1)
2. [Biểu diễn lớp trong UML](https://braniumacademy.net/courses/java/lessons/java-class-uml-relationship/#s2)
3. [Phân loại sơ đồ lớp](https://braniumacademy.net/courses/java/lessons/java-class-uml-relationship/#s3)
4. [Quan hệ dependency](https://braniumacademy.net/courses/java/lessons/java-class-uml-relationship/#s4)
5. [Quan hệ association](https://braniumacademy.net/courses/java/lessons/java-class-uml-relationship/#s5)
6. [Quan hệ agregation](https://braniumacademy.net/courses/java/lessons/java-class-uml-relationship/#s6)
7. [Quan hệ composition](https://braniumacademy.net/courses/java/lessons/java-class-uml-relationship/#s7)
8. [Bài tập thực hành](https://braniumacademy.net/courses/java/lessons/java-class-uml-relationship/#s8)

### Giới thiệu

* Khi triển khai các hệ thống lớn, để đảm bảo tính nhất quán và chuyên nghiệp ta phải có bản vẽ thiết kế sơ đồ lớp để biểu diễn chúng.
* Mục đích là nhằm thuận tiện cho việc triển khai, bảo trì, giám sát về sau.
* Sơ đồ lớp biểu diễn quan hệ của các lớp, mỗi lớp lại được biểu diễn trên lược đồ UML.
* Giữa hai lớp độc lập trong Java luôn tồn tại các mối quan hệ có liên quan đến nhau. Nội dung bài học này ta sẽ tìm hiểu về các mối quan hệ đó.
* Các mối quan hệ giữa hai lớp có thể là:
  + Lớp này sử dụng lớp kia(dependency).
  + Lớp này triển khai lớp kia(implementation).
  + Lớp này kế thừa lớp kia(inheritance).
  + Lớp này là thành phần chặt của lớp kia(composition).
  + Lớp này là thành phần lỏng của lớp kia(agregation).
  + …
* Nội dung của bài sẽ tập trung vào quan hệ agregation và các biến thể của nó. Về quan hệ kế thừa, triển khai ta sẽ tìm hiểu sau trong chương 8, 9.

### Biểu diễn lớp trong UML

* Sử dụng các công cụ hỗ trợ vẽ UML [online](https://app.diagrams.net/) hoặc offline mà bạn thích để vẽ.
* Đầu tiên ta nhắc lại về class và object: class là bản vẽ để từ đó tạo ra các đối tượng thực thể object. Nói cách khác, class là cái khuôn, object là sản phẩm tạo ra bởi cái khuôn đó.

Ảnh có chứa văn bản, biểu đồ, thiết kế

Mô tả được tạo tự động

* Xét hình trên: class mô tả con cún có các đặc điểm về màu lông, màu mắt, chiều cao, cân nặng, chiều dài cơ thể.
* Khi tạo đối tượng con cún thì các thông số trên sẽ chi tiết hơn: lông màu vàng, mắt màu nâu, cao 17in, dài 35in, nặng 15.5kg chẳng hạn.
* Mỗi đối tượng thực tế tạo ra từ cái khuôn class có các thông số và hành vi giống nhau nhưng khác giá trị. Ví dụ ta có con chó mực thì nó màu đen, nặng 30kg…
* Như vậy class chỉ nêu ra các đặc điểm chung, không nêu giá trị cụ thể của các đặc điểm đó. Tương tự cho hành vi: cho biết bắt tay nhưng có con chó biết bắt tay bằng cả 2 tay, có con chỉ biết bắt 1 tay, có con bắt 4 tay…
* Để mô tả cụ thể từng đối tượng thì rất vô cùng vô tận nên ta chỉ mô tả class với những đặc điểm chung nhất, còn về chi tiết sẽ do từng object(instance) sinh ra từ class đó quyết định.
* Bây giờ ta nói về quy ước biểu diễn lớp trong UML: một lớp đầy đủ thường gồm 3 phần: tên lớp, các thuộc tính, các phương thức.

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, số

Mô tả được tạo tự động

* Tên lớp xuất hiện đầu tiên, trên cùng. Đặt đúng quy tắc đặt tên lớp: sử dụng danh từ, cụm danh từ. Ví dụ: Dog, Cat, PaymentMethod, BillingAddress, PlayingScore…
* Tiếp theo là các thuộc tính(attribute/field):
  + Thuộc tính private bắt đầu với dấu trừ – ở đầu sau đó là kiểu của thuộc tính đặt sau dấu hai chấm, ví dụ -fullName: String.
  + Thuộc tính public tương tự nhưng bắt đầu với dấu cộng +: +fullName: String.
  + Thuộc tính protected bắt đầu với dấu thăng #: #empId: String.
  + Thuộc tính static có thể là private, protected hoặc public và được gạch chân.
* Phần cuối cùng trong UML là danh sách các phương thức:
  + Phương thức private bắt đầu với dấu -. Ví dụ: – createStudent(): Student.
  + Phương thức protected bắt đầu với dấu #. Ví dụ: # showInfo(): void.
  + Phương thức public bắt đầu với dấu cộng +. Ví dụ: + doHomework(subject: String): boolean.
  + Phương thức static có thể là 1 trong 3 loại trên và có gạch chân.
  + Phương thức cũng có kiểu trả về để sau dấu hai chấm. Ví dụ: +play(gameName: String): int.
  + Nếu phương thức có tham số, liệt kê tên trước : kiểu tham số. Ví dụ: +add(a: int, b: int, c: float): float.
* Ví dụ UML mô tả lớp sinh viên với các thông tin: id tự tăng, id, gpa, fullName, major. Hành vi: thiết lập họ và tên, làm bài kiểm tra, làm bài tập, nộp học phí.

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, số

Mô tả được tạo tự động

### Phân loại sơ đồ lớp

* Có 3 dạng sơ đồ lớp tùy thuộc vào mức độ và quy trình phát triển sản phẩm phần mềm của bạn đang ở pha nào.
* Một sơ đồ lớp có thể là sự kết hợp của nhiều loại:
  + Sơ đồ mức khái niệm: thể hiện khái niệm của lớp trong miền ứng dụng.
  + Sơ đồ mức đặc tả: tập trung vào giao diện của tầng dữ liệu trừu tượng của phần mềm.
  + Sơ đồ mức triển khai: mô tả cách các lớp sẽ được triển khai như nào.
* Sơ đồ lớp mức khái niệm: mới chỉ nêu lên vấn đề, chưa đi vào chi tiết.

Ảnh có chứa văn bản, ảnh chụp màn hình, số, Phông chữ

Mô tả được tạo tự động

* Sơ đồ mức đặc tả: mô tả các trường thuộc tính, các phương thức cùng các tham số có thể có hoặc chưa có kiểu dữ liệu.

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, số

Mô tả được tạo tự động

* Sơ đồ mức triển khai: cao hơn sơ đồ mức đặc tả ở chỗ có giá trị cụ thể gán cho thuộc tính.

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, số

Mô tả được tạo tự động

### Quan hệ dependency

* Quan hệ phụ thuộc xảy ra khi một đối tượng của một lớp A sử dụng đối tượng của lớp B trong phương thức của nó và object của lớp B không được lưu trữ trong bất kỳ thuộc tính/trường nào của lớp A.
* Ví dụ trong code:

package net.braniumacademy.chapter6.lesson9;

// lớp Student

public class Student {

private static int autoId = 1000;

private String studentId;

private String fullName;

private String major;

private float gpa;

public String getStudentId() {

return studentId;

}

// ...

}

// lớp Main:

public class Main {

// ...

public void showInfo(Student student) {

System.out.println(STR."Student id: \{student.getStudentId()}");

// ...

}

}

* Ta biểu diễn quan hệ phụ thuộc bằng đường nét đứt có mũi tên cắm vào lớp bị sử dụng: lớp Main sử dụng Student.

Ảnh có chứa văn bản, ảnh chụp màn hình, hàng, Phông chữ

Mô tả được tạo tự động

### Quan hệ association

* Quan hệ liên kết thể hiện sự kết nối giữa hai lớp có liên quan đến nhau, biểu diễn bằng một đường nét liền nối hai lớp.
* Ví dụ quan hệ giữa lớp điều khiển và lớp biên.
* Loại quan hệ này thường chia thành hai loại cụ thể hơn:
  + Aggregation: quan hệ thành phần/liên kết lỏng.
  + Composition: quan hệ thành phần/liên kết chặt.
* Số lượng liên quan có thể là quan hệ:
  + 1 – 1
  + 1 – nhiều
  + nhiều nhiều
* Quan hệ 1 – 1 có thể là:
  + 0..1: không hoặc 1.
  + 1: chính xác 1.
* Phần nhiều có thể là:
  + 1..\*: 1 hoặc nhiều.
  + \* 0 hoặc nhiều.

Ảnh có chứa văn bản, ảnh chụp màn hình, hàng, Phông chữ

Mô tả được tạo tự động

### Quan hệ agregation

* Là loại quan hệ thành phần lỏng, biến thể đặc biệt hơn của quan hệ liên kết.
* Nó thể hiện A là một phần cấu thành của B hoặc A là một phần của B.
* Nhiều đối tượng của lớp A có thể là thành phần của lớp B.
* Đối tượng của lớp A và lớp B tồn tại độc lập, không ảnh hưởng dây chuyền tới nhau. Tức là khi đối tượng của lớp A có tồn tại hay không thì không ảnh hưởng tới đối tượng lớp B và ngược lại.
* Biểu diễn quan hệ này bằng đường nối nét liền có một hình thoi rỗng cắm vào lớp sở hữu.
* Ví dụ: sinh viên có họ và tên, ô tô có động cơ, máy tính có con vi xử lý, ram, màn hình…

class Computer {

private String id;

private CPU cpu;

private Monitor monitor;

private RAM ram;

// ...

public void setId(String id) {

this.id = id;

}

// các đối tượng thành phần sẽ được gắn giá trị khi tính toán hoặc được cung cấp sau

public void setCpu(CPU cpu) {

this.cpu = cpu;

}

public void setMonitor(Monitor monitor) {

this.monitor = monitor;

}

public void setRam(RAM ram) {

this.ram = ram;

}

}

class CPU {

// số lõi, số luồng, hiệu năng...

}

class RAM {

// loại ram, kích thước, tốc độ, dung lượng...

}

class Monitor {

// hãng sản xuất, kích thước, độ phân giải...

}

* Biểu diễn trong UML:

Ảnh có chứa văn bản, ảnh chụp màn hình, biểu đồ, số

Mô tả được tạo tự động

* Mô tả: một máy tính có thành phần cấu thành là:
  + 1 con cpu, ví dụ của AMD hoặc Intel, Snapdrangon…
  + 1 hoặc nhiều thanh RAM, ví dụ của hãng Sandisk, Samsung, Kingston, Crucial, Gigabyte…
  + 1 máy tính có thể có 1 hoặc nhiều màn hình: 1 màn của Dell, 1 màn Samsung, 1 màn HP…
  + Và các thuộc tính khác.
* Biểu hiện trong code: đối tượng thành phần sẽ được cung cấp sau trong các phương thức khác constructor khi cần tới.

### Quan hệ composition

* Là biến thể của quan hệ liên kết. Trong đó hai lớp liên kết chặt chẽ với nhau. Ta còn gọi quan hệ này là quan hệ thành phần chặt.
* Tự bản thân 1 đối tượng thành phần không thể tồn tại nếu thiếu đối tượng chủ.
* Khi đối tượng chủ bị hủy thì đối tượng thành phần cũng sẽ bị hủy theo.
* Ví dụ mỗi người sẽ gắn liền với một mã căn cước công dân tới hết đời. Mã này sẽ không được sử dụng cho ai khác ngoài bạn.
* Mỗi chiếc điện thoại sẽ có 1 bo mạch chủ và có mã MAC định danh phần cứng duy nhất. Thiếu nó hoặc bo mạch bị hỏng thì smartphone cũng không thể sử dụng được nữa.
* Khác biệt với quan hệ aggregation: trong quan hệ composition, đối tượng thành phần được khởi tạo trực tiếp ngay khi khai báo thuộc tính hoặc được gán trong constructor.
* Ví dụ:

class Computer {

private String id;

private CPU cpu;

private Monitor monitor;

private RAM ram;

public Computer(String id, CPU cpu, Monitor monitor, RAM ram) {

this.id = id;

// các đối tượng thành phần chặt được khởi tạo ngay khi tạo object chủ

this.cpu = cpu; // không thể thiếu

this.monitor = monitor; // không thể thiếu

this.ram = ram; // không thể thiếu

}

// ...

}

class CPU {

// số lõi, số luồng, hiệu năng...

}

class RAM {

// loại ram, kích thước, tốc độ, dung lượng...

}

class Monitor {

// hãng sản xuất, kích thước, độ phân giải...

}

* Biểu diễn trong UML: sử dụng đường nối liền với hình thoi đặc ở 1 đầu và phía có hình thoi cắm vào lớp chủ thể, phần đuôi cắm vào lớp thành phần.

Ảnh có chứa văn bản, ảnh chụp màn hình, biểu đồ, số

Mô tả được tạo tự động

# Bài 6.10. Đọc file với Scanner

### Nội dung bài học

1. [Tìm hiểu về lớp File](https://braniumacademy.net/courses/java/lessons/java-read-file-scanner/#s1)
2. [Đọc file với Scanner](https://braniumacademy.net/courses/java/lessons/java-read-file-scanner/#s2)
3. [Ví dụ minh họa](https://braniumacademy.net/courses/java/lessons/java-read-file-scanner/#s3)
4. [Lưu ý trên máy Mac](https://braniumacademy.net/courses/java/lessons/java-read-file-scanner/#s4)
5. [Bài tập thực hành](https://braniumacademy.net/courses/java/lessons/java-read-file-scanner/#s5)

**Đọc file trên máy Mac**

### Tìm hiểu về lớp File

* Có rất nhiều cách khác nhau để đọc file trong Java, nội dung này sẽ giới thiệu với bạn cách đọc file đơn giản nhất, sử dụng lớp Scanner mà ta đã quen thuộc từ đầu khóa học tới giờ.
* Chi tiết hơn về các cách đọc file khác, bạn vui lòng tham khảo trong **chương 13: thao tác với file và kết nối cơ sở dữ liệu** nhé.
* Là một lớp được xây dựng sẵn trong thư viện Java để tạo ra các đối tượng file đại diện cho các đường dẫn tệp và thư mục.
* Một số phương thức thường dùng:
  + **File(pathName):** Tạo đối tượng file từ đường dẫn cho trong tham số. Nếu tham số là một string rỗng thì phương thức sẽ tạo đối tượng file với đường dẫn là rỗng.
  + **exists():** Kiểm tra xem tệp hay thư mục được thể hiện trong đối tượng file có tồn tại hay không.
  + **getName():** Trả về tên tệp hay thư mục đại diện bởi đối tượng file. Nếu tên rỗng ta sẽ nhận được một String rỗng. Phương thức này chỉ trả về tên và đuôi mở rộng(nếu có). Không trả về đường dẫn của tệp hay thư mục.
  + **getAbsoluteFile():** Trả về một đối tượng file tuyệt đối với tên và đường dẫn đầy đủ của tệp/thư mục.
  + **getAbsolutePath():** Trả về một String là đường dẫn tuyệt đối của đối tượng file này.
  + **canRead():** Kiểm tra xem file được chỉ ra bởi đối tượng file có thể đọc hay không.
  + **canWrite():** Kiểm tra xem file được chỉ ra bỏi đối tượng file có thể ghi hay không.
  + **length():** Trả về kích thước của file được chỉ ra trong đối tượng file này ở đơn vị byte. Nếu đường dẫn trỏ đến thư mục thì kích thước sẽ không xác định được chính xác.
  + **isFile():** Kiểm tra xem đối tượng file này có đang trỏ đến file hay không.
  + **isDirectory():** Kiểm tra xem đối tượng file này có đang trỏ đến một thư mục hay không.
  + **isHidden():** Kiểm tra xem file hiện thời có đang bị ẩn trên hệ điều hành mà nó đang chạy hay không.

#### Các phương thức thao tác tạo, đổi tên, xóa tệp, thư mục:

* + **renameTo(newFile):** Đổi tên file được trỏ tới bởi đối tượng file hiện thời thành file mới với tên được cung cấp bởi đối tượng newFile trong tham số. Nếu việc đổi tên thực hiện thành công thì trả về true và ngược lại trả về false.
  + **mkdir():** Tạo thư mục đích với tên được trỏ đến bởi đối tượng file này. Trả về true nếu tạo file thành công và false nếu ngược lại.
  + **mkdirs():** Tạo thư mục đích và tất cả các thư mục cha trong đường dẫn được trỏ tới bởi đối tượng file này. Trả về true nếu tạo thành công và false nếu ngược lại.
  + **delete():** Xóa tệp hoặc thư mục chỉ ra trong đường dẫn của đối tượng file hiện thời. Nếu là thư mục thì muốn xóa được nó phải rỗng. Trả về true nếu xóa thành công và false nếu ngược lại.
  + **deleteOnExist():** Xóa tệp hay thư mục nếu nó tồn tại.

#### Ví dụ 1: kiểm tra thông tin file/thư mục.

import java.io.File;

public class ReadFile {

public static void main(String[] args) {

File file = new File("D:\\style2.txt");

System.out.println("Đây là một tập tin? " + file.isFile());

System.out.println("Đây là một thư mục? " + file.isDirectory());

System.out.println("Có thể đọc file này? " + file.canRead());

System.out.println("Có thể ghi thêm vào file này? " + file.canWrite());

System.out.println("Tên file: " + file.getName());

System.out.println("Đường dẫn đầy đủ: " + file.getAbsolutePath());

System.out.println("Kích thước file: " + file.length() + " byte");

}

}

Đây là một tập tin? true

Đây là một thư mục? false

Có thể đọc file này? true

Có thể ghi thêm vào file này? true

Tên file: style2.txt

Đường dẫn đầy đủ: D:\style2.txt

Kích thước file: 438 byte

**Ví dụ 2: tạo file trong thư mục gốc project IntelliJ:**

package net.braniumacademy.chapter6.lesson10;

import java.io.File;

import java.io.IOException;

public class Lesson610 {

public static void main(String[] args) throws IOException {

File file = new File("input.txt");

if (file.createNewFile()) {

System.out.println("==> Tạo file thành công!");

} else {

System.out.println("==> Tạo file thất bại!");

}

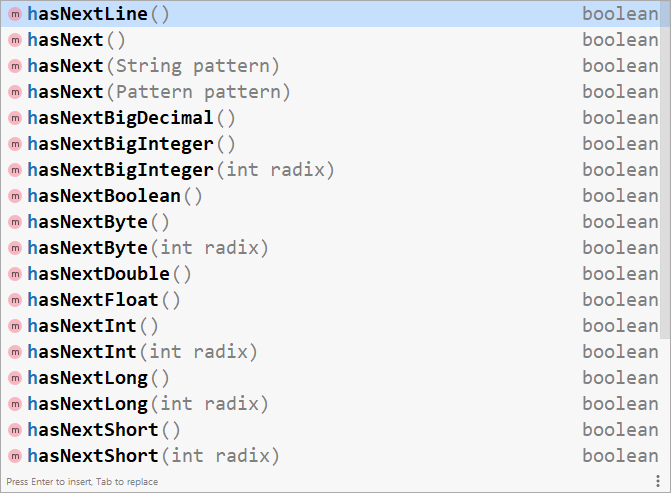
}

}

### Đọc file với Scanner

Để thực hiện đọc file với đối tượng của Scanner, ta làm như sau:

* Đầu tiên tạo đối tượng file với lớp File và cung cấp đầy đủ đường dẫn, tên file chính xác.
* Sau đó ta tạo đối tượng của Scanner với tham số truyền vào là đối tượng file ở trên.
* Thêm xử lý ngoại lệ vào khai báo phương thức hoặc cho khai báo của đối tượng scanner vào try-catch.
* Duyệt file và đọc ra các giá trị như mong muốn. Sử dụng vòng lặp ở bước này. Lưu ý kiểm tra xem nội dung cần đọc có tồn tại tiếp hay không.
* Sau khi đọc xong chủ động đóng file lại với phương thức close().
* Các phương thức dùng để kiểm tra dữ liệu trong file có tồn tại nữa hay không:

[](https://braniumacademy.net/wp-content/uploads/2020/09/vi-du-cac-phuong-thuc-hasNextX.png)

### Ví dụ minh họa

* **Ví dụ 1: đọc ra thông tin của các sinh viên cho trong file txt.**
* File input.txt:

20

Ha Noi

Tran Van Hung

hung@xmail.com

3.36

25

Thai Binh

Ngo Van Trung

ngtrung@xmail.com

3.66

21

Da Nang

Nguyen Thi Loan

loannana@xmail.com

3.58

18

Ho Chi Minh City

Le Hoang Yen

lehoangyen@xmail.com

2.64

package net.braniumacademy.chapter6.lesson10;

import java.io.File;

import java.io.FileNotFoundException;

import java.util.ArrayList;

import java.util.Scanner;

public class Lesson610 {

public static void main(String[] args) throws FileNotFoundException {

// tạo đối tượng file trỏ tới file cần đọc

File inputFile = new File("E:/Documents/input.txt");

// tạo đối tượng scanner để đọc dữ liệu từ file

Scanner scanner = new Scanner(inputFile);

// tạo arraylist để chứa danh sách sinh viên

ArrayList<Student> students = new ArrayList<>();

// gọi method đọc file

readStudentFromFile(students, scanner);

scanner.close(); // đóng luồng đọc file

showStudentInfo(students); // hiển thị thông tin lên màn hình

}

/\*\*

\* Phương thức đọc ra thông tin sinh viên trong file.

\*

\* @param students danh sách sinh viên đọc được

\* @param scanner đối tượng đọc file

\*/

private static void readStudentFromFile(ArrayList<Student> students, Scanner scanner) {

while (scanner.hasNextLine()) {

var age = scanner.nextInt();

scanner.nextLine();

var address = scanner.nextLine();

var fullName = scanner.nextLine();

var email = scanner.nextLine();

var gpa = scanner.nextFloat();

var student = new Student(fullName, address, email, gpa, age);

students.add(student);

}

}

/\*\*

\* Phương thức hiển thị thông tin sinh viên dạng bảng.

\*

\* @param students danh sách sinh viên cần hiển thị.

\*/

private static void showStudentInfo(ArrayList<Student> students) {

System.out.println("================= THÔNG TIN SINH VIÊN =================");

System.out.printf("%-30s%-25s%-25s%-10s%-10s\n",

"Họ và tên", "Địa chỉ", "Email", "Điểm gpa", "Tuổi");

for (var student : students) {

System.out.printf("%-30s%-25s%-25s%-10.2f%-10d\n",

student.getFullName(), student.getAddress(), student.getEmail(),

student.getGpa(), student.getAge());

}

}

}

/\*\*

\* Lớp mô tả thông tin sinh viên.

\*/

class Student {

private String fullName;

private String address;

private String email;

private float gpa;

private int age;

public Student() {

}

public Student(String fullName, String address, String email, float gpa, int age) {

this.setFullName(fullName);

this.setAddress(address);

this.setEmail(email);

this.setGpa(gpa);

this.setAge(age);

}

public String getFullName() {

return fullName;

}

public void setFullName(String fullName) {

this.fullName = fullName;

}

public String getAddress() {

return address;

}

public void setAddress(String address) {

this.address = address;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public float getGpa() {

return gpa;

}

public void setGpa(float gpa) {

this.gpa = gpa;

}

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

}

===================================== THÔNG TIN SINH VIÊN =======================================

Họ và tên Địa chỉ Email Điểm gpa Tuổi

Tran Van Hung Ha Noi hung@xmail.com 3.25 20

Ngo Van Trung Thai Binh ngtrung@xmail.com 3.66 25

Nguyen Thi Loan Da Nang loannana@xmail.com 3.58 21

Le Hoang Yen Ho Chi Minh City lehoangyen@xmail.com 2.64 18

* **Ví dụ 2: đọc danh sách sinh viên, trong đó thông tin về mỗi sinh viên cho trên 1 dòng.**
* Nội dung file input.txt đặt trong thư mục gốc project của IntelliJ:

S1001,Lê Bá Nam,Quảng Ninh,CNTT,3.26

S1002,Hoàng Quốc Tuấn,Hồ Chí Minh,CNTT,3.54

S1003,Mai Thị Diệu Linh,Vũng Tàu,CNTT,3.71

S1004,Ngô Quang Trường,Bình Dương,CNTT,3.33

S1005,Hồ Hoài Anh,Quảng Bình,CNTT,3.12

S1006,Nguyễn Ngọc Mai,Đà Nẵng,CNTT,3.95

S1007,Nguyễn Thị Ngọc Linh,Hà Nội,CNTT,3.18

S1008,Trần Quang Khải,Bắc Giang,CNTT,3.29

S1009,Ma Thùy Dung,Thái Nguyên,CNTT,3.37

S1010,Lương Đình Vinh,Nghệ An,CNTT,3.59

package net.braniumacademy.chapter6.lesson10;

import java.io.File;

import java.io.FileNotFoundException;

import java.util.ArrayList;

import java.util.Scanner;

public class Lesson610 {

public static void main(String[] args) throws FileNotFoundException {

// tạo đối tượng file trỏ tới file cần đọc

File inputFile = new File("input.txt");

// tạo đối tượng scanner để đọc dữ liệu từ file

Scanner scanner = new Scanner(inputFile);

// tạo arraylist để chứa danh sách sinh viên

ArrayList<Student> students = new ArrayList<>();

// gọi method đọc file

readStudentFromFile(students, scanner);

scanner.close(); // đóng luồng đọc file

showStudentInfo(students); // hiển thị thông tin lên màn hình

}

/\*\*

\* Phương thức đọc ra thông tin sinh viên trong file.

\*

\* @param students danh sách sinh viên đọc được

\* @param scanner đối tượng đọc file

\*/

private static void readStudentFromFile(ArrayList<Student> students, Scanner scanner) {

while (scanner.hasNextLine()) {

var data = scanner.nextLine().split(",");

var gpa = Float.parseFloat(data[4]);

var student = new Student(data[0], data[1], data[2], data[3], gpa);

students.add(student);

}

}

/\*\*

\* Phương thức hiển thị thông tin sinh viên dạng bảng.

\*

\* @param students danh sách sinh viên cần hiển thị.

\*/

private static void showStudentInfo(ArrayList<Student> students) {

var line = "===========================";

System.out.println(STR."\{line} THÔNG TIN SINH VIÊN \{line}");

System.out.printf("%-5s%-10s%-25s%-15s%-10s%-10s\n",

"STT", "Mã SV", "Họ và tên", "Địa chỉ", "C.Ngành", "Điểm gpa");

var counter = 1;

for (var student : students) {

System.out.printf("%-5d%-10s%-25s%-15s%-10s%-10s\n",

counter++, student.getStudentId(), student.getFullName(),

student.getAddress(), student.getMajor(), student.getGpa());

}

}

}

/\*\*

\* Lớp mô tả thông tin sinh viên.

\*/

class Student {

private String studentId;

private String fullName;

private String address;

private String major;

private float gpa;

public Student() {

}

public Student(String studentId, String fullName,

String address, String major, float gpa) {

this.setStudentId(studentId);

this.setFullName(fullName);

this.setAddress(address);

this.setMajor(major);

this.setGpa(gpa);

}

public String getStudentId() {

return studentId;

}

public void setStudentId(String studentId) {

this.studentId = studentId;

}

public String getFullName() {

return fullName;

}

public void setFullName(String fullName) {

this.fullName = fullName;

}

public String getAddress() {

return address;

}

public void setAddress(String address) {

this.address = address;

}

public String getMajor() {

return major;

}

public void setMajor(String major) {

this.major = major;

}

public float getGpa() {

return gpa;

}

public void setGpa(float gpa) {

this.gpa = gpa;

}

}

=========================== THÔNG TIN SINH VIÊN ===========================

STT Mã SV Họ và tên Địa chỉ C.Ngành Điểm gpa

1 S1001 Lê Bá Nam Quảng Ninh CNTT 3.26

2 S1002 Hoàng Quốc Tuấn Hồ Chí Minh CNTT 3.54

3 S1003 Mai Thị Diệu Linh Vũng Tàu CNTT 3.71

4 S1004 Ngô Quang Trường Bình Dương CNTT 3.33

5 S1005 Hồ Hoài Anh Quảng Bình CNTT 3.12

6 S1006 Nguyễn Ngọc Mai Đà Nẵng CNTT 3.95

7 S1007 Nguyễn Thị Ngọc Linh Hà Nội CNTT 3.18

8 S1008 Trần Quang Khải Bắc Giang CNTT 3.29

9 S1009 Ma Thùy Dung Thái Nguyên CNTT 3.37

10 S1010 Lương Đình Vinh Nghệ An CNTT 3.59

### Lưu ý trên máy Mac

* Để tạo file **.txt** hoặc đuôi bất kì ta sử dụng phần mềm **TextEdit** và nhớ đặt tên file, chỉ định đuôi file theo mong muốn và chọn lưu vào thư mục đích.
* Để lấy đường dẫn của một file bất kì, mở **Finder**-> tìm file cần lấy đường dẫn -> nhấn phím **control** và **nhấn chuột vào file cần lấy đường dẫn** -> nhấn phím **option** -> chọn **Copy “tên file.đuôi file” as pathName**. Sau đó quay sang công cụ lập trình pate vào là xong.
* Để đọc file trong cùng project IntelliJ ta chỉ cần tạo file tại tên thư mục project và điền tên file, đuôi file vào không cần đường dẫn.
* Để đọc file trong thư mục/gói con của cùng project IntelliJ ta thêm đường dẫn tới file đó tính từ thư mục gốc project.
* Tất cả các file cần đọc phải có tên file và đuôi file đầy đủ mới có hiệu lực. Ví dụ input.txt.
* Các file không nằm ở thư mục gốc project phải có đường dẫn tuyệt đối hoặc tương đối.

# Bài 6.12. Ghi file với PrintWriter

### Nội dung bài học

1. [Tạo thư mục, file bằng File object](https://braniumacademy.net/courses/java/lessons/java-write-file-printwriter/#s1)
2. [Lớp PrintWriter](https://braniumacademy.net/courses/java/lessons/java-write-file-printwriter/#s2)
3. [Lớp FileWriter](https://braniumacademy.net/courses/java/lessons/java-write-file-printwriter/#s3)
4. [Lớp FileOuputStream](https://braniumacademy.net/courses/java/lessons/java-write-file-printwriter/#s8)
5. [Ghi dữ liệu ra màn hình](https://braniumacademy.net/courses/java/lessons/java-write-file-printwriter/#s4)
6. [Ghi dữ liệu ra file](https://braniumacademy.net/courses/java/lessons/java-write-file-printwriter/#s5)
7. [Ví dụ minh họa](https://braniumacademy.net/courses/java/lessons/java-write-file-printwriter/#s6)
8. [Bài tập thực hành](https://braniumacademy.net/courses/java/lessons/java-write-file-printwriter/#s7)

### Tạo thư mục, file bằng File object

* Một số các phương thức của lớp File dùng để thao tác với file, thư mục:
  + **renameTo(newFile):** Đổi tên file được trỏ tới bởi đối tượng file hiện thời thành file mới với tên được cung cấp bởi đối tượng newFile trong tham số. Nếu việc đổi tên thực hiện thành công thì trả về true và ngược lại trả về false.
  + **createNewFile():** Tạo file mới với tên được cung cấp trong tham số của constructor nếu như file được chỉ ra đó chưa tồn tại trong đường dẫn. Trả về true nếu quá trình tạo file mới thành công và false trong trường hợp ngược lại.
  + **mkdir():** Tạo thư mục đích với tên được trỏ đến bởi đối tượng file này. Trả về true nếu tạo file thành công và false nếu ngược lại.
  + **mkdirs():** Tạo thư mục đích và tất cả các thư mục cha trong đường dẫn được trỏ tới bởi đối tượng file này. Trả về true nếu tạo thành công và false nếu ngược lại.
  + **delete():** Xóa tệp hoặc thư mục chỉ ra trong đường dẫn của đối tượng file hiện thời. Nếu là thư mục thì muốn xóa được nó phải rỗng. Trả về true nếu xóa thành công và false nếu ngược lại.
  + **deleteOnExist():** Xóa tệp hay thư mục nếu nó tồn tại.
* Ví dụ 1:

var filePath = "output.txt";

File file = new File(filePath);

var isCreateNewFileSuccess = file.createNewFile();

System.out.println("Tạo mới file thành công? "

+ isCreateNewFileSuccess);

// nếu file cần tạo chưa tồn tại:

Tạo mới file thành công? True

// nếu file cần tạo mới đã tồn tại trước đó:

Tạo mới file thành công? false

* Ví dụ 2:

var filePath = "output.txt";

File file = new File(filePath);

File newFile = new File("OUTPUT.OUT");

file.renameTo(newFile);

System.out.println("File cũ: " + file.getName());

System.out.println("File mới: " + newFile.getName());

File cũ: output.txt

File mới: OUTPUT.OUT

### Lớp PrintWriter

* Tại sao phải ghi dữ liệu ra file? Lý do đơn giản vì dữ liệu trong các biến của chương trình là tạm thời. Chúng sẽ bị hủy khi chương trình không còn hoạt động nữa. Để lưu trữ và sử dụng lâu dài các dữ liệu ta phải ghi dữ liệu ra các file. Các file này có thể có các định dạng khác nhau. Ví dụ file âm thanh, hình ảnh, text, …
* Dùng để ghi dữ liệu dạng text ra file. Đây là dạng dữ liệu văn bản tường minh mà con người có thể đọc hiểu và sử dụng hàng ngày. Ví dụ như file word, pdf…
* Sau khi tạo đối tượng của lớp PrintWriter ta sử dụng các phương thức print, println, printf như thao tác hiển thị dữ liệu lên màn hình.
* Một số các constructor của lớp PrintWriter:
  + **PrintWriter(String fileName):** Tạo một đối tượng PrintWriter đổ dữ liệu vào tên file đã chỉ ra trong tham số. Không tự động xóa bộ đệm.
  + **PrintWriter(String fileName, String csn):** Tạo ra một đối tượng PrintWriter đổ dữ liệu vào file có tên cho trong tham số thứ nhất và hỗ trợ tập csn trong tham số thứ hai. Không tự động xóa bộ đệm.(csn = character-encoding scheme: lược đồ mã hóa kí tự)
  + **PrintWriter(File file):**Tạo một đối tượng PrintWriter trỏ đến đối tượng File đã được chỉ ra trong tham số. Không tự động xóa bộ đệm.
  + **PrintWriter(File file, String csn):** Tạo một đối tượng PrintWriter trỏ đến đối tượng File đã được chỉ ra trong tham số thứ nhất và hỗ trợ tập csn cho trong tham số thứ hai. Không tự động xóa bộ đệm.
  + **PrintWriter(OutputStream out):** Tạo một đối tượng PrintWriter mới từ đối tượng OutputStream cho trước. Không tự động xóa bộ đệm.
  + **PrintWriter(OutputStream out, boolean autoFlush):** Tạo một đối tượng PrintWriter mới từ đối tượng OutputStream cho trước và thiết lập kích hoạt hoặc không tự động xóa bộ đệm.
  + **PrintWriter(Writer out):** Tạo một đối tượng PrintWriter từ một đối tượng Writer cho trước.
  + **PrintWriter(Writer out, boolean autoFlush):** Tạo một đối tượng PrintWriter từ một đối tượng Writer cho trước và thiết lập auto flush.
* Ví dụ tạo đối tượng của lớp PrintWriter:

// tạo đối tượng PrintWriter trỏ trực tiếp đến một file

PrintWriter printWriter = new PrintWriter("output.txt", "UTF\_8" );

// tạo đối tượng PrintWriter để in ra màn hình

PrintWriter printWriter = new PrintWriter(System.out);

// tạo đối tượng PrintWriter sử dụng đối tượng File

var filePath = "output.txt";

File file = new File(filePath);

PrintWriter printWriter = new PrintWriter(file, "UTF\_8" );

// tạo đối tượng PrintWriter sử dụng đối tượng của FileWriter

var filePath = "output.txt";

File file = new File(filePath);

// sử dụng FileWriter nếu muốn thêm dữ liệu vào file

// ở chế độ append

FileWriter fileWriter = new FileWriter(file, true);

PrintWriter printWriter = new PrintWriter(fileWriter, true);

### Lớp FileWriter

* Là một lớp tiện dụng để ghi dữ liệu dạng text ra file. Giả định rằng các mặc định mã hóa ký tự và mặc định kích thước bộ đệm đã được chấp thuận.
* Nếu file để thao tác chưa tồn tại sẽ được tạo mới.
* Ở một số nền tảng, mỗi file tại 1 thời điểm chỉ được sử dụng bởi 01 đối tượng FileWriter. Do vậy việc tạo mới đối tượng từ file đang được sử dụng bởi một object của FileWriter khác sẽ thất bại.
* Một số constructor của lớp FileWriter:
  + **FileWriter(String fileName):** Tạo đối tượng FileWriter từ một tên file cho trước trong tham số.
  + **FileWriter(String fileName, boolean append):** Tạo đối tượng FileWriter từ một tên file nào đó cho trước trong tham số. Đồng thời xác định xem có kích hoạt chế độ ghi thêm vào cuối file hay không.
  + **FileWriter(File file):** Tạo đối tượng FileWriter từ một đối tượng file cho trước trong tham số.
  + **FileWriter(File file, boolean append):** Tạo đối tượng FileWriter từ một đối tượng file cho trước trong tham số. Đồng thời xác định xem có kích hoạt chế độ ghi thêm vào cuối file hay không.

### Lớp FileOuputStream

* Là một luồng dữ liệu dùng để ghi dữ liệu ra file hoặc một FileDescriptor.
* Nếu file cần ghi dữ liệu chưa tồn tại, nó sẽ tự tạo mới file để tiến hành ghi dữ liệu.
* Trên một số hệ điều hành chỉ cho phép một file được mở bởi duy nhất một đối tượng FileOutputStream tại 1 thời điểm. Do đó việc tạo mới đối tượng FileOuputStream với đối tượng file đang bị chiếm bởi một FileOuputStream nào đó sẽ thất bại.
* Lớp này dùng để ghi luồng các byte dữ liệu thô như dữ liệu về ảnh, video, audio. Nếu ghi luồng dữ liệu văn bản ta nên sử dụng FileWriter thay cho FileOuputStream.
* Sau đây là một số constructor của lớp này:
  + FileOutputStream(File file): tạo mới đối tượng FileOutputStream để ghi dữ liệu vào file được chỉ định trong tham số.
  + FileOutputStream(File file, boolean append): giống như trên nhưng chỉ định thêm chế độ ghi file là ghi đè hay thêm dữ liệu vào cuối.
  + FileOutputStream(String fileName): tạo mới đối tượng FileOutputStream để ghi dữ liệu vào file có tên được chỉ định trong tham số.
  + FileOutputStream(String fileName, boolean append): giống như trên nhưng có thêm chế độ ghi file.
  + FileOutputStream(FileDescriptor fdObj): tạo đối tượng FileOutputStream để ghi dữ liệu vào một đốit tượng mô tả file đại diện cho một kết nối đang tồn tại tới một file nào đó trong hệ thống.

### Ghi dữ liệu ra màn hình

* Để ghi dữ liệu ra màn hình, ta tạo đối tượng PrintWriter với tham số nhận vào là System.out.
* Sau đó sử dụng các phương thức như print(), printf(), println().
* Ví dụ:

// tạo đối tượng PrintWriter để đẩy dữ liệu ra màn hình console

PrintWriter printWriter = new PrintWriter(System.out, true);

// sử dụng như ghi dữ liệu ra màn hình trước nay

printWriter.println("Hello World!");

printWriter.println("Hey, who you are?");

printWriter.printf("My average grade is %5.2f", 3.26f);

* Nếu không để chế độ auto flush thì dữ liệu sẽ không tự động đẩy ra. Và bộ đệm cho từng dòng không được xóa với các phương thức printf, println, và format.

// tạo đối tượng PrintWriter để đẩy dữ liệu ra màn hình console

PrintWriter printWriter = new PrintWriter(System.out);

// sử dụng như ghi dữ liệu ra màn hình trước nay

printWriter.println("Hello World!");

printWriter.print("Hey, who you are?");

printWriter.flush(); // đẩy dữ liệu ra và xóa bộ đệm

printWriter.printf("\nMy average grade is %5.2f", 3.26f);

### Ghi dữ liệu ra file

* Các bước để ghi dữ liệu ra file:
  1. **Tạo file**: có sẵn hoặc sử dụng lớp File/FileWriter để tạo
  2. **Mở file**: thực hiện qua đối tượng của PrintWriter.
  3. **Ghi dữ liệu ra file**: sử dụng các phương thức như print(), printf(), println().
  4. **Đóng file**: sử dụng phương thức close() của lớp PrintWriter, FileWriter. Việc đóng file và các luồng dữ liệu phải được thực hiện theo thứ tự ngược với việc mở file/luồng dữ liệu.
* Khi đọc dữ liệu đã ghi ta cần phải đọc đúng thứ tự và kiểu như đã ghi trong file nếu không sẽ báo lỗi.
* Nếu bạn không đóng file vừa ghi với phương thức **close()**hoặc không sử dụng try-with resource thì dữ liệu sẽ không được lưu vào file.
* Ví dụ:

var filePath = "output.txt";

File file = new File(filePath);

FileWriter fileWriter = new FileWriter(file, true);

PrintWriter printWriter = new PrintWriter(fileWriter, true);

// sử dụng như ghi dữ liệu ra màn hình trước nay

printWriter.println("Hello World!");

printWriter.print("Hey, who you are?");

printWriter.flush(); // đẩy dữ liệu ra và xóa bộ đệm

printWriter.printf("\nMy average grade is %5.2f", 3.26f);

printWriter.close();

fileWriter.close();

Hello World!

Hey, who you are?

My average grade is 3.26

### Ví dụ minh họa

* **Ví dụ 1: ghi đè dữ liệu cũ trong file**.
* Với trường hợp này ta sẽ cho tham số thứ hai của FileWriter hoặc FileOuputStream là false.

private static void saveDataToFile(ArrayList<Student> students) throws IOException {

File file = new File(FILE\_NAME);

// cách 1: sử dụng FileOutputStream với tham số thứ hai là false hoặc để trống

// PrintWriter printWriter = new PrintWriter(new FileOutputStream(file, fase));

// cách 2: sử dụng đối tượng của FileWriter với tham số thứ hai false hoặc để trống

PrintWriter printWriter = new PrintWriter(new FileWriter(file, false));

// tiến hành ghi file

for (var student : students) {

printWriter.println(STR."\{student.getStudentId()},\{student.getFullName()},"

+ STR."\{student.getAddress()},\{student.getMajor()},\{student.getGpa()}");

}

printWriter.close();

}

* Code đầy đủ file Student.java:

package net.braniumacademy.chapter6.lesson12;

public class Student {

private String studentId;

private String fullName;

private String address;

private String major;

private float gpa;

public Student() {

}

public Student(String studentId, String fullName,

String address, String major, float gpa) {

this.setStudentId(studentId);

this.setFullName(fullName);

this.setAddress(address);

this.setMajor(major);

this.setGpa(gpa);

}

public String getStudentId() {

return studentId;

}

public void setStudentId(String studentId) {

this.studentId = studentId;

}

public String getFullName() {

return fullName;

}

public void setFullName(String fullName) {

this.fullName = fullName;

}

public String getAddress() {

return address;

}

public void setAddress(String address) {

this.address = address;

}

public String getMajor() {

return major;

}

public void setMajor(String major) {

this.major = major;

}

public float getGpa() {

return gpa;

}

public void setGpa(float gpa) {

this.gpa = gpa;

}

}

* Code đầy đủ file Lesson612.java:

package net.braniumacademy.chapter6.lesson12;

import java.io.\*;

import java.util.ArrayList;

import java.util.Scanner;

public class Lesson12 {

public static final String FILE\_NAME = "STUDENT.DAT";

public static void main(String[] args) throws IOException {

// tạo arraylist để chứa danh sách sinh viên

ArrayList<Student> students = new ArrayList<>();

// gọi method đọc file

readStudentFromFile(students);

showStudentInfo(students); // hiển thị thông tin lên màn hình

System.out.println("==> Nhập thông tin sinh viên mới: ");

addNewStudent(students);

System.out.println("Danh sách sinh viên sau khi thêm mới: ");

showStudentInfo(students);

// ghi danh sách sinh viên ra file

saveDataToFile(students);

}

private static void saveDataToFile(ArrayList<Student> students) throws IOException {

File file = new File(FILE\_NAME);

// cách 1:

// PrintWriter printWriter = new PrintWriter(new FileOutputStream(file));

// cách 2:

PrintWriter printWriter = new PrintWriter(new FileWriter(file));

for (var student : students) {

printWriter.println(STR."\{student.getStudentId()},\{student.getFullName()},"

+ STR."\{student.getAddress()},\{student.getMajor()},\{student.getGpa()}");

}

printWriter.close();

}

private static void addNewStudent(ArrayList<Student> students) {

Scanner keyboardReader = new Scanner(System.in);

System.out.println("Mã SV: ");

var studentId = keyboardReader.nextLine().trim().toUpperCase();

System.out.println("Họ và tên: ");

var fullName = keyboardReader.nextLine();

System.out.println("Địa chỉ(thành phố): ");

var address = keyboardReader.nextLine();

System.out.println("Chuyên ngành: ");

var major = keyboardReader.nextLine();

System.out.println("Điểm gpa: ");

var gpa = keyboardReader.nextFloat();

var student = new Student(studentId, fullName, address, major, gpa);

students.add(student);

}

/\*\*

\* Phương thức đọc ra thông tin sinh viên trong file.

\*

\* @param students danh sách sinh viên đọc được

\*/

private static void readStudentFromFile(ArrayList<Student> students) throws FileNotFoundException {

// tạo đối tượng file trỏ tới file cần đọc

File inputFile = new File(FILE\_NAME);

// tạo đối tượng fileReader để đọc dữ liệu từ file

Scanner fileReader = new Scanner(inputFile);

while (fileReader.hasNextLine()) {

var data = fileReader.nextLine().split(",");

if (data.length > 0) {

var gpa = Float.parseFloat(data[4]);

var student = new Student(data[0], data[1], data[2], data[3], gpa);

students.add(student);

}

}

fileReader.close(); // đóng luồng đọc file

}

/\*\*

\* Phương thức hiển thị thông tin sinh viên dạng bảng.

\*

\* @param students danh sách sinh viên cần hiển thị.

\*/

private static void showStudentInfo(ArrayList<Student> students) {

var line = "===========================";

System.out.println(STR."\{line} THÔNG TIN SINH VIÊN \{line}");

System.out.printf("%-5s%-10s%-25s%-15s%-10s%-10s\n",

"STT", "Mã SV", "Họ và tên", "Địa chỉ", "C.Ngành", "Điểm gpa");

var counter = 1;

for (var student : students) {

System.out.printf("%-5d%-10s%-25s%-15s%-10s%-10s\n",

counter++, student.getStudentId(), student.getFullName(),

student.getAddress(), student.getMajor(), student.getGpa());

}

}

}

* **Ví dụ 2: ghi thêm dữ liệu vào cuối file**.
* Ta cho tham số thứ hai của FileWriter hoặc FileOuputStream là true.

private static void saveDataToFile(ArrayList<Student> students) throws IOException {

File file = new File(FILE\_NAME);

// cách 1: sử dụng FileOutputStream

// PrintWriter printWriter = new PrintWriter(new FileOutputStream(file, true));

// cách 2: sử dụng FileWriter

PrintWriter printWriter = new PrintWriter(new FileWriter(file, true));

for (var student : students) {

printWriter.println(STR."\{student.getStudentId()},\{student.getFullName()},"

+ STR."\{student.getAddress()},\{student.getMajor()},\{student.getGpa()}");

}

printWriter.close();

}

* Kết quả: nội dung cũ trong file STUDENT.DAT được giữ nguyên và bổ sung dữ liệu mới vào cuối.

==> dữ liệu cũ

S1001,Lê Bá Nam,Quảng Ninh,CNTT,3.26

S1002,Hoàng Quốc Tuấn,Hồ Chí Minh,CNTT,3.54

S1003,Mai Thị Diệu Linh,Vũng Tàu,CNTT,3.71

S1004,Ngô Quang Trường,Bình Dương,CNTT,3.33

S1005,Hồ Hoài Anh,Quảng Bình,CNTT,3.12

S1006,Nguyễn Ngọc Mai,Đà Nẵng,CNTT,3.95

S1007,Nguyễn Thị Ngọc Linh,Hà Nội,CNTT,3.18

S1008,Trần Quang Khải,Bắc Giang,CNTT,3.29

S1009,Ma Thùy Dung,Thái Nguyên,CNTT,3.37

S1010,Lương Đình Vinh,Nghệ An,CNTT,3.59

S1011,Nguyễn Thị Loan,Hà Nội,CNTT,3.65

S1012,Nguyễn Như Hoa,Nghệ An,QTKD,3.27

<== ==> dữ liệu mới ghi vào file

S1001,Lê Bá Nam,Quảng Ninh,CNTT,3.26

S1002,Hoàng Quốc Tuấn,Hồ Chí Minh,CNTT,3.54

S1003,Mai Thị Diệu Linh,Vũng Tàu,CNTT,3.71

S1004,Ngô Quang Trường,Bình Dương,CNTT,3.33

S1005,Hồ Hoài Anh,Quảng Bình,CNTT,3.12

S1006,Nguyễn Ngọc Mai,Đà Nẵng,CNTT,3.95

S1007,Nguyễn Thị Ngọc Linh,Hà Nội,CNTT,3.18

S1008,Trần Quang Khải,Bắc Giang,CNTT,3.29

S1009,Ma Thùy Dung,Thái Nguyên,CNTT,3.37

S1010,Lương Đình Vinh,Nghệ An,CNTT,3.59

S1011,Nguyễn Thị Loan,Hà Nội,CNTT,3.65

S1012,Nguyễn Như Hoa,Nghệ An,QTKD,3.27

S1013,Ngô Văn Khải,Hải Dương,CNTT,3.22

**Bài 7.1. Lớp Date và SimpleDateFormat**

**Nội dung bài học**

1. Lớp Date
2. Định dạng thời gian với SimpleDateFormat
3. Các kí tự định dạng thời gian với SimpleDateFormat
4. Ví dụ minh họa
5. Bài tập thực hành

Video lý thuyết

Video minh họa

**Lớp Date**

* Lớp Date dùng để tạo đối tượng đại diện cho thời gian ngày giờ trong Java.
* Bạn có thể tạo đối tượng Date để đại diện cho thời gian hiện thời hoặc một thời điểm được chỉ định nào đó với độ chính xác tính theo mili giây.
* Lớp Date ta đang nói tới nằm trong gói java.util.
* Các phương thức thường dùng và mô tả của nó như sau:
  + Date(): Phương thức này tạo một đối tượng với thời gian là ngày giờ hiện tại của hệ thống.
  + Date(long milisec): Phương thức này tạo ra một đối tượng mới với thời gian cho trước trong tham số. Thời gian sẽ được tính theo mili giây với gốc thời gian là 0h00’00’’ ngày 1 tháng 1 năm 1970.
  + boolean after(Date date): So sánh xem ngày giờ hiện thời có muộn hơn ngày giờ cho trong tham số không. Nếu đúng thì return true và ngược lại return false.
  + boolean before(Date date): So sánh xem ngày giờ hiện thời có sớm hơn ngày giờ cho trong tham số hay không. Return true nếu đúng và false nếu ngược lại.
  + int compareTo(Date d ate): So sánh hai đối tượng ngày giờ xem chúng tương quan nhau như thế nào. Nếu đối tượng hiện thời sớm hơn đối tượng trong tham số thì trả về giá trị âm. Trả về 0 nếu hai đối tượng tương đương. Trả về giá trị dương nếu đối tượng hiện thời muộn hơn đối tượng trong tham số.
  + long getTime(): Trả về số mili giây của đối tượng ngày giờ hiện thời với mốc thời gian tính từ 1/1/1970, 00:00:00:000.
  + void setTime(long time): Thiết lập thời gian mới cho đối tượng hiện thời. Gốc thời gian được tính là từ 1/1/1970, 00:00:00:000.
* Ví dụ:

Date date = new Date();

System.out.println("Thời gian hiện tại là: " + date);

System.out.println("Thời gian tính theo mili giây: " + date.getTime());

// Kết quả:

Thời gian hiện tại là: Thu Sep 24 11:27:18 ICT 2020

Thời gian tính theo mili giây: 1600921638110

**Định dạng thời gian với SimpleDateFormat**

* Khi muốn chuyển đổi định dạng ngày giờ sang định dạng ta mong muốn, ta sử dụng lớp này.
* Ví dụ:

var format = "E dd/MM/yyyy, hh:mm:ss";

SimpleDateFormat dateFormat = new SimpleDateFormat(format);

Date date = new Date();

System.out.println("Thời gian sau khi định dạng: ");

System.out.println(dateFormat.format(date));

Thời gian sau khi định dạng:

Thu 24/09/2020, 12:33:50

**Các kí tự định dạng thời gian với SimpleDateFormat**

* Các kí tự sau đây được dùng để định dạng ngày giờ và thời gian theo mong muốn:

|  |  |  |
| --- | --- | --- |
| **Kí tự** | **Ý nghĩa** | **Ví dụ kết quả** |
| G | Kỉ nguyên | AD(Sau công nguyên) |
| y | Năm ở dạng 4 con số | 2025 |
| M | Tháng của năm | 10 |
| D | Ngày của năm | 235 |
| d | Ngày của tháng | 22 |
| h | Giờ ở định dạng 12 giờ(AM, PM) | 12 |
| H | Giờ ở định dạng 24 giờ | 21 |
| m | Phút của giờ | 59 |
| s | Số giây của phút | 50 |
| S | Số mili giây | 125 |
| E | Ngày trong tuần | Monday |
| F | Ngày của tuần trong tháng | 3 |
| W | Tuần của tháng | 2 |
| w | Tuần của năm | 41 |
| a | AM hoặc PM | AM |
| k | Giờ trong ngày(1-24) | 23 |
| K | Giờ của hệ 12 giờ trong ngày | 11 |
| z | Múi giờ | ICT |

* Ví dụ:

var format = "E dd/MM/yyyy, hh:mm:ss";

SimpleDateFormat dateFormat = new SimpleDateFormat(format);

Date date = new Date();

System.out.println("Thời gian sau khi định dạng: ");

System.out.println(dateFormat.format(date));

System.out.print("Ngày của năm: ");

SimpleDateFormat dayOfYear = new SimpleDateFormat("D");

System.out.println(dayOfYear.format(date));

System.out.print("Múi giờ: ");

SimpleDateFormat timeZone = new SimpleDateFormat("z");

System.out.println(timeZone.format(date));

// Kết quả:

Thời gian sau khi định dạng:

Thu 24/09/2020, 12:50:55

Ngày của năm: 268

Múi giờ: ICT

**Ví dụ minh họa**

* Ví dụ sau sẽ nhập vào ngày tháng năm sinh của một người và tính xem đến bây giờ họ đã bao nhiêu tuổi.

package lesson38;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.Scanner;

public class Example1 {

public static void main(String[] args) {

var format = "dd/MM/yyyy";

SimpleDateFormat dateFormat = new SimpleDateFormat(format);

var input = new Scanner(System.in);

System.out.println("Nhập vào ngày tháng năm sinh ở định dạng: dd/MM/yyyy");

var birthday = input.nextLine();

try {

Date myBirthday = dateFormat.parse(birthday);

var age = findAge(myBirthday.getTime());

System.out.println("Ngày sinh: " + dateFormat.format(myBirthday));

System.out.println("Ngày hiện tại: " + dateFormat.format(new Date()));

System.out.printf("Tuổi hiện tại: %5.2f", age);

} catch (ParseException e) {

e.printStackTrace();

}

}

/\*\*

\* tính tuổi của một người đến thời điểm hiện tại dựa vào ngày sinh

\*

\* @param time ngày sinh tính theo mili giây

\* @return số năm đã sống tính đến thời điểm chạy chương trình

\*/

private static float findAge(long time) {

var currentTime = new Date().getTime();

var hours = 365 \* 24 + 6;

var min = hours \* 60;

var sec = min \* 60;

var milisec = sec \* 1000L;

return 1.0f \* (currentTime - time) / (milisec);

}

}

Nhập vào ngày tháng năm sinh ở định dạng: dd/MM/yyyy

15/05/2008

Ngày sinh: 15/05/2008

Ngày hiện tại: 24/09/2020

Tuổi hiện tại: 12.36

**Bài 7.2. Trích xuất thông tin từ đối tượng lớp Date**

**Nội dung bài học**

1. Tìm hiểu lớp Calendar
2. Các bước để trích xuất thông tin từ đối tượng của Calendar
3. Ví dụ minh họa

Video lý thuyết

Video minh họa

**Tìm hiểu lớp Calendar**

* Lớp Calendar là một abstract class chứa các phương thức hỗ trợ việc chuyển đổi giữa các thể hiện cụ thể của thời gian và tập các trường có sẵn của nó như YEAR, MONTH, HOUR,….
* Một số trường và phương thức:

|  |  |
| --- | --- |
| **Trường/phương thức** | **Mô tả** |
| YEAR | Năm |
| MONTH | Tháng, tháng đầu tiên của năm là 0 |
| WEEK\_OF\_YEAR | Tuần của năm, bắt đầu từ 1 |
| WEEK\_OF\_MONTH | Tuần của tháng, bắt đầu từ 1 |
| DATE | Ngày của tháng, bắt đầu từ 1 |
| DAY\_OF\_MONTH | Ngày của tháng, bắt đầu từ 1 |
| DAY\_OF\_YEAR | Ngày của năm, bắt đầu từ 1 |
| DAY\_OF\_WEEK | Ngày của tuần, bắt đầu từ chủ nhật |
| AM\_PM | Chỉ định hậu tố AM trước 12h trưa và PM từ 12h trưa |
| HOUR | Giờ của buổi sáng và chiều, tức AM/PM bắt đầu từ 0 |
| HOUR\_OF\_DAY | Giờ của ngày, tức 24 giờ |
| MINUTE | Phút của giờ |
| SECOND | Giấy của phút |
| MILISECOND | Mili giây của một giây |
| SUNDAY, MONDAY,… | Chỉ định thứ cụ thể của tuần |
| JANUARY,…, DECEMBER | Chỉ định tháng cụ thể của năm |
| AM/PM | Chỉ định giờ buổi sáng(AM) hoặc buổi chiều(PM) |
| getInstance() | Trả về đối tượng Calendar đại diện cho thời gian hiện thời |
| setTime(Date date) | Thiết lập thời gian là một đối tượng Date |
| set(int field, int value) | Thiết lập giá trị value cho một trường nào đó |
| set(int year, int month, int date) | Thiết lập năm, tháng, ngày |
| set(int year, int month, int date, int hourOfDay, int minute) | Thiết lập năm, tháng, ngày, giờ, phút |
| setTimeZone(TimeZone value) | Thiết lập múi giờ |
| setFirstDayOfWeek(int value) | Thiết lập ngày đầu tiên của tuần |
| setTimeInMilis(long value) | Thiết lập thời gian ở mili giây |
| add(int field, int amount) | Cộng hoặc trừ một lượng amount vào trường thời gian nào đó |
| clear() | Đưa giá trị các trường thời gian về mặc định |
| clear(int field) | Đưa giá trị của một trường xác định về mặc định |
| compareTo(Calendar other) | So sánh giá trị thời gian mili giây của hai calendar |
| get(int field) | Lấy giá trị của một trường thời gian nào đó |
| getTime() | Trả về đối tượng Date đại diện cho Calendar hiện thời |
| getTimeInMilis() | Lấy giá trị thời gian ở mili giây |
| getTimeZone() | Lấy múi giờ của đối tượng Calendar hiện thời |

**Các bước để trích xuất thông tin từ đối tượng của Calendar**

* Các bước để trích xuất thông tin của một đối tượng lớp Date như sau:
  + Tạo đối tượng của lớp Date
  + Tạo đối tượng của lớp Calendar: Calendar.getInstance()
  + Thiết lập các trường giá trị mong muốn qua các phương thức setX, ví dụ setTime(date)
  + Lấy giá trị của các trường ra thông qua các phương thức get(field) hoặc getX()
* Ví dụ:

var format = "dd/MM/yyyy";

SimpleDateFormat dateFormat = new SimpleDateFormat(format);

var dateStr = "27/04/1999";

Date date = dateFormat.parse(dateStr);

Calendar calendar = Calendar.getInstance();

calendar.setTime(date);

// đọc các trường giá trị mong muốn ra

System.out.println("Ngày: " + calendar.get(Calendar.DAY\_OF\_MONTH));

System.out.println("Tháng" + calendar.get(Calendar.MONTH));

System.out.println("Năm: " + calendar.get(Calendar.YEAR));

System.out.println("Ngày của tuần: " + calendar.get(Calendar.DAY\_OF\_WEEK));

System.out.println("Múi giờ: " + calendar.getTimeZone().getID());

Ngày: 27

Tháng3

Năm: 1999

Ngày của tuần: 3

Múi giờ: Asia/Bangkok

**Ví dụ minh họa**

package lesson39;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Calendar;

import java.util.Date;

public class Bai39 {

public static void main(String[] args) throws ParseException {

var format = "dd/MM/yyyy";

SimpleDateFormat dateFormat = new SimpleDateFormat(format);

var dateStr = "27/04/1999";

Date date = dateFormat.parse(dateStr);

Calendar calendar = Calendar.getInstance();

calendar.setTime(date);

// đọc các trường giá trị mong muốn ra

System.out.println("Ngày: " + calendar.get(Calendar.DAY\_OF\_MONTH));

System.out.println("Tháng" + calendar.get(Calendar.MONTH));

System.out.println("Năm: " + calendar.get(Calendar.YEAR));

System.out.println("Ngày của tuần: " + calendar.get(Calendar.DAY\_OF\_WEEK));

System.out.println("Múi giờ: " + calendar.getTimeZone().getID());

}

}

Ngày: 27

Tháng3

Năm: 1999

Ngày của tuần: 3

Múi giờ: Asia/Bangkok

**Bài 7.3. Lớp LocalDate**

**Nội dung bài học**

1. Lớp LocalDate
2. Các phương thức thường dùng
3. Các phương thức getX
4. Các phương thức kiểm tra
5. Các phương thức thay đổi trường giá trị ngày tháng năm
6. Các phương thức tạo lập đối tượng LocalDate
7. Ví dụ minh họa

Video lý thuyết

Video minh họa

**Lớp LocalDate**

* Là một lớp final, dùng để tạo đối tượng biểu đạt thông tin ngày tháng năm.
* Lớp này không chứa múi giờ do đó không áp dụng với định dạng múi giờ nếu không sẽ xảy ra ngoại lệ.
* Lớp LocalDate nằm trong gói java.time

**Các phương thức thường dùng**

* **adjustInto(Temporal tem):** Thiết lập một đối tượng thời gian với cùng thời gian như trong tham số.
* **atTime(int hour, int minute):** Kết hợp ngày hiện thời với thời gian giờ phút để tạo một đối tượng LocalDateTime.
* **atTime(int hour, int minute, int second):**Kết hợp ngày hiện thời với thời gian giờ phút giây để tạo một đối tượng LocalDateTime.
* **atTime(int hour, int minute, int second, int nanoSec):** Kết hợp ngày hiện thời với thời gian giờ phút giây và nono giây để tạo một đối tượng LocalDateTime.
* **atTime(LocalTime time):** Kết hợp ngày hiện thời với thời gian cho trong tham số để tạo một đối tượng LocalDateTime.
* **compareTo(ChronoLocalDate other):** So sánh ngày hiện thời với một ngày nào đó.
* **format(DateTimeFormatter formatter):** Định dạng ngày hiện thời với formatter cho trước.
* **from(TemporalAccessor tem):** Tạo một đối tượng LocalDate từ tham số nhận được.

**Các phương thức getX**

* **getDayOfMonth():** Lấy ngày của tháng trong đối tượng hiện thời.
* **getDayOfWeek():** Lấy ngày của tuần trong đối tượng hiện thời.
* **getDayOfYear():** Lấy ngày của năm trong đối tượng hiện thời.
* **getEra():** Lấy kỉ nguyên của đối tượng hiện thời.
* **getMonth():** Lấy tháng của năm, trả về enum Month.
* **getMonthValue():** Lấy giá trị số của tháng của đối tượng hiện thời, từ 1-12.
* **getYear():** Lấy năm của đối tượng hiện thời.

**Các phương thứ kiểm tra**

* **isAfter(ChronoLocalDate other):** Kiểm tra xem ngày hiện thời có muộn hơn ngày trong tham số hay không.
* **isBefore(ChronoLocalDate other):** Kiểm tra xem ngày hiện thời có sớm hơn ngày trong tham số hay không.
* **isLeapYear():** Kiểm tra xem năm hiện tại có phải năm nhuận không.

**Các phương thức thay đổi giá trị của các trường ngày tháng năm**

* **minusMonths(long n):** Trả về bản copy của đối tượng hiện thời giảm thời gian đi n tháng.
* **minusDays(long n):** Trả về bản copy của đối tượng hiện thời giảm thời gian đi n ngày.
* **minusWeeks(long n):** Trả về bản copy của đối tượng hiện thời giảm thời gian đi n tuần.
* **minusYears(long n):** Trả về bản copy của đối tượng hiện thời giảm thời gian đi n năm.
* **plusYears(long n):** Trả về bản sao của đối tượng hiện thời với số năm được thêm vào.
* **plusMonths(long n):** Trả về bản sao của đối tượng hiện thời với số tháng được thêm vào.
* **plusWeeks(long n):** Trả về bản sao của đối tượng hiện thời với số tuần được thêm vào.
* **plusDays(long n):** Trả về bản sao của đối tượng hiện thời với số ngày được thêm vào.

**Các phương thức tạo lập đối tượng LocalDate**

* **now():** Trả về đối tượng LocalDate đại diện cho ngày hiện thời theo thời gian hệ thống.
* **of(int year, int month, int day):** Tạo một thể hiện của LocalDate từ năm, tháng, ngày cho trước.
* **of(int year, Month month, int day):** Tạo một thể hiện của LocalDate từ năm, tháng, ngày cho trước.
* **of(int year, int dayOfYear):** Tạo một thể hiện của LocalDate từ năm và ngày của năm.
* **parse(CharSequence text):** Tạo đối tượng LocalDate từ một chuỗi với định dạng cho trước.
* **parse(CharSequence text, DateTimeFormatter formatter):** Tạo đối tượng LocalDate từ một chuỗi với định dạng trong formatter.

Chi tiết về các phương thức trên và các phương thức khác tại: [**ĐÂY**](https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/time/LocalDate.html)

**Ví dụ minh họa**

package lesson40;

import java.time.LocalDate;

public class Bai40 {

public static void main(String[] args) {

LocalDate today = LocalDate.now(); // ngày hiện thời

// ngày 25/10/2025

LocalDate date1 = LocalDate.of(2025, 10, 25);

System.out.println("Thông tin ngày hiện thời: " + today);

System.out.println("Thông tin ngày trong date1: " + date1);

// thay đổi vài năm thông tin ngày hiện thời

var hundredYearsLatter = today.plusYears(100);

System.out.println("100 năm sau: " + hundredYearsLatter);

}

}

Thông tin ngày hiện thời: 2020-09-25

Thông tin ngày trong date1: 2025-10-25

100 năm sau: 2120-09-25

**Bài 7.4. Lớp LocalTime**

**Nội dung bài học**

1. Lớp LocalTime
2. Các trường hằng số
3. Các phương thức get
4. Các phương thức so sánh
5. Các phương thức thay đổi giá trị các trường thời gian
6. Các phương thức tạo lập đối tượng của LocalTime
7. Các phương thức khác
8. Ví dụ minh họa

Video lý thuyết

Video minh họa

**Lớp LocalTime**

* Là một lớp final dùng để tạo các đối tượng đại diện cho thời gian về giờ phút giây.
* Thời gian trong đối tượng này không chứa múi giờ nên không áp dụng lấy múi giờ được.
* Độ chính xác thời gian của lớp này tính theo nano giây(9 chữ số).
* Lớp LocalTime nằm trong gói java.time.

**Các trường hằng số**

* **MAX**: Giá trị tối đa được hỗ trợ bởi LocalTime: 23:59:59.999999999.
* **MIN**: Giá trị tối thiểu được hỗ trợ bởi LocalTime: 00:00.
* **MIDNIGHT**: Thời gian giữa đêm. Bắt đầu của ngày: 00:00.
* **NOON**: Giữa trưa: 12:00.

**Các phương thức get**

* **get(TemporalField field):** Lấy giá trị của một trường xác định và trả về giá trị kiểu int.
* **getHour():** Lấy giờ.
* **getMinute():** Lấy phút.
* **getSecond():** Lấy giây.
* **getNano():** Lấy nano giây

**Các phương thức so sánh**

* **compareTo(LocalTime time):** So sánh với đối tượng thời gian khác.
* **isAfter(LocalTime other):** Kiểm tra xem thời gian hiện tại có muộn hơn thời gian trong tham số hay không.
* **isBefore(LocalTime other):** Kiểm tra xem thời gian hiện tại có sớm hơn thời gian trong tham số hay không.

**Các phương thức thay đổi giá trị các trường thời gian**

* **minusHours(long hour):** Tạo bản sao của đối tượng hiện thời với số giờ được trừ đi là hour.
* **minusMinutes(long min):** Tạo bản sao của đối tượng hiện thời với số phút được trừ đi là min.
* **minusSeconds(long sec):** Tạo bản sao của đối tượng hiện thời với số giây được trừ đi một lượng sec.
* **minusNanos(long nano):** Tạo bản sao của đối tượng hiện thời với thời gian được trừ đi một lượng nano giây.
* **plusHours(long hour):** Tạo một bản sao của đối tượng hiện thời với số giờ được cộng thêm một lượng hour.
* **plusMinutes(long min):** Tạo một bản sao của đối tượng hiện thời với số phút được cộng thêm một lượng min.
* **plusSeconds(long sec):** Tạo một bản sao của đối tượng hiện thời với số giây được cộng thêm một lượng sec.
* **plusNanos(long nano):** Tạo một bản sao của đối tượng hiện thời với số nano giây được cộng thêm một lượng nano.

**Các phương thức tạo lập đối tượng của LocalTime**

* **now():** Tạo đối tượng LocalTime lấy thời gian từ hệ thống.
* **of(int hour, int min):** Tạo đối tượng LocalTime với thời gian giờ phút cho trước.
* **of(int hour, int min, int sec):** Tạo đối tượng LocalTime với thời gian giờ phút giây cho trước.
* **of(int hour, int min, int sec, int nano):** Tạo đối tượng LocalTime với thời gian giờ phút giây, nano giây cho trước.
* **parse(CharSequence text):** Tạo đối tượng LocalTime từ thời gian trong String ví dụ 10:15.
* **parse(CharSequence text, DateTimeFormatter formatter):** Tạo đối tượng LocalTime từ thời gian trong String với định dạng cho trước trong formatter.

**Các phương thức khác**

* **atDate(LocalDate date):** Kết hợp với đối tượng của LocalDate tạo đối tượng LocalDateTime.
* **format(DateTimeFormatter formatter):** Định dạng thời gian hiện thời sử dụng đối tượng DateTimeFormatter cho trước.
* **from(TemporalAccessor tem):** Tạo đối tượng thời gian từ tham số.
* **toNanoOfDay():** Trích xuất thời gian hiện tại ra dạng nano giây.
* **toSecondOfDay():** Trích xuất thời gian hiện tại ra ở dạng giây.

Chi tiết về các phương thức trên và các phương thức khác bạn tham khảo tại [**ĐÂY**](https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/time/LocalTime.html).

**Ví dụ minh họa**

package lesson41;

import java.time.LocalTime;

public class Bai41{

public static void main(String[] args) {

LocalTime now = LocalTime.

();

LocalTime other = LocalTime.

(21, 15, 50);

LocalTime myLocalTime = LocalTime.

("23:54:56");

System.

.println("Thời gian hiện thời: " + now);

System.

.println("Thời gian trong other: " + other);

System.

.println("Thời gian trong myLocalTime: " + myLocalTime);

myLocalTime = myLocalTime.minusHours(10);

myLocalTime = myLocalTime.minusMinutes(50);

System.

.println("Thời gian mới trong myLocalTime: " + myLocalTime);

}

}

Thời gian hiện thời: 16:03:29.481015100

Thời gian trong other: 21:15:50

Thời gian trong myLocalTime: 23:54:56

Thời gian mới trong myLocalTime: 13:04:56

**Bài 7.5. Lớp LocalDateTime**

**Nội dung bài học**

1. Lớp LocalDateTime
2. Các trường
3. Các phương thức
4. Ví dụ minh họa

Video lý thuyết

Video minh họa

**Lớp LocalDateTime**

* Là một lớp final để tạo đối tượng đại diện cho thời gian ngày giờ. Tức bao gồm cả ngày tháng năm, giờ phút giây.
* Lớp này không hỗ trợ múi giờ nên không áp dụng để lấy múi giờ.
* Độ chính xác thời gian được hỗ trợ bởi lớp này là nano giây.
* Lớp LocalDateTime nằm trong gói java.time.

**Các trường**

* **MAX**: Giá trị lớn nhất được hỗ trợ bởi lớp LocalDateTime, tức 999999999-12-31T23:59:59.999999999.
* **MIN**: Giá trị nhỏ nhất được hỗ trợ bởi lớp LocalDateTime, tức -999999999-01-01T00:00:00.

**Các phương thức thường dùng**

* **format(DateTimeFormatter formatter):** Trả về chuỗi định dạng thời gian với định dạng cho trước.
* **from(TemporalAccessor tem):** Tạo đối tượng LocalDateTime từ đối tượng cho trước.
* Các phương thức **getX()** của hai lớp LocalDate, LocalTime.
* Các phương thức kiểm tra đối sánh hai đối tượng LocalDateTime như trong lớp LocalDate, LocalTime.
* Các phương thức **minusX()**, **plusX()** của cả hai lớp LocalDate, LocalTime.
* **now():** Trả về đối tượng LocalDateTime với ngày giờ lấy từ hệ thống.
* **of(int year, int month, int dayOfMonth, int hour, int minute):** Tạo đối tượng LocalDateTime từ các tham số cho trước.
* **of(int year, int month, int dayOfMonth, int hour, int minute, int second):** Tạo đối tượng LocalDateTime từ các tham số cho trước.
* **of(int year, int month, int dayOfMonth, int hour, int minute, int second, int nanoSec):** Tạo đối tượng LocalDateTime từ các tham số cho trước.
* **of(int year, Month month, int dayOfMonth, int hour, int minute):** Tạo đối tượng LocalDateTime từ các tham số cho trước.
* **of(int year, Month month, int dayOfMonth, int hour, int minute, int second):** Tạo đối tượng LocalDateTime từ các tham số cho trước.
* **of(int year, Month month, int dayOfMonth, int hour, int minute, int second, int nanoSec):** Tạo đối tượng LocalDateTime từ các tham số cho trước.
* **parse(CharSequence text):** Chuyển đổi một chuỗi ngày giờ thành đối tượng của LocalDateTime tương ứng. Ví dụ 2005-01-31T01:25:36.
* **parse(CharSequence text, DateTimeFormatter formatter):** Chuyển đổi một chuỗi ngày giờ thành đối tượng của LocalDateTime tương ứng với định dạng cho trước.

Thông tin chi tiết về các phương thức trên và các phương thức khác bạn tham khảo tại [**ĐÂY**](https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/time/LocalDateTime.html).

**Ví dụ minh họa**

package lesson42;

import java.time.LocalDateTime;

public class Bai42 {

public static void main(String[] args) {

// lấy thời gian hiện thời

LocalDateTime now = LocalDateTime.now();

// tạo đối tượng đại diện cho thời điểm cụ thể:

LocalDateTime myBirthday = LocalDateTime.of(1999, 5, 20, 15, 35, 19);

LocalDateTime otherDateTime = LocalDateTime.parse("2005-12-10T20:15:36");

// hiển thị thông tin ngày giờ và thời gian ra

System.out.println("Thời gian hiện tại: " + now);

System.out.println("Sinh nhật: " + myBirthday);

System.out.println("Khác: " + otherDateTime);

// thay đổi:

var newOther = otherDateTime.plusMonths(5);

newOther = newOther.plusHours(20);

newOther = newOther.plusMinutes(50);

System.out.println("Sau khi thay đổi: " + newOther);

}

}

Thời gian hiện tại: 2020-09-25T16:36:04.835593

Sinh nhật: 1999-05-20T15:35:19

Khác: 2005-12-10T20:15:36

Sau khi thay đổi: 2006-05-11T17:05:36

# Bài 7.6. Lớp DateTimeFormatter

### Nội dung bài học

1. [Lớp DateTimeFormatter](https://braniumacademy.net/courses/java/lessons/java-datetimeformatter/#s1)
2. [Các mẫu kí tự định dạng ngày giờ và thời gian](https://braniumacademy.net/courses/java/lessons/java-datetimeformatter/#s2)
3. [Ví dụ minh họa](https://braniumacademy.net/courses/java/lessons/java-datetimeformatter/#s3)

Video lý thuyết

Video minh họa

### Lớp DateTimeFormatter

* Là một lớp dùng để định dạng các đối tượng ngày giờ khi muốn tạo mới hoặc xuất dữ liệu ngày giờ.
* Lớp này cung cấp các dạng thực thi phổ biến với các cách khác nhau:
  + Sử dụng các hằng số được định nghĩa sẵn, ví dụ ISO\_LOCAL\_DATE
  + Sử dụng các kí tự định dạng mẫu như dd-MMM-uuuu
  + Sử dụng các style đã bản địa hóa như long hay medium
* Các formatter phức tạp hơn được cung cấp trong lớp DateTimeFormatterBuilder.
* Trong nội dung bài học này ta chỉ tập trung vào việc áp dụng DateTimeFormatter với các kí tự định dạng mẫu.
* Các cách khác bạn tự chủ động tra tài liệu trên internet.
* Các phương thức thường dùng:
  + **format(TemporalAccessor temp):** Định dạng đối tượng ngày giờ khác null trong tham số với formatter hiện tại. Trả về một String kết quả khác null sau khi định dạng.
  + **ofPattern(String parttern):** Tạo đối tượng của lớp với mẫu định dạng khác null cho trước. Trả về một đối tượng DateTimeFormatter khác null.
  + **parse(CharSequence text):** Chuyển đổi text khác null thành đối tượng date-time tương ứng.

### Các mẫu kí tự định dạng ngày giờ và thời gian

|  |  |  |
| --- | --- | --- |
| **Kí tự** | **Ý nghĩa** | **Ví dụ** |
| G | Kỉ nguyên | AD, A |
| u | Năm | 2024, 24 |
| y | Năm của kỉ nguyên | 2025, 25 |
| D | Ngày của năm | 321 |
| M/L | Tháng của năm | 7, 07, Jul, July, J |
| d | Ngày của tháng | 30 |
| q/Q | Quý của năm | 3, 03, Q3, 3rd quarter |
| W | Tuần của tháng | 1 |
| E | Ngày của tuần | Tue, Tuesday |
| a | Buổi của ngày, tức sáng hay chiều | AM, PM |
| h | Giờ đồng hồ 12h trong ngày(1-12) | 12 |
| K | Giờ AM, PM(0-11) | 0 |
| k | Giờ đồng hồ trong ngày(1-24) | 24 |
| H | Giờ trong ngày(0-23) | 23 |
| m | Phút của một giờ | 31 |
| s | Giây của một phút | 52 |
| S | Mili giây | 585 |
| A | Mili giây của ngày | 1520 |
| n | Nano giây của một giây | 987654321 |
| N | Nano giây của ngày | 1234000000 |
| V | Mã múi giờ | Asia/Bangkok |
| v | Tên múi giờ generic | Parcific Time |
| z | Tên múi giờ | Parcific Standard Time |
| O | Múi giờ địa phương | GMT+7 |
| X/x/Z | Chênh lệch so với múi giờ số 0 | +07, +0700 |

Chi tiết về các thành phần khác của lớp bạn có thể tham khảo tại [**ĐÂY**](https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/time/format/DateTimeFormatter.html).

### Ví dụ minh họa

**Ví dụ 1: chuyển đổi ngày riêng và giờ riêng.**

package lesson43;

import java.time.LocalDateTime;

public class Bai43 {

public static void main(String[] args) {

// định dạng ngày giờ

DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MMMM/uuuu");

LocalDateTime dateTime = LocalDateTime.now();

System.out.println(formatter.format(dateTime));

// định dạng thời gian

formatter = DateTimeFormatter.ofPattern("HH:mm:ss:nnnnnnnnn");

LocalTime localTime = LocalTime.now();

System.out.println(localTime.format(formatter));

// định dạng múi giờ

formatter = DateTimeFormatter.ofPattern("HH:mm:ss O");

ZonedDateTime zonedDateTime = ZonedDateTime.now();

System.out.println(zonedDateTime.format(formatter));

}

}

// kết quả:

25/September/2020

11:55:29:229554200

11:55:29 GMT+7

* **Ví dụ 2: chuyển đổi cả ngày và giờ.**

DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/uuuu HH:mm:ss:SSS");

LocalDateTime myDateTime = LocalDateTime.parse("12/12/2012 21:15:27:699", formatter);

System.out.println(myDateTime);

// kết quả:

2012-12-12T21:15:27.699

* **Ví dụ 3**:**tìm số ngày chênh lệch giữa hai thời điểm.**

public class Test {

public static void main(String[] args) {

DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy");

LocalDate date1 = LocalDate.parse("01/06/2025", formatter);

LocalDate date2 = LocalDate.parse("25/05/2025", formatter);

var numOfDay = ChronoUnit.DAYS.between(date2, date1); // date2 - date1

System.out.println("Số ngày từ " + date1.format(formatter)

+ " đến " + date2.format(formatter) + " là: " + numOfDay);

}

}

Số ngày từ 01/06/2025 đến 25/05/2025 là: 7

* Ở trên ta sử dụng ChronoUnit.DAYS.between(date2, date1) để tính độ lệch ngày giữa hai LocalDate cho trước.
* Nếu ta đổi thứ tự date2 và date1 thì kết quả có thể âm. Do đó để chính xác ta lấy trị tuyệt đối của nó:

public class Test {

public static void main(String[] args) {

DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy");

LocalDate date1 = LocalDate.parse("01/01/2025", formatter);

LocalDate date2 = LocalDate.parse("25/05/2025", formatter);

var numOfDay = Math.abs(ChronoUnit.DAYS.between(date1, date2)); // date2 - date1

System.out.println("Số ngày từ " + date1.format(formatter)

+ " đến " + date2.format(formatter) + " là: " + numOfDay);

}

}

Số ngày từ 01/01/2025 đến 25/05/2025 là: 144

* **Ví dụ 4: Tìm ngày kết thúc khóa học.**
  + Yêu cầu: Mỗi khóa học của Branium Academy có ngày bắt đầu và ngày kết thúc. Cho biết ngày bắt đầu của khóa học và thời gian học tính theo tuần, bạn hãy cho biết ngày kết thúc của khóa học?

import java.time.LocalDate;

import java.time.format.DateTimeFormatter;

import java.util.Scanner;

public class EndDate {

private static String findEndDate(String start, int duration) {

var formatStr = "dd/MM/yyyy";

DateTimeFormatter formatter = DateTimeFormatter.ofPattern(formatStr);

LocalDate startDate = LocalDate.parse(start, formatter);

LocalDate endDate = startDate.plusWeeks(duration);

return formatter.format(endDate);

}

public static void main(String[] args) {

var scanner = new Scanner(System.in);

System.out.print("Ngày bắt đầu: ");

String startDateStr = scanner.nextLine();

System.out.print("Số tuần học: ");

int duration = scanner.nextInt();

System.out.println("=============================");

System.out.println("Ngày bắt đầu: " + startDateStr);

System.out.println("Ngày kết thúc: " + findEndDate(startDateStr, duration));

}

}

Ngày bắt đầu: 01/01/2025

**Số tuần học: 30**

**=============================**

Ngày bắt đầu: 01/01/2025

Ngày kết thúc: 30/07/2025

# Bài 8.1. Lớp cha, lớp con

### Nội dung bài học

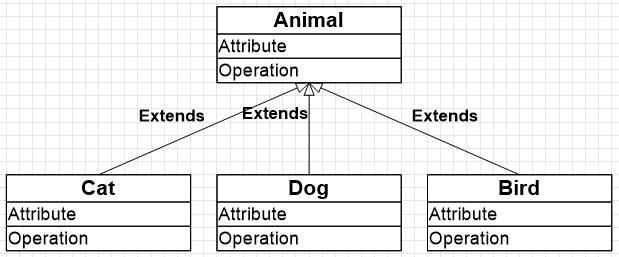
1. [Các khái niệm](https://braniumacademy.net/courses/java/lessons/java-sub-class/#s1)
2. [Những gì được kế thừa?](https://braniumacademy.net/courses/java/lessons/java-sub-class/#s2)
3. [Java chỉ hỗ trợ kế thừa đơn](https://braniumacademy.net/courses/java/lessons/java-sub-class/#s3)
4. [Khai báo kế thừa](https://braniumacademy.net/courses/java/lessons/java-sub-class/#s4)
5. [Ví dụ minh họa](https://braniumacademy.net/courses/java/lessons/java-sub-class/#s5)
6. [Bài tập thực hành](https://braniumacademy.net/courses/java/lessons/java-sub-class/#s6)

Video lý thuyết

Video minh họa

### Các khái niệm

* Đây là đặc trưng thứ hai của ngôn ngữ lập trình hướng đối tượng Java(OOP).
* Kế thừa(**inheritance**) là một kĩ thuật trong Java cho phép một lớp kế thừa lại những tính năng(trường, phương thức) của một lớp khác.
* Ta còn gọi tính chất kế thừa là mở rộng từ một lớp(extend).
* Liên quan đến khái niệm kế thừa, ta có các khái niệm sau:
  + **Lớp cha**(super class, base class, father class): là các lớp được lớp khác kế thừa.
  + **Lớp con**(sub class, derived class, child class): là các lớp đi kế thừa lớp khác. Lớp con có thể bổ sung thêm các trường và các phương thức của riêng nó.
  + **Tính tái sử dụng(reusability):** kế thừa hỗ trợ tái sử dụng lại code. Thay vì tạo một lớp từ đầu với tất cả các trường, phương thức. Ta kế thừa từ một lớp đã có nào đó có chứa các tính năng ta cần. Do đó tái sử dụng lại các trường, phương thức của lớp đã có và chỉ cần bổ sung thêm các thông tin riêng.
  + **Tính tổng quát hóa:** lớp cha càng ở trên cao thì càng tổng quát, chứa càng ít thông tin nhưng là những thông tin chung nhất có ở tất cả các lớp con của nó.
  + **Tính cụ thể hóa:** lớp con càng ở phía dưới càng có những đặc trưng riêng không có ở lớp cha.
  + **Quan hệ là một(is a):** lớp con là một ví dụ khác cụ thể hơn của lớp cha. Ví dụ: mèo là động vật, cá là động vật, người là động vật,…Do đó ta có thể cho biến của lớp cha tham chiếu tới đối tượng của mọi lớp con của nó. Hoặc ta có thể trả về đối tượng của lớp con trong các phương thức có kiểu trả về của lớp cha(Ví dụ phía dưới).
* Ví dụ hình ảnh kế thừa: Lớp Cat, Dog, Bird kế thừa từ Animal.

[](https://braniumacademy.net/wp-content/uploads/2020/08/vi-du-ke-thua-UML.png)

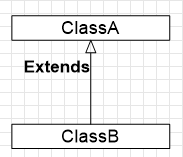
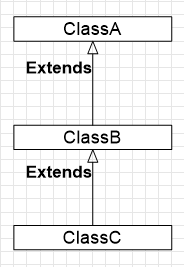
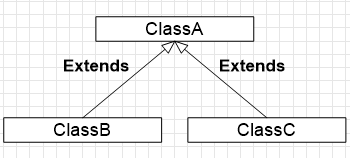
* **Mục đích của kế thừa:** Tái sử dụng lại code của các lớp khác. Do đó tiết kiệm thời gian, chi phí khi làm một sản phẩm nào đó.

### Những gì được kế thừa?

* Khi một lớp con nào đó kế thừa một lớp cha, lớp con sẽ kế thừa tất cả các trường và phương thức có access modifier public và protected của lớp cha.
* Khi đó các thành phần được kế thừa sẽ được truy cập một cách trực tiếp từ trong lớp con mà không gặp bất cứ trở ngại nào. Giống như là lớp con tự định nghĩa ra các thành phần đó vậy.
* Các thành phần có access modifier là private default access của lớp cha chỉ có thể được truy cập trực tiếp từ lớp con nếu lớp con và lớp cha nằm trong cùng gói.
* Các thành phần private của lớp cha không được kế thừa bởi lớp con. Do đó đảm bảo tính đóng gói dữ liệu, thứ gì private thì của lớp nào thì chỉ có lớp đó mới có thể sử dụng trực tiếp.
* Các phương thức khởi tạo của lớp cha không được kế thừa bởi lớp con. Nhưng lớp con có thể truy cập đến phương thức khởi tạo của lớp cha qua keyword super.
* Vậy là ta đã rõ những gì được sử dụng lại ở lớp con trong mối quan hệ kế thừa của Java.

### Java chỉ hỗ trợ kế thừa đơn

* Java chỉ hỗ trợ đơn kế thừa với lớp, tức là một lớp chỉ có thể có một lớp cha trực tiếp. Giống như mỗi người chỉ có 1 cha trực tiếp sinh ra mình vậy.
* Nếu muốn đa kế thừa, tức là một lớp kế thừa từ nhiều cha khác nhau thì ta sử dụng interface. Nội dung này sẽ trình bày cụ thể trong bài về interface.
* Vậy tại sao lại chỉ hỗ trợ đơn kế thừa? Lý do là khi đa kế thừa có thể xảy ra nhiều vấn đề phức tạp rối rắm. Chúng ta nên đơn giản hóa vấn đề chứ không nên phức tạp hóa nó. Ví dụ nếu đa kế thừa sẽ có thể xảy ra trường hợp cả hai lớp cha có cùng phương thức, cùng tham số, cùng kiểu trả về và thực hiện chức năng giống nhau->dư thừa và khó mà nhận diện được phương thức đang được sử dụng là từ lớp cha nào.
* Các dạng kế thừa với lớp:
  + **Kế thừa đơn:** 1 lớp con chỉ kế thừa từ 1 lớp cha và 1 lớp cha chỉ có 1 lớp con.
  + **Kế thừa nhiều bậc**(thế hệ): một lớp con có thể có nhiều lớp cha, ông, cụ, kị. Đồng thời lớp con đó lại là lớp cha, ông, cụ, kị của các lớp con cháu khác
  + **Kế thừa phân cấp**(phân nhánh): một lớp có thể là cha của nhiều hơn 1 lớp con.

[](https://braniumacademy.net/wp-content/uploads/2020/08/vi-du-single_inheritance.png)Ví dụ kế thừa đơn[](https://braniumacademy.net/wp-content/uploads/2020/08/vi-du-multilevel_inheritance.png)Ví dụ kế thừa nhiều bậc[](https://braniumacademy.net/wp-content/uploads/2020/08/vi-du-hierachical_inheritance.png)Ví dụ kế thừa phân cấp

### Khai báo kế thừa

* Sử dụng keyword extends để thể hiện lớp A kế thừa B.
* Tổng quát:

**access non-access class A extends B {**

**// ...**

**}**

* Trong đó:
  + A là lớp con hay lớp dẫn xuất, lớp đi kế thừa lớp khác. Nó đứng trước keyword extends
  + Keyword extends là bắt buộc để thể hiện quan hệ kế thừa.
  + Sau keyword extends là B, chính là lớp cha, lớp cơ sở, lớp cho lớp khác kế thừa.
  + Trong thân của lớp con có thể chứa những khai báo riêng của chính nó mà không có ở lớp cha.

### Ví dụ minh họa

* Ví dụ kế thừa đơn:

package lesson44;

public class Animal {

private String name;

private float weight;

public void move() {

// do something...

}

public void eat() {

// do something...

}

}

class Cat extends Animal {

private String color;

private String favoriteFood;

public void catchMouse() {

// do something...

}

public void washFace() {

// do something...

}

}

* Ví dụ kế thừa nhiều thế hệ(bậc):

package lesson44;

public class Person {

private String id;

private String fullName;

private void move() {

// ...

}

}

class Student extends Person {

private String studentId;

private float avgGrade;

public void doHomeWork() {

//...

}

}

class GraduatedStudent extends Student {

private String level;

private float startingSalary;

private int graduatedYear;

public void workInOffice() {

//...

}

}

* Ví dụ kế thừa phân cấp:

package lesson44;

public class Animal {

private String name;

private float weight;

public void move() {

// do something...

}

public void eat() {

// do something...

}

}

class Cat extends Animal {

private String color;

private String favoriteFood;

public void catchMouse() {

// do something...

}

public void washFace() {

// do something...

}

}

public class Dog extends Animal {

private String species;

public void bark() {

//...

}

public void biteOther() {

//...

}

}

* Ví dụ sử dụng kế thừa trả về đối tượng các lớp con trong phương thức có kiểu trả về là lớp cha. Sau đây là các lớp trong quan hệ kế thừa:

public class Person {

private String id;

private String fullName;

private void move() {

// ...

}

}

class Student extends Person {

private String studentId;

private float avgGrade;

public void doHomeWork() {

//...

}

}

class GraduatedStudent extends Student {

private String level;

private float startingSalary;

private int graduatedYear;

public void workInOffice() {

System.out.println("He is coding a function...");

}

}

class Teacher extends Person {

private String teacherId;

private long salary;

private float experience;

public void teach(String subject) {

System.out.println("Teacher is teaching " + subject);

}

}

* Vì Student, GraduatedStudent, Teacher đều là Person(một người nào đó) nên ta có thể làm như sau:

class Main {

public static void main(String[] args) {

var student = getChildObject("Student");

var graduatedStudent = getChildObject("GraduatedStudent");

var teacher = getChildObject("Teacher");

var person = getChildObject("");

// kiểm tra kiểu của đối tượng đã tạo

checkType(student);

checkType(teacher);

checkType(person);

checkType(graduatedStudent);

}

private static Person getChildObject(String childType) {

if(childType.compareTo("Student") == 0) {

return new Student(); // nếu đối tượng cần tạo có kiểu là Student

} else if(childType.compareTo("Teacher") == 0) {

return new Teacher(); // nếu đối tượng cần tạo có kiểu là Teacher

} else if(childType.compareTo("GraduatedStudent") == 0) {

return new GraduatedStudent(); // nếu đối tượng cần tạo có kiểu là GraduatedStudent

}

return new Person(); // mặc định trả về đối tượng của Person

}

private static void checkType(Object obj) {

if(obj instanceof GraduatedStudent) {

System.out.println("Đây là một đối tượng GraduatedStudent.");

} else if(obj instanceof Student) {

System.out.println("Đây là một đối tượng GraduatedStudent.");

} else if(obj instanceof Teacher) {

System.out.println("Đây là một đối tượng Teacher.");

} else if(obj instanceof Person) {

System.out.println("Đây là một đối tượng Person.");

} else {

System.out.println("Đối tượng của một kiểu khác.");

}

}

}

Đây là một đối tượng GraduatedStudent.

Đây là một đối tượng Teacher.

Đây là một đối tượng Person.

Đây là một đối tượng GraduatedStudent.

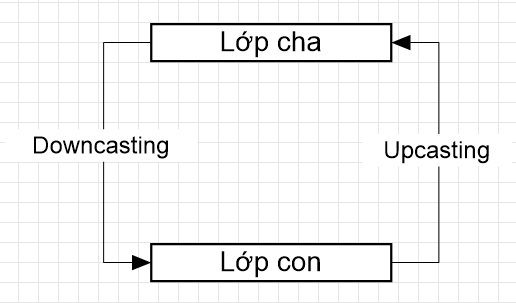
**Bài 8.2. Ép kiểu trong mối quan hệ kế thừa**

**Nội dung bài học**

1. Tổng quát
2. Ép kiểu từ dưới lên
3. Ép kiểu từ trên xuống

**Tổng quát**

* Ép kiểu giữa các đối tượng có thể thực hiện được nhưng chỉ trong mối quan hệ kế thừa.
* Trong mối quan hệ kế thừa, thứ bậc được xác định như sau: lớp cha càng cao càng ở trên cùng, lớp con càng thấp càng ở dưới cùng. Giống như: cụ->ông->cha->con->cháu.
* Có hai hình thức ép kiểu:
  + Ép kiểu từ đối tượng lớp con lên đối tượng lớp cha gọi là upcasting.
  + Ép kiểu từ đối tượng lớp cha xuống đối tượng lớp con gọi là downcasting.

[](https://braniumacademy.net/wp-content/uploads/2020/08/vi-du-ep-kieu-doi-tuong-tong-quat.png)

**Ép kiểu từ dưới lên**

* Là cách ép kiểu đối tượng trong đó đối tượng lớp con được ép thành đối tượng lớp cha.
* Cách ép kiểu này được thực hiện ngầm định.
* Ép kiểu theo cách này cho phép ta sử dụng các tính năng của lớp cha và chỉ một phần các tính năng của lớp con.
* Cụ thể các tính năng của lớp con được sử dụng là các phương thức override(học sau).
* Ví dụ:

package lesson45;

public class Animal {

private String name;

private float weight;

public Animal() {

}

public Animal(String name, float weight) {

this.name = name;

this.weight = weight;

}

public void move() {

System.out.println(name + " đang di chuyển...");

}

public void eat() {

System.out.println(name + " đang ăn...");

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public float getWeight() {

return weight;

}

public void setWeight(float weight) {

this.weight = weight;

}

}

class Cat extends Animal {

private String color;

private String favoriteFood;

public Cat() {

super();

}

public Cat(String name, float weight, String color, String favoriteFood) {

super(name, weight);

this.color = color;

this.favoriteFood = favoriteFood;

}

public void catchMouse() {

System.out.println(getName() + " đang rình bắt chuột...");

}

public void washFace() {

System.out.println(getName() + " đang rửa mặt...");

}

public String getColor() {

return color;

}

public void setColor(String color) {

this.color = color;

}

public String getFavoriteFood() {

return favoriteFood;

}

public void setFavoriteFood(String favoriteFood) {

this.favoriteFood = favoriteFood;

}

}

package lesson45;

public class Bai45 {

public static void main(String[] args) {

Animal cat = new Cat("Tom", 2.0f, "Gray", "Fish");

cat.eat(); // ok

cat.move(); // ok

cat.catchMouse(); // error

cat.washFace(); // error

}

}

**Ép kiểu từ trên xuống**

* Là cách ép kiểu đối tượng trong đó đối tượng của lớp cha được ép xuống đối tượng của lớp con.
* Cách ép kiểu này phải thực hiện tường minh.
* Sau khi ép kiểu ta có thể sử dụng đầy đủ tính năng của đối tượng lớp con.
* Ví dụ:

package lesson45;

public class Bai45 {

public static void main(String[] args) {

Animal animal = new Cat("Tom", 2.5f, "Black", "Fish");

Cat tom = (Cat)animal;

tom.eat(); // ok

tom.move(); // ok

tom.washFace(); // ok

tom.catchMouse(); // ok

}

}

* Để đảm bảo ép kiểu không xảy ra lỗi, ta kiểm tra đối tượng cần ép với instanceof. Nếu không bạn sẽ nhận được ClassCastException.
* Ví dụ:

package lesson45;

public class Bai45 {

public static void main(String[] args) {

Animal animal = new Animal("Tom", 2.0f);

Animal cat = new Cat("Tom", 2.5f, "Black", "Fish");

Animal fish = new Fish();

if(cat instanceof Cat) { // true

Cat tom = (Cat) cat;

//...

}

if(fish instanceof Cat) { // false

Cat tom = (Cat) fish;

}

if(animal instanceof Cat) { // false

Cat tom = (Cat) animal;

}

}

}

* Trong ví dụ trên, đối tượng fish không phải một thể hiện của Cat nên không thực hiện ép kiểu. Tương tự cho đối tượng animal. Đối tượng cat là một thể hiện của Cat nên toán tử instanceof trả về true-> ép kiểu thành công.

**Bài 8.3. Các thành phần protected**

**Nội dung bài học**

1. Mục đích sử dụng
2. Các trường protected
3. Các phương thức protected
4. Tổng hợp phạm vi khả dụng của các access modifier

Video lý thuyết

Video minh họa

**Mục đích sử dụng**

* Ta sử dụng keyword protected khi muốn thiết kế một lớp với mục đích cho phép kế thừa. Ví dụ ta muốn người sử dụng kế thừa các thư viện mà ta thiết kế ra. Thường áp dụng với lớp abstract.
* Khi muốn tạo một lớp mà các tính năng của nó chỉ được sử dụng trong các lớp con của nó mà không phải cho các lớp khác.

**Các trường protected**

* Thực tế ta ít khi sử dụng các trường protected. Tất cả các dữ liệu của một lớp ta để là private.
* Khi sử dụng trường protected sẽ không đảm bảo tính đóng gói dữ liệu và có thể xảy ra các nguy cơ nguy hiểm tiềm tàng. Ví dụ các hacker sẽ lợi dụng kẽ hở là các lớp con có thể truy cập trực tiếp vào các dữ liệu protected của lớp cha. Chúng chỉ cần khai báo một lớp con rồi kế thừa lớp cha có dữ liệu cần truy cập. Vậy là họ đã có thể dễ dàng có được thông tin mong muốn.
* Không sử dụng các trường protected chỉ vì muốn được truy cập trực tiếp vào các thành phần dữ liệu của lớp cha.
* Ví dụ:

package lesson46;

import java.util.Date;

public class Person {

protected String fullName;

private Date dateOfBirth;

//...

}

class Student extends Person {

private String studentId;

//...

public static void main(String[] args) {

Student nam = new Student();

nam.fullName = "Trần Văn Nam"; //ok but don't do this

}

}

* Thay vào đó hãy thực hiện như sau:

package lesson46;

import java.util.Date;

public class Person {

private String fullName;

private Date dateOfBirth;

//...

public String getFullName() {

return fullName;

}

public void setFullName(String fullName) {

this.fullName = fullName;

}

public Date getDateOfBirth() {

return dateOfBirth;

}

public void setDateOfBirth(Date dateOfBirth) {

this.dateOfBirth = dateOfBirth;

}

}

class Student extends Person {

private String studentId;

//...

public static void main(String[] args) {

Student nam = new Student();

nam.setFullName("Trần Văn Nam"); // very good!

}

}

**Các phương thức protected**

* Các outer class, interface không thể khai báo với access modifier là protected.
* Ví dụ:

package lesson46;

import java.util.Date;

protected class Person { // error!

private String fullName;

private Date dateOfBirth;

//...

public String getFullName() {

return fullName;

}

public void setFullName(String fullName) {

this.fullName = fullName;

}

public Date getDateOfBirth() {

return dateOfBirth;

}

public void setDateOfBirth(Date dateOfBirth) {

this.dateOfBirth = dateOfBirth;

}

}

* Các phương thức protected thường hay được sử dụng trong mối quan hệ kế thừa. Mục đích là cung cấp các tính năng cho chỉ những lớp con trong chuỗi kế thừa. Bạn có thể hình dung việc này giống như là các bí kíp gia truyền của một dòng họ nào đó. Chỉ những người là con cháu dòng dõi của dòng họ này mới được truyền đạt lại. Người ngoài không có cơ hội.
* Ví dụ:

package lesson46;

public class Father {

private String bankAccPassword; // mật khẩu tk ngân hàng

//...

protected void setBankAccPassword(String newPws) {

bankAccPassword = newPws;

}

}

package lesson46;

public class Child extends Father {

// các trường dữ liệu...

//...

// phương thức cập nhật mật khẩu:

public void updateBankInfo(String newPassword) {

if(newPassword.contains("^[a-zA-Z0-9/+-]$")) { // các điều kiện...

setBankAccPassword(newPassword); // thiết lập mật khẩu ok

}

}

}

// lớp Test trong một gói khác

package other;

import lesson46.Child;

import lesson46.Father;

public class Test {

public static void main(String[] args) {

Father myFather = new Father();

Child child = new Child();

myFather.setBankAccPassword("0123456\*Avx$"); // error!

child.updateBankInfo("0123456\*Avx$"); // ok

}

}

* Các protected constructor được sử dụng để ngăn cấm việc tạo đối tượng của một lớp từ bên ngoài gói.
* Ví dụ:

package lesson46;

import java.util.Date;

public class Person {

private String fullName;

private Date dateOfBirth;

//...

protected Person() { // protected constructor

fullName = "No Name";

dateOfBirth = new Date();

}

public String getFullName() {

return fullName;

}

public void setFullName(String fullName) {

this.fullName = fullName;

}

public Date getDateOfBirth() {

return dateOfBirth;

}

public void setDateOfBirth(Date dateOfBirth) {

this.dateOfBirth = dateOfBirth;

}

}

**Tổng hợp phạm vi khả dụng của các access modifier**

* Các thành phần private: chỉ khả dụng trong phạm vi lớp chứa nó.
* Các thành phần public: khả dụng cho toàn bộ mọi lớp.
* Các thành phần protected: khả dụng trong bản thân lớp chứa nó, các lớp trong cùng package, các lớp con của lớp hiện thời.
* Các thành phần mặc định(không có access modifier): khả dụng trong lớp chứa nó và các lớp trong cùng package.

# Bài 8.4. Ghi đè phương thức

### Nội dung bài học

1. [Mục đích sử dụng](https://braniumacademy.net/courses/java/lessons/java-overload-vs-override/#s1)
2. [Cú pháp](https://braniumacademy.net/courses/java/lessons/java-overload-vs-override/#s2)
3. [Các thành phần final](https://braniumacademy.net/courses/java/lessons/java-overload-vs-override/#s3)
4. [Quy tắc ghi đè phương thức](https://braniumacademy.net/courses/java/lessons/java-overload-vs-override/#s4)
5. [Sử dụng super keyword](https://braniumacademy.net/courses/java/lessons/java-overload-vs-override/#s5)
6. [Ví dụ minh họa](https://braniumacademy.net/courses/java/lessons/java-overload-vs-override/#s6)
7. [Bài tập thực hành](https://braniumacademy.net/courses/java/lessons/java-overload-vs-override/#s7)

Video lý thuyết

Video minh họa

### Mục đích sử dụng

* Override hay ghi đè có thể hiểu đơn giản là kĩ thuật trong Java cho phép định nghĩa lại phương thức của lớp cha theo cách của lớp con.
* Mục đích chính của override là lớp cha tạo ra các phương thức chung mà tất cả các lớp con có thể sử dụng và cho phép lớp con định nghĩa lại phương thức của lớp cha cho phù hợp với đặc trưng cụ thể của lớp con. Ví dụ tất cả các động vật đều di chuyển. Nhưng con mèo thì di chuyển bằng 4 chân, trên mặt đất. Cá bơi bằng vây, dưới nước. Còn chim bay bằng cánh, trên không. Vậy để mô tả các cách di chuyển phù hợp với từng loài động vật ta phải override lại phương thức move(di chuyển) của lớp cha(Animal).
* Override trong hướng đối tượng chỉ xảy ra với các phương thức không static, final và được kế thừa bởi lớp con.

### Cú pháp

* Để override một phương thức mà lớp con được kế thừa từ lớp cha. Ta gõ lại y nguyên kiểu trả về, tên phương thức, danh sách, thứ tự tham số.
* Phần access modifier của phương thức override ở lớp con có tầm khả dụng rộng hơn hoặc bằng của phương thức nguyên bản trong lớp cha.
* Nội dung của phương thức override trong lớp con được phép thay đổi cho phù hợp với ngữ cảnh của lớp con. Nhưng phải đảm bảo đúng mục đích sử dụng của phương thức.
* Có thể tái sử dụng lại các thành phần đã định nghĩa của phương thức nguyên bản qua lời gọi super.tên\_phương\_thức.
* Sử dụng @Override ở trước phương thức để đánh dấu một phương thức là đang override chứ không phải khai báo mới trong lớp con.
* Ví dụ:

import java.util.Date;

public class Person {

private String fullName;

private Date dateOfBirth;

//...

// phương thức gốc trong lớp cha

public void work() {

System.out.println("Người đang làm việc...");

}

}

class Student extends Person {

private String studentId;

//...

@Override

public void work() { // ghi đè phương thức gốc trong lớp cha

super.work(); // tái sử dụng phương thức nguyên bản trong lớp cha

System.out.println("Sinh viên đang học bài...");

}

}

* Khi ta tạo đối tượng để thực thi chương trình thì chương trình sẽ tự biết lấy đối tượng của lớp nào để chạy một cách chính xác:

public class Test {

public static void main(String[] args) {

Person person = new Person();

Student student = new Student();

Person nam = new Student();

person.work(); // gọi phương thức từ đối tượng của person

System.out.println("=====================================");

student.work(); // gọi phương thức từ đối tượng của Student

System.out.println("=====================================");

nam.work(); // gọi phương thức từ đối tượng person tham chiếu đến Student

}

}

Người đang làm việc...

=====================================

Người đang làm việc...

Sinh viên đang học bài...

=====================================

Người đang làm việc...

Sinh viên đang học bài...

* Trong ví dụ trên, thấy rằng nếu tham chiếu bằng đối tượng của Person thì chắc chắn rằng chương trình sẽ gọi tới phương thức work() nguyên bản trong lớp Person. Nhưng nếu tham chiếu bằng đối tượng của Student hoặc ép kiểu đối tượng của Student lên Person thì chương trình sẽ gọi phương thức work đã override trong Student.
* Như vậy việc gọi phương thức gốc hay đã override chỉ phụ thuộc vào kiểu của đối tượng được tạo. Tức khi tạo với keyword new chứ không phụ thuộc vào kiểu của biến đang tham chiếu(ở vế trái dấu =).

### Các thành phần final

* Lớp final là lớp có từ khóa final trong khai báo, lớp final không thể kế thừa.
* Có 2 mục đích chính khi sử dụng final class:
  + Ngăn cấm các lớp khác kế thừa lớp này. Ví dụ các lớp bao của các kiểu nguyên thủy là các lớp final
  + Để tạo các lớp bất biến(immutable class). Các lớp bất biến là các lớp mà khi đối tượng của lớp đó đã được tạo thì ta không thể thay đổi nội dung trong đối tượng đó. Ví dụ điển hình là lớp String.

public final class String

implements java.io.Serializable, Comparable<String>, CharSequence,

Constable, ConstantDesc {

//….

* Phương thức final là phương thức có non-access modifier là final và nó không thể override.
* Ví dụ:

public final void setFullName(String fullName) {

this.fullName = fullName;

}

public final Date getDateOfBirth() {

return dateOfBirth;

}

* Ví dụ kế thừa lớp final:

import java.util.Date;

public final class Person {

private String fullName;

private Date dateOfBirth;

//...

public final void setFullName(String fullName) {

this.fullName = fullName;

}

public final Date getDateOfBirth() {

return dateOfBirth;

}

}

class Student extends Person { // error

private String studentId;

//...

}

* Ví dụ override phương thức final:

import java.util.Date;

public class Person {

private String fullName;

private Date dateOfBirth;

public final void setFullName(String fullName) {

this.fullName = fullName;

}

public void work() {

System.out.println("Người đang làm việc...");

}

}

class Student extends Person {

private String studentId;

//...

@Override

public void work() { // ok

super.work();

System.out.println("Sinh viên đang học bài...");

}

@Override

public void setFullName(String fullName) { // error

super.setFullName(fullName);

}

}

### Quy tắc ghi đè phương thức

* Danh sách tham số của phương thức ghi đè phải giống chính xác về kiểu và thứ tự như trong lớp cha.
* Kiểu trả về, tên, chức năng của phương thức ghi đè phải giống y hệt như phương thức gốc trong lớp cha.
* Cấp độ khả dụng của access modifier của phương thức ghi đè không được nhỏ hơn trong phương thức gốc. Ví dụ phương thức gốc có access modifier là public thì phương thức override không thể có access modifier là private hoặc protected.
* Phương thức override chỉ thực hiện thành công khi nó nằm trong lớp cha của lớp hiện thời.
* Phương thức final không thể override bởi lớp con.
* Phương thức static không thể override nhưng có thể khai báo lại.
* Lớp con cùng gói với lớp cha có thể override bất kì phương thức không private, final, static của lớp cha.
* Lớp con không cùng gói với lớp cha chỉ có thể override các phương thức không static, final và có access modifier là public, protected.
* Phương thức khởi tạo không thể override.

### Sử dụng super keyword

* Sử dụng super keyword để:
  + Gọi đến phương thức khởi tạo của lớp cha.
  + Gọi đến các trường được kế thừa từ lớp cha.
  + Gọi đến các phương thức khác và phương thức nguyên bản đang được override ở lớp cha.

### Ví dụ minh họa

package lesson47;

public class Animal {

public void move() {

System.out.println("Animal is is moving...");

}

}

class Fish extends Animal {

@Override

public void move() {

System.out.println("Fish is moving by swimming...");

}

}

class Cat extends Animal {

@Override

public void move() {

System.out.println("Cat is moving by running on the ground...");

}

}

class Bird extends Animal {

@Override

public void move() {

System.out.println("Bird is moving by flying...");

}

}

* Lớp để test:

package lesson47;

public class Test {

public static void main(String[] args) {

Animal cat = new Cat();

Animal fish = new Fish();

Animal bird = new Bird();

Animal animal = new Animal();

cat.move();

fish.move();

bird.move();

animal.move();

}

}

Cat is moving by running on the ground...

Fish is moving by swimming...

Bird is moving by flying...

Animal is is moving...

**Bài 8.5. Kế thừa với lập trình giao diện**

**Nội dung bài học**

1. Tìm hiểu các thành phần của kiến trúc JFC/Swing
2. Vẽ đường tròn
3. Vẽ hình tròn và đổ màu cho nó
4. Vẽ hình chữ nhật bên trong hình tròn
5. Bài tập thực hành

**Tìm hiểu các thành phần của kiến trúc JFC/Swing**

* Ta đã tạo các ứng dụng Java đơn giản trên giao diện Console nhàm chán. Bài này ta sẽ tìm hiểu vài cú pháp để tạo ra giao diện đẹp mắt hấp dẫn hơn bằng ngôn ngữ Java cùng với các kiến thức vừa học nhé.
* Ta sẽ tạo ra các hình 2D đơn giản trên cửa sổ màn hình dạng như này:

Ảnh có chứa ảnh chụp màn hình, Nhiều màu sắc, biểu đồ, Đồ họa

Mô tả được tạo tự động

* Để tạo ra được giao diện cửa sổ trong Java Swing ta kế thừa lớp **JFrame**.
* Để đặt tiêu đề cho frame của bạn, sử dụng phương thức **setTitle(String title)**;
* Để thiết lập kích thước(pixel) ta dùng phương thức**setSize(int width, int height)**;
* Để thiết lập vị trí hiển thị của frame ta dùng phương thức **setLocation(int xPos, int yPos)**. Gốc tọa độ tính từ góc trái trên, x tăng từ trái qua phải, y tăng từ trên xuống dưới.

Ảnh có chứa văn bản, ảnh chụp màn hình, màn hình, phần mềm

Mô tả được tạo tự động

* Để căn frame ra giữa màn hình ta sử dụng phương thức **setLocationRelativeTo(null)**;
* Để cho frame(cửa sổ) hiển thị lên màn hình ta sử dụng phương thức **setVisible(true)**;
* Để thiết lập độ trong suốt cho cửa sổ ta sử dụng phương thức**setOpacity(float value)**; Giá trị value từ 0 đến 1.
* Để thoát chương trình khi nhấn nút x màu đỏ góc phải trên của frame ta gọi **setDefaultCloseOperation(EXIT\_ON\_CLOSE)**;
* Để thiết lập màu background cho frame ta gọi **setBackground(Color c)**.
* Để tùy chỉnh vẽ lại giao diện theo mong muốn của bản thân ta ghi đè phương thức paint(Graphics g)
* Ví dụ:

import javax.swing.\*;

public class MyFrame extends JFrame {

public MyFrame(String title) {

this.setTitle(title);

this.setSize(400, 400);

this.setLocation(200, 300);

this.setDefaultCloseOperation(EXIT\_ON\_CLOSE);

}

public static void main(String[] args) {

MyFrame frame = new MyFrame("My First JFrame Application");

frame.setVisible(true);

}

}

Ảnh có chứa văn bản, ảnh chụp màn hình, màn hình, phần mềm

Mô tả được tạo tự động

**Vẽ đường tròn**

* Để vẽ đường tròn ta gọi phương thức **drawOval(int x, int y, int width, int height)**.
* Trong đó cho width = height. Giá trị x, y là tọa độ góc trái trên hình vuông chứa gọn hình tròn trong đó.
* Giá trị width, height là đường kính của hình oval.
* Để thiết lập độ dày của đường tròn ta gọi **setStroke(new BasicStroke(int width))**; trên đối tượng **Graphics2D**.
* Trong đó with là độ dày tính theo số **pixel** trên màn hình máy tính.
* Để tạo màu cho đường tròn ta gọi **setColor(Color c)** trên đối tượng **Graphics** hoặc **Graphics2D**.
* Để căn giữa màn hình ta lấy tâm của cửa sổ(width/2, height/2) trừ đi bán kính của hình oval.
* Cụ thể:

import javax.swing.\*;

import java.awt.\*;

public class MyFrame extends JFrame {

private int width; // chiều rộng Ox

private int height; // chiều cao Oy

public MyFrame(String title, int width, int height) {

this.width = width;

this.height = height;

this.setTitle(title);

this.setSize(width, height);

this.setLocation(200, 300);

this.setDefaultCloseOperation(EXIT\_ON\_CLOSE);

}

@Override

public void paint(Graphics g) {

var g2d = (Graphics2D) g; // ép kiểu sang Graphics2D

g.setColor(Color.red); // thiết lập màu

g2d.setStroke(new BasicStroke(2)); // gọi thiết lập độ rộng nét vẽ

int radius = 150; // bán kính đường tròn

g.drawOval(width / 2 - radius, height / 2 - radius,

2 \* radius, 2 \* radius);

}

public static void main(String[] args) {

int width = 400; // chiều rộng cửa sổ

int height = 400; // chiều cao cửa sổ

String title = "My First JFrame Application";

MyFrame frame = new MyFrame(title, width, height);

frame.setVisible(true);

}

}

Ảnh có chứa ảnh chụp màn hình, văn bản, vòng tròn, biểu đồ

Mô tả được tạo tự động

* Để vẽ các loại hình khác:
  + Vẽ đường thẳng: **drawLine(int x1, int y1, int x2, int y2)**;
  + Vẽ đa giác:**drawPoligon(int[] xPoints, int[] yPoints, int points)**;
  + Vẽ ảnh: **drawImage(Image img, int x, int y, ImageObserver obs)**;
  + Vẽ hình chữ nhật: **drawRect(int x, int y, int width, int height)**;
  + Vẽ hình chữ nhật bo cạnh: **drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)**;
  + Các hình và thành phần khác bạn gõ **g.** sẽ hiện ra trong phương thức **paint()** nạp chồng.

**Vẽ hình tròn và đổ màu cho nó**

* Bây giờ ta vẽ hình tròn đặc với fillOval().
* Các hình khối đặc khác ta thay drawX() bằng fillX().
* Ví dụ sau vẽ hình tròn đặc màu đỏ:

import javax.swing.\*;

import java.awt.\*;

public class MyFrame extends JFrame {

private int width; // chiều rộng Ox

private int height; // chiều cao Oy

public MyFrame(String title, int width, int height) {

this.width = width;

this.height = height;

this.setTitle(title);

this.setSize(width, height);

this.setLocation(200, 300);

this.setDefaultCloseOperation(EXIT\_ON\_CLOSE);

}

@Override

public void paint(Graphics g) {

var g2d = (Graphics2D) g; // ép kiểu sang Graphics2D

g.setColor(Color.red); // thiết lập màu

g2d.setStroke(new BasicStroke(2)); // gọi thiết lập độ rộng nét vẽ

int radius = 150; // bán kính đường tròn

g.fillOval(width / 2 - radius, height / 2 - radius,

2 \* radius, 2 \* radius);

}

public static void main(String[] args) {

int width = 400; // chiều rộng cửa sổ

int height = 400; // chiều cao cửa sổ

String title = "My First JFrame Application";

MyFrame frame = new MyFrame(title, width, height);

frame.setVisible(true);

}

}

Ảnh có chứa ảnh chụp màn hình, văn bản, vòng tròn, Nhiều màu sắc

Mô tả được tạo tự động

* Ví dụ vẽ hình chữ nhật bo cạnh đặc màu đỏ:

import javax.swing.\*;

import java.awt.\*;

public class MyFrame extends JFrame {

private int width; // chiều rộng Ox

private int height; // chiều cao Oy

public MyFrame(String title, int width, int height) {

this.width = width;

this.height = height;

this.setTitle(title);

this.setSize(width, height);

this.setLocationRelativeTo(null);

this.setDefaultCloseOperation(EXIT\_ON\_CLOSE);

}

@Override

public void paint(Graphics g) {

var g2d = (Graphics2D) g; // ép kiểu sang Graphics2D

g.setColor(Color.red); // thiết lập màu

g2d.setStroke(new BasicStroke(2)); // gọi thiết lập độ rộng nét vẽ

int radius = 125; // bán kính đường tròn

g.fillRoundRect(width / 2 - radius, height / 2 - radius,

2 \* radius, 2 \* radius, 20, 20);

}

public static void main(String[] args) {

int width = 400; // chiều rộng cửa sổ

int height = 400; // chiều cao cửa sổ

String title = "My First JFrame Application";

MyFrame frame = new MyFrame(title, width, height);

frame.setVisible(true);

}

}

Ảnh có chứa văn bản, ảnh chụp màn hình, Hình chữ nhật, phần mềm

Mô tả được tạo tự động

**Vẽ hình chữ nhật bên trong hình tròn**

* Bây giờ ta đã biết vẽ hình chữ nhật, tròn, đa giác,…
* Tiếp theo ta sẽ chồng lấn các hình lên nhau. Lưu ý rằng hình nào vẽ trước sẽ nằm dưới, hình vẽ sau sẽ chồng lấn bên trên.
* Ta vẽ hình chữ nhật bên trong đường tròn:

import javax.swing.\*;

import java.awt.\*;

public class MyFrame extends JFrame {

private int width; // chiều rộng Ox

private int height; // chiều cao Oy

public MyFrame(String title, int width, int height) {

this.width = width;

this.height = height;

this.setTitle(title);

this.setSize(width, height);

this.setLocationRelativeTo(null);

this.setDefaultCloseOperation(EXIT\_ON\_CLOSE);

}

@Override

public void paint(Graphics g) {

var g2d = (Graphics2D) g; // ép kiểu sang Graphics2D

g.setColor(Color.red); // thiết lập màu

g2d.setStroke(new BasicStroke(2)); // gọi thiết lập độ rộng nét vẽ

drawRoundRect(g); // vẽ hình vuông bo cạnh trước

drawCircle(g); // vẽ đường tròn sau

}

/\*\*

\* Phương thức vẽ đường tròn bán kính radius với màu được chỉ định

\* @param g một đối tượng kiểu Graphics hoặc Graphics2D

\*/

private void drawCircle(Graphics g) {

int radius = 150;

g.setColor(Color.green);

g.fillOval(width / 2 - radius, height / 2 - radius,

2 \* radius, 2 \* radius);

}

/\*\*

\* Phương thức vẽ hình vuông/chữ nhật bo cạnh với màu cho trước.

\* @param g một đối tượng kiểu Graphics hoặc Graphics2D

\*/

private void drawRoundRect(Graphics g) {

int radius = 125; // bán kính đường tròn

g.fillRoundRect(width / 2 - radius, height / 2 - radius,

2 \* radius, 2 \* radius, 20, 20);

}

public static void main(String[] args) {

int width = 400; // chiều rộng cửa sổ

int height = 400; // chiều cao cửa sổ

String title = "My First JFrame Application";

MyFrame frame = new MyFrame(title, width, height);

frame.setVisible(true);

}

}

Ảnh có chứa ảnh chụp màn hình, văn bản, biểu đồ, Nhiều màu sắc

Mô tả được tạo tự động

import javax.swing.\*;

import java.awt.\*;

public class MyFrame extends JFrame {

private final int width; // chiều rộng Ox

private final int height; // chiều cao Oy

public MyFrame(String title, int width, int height) {

this.width = width;

this.height = height;

this.setTitle(title);

this.setSize(width, height);

this.setLocationRelativeTo(null);

this.setDefaultCloseOperation(EXIT\_ON\_CLOSE);

}

@Override

public void paint(Graphics g) {

var g2d = (Graphics2D) g; // ép kiểu sang Graphics2D

g2d.setStroke(new BasicStroke(2)); // gọi thiết lập độ rộng nét vẽ

drawCircle(g); // vẽ đường tròn trước

drawRoundRect(g); // vẽ hình vuông bo cạnh sau

}

/\*\*

\* Phương thức vẽ đường tròn bán kính radius với màu được chỉ định

\* @param g một đối tượng kiểu Graphics hoặc Graphics2D

\*/

private void drawCircle(Graphics g) {

int radius = 150;

g.setColor(Color.green);

g.drawOval(width / 2 - radius, height / 2 - radius,

2 \* radius, 2 \* radius);

}

/\*\*

\* Phương thức vẽ hình vuông/chữ nhật bo cạnh với màu cho trước.

\* @param g một đối tượng kiểu Graphics hoặc Graphics2D

\*/

private void drawRoundRect(Graphics g) {

int radius = 100; // bán kính đường tròn

g.setColor(Color.red); // set màu cho hình chữ nhật

g.fillRoundRect(width / 2 - radius, height / 2 - radius,

2 \* radius, 2 \* radius, 20, 20);

}

public static void main(String[] args) {

int width = 400; // chiều rộng cửa sổ

int height = 400; // chiều cao cửa sổ

String title = "My First JFrame Application";

MyFrame frame = new MyFrame(title, width, height);

frame.setVisible(true);

}

}

Ảnh có chứa văn bản, ảnh chụp màn hình, biểu đồ, Nhiều màu sắc

Mô tả được tạo tự động

* Hoặc vẽ hình đặc:

import javax.swing.\*;

import java.awt.\*;

public class MyFrame extends JFrame {

private final int width; // chiều rộng Ox

private final int height; // chiều cao Oy

public MyFrame(String title, int width, int height) {

this.width = width;

this.height = height;

this.setTitle(title);

this.setSize(width, height);

this.setLocationRelativeTo(null);

this.setDefaultCloseOperation(EXIT\_ON\_CLOSE);

}

@Override

public void paint(Graphics g) {

var g2d = (Graphics2D) g; // ép kiểu sang Graphics2D

g2d.setStroke(new BasicStroke(2)); // gọi thiết lập độ rộng nét vẽ

drawCircle(g); // vẽ đường tròn sau

drawRoundRect(g); // vẽ hình vuông bo cạnh trước

}

/\*\*

\* Phương thức vẽ đường tròn bán kính radius với màu được chỉ định

\* @param g một đối tượng kiểu Graphics hoặc Graphics2D

\*/

private void drawCircle(Graphics g) {

int radius = 150;

g.setColor(Color.green);

g.fillOval(width / 2 - radius, height / 2 - radius,

2 \* radius, 2 \* radius);

}

/\*\*

\* Phương thức vẽ hình vuông/chữ nhật bo cạnh với màu cho trước.

\* @param g một đối tượng kiểu Graphics hoặc Graphics2D

\*/

private void drawRoundRect(Graphics g) {

int radius = 100; // bán kính đường tròn

g.setColor(Color.red); // set màu cho hình chữ nhật

g.fillRoundRect(width / 2 - radius, height / 2 - radius,

2 \* radius, 2 \* radius, 20, 20);

}

public static void main(String[] args) {

int width = 400; // chiều rộng cửa sổ

int height = 400; // chiều cao cửa sổ

String title = "My First JFrame Application";

MyFrame frame = new MyFrame(title, width, height);

frame.setVisible(true);

}

}

Ảnh có chứa văn bản, ảnh chụp màn hình, biểu đồ, Nhiều màu sắc

Mô tả được tạo tự động

* Để thiết lập kích thước ưa thích ta gọi phương thức **setPreferredSize(Dimension d)**;
* Để đổ màu cho background ta gọi **setBackground(Color c)** sau khi **setPreferredSize()**;
* Để ngăn cửa sổ phóng to ta gọi **setResizable(false)**;

import javax.swing.\*;

import java.awt.\*;

public class MyFrame extends JFrame {

private final int width;

private final int height;

public MyFrame(int width, int height, String title) {

this.width = width;

this.height = height;

this.setTitle(title);

this.setSize(width, height);

//this.setLocation(100, 150);

this.setLocationRelativeTo(null);

this.setDefaultCloseOperation(EXIT\_ON\_CLOSE);

this.setResizable(false);

this.setPreferredSize(new Dimension(width, height));

this.setBackground(Color.YELLOW);

this.pack();

}

@Override

public void paint(Graphics g) {

var g2d = (Graphics2D) g;

g2d.setStroke(new BasicStroke(2));

drawCircle(g);

drawRoundRect(g);

}

private void drawCircle(Graphics g) {

int radius = 150;

g.setColor(Color.green);

g.drawOval(width / 2 - radius, height / 2 - radius,

2 \* radius, 2 \* radius);

}

private void drawRoundRect(Graphics g) {

int width = 200;

int height = 200;

g.setColor(Color.red);

g.fillRoundRect(this.width / 2 - width / 2,

this.height / 2 - height / 2, width, height,

20, 20);

}

public static void main(String[] args) {

var title = "My First JFrame Application";

int width = 400;

int height = 400;

MyFrame frame = new MyFrame(width, height, title);

frame.setVisible(true);

}

}

Ảnh có chứa văn bản, ảnh chụp màn hình, Nhiều màu sắc, màu vàng

Mô tả được tạo tự động

# Bài 8.6. Ghi đè vs nạp chồng

### Nội dung bài học:

1. Khái niệm
2. Sự khác nhau
3. Ví dụ minh họa

### Khái niệm

* **Nạp chồng** là kĩ thuật trong Java cho phép 2 hoặc nhiều phương thức có cùng tên nhưng khác tham số(về kiểu, số lượng, thứ tự).
* **Ghi đè** là kĩ thuật trong Java cho phép lớp con định nghĩa lại phương thức của lớp cha theo cách của lớp con.

#### Sự khác nhau

|  |  |
| --- | --- |
| **Nạp chồng** | **Ghi đè** |
| Thuộc loại đa hình tại thời điểm biên dịch chương trình. | Thuộc loại đa hình tại thời điểm chương trình chạy. |
| Cung cấp cùng chức năng cho nhiều kiểu tham số khác nhau. | Tăng mức độ tương thích của một phương thức của lớp cha trong ngữ cảnh sử dụng của lớp con. |
| Thường xảy ra trong 1 lớp độc lập hoặc trong mối quan hệ kế thừa. | Chỉ xảy ra tại lớp con trong mối quan hệ kế thừa. |
| Phương thức overload bắt buộc phải cùng tên và khác dấu hiệu nhận biết về tham số. Tức phải khác kiểu, thứ tự và số lượng tham số. | Phương thức override phải có dấu hiệu nhận biết giống hệt như phương thức gốc trong lớp cha. |
| Trong overloading kiểu trả về có thể giống hoặc khác nhau nhưng bắt buộc khác tham số. | Kiểu trả về phải giữ nguyên bản như trong phương thức gốc. |
| Có thể xảy ra ở mọi phương thức, không giới hạn bởi access hay non-access modifier. | Chỉ xảy ra ở các phương thức được kế thừa lại và có thể truy cập trực tiếp từ lớp con. Không xảy ra với các phương thức private, final, static. |

### Ví dụ minh họa

* Nạp chồng:

package lesson48;

public class MyCalculator {

// phương thức overload với tham số là int

public static int add(int a, int b) {

return a + b;

}

// phương thức overload với tham số là long

public static long add(long a, long b) {

return a + b;

}

// phương thức overload với tham số là double

public static double add(double a, double b) {

return a + b;

}

public static void main(String[] args) {

// sử dụng phương thức overload

double sumDouble = add(3.25, 7.85);

long sumLong = add(15536L, 15454L);

int sumInt = add(1314, 578);

//...

}

}

* Ghi đè:

package lesson48;

public class Animal {

public void move() {

System.out.println("Animal is is moving...");

}

}

class Fish extends Animal {

@Override

public void move() {

System.out.println("Fish is moving by swimming...");

}

}

class Bird extends Animal {

@Override

public void move() {

System.out.println("Bird is moving by flying...");

}

}

class Cat extends Animal {

@Override

public void move() {

System.out.println("Cat is moving by running on the ground...");

}

}

* Lớp Test:

package lesson48;

public class Test {

public static void main(String[] args) {

Animal fish = new Fish();

Animal cat = new Cat();

Animal bird = new Bird();

Animal animal = new Animal();

// gọi phương thức override

animal.move();

fish.move();

cat.move();

bird.move();

}

}

Animal is is moving...

Fish is moving by swimming...

Cat is moving by running on the ground...

Bird is moving by flying...

**Bài 8.7. Lớp Object và một số phương thức quan trọng**

**Nội dung bài học**

1. Giới thiệu lớp Object
2. Các phương thức thường dùng
3. Ghi đè toString()
4. Ghi đè equals()

Video lý thuyết

Video minh họa

**Giới thiệu lớp Object**

* Lớp Object là tổ tiên của mọi lớp trong Java. Nếu một lớp không chỉ rõ lớp cha trực tiếp thì lớp cha trực tiếp của nó chính là lớp Object.
* Lớp Object sẽ hữu ích trong trường hợp bạn muốn tham chiếu đến bất kì đối tượng nào có kiểu chưa được biết trước.
* Ví dụ:

package lesson49;

public class Car {

private String brand;

private String name;

private float weight;

public void speedUp() {

//...

}

public void brake() {

//...

}

}

* Tương đương với:

package lesson49;

public class Car extends Object {

private String brand;

private String name;

private float weight;

public void speedUp() {

//...

}

public void brake() {

//...

}

}

* Và ta có thể sử dụng kiểu Object để tham chiếu đến đối tượng của Car:

package lesson49;

public class Test {

public static void main(String[] args) {

Car car = new Car(); // ok

Object aCar = new Car(); // ok

}

}

**Các phương thức thường dùng**

* Sau đây là các phương thức của lớp Object thường dùng và mô tả:

|  |  |
| --- | --- |
| **Phương thức** | **Mô tả** |
| Class<?> getClass() | Trả về kiểu(lớp) của đối tượng hiện thời. |
| boolean equals(Object obj) | Trả về giá trị kiểu boolean kết luận liệu hai đối tượng đang so sánh có tương đương hay không. |
| int hashCode() | Trả về mã băm định danh của đối tượng hiện thời. Mỗi đối tượng trong Java khi được tạo ra sẽ có một mã băm duy nhất giống như số chứng minh thư hay căn cước công dân. |
| String toString() | Trả về một String mô tả vắn tắt thông tin về đối tượng hiện thời. |

* Ở mức độ sử dụng cơ bản, ta thường hay override hai phương thức là equals() và toString() trong các lớp con như sau:

**Ghi đè toString()**

* Với phương thức toString(), dùng để mô tả thông tin về:
  + Tên lớp
  + Các trường và giá trị tương ứng
* Ví dụ sau ghi đè phương thức toString để mô tả thông tin của đối tượng lớp Car:

package lesson49;

public class Car {

private String brand;

private String name;

private float weight;

public Car() {

}

public Car(String brand, String name, float weight) {

this.brand = brand;

this.name = name;

this.weight = weight;

}

public void speedUp() {

//...

}

public void brake() {

//...

}

@Override

public String toString() {

return "Car{" +

"brand='" + brand + '\'' +

", name='" + name + '\'' +

", weight=" + weight +

'}';

}

}

* Ta có thể override phương thức toString() trên với biến thể như sau:

@Override

public String toString() {

return getClass().getName() + "[" +

"brand='" + brand + '\'' +

", name='" + name + '\'' +

", weight=" + weight +

']';

}

* Khi ta sử dụng đối tượng của một lớp làm đối số của phương thức như println() thì chương trình Java sẽ tự động gọi đến phương thức toString của kiểu đang sử dụng.

package lesson49;

public class Test {

public static void main(String[] args) {

Car car = new Car("Vinfast", "President", 2250);

System.out.println(car);

System.out.println("==========================");

// gán giá trị ra một String:

String carInfo = car.toString();

System.out.println(carInfo);

}

}

lesson49.Car[brand='Vinfast', name='President', weight=2250.0]

==========================

lesson49.Car[brand='Vinfast', name='President', weight=2250.0]

* Nếu lớp cha đã override lại phương thức toString() và có getClass().getName() thì ta chỉ cần bổ sung thêm thông tin như sau:

package lesson49;

public class Car {

private String brand;

private String name;

private float weight;

public Car() {

}

public Car(String brand, String name, float weight) {

this.brand = brand;

this.name = name;

this.weight = weight;

}

public void speedUp() {

//...

}

public void brake() {

//...

}

@Override

public String toString() {

return getClass().getName() + "[" +

"brand='" + brand + '\'' +

", name='" + name + '\'' +

", weight=" + weight +

']';

}

}

class ElectricCar extends Car {

private float voltage; // hiệu điện thế

private float bateryCapacity; // dung lượng pin

public ElectricCar(float voltage, float bateryCapacity) {

this.voltage = voltage;

this.bateryCapacity = bateryCapacity;

}

public ElectricCar(String brand, String name, float weight,

float voltage, float bateryCapacity) {

super(brand, name, weight);

this.voltage = voltage;

this.bateryCapacity = bateryCapacity;

}

//...

@Override

public String toString() {

return super.toString() + "[" +

"voltage=" + voltage +

", bateryCapacity=" + bateryCapacity +

']';

}

}

* Chạy chương trình:

package lesson49;

public class Test {

public static void main(String[] args) {

Car car = new Car("Vinfast", "President", 2250);

System.out.println(car);

System.out.println("==========================");

// gán giá trị ra một String:

ElectricCar eCar = new ElectricCar("Tesla", "Model S", 2325, 120, 900000);

System.out.println(eCar);

}

}

lesson49.Car[brand='Vinfast', name='President', weight=2250.0]

==========================

lesson49.ElectricCar[brand='Tesla', name='Model S', weight=2325.0][voltage=120.0, bateryCapacity=900000.0]

**Ghi đè equals()**

* Phương thức này dùng để kiểm tra xem hai đối tượng có tương đương hay không.
* Hai đối tượng gọi là tương đương nếu chúng cùng tham chiếu đến 1 đối tượng.
* Lưu ý rằng cho dù hai đối tượng cùng lớp có thông tin giống hệt nhau nhưng chưa chắc chúng đã tương đương. Tương đương của phương thức equals này mang ý nghĩa trùng khớp duy nhất.
* Giả sử rằng ta quy ước hai đối tượng có cùng 1 số thông tin nào đó là hai đối tượng tương đương, ta sẽ override lại phương thức equals.
* Ví dụ hai sinh viên cùng mã, cùng họ tên, địa chỉ, email… thì chắc chắn rằng hai sinh viên đó được coi là tương đương, hay cùng là 1 đối tượng.
* Các bước để override phương thức equals:
  + Bước 1: Đặt tên tham số là otherObject để phân biệt với đối tượng hiện thời
  + Bước 2: Kiểm tra xem nếu đối tượng hiện thời và tham số là đồng nhất, return true
  + Bước 3: Nếu otherObject là null return false
  + Bước 4: Nếu otherObject không cùng kiểu, return false
  + Bước 5: Ép kiểu otherObject sang kiểu của lớp hiện thời
  + Bước 6: So sánh các trường của hai đối tượng và trả về kết quả. Sử dụng == với kiểu nguyên thủy và Objects.equals() với các trường tham chiếu.
* Ví dụ:

package lesson49;

import java.util.Date;

import java.util.Objects;

public class Student {

private String id;

private String name;

private String email;

private String address;

private Date dateOfBirth;

public Student() {

}

public Student(String id, String name, String email,

String address, Date dateOfBirth) {

this.id = id;

this.name = name;

this.email = email;

this.address = address;

this.dateOfBirth = dateOfBirth;

}

@Override

public String toString() {

return "Student{" +

"id='" + id + '\'' +

", name='" + name + '\'' +

", email='" + email + '\'' +

", address='" + address + '\'' +

", dateOfBirth=" + dateOfBirth +

'}';

}

@Override

public boolean equals(Object otherObject) { // bước 1

if (this == otherObject) return true; // bước 2

if (otherObject == null // bước 3

|| getClass() != otherObject.getClass() // bước 4

) return false;

Student other = (Student) otherObject; // bước 5

return Objects.equals(id, other.id) && // bước 6

Objects.equals(name, other.name) &&

Objects.equals(dateOfBirth, other.dateOfBirth);

}

@Override

public int hashCode() {

return Objects.hash(id, name, dateOfBirth);

}

}

* Ví dụ sử dụng:

package lesson49;

import java.text.ParseException;

import java.text.SimpleDateFormat;

public class Test {

public static void main(String[] args) throws ParseException {

var format = "dd/MM/yyyy";

var dateFormat = new SimpleDateFormat(format);

var dob = dateFormat.parse("22/10/2005");

Student nam = new Student("STU100001", "Trần Văn Nam",

"nam@xmail.com", "Hà Nội", dob);

Student other = new Student("STU100001", "Trần Văn Nam",

"nam@xmail.com", "Hà Nội", dob);

System.out.println(nam.equals(other)); // true

}

}

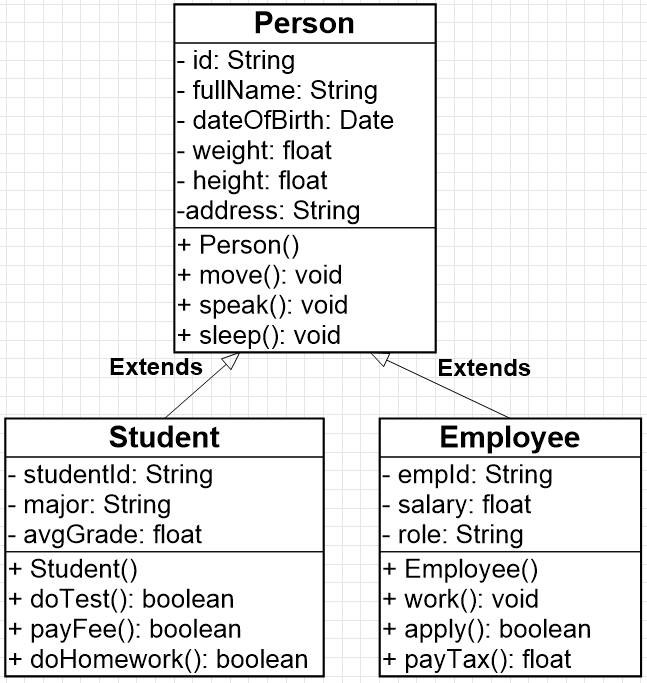
**Bài 8.8. Gợi ý thiết kế kế thừa trong Java**

**Nội dung bài học**

1. Đặt các thành phần chung nhất ở lớp cha
2. Không sử dụng trường protected
3. Sử dụng kế thừa để mô hình hóa mối quan hệ “is-a”
4. Không sử dụng kế thừa trừ khi tất cả các phương thức được kế thừa đều có ý nghĩa
5. Không thay đổi mục đích sử dụng của phương thức override
6. Sử dụng tính chất đa hình thay cho kiểu cụ thể

**Đặt các thành phần chung nhất ở lớp cha**

* Tất cả các lớp con khi kế thừa lớp cha sẽ sử dụng được các thành phần chung có ở lớp cha.
* Do đó ta sẽ đưa các trường, các phương thức có chung ở tất cả các lớp con vào lớp cha.
* Các lớp con sau đó chỉ bổ sung thêm các thành phần riêng của chính bản thân nó.
* Ví dụ:

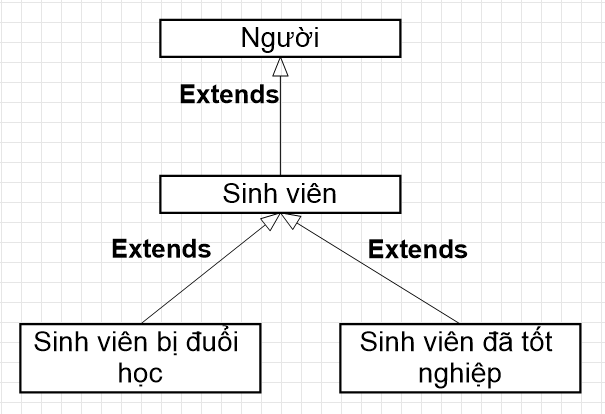
[](https://braniumacademy.net/wp-content/uploads/2020/09/vi-du-thiet-ke-ke-thua-111.png)Dữ liệu và hành vi lớp cha, lớp con

**Không sử dụng trường protected**

* Để đảm bảo tính đóng gói dữ liệu, tất cả dữ liệu của từng lớp ta để private.
* Sử dụng trường protected sẽ thuận tiện cho truy cập trực tiếp từ trong lớp con. Tuy nhiên nó lại không hề an toàn và có nguy cơ bị lộ thông tin nhạy cảm. Lí do là các lớp con và các lớp cùng gói đều có thể truy cập trực tiếp vào thành phần protected. Khi đó những kẻ có ý đồ xấu sẽ lợi dụng kẽ hở này để tạo một lớp con của lớp có thông tin cần truy cập. Hoặc chỉ cần tạo một lớp trong cùng gói với lớp có thông tin cần truy cập là họ có thể lấy được các thông tin mong muốn.
* Các phương thức protected thì lại hữu ích.
* Các phương thức protected khi muốn sử dụng thường sẽ phải override lại.
* Chúng chỉ có thể được sử dụng bởi lớp con hoặc các lớp trong cùng gói. Như vậy sẽ an toàn hơn.

**Sử dụng kế thừa để mô hình hóa mối quan hệ “is-a”**

* Chỉ sử dụng kế thừa để thể hiện mối quan hệ là một cái gì đó. Không lạm dụng kế thừa chỉ vì muốn tái sử dụng lại code và không quan tâm những vấn đề khác.
* Ví dụ, để thể hiện mối quan hệ giữa người, sinh viên, sinh viên đã tốt nghiệp, sinh viên bị đuổi học. Ta sử dụng mối quan hệ kế thừa:

[](https://braniumacademy.net/wp-content/uploads/2020/09/vi-du-quan-he-ke-thua222.png)Ví dụ mối quan hệ is-a

* Nhưng giả sử ta có một bảng điểm, trong đó chứa các đầu điểm, học lực và thông tin sinh viên. Nếu ta lười code thì ta có thể nghĩ đến việc cho bảng điểm kế thừa sinh viên. Sau đó sử dụng thông tin của sinh viên bình thường. Nếu làm vậy sẽ là lạm dụng tính chất kế thừa và không thể hiện được mối quan hệ kế thừa ở đây. Bởi đơn giản bảng điểm không phải là lớp con của sinh viên mà là một thành phần độc lập có liên quan đến ít nhất một khóa học nào đó mà sinh viên tham gia.

**Không sử dụng kế thừa trừ khi tất cả các phương thức được kế thừa đều có ý nghĩa**

* Giả sử ta có lớp Người với các hành động như ăn, uống, nghỉ ngơi, giao tiếp, làm việc. Lớp Nhân viên kế thừa từ người và có thêm các phương thức về chấm công, tính lương, nhận lương, đi công tác, nhận nhiệm vụ, thăng chức.
* Ta muốn tạo lớp mô tả thông tin sinh viên. Rõ ràng ta sẽ không kế thừa từ lớp nhân viên vì các hành động của nhân viên không có ý nghĩa với sinh viên. Vì không phải mọi sinh viên đều đi làm, nên không có chấm công, không có tính lương, không nhận lương, không đi công tác…
* Như vậy ta sẽ chọn kế thừa từ lớp Người. Vì mọi hành vi của người đều có ở sinh viên. Sau đó ta bổ sung thêm các hành vi đặc trưng của sinh viên: học bài, đi thi, nộp học phí, tham gia hoạt động ngoại khóa, đi tình nguyện….
* Rõ ràng rằng việc kế thừa cần phải được xem xét cẩn thận và hợp lý.

**Không thay đổi mục đích sử dụng của phương thức override**

* Khi thực hiện override một phương thức của lớp cha. Bạn hãy thực hiện giữ đúng mục đích sử dụng của phương thức.
* Không tự ý thay đổi mục đích sử dụng của một phương thức thành một hành động khác.
* Điểu này nhằm đảm bảo tính nhất quán trong toàn bộ chuỗi kế thừa của lớp.
* Ví dụ cùng là phương thức di chuyển nhưng mỗi loài động vật lại có cách di chuyển khác nhau: chim bay trên không, cá bơi trong nước, bò đi trên mặt đất. Tất cả đề ám chỉ cách di chuyển của từng loài động vật trong lớp con. Không tự ý thay đổi mục đích sử dụng của phương thức này trong bất kì lớp con nào. Ví dụ trong phương thức di chuyển của cá lại viết code thể hiện việc ăn uống của cá->không làm như vậy.

**Sử dụng tính chất đa hình thay cho kiểu cụ thể**

* Khi bạn có một vấn đề cần xử lý có dạng:
  + Nếu x là đối tượng kiểu T1 thì thực hiện hành động 1
  + Nếu x là đối tượng kiểu T2 thì thực hiện hành động 2
  + Nếu x là đối tượng kiểu T3 thì thực hiện hành động 3
* Hãy nghĩ ngay đến việc sử dụng tính đa hình. Tức là tạo biến có kiểu là kiểu của lớp cha hoặc interface nhưng tham chiếu đến các đối tượng của chính lớp cha đó hoặc lớp con tương ứng.
* Khi đó chương trình sẽ tự động dựa vào đối tượng đang được tham chiếu để xác định chính xác phương thức cần gọi.
* Code sử dụng tính chất đa hình sẽ dễ bảo trì và mở rộng hơn so với code sử dụng nhiều kiểu khác nhau để thực hiện cùng chức năng.
* Ví dụ:

package lesson50;

public class Animal {

public void move() {

System.out.println("Animal is is moving...");

}

}

class Fish extends Animal {

@Override

public void move() {

System.out.println("Fish is moving by swimming...");

}

}

class Cat extends Animal {

@Override

public void move() {

System.out.println("Cat is moving by running on the ground...");

}

}

class Bird extends Animal {

@Override

public void move() {

System.out.println("Bird is moving by flying...");

}

}

* Sử dụng các lớp trên:

package lesson50;

public class Test {

public static void main(String[] args) {

Animal fish = new Fish();

Animal cat = new Cat();

Animal bird = new Bird();

Animal animal = new Animal();

// gọi phương thức override sử dụng tính đa hình

animal.move();

fish.move();

cat.move();

bird.move();

}

}

**Bài 9.1. Các lớp trừu tượng – Abtract classes**

**Nội dung bài học**

1. Khái niệm và mục đích sử dụng
2. Các đặc điểm
3. Ví dụ minh họa
4. Bài tập thực hành

Video lý thuyết

Video minh họa

**Khái niệm và mục đích sử dụng**

* Lớp abstract là lớp có chứa keyword abstract trong khai báo lớp.
* Phương thức abstract là phương thức có keyword abstract trong khai báo phương thức.
* Phương thức abstract chỉ chứa khai báo, không chứa phần thân định nghĩa hành vi của phương thức.
* Ví dụ:

package lesson51;

// lớp abstract

public abstract class Shape {

// phương thức abstract

public abstract void draw();

}

* Mục đích của phương thức abstract là nêu ra hành động muốn thực hiện để các lớp con của nó thực hiện. Tức là phương thức này nói sẽ làm gì, cần làm gì nhưng không chỉ rõ làm cách nào, như thế nào.
* Lớp abstract thường chứa các phương thức abstract. Cũng có trường hợp lớp abstract không chứa phương thức abstract nào.
* Lớp abstract thường được thiết kế ra để làm lớp cha chung của một nhóm lớp con trong một hệ thống nào đó kế thừa nó.
* Lớp abstract là các lớp không hoàn thiện và không có nhiều ý nghĩa khi sử dụng đơn độc trong ứng dụng. Ví dụ lớp Shape2D, Vehicle, Animal, ElectrictDevice, Employee…
* Sử dụng lớp abstract nếu muốn đạt được tính trừu tượng trong Java. Tức là nhìn vấn đề một cách tổng thể, thay vì nhìn cụ thể.
* Ví dụ để dùng smartphone, ta chỉ cần sạc đầy, mở khóa màn hình, mở ứng dụng và dùng. Người dùng chỉ quan tâm như thế. Vậy là đủ. Họ chẳng quan tâm khi cắm sạc thì điện đi như nào, các bước để làm điện tích tụ đầy trong pin ra sao, khi nó đầy thì hệ thống làm gì… Rồi khi mở khóa màn hình, người dùng chỉ cần vẽ hình hoặc dùng vân tay, mã số, face id. Họ k cần quan tâm làm như nào hệ thống xử lý được việc xác nhận mật khẩu mở màn hình đó. Làm sao để từ bước đọc vân tay vào từ cảm biến, đưa vào xử lý, lưu trữ kết quả ở đâu, tính toán như nào…. Các abstract class, abtract method và tiếp sau bài này là interface sẽ giúp đơn giản hóa vấn đề.

**Các đặc điểm**

* Abstract method chỉ chứa khai báo không chứa định nghĩa.
* Abstract method không thể chứa access modifier là private.
* Một lớp chứa phương thức abstract bắt buộc phải là lớp abstract. Nhưng một lớp abstract không nhất thiết phải có phương thức abstract.
* Abtract class không thể dùng để tạo đối tượng nhưng có thể dùng làm kiểu của biến để tham chiếu đến các đối tượng của các lớp con của nó.
* Abtract class có thể chứa cả các thành phần đầy đủ của lớp: các trường, phương thức đã thực thi và các phương thức abstract…
* Lớp con kế thừa lớp abstract có 2 tùy chọn:
  + Một là thực thi các phương thức abstract của lớp cha.
  + Hai là trở thành một lớp abstract.
* Ví dụ:

package lesson51;

import java.util.Date;

public abstract class Person {

private String id; // số chứng minh thư/căn cước/hộ chiếu

private String name; // họ tên

private String address; // địa chỉ

private Date dateOfBirth; // ngày sinh

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

// hành động kiếm tiền của người nói chung

abstract void earnMoney();

}

* Ví dụ sau bị lỗi:

package lesson51;

public class Shape { // error, thiếu abstract trong khai báo lớp

public abstract void draw();

}

* Ví dụ sau là implement một lớp abstract:

package lesson51;

import java.util.Date;

public abstract class Person {

private String id; // số chứng minh thư/căn cước/hộ chiếu

private String name; // họ tên

private String address; // địa chỉ

private Date dateOfBirth; // ngày sinh

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

// hành động kiếm tiền của người nói chung

abstract void earnMoney();

}

class Student extends Person {

private String studentId;

private String major;

private float avgGrade;

//...

@Override

void earnMoney() {

System.out.println("Sinh viên kiếm tiền bằng cách đi làm thêm...");

}

}

* **Khi tạo đối tượng để gọi phương thức. Ta thường sử dụng tính đa hình, tức dùng biến kiểu của lớp cha tham chiếu đến đối tượng lớp con.**

package lesson51;

public class Test {

public static void main(String[] args) {

Person person = new Person(); // error

Person nam = new Student(); // ok

nam.earnMoney(); // sinh viên Nam kiếm tiền bằng cách của mình

}

}

**Ví dụ minh họa**

* Ví dụ sau là một ví dụ điển hình của lớp abstract

[Ảnh có chứa văn bản, Phông chữ, hàng, số

Mô tả được tạo tự động](https://braniumacademy.net/wp-content/uploads/2020/08/abstract-class-ex1.png)

* Thực thi chi tiết:

package lesson51;

public abstract class Shape {

public abstract void draw();

}

class Circle extends Shape {

@Override

public void draw() {

System.out.println("Vẽ hình tròn...");

}

}

class Triangle extends Shape {

@Override

public void draw() {

System.out.println("Vẽ tam giác...");

}

}

* Ví dụ đầy đủ về implement lớp Person:

[Ảnh có chứa văn bản, Phông chữ, ảnh chụp màn hình, số

Mô tả được tạo tự động](https://braniumacademy.net/wp-content/uploads/2020/08/abstract-class-ex2.png)

* Thực thi chi tiết:

package lesson51;

import java.util.Date;

public abstract class Person {

private String id; // số chứng minh thư/căn cước/hộ chiếu

private String name; // họ tên

private String address; // địa chỉ

private Date dateOfBirth; // ngày sinh

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

// hành động kiếm tiền của người nói chung

public abstract void earnMoney();

}

class Student extends Person {

private String studentId;

private String major;

private float avgGrade;

//...

@Override

void earnMoney() {

System.out.println("Sinh viên kiếm tiền bằng cách đi làm thêm...");

}

}

class Instructor extends Person {

//...

@Override

void earnMoney() {

System.out.println("Giảng viên kiếm tiền bằng đi dạy");

}

}

class Programmer extends Person {

//...

@Override

void earnMoney() {

System.out.println("Lập trình viên kiếm tiền bằng đi code dạo");

}

}

* Ví dụ chạy chương trình:

package lesson51;

public class Test {

public static void main(String[] args) {

Person nam = new Student();

Person thanh = new Instructor();

Person linh = new Programmer();

// gọi tới hành động của từng đối tượng:

nam.earnMoney();

System.out.println("===============================");

thanh.earnMoney();

System.out.println("===============================");

linh.earnMoney();

}

}

Sinh viên kiếm tiền bằng cách đi làm thêm...

===============================

Giảng viên kiếm tiền bằng đi dạy

===============================

Lập trình viên kiếm tiền bằng đi code dạo

**Bài 9.2. Tổng quan về giao diện – Interfaces**

**Nội dung bài học**

1. Khái niệm và đặc trưng
2. Khai báo interface
3. Thực thi interface
4. Kế thừa interface
5. Ví dụ minh họa

Video lý thuyết

Video minh họa

**Khái niệm và đặc trưng**

* Cùng với class, interface là một kiểu tham chiếu trong Java.
* Interface là một kiểu tập hợp của các phương thức trừu tượng và các hằng số.
* Các thành phần có thể chứa trong interface gồm: các phương thức abstract, các phương thức default, các phương thức static, các phương thức private, private static, các hằng số.
* Interface mặc định là abstract nên ta không cần sử dụng abstract khi khai báo interface.
* Các trường khai báo trong interface mặc định là public static final nên ta có thể bỏ qua ba keyword này khi khai báo trường dữ liệu trong interface.
* Các phương thức trong interface mặc định là public và abstract nên ta bỏ hai keyword này khỏi khai báo phương thức của interface nếu không có gì đặc biệt.
* Sử dụng interface để:
  + Đạt được tính trừu tượng hoàn toàn trong Java.
  + Đạt được mục đích đa kế thừa trong Java.
  + Đạt được tính chất kết nối lỏng lẻo(loose coupling) trong Java.
  + Để giao tiếp giữa các lớp không nằm trong chuỗi kế thừa muốn có chung hành động nào đó.
* Giống như abstract class, interface không thể khởi tạo đối tượng. Interface không có phương thức khởi tạo.
* Một lớp extends một lớp khác nhưng implements 1 hoặc nhiều interface.
* Ta thường tạo biến kiểu của interface sau đó cho nó tham chiếu đến đối tượng lớp triển khai interface đó.

**Khai báo interface**

* Sử dụng keyword interface để khai báo interface.
* Cú pháp tổng quát:

**interface name {**

**// các hằng số**

**// các phương thức private**

**// các phương thức static**

**// các phương thức mặc định**

**// các phương thức trừu tượng**

**}**

* Trong đó:
  + Keyword interface là bắt buộc phải có để khai báo interface.
  + Name là tên của interface. Tên interface có thể là danh từ nếu nó đại diện cho một họ các lớp nào đó như List, Map.
  + Tên interface thường là tính từ và viết hoa chữ cái đầu mỗi từ, ví dụ: Clonable, Iterable, Comparable, Serializable…
  + Các hằng số là dữ liệu của interface.
  + Các phương thức private thường là các phương thức bổ trợ nội bộ của interface.
  + Các phương thức static nhằm định sẵn một chức năng chung không cho phép override.
  + Các phương thức mặc định viết sẵn thực thi cơ bản và có thể được override bởi các lớp implements nó.
  + Các phương thức trừu tượng là thành phần được thực thi bởi các lớp con implement interface.
* Ví dụ về interface:

package lesson52;

interface Drawable {

void draw(); // vẽ

void erase(); // tẩy xóa

}

* Ví dụ về một interface trong hệ thống bắt sự kiện của chuột:

public interface MouseListener extends EventListener {

public void mouseClicked(MouseEvent e);

public void mousePressed(MouseEvent e);

public void mouseReleased(MouseEvent e);

public void mouseEntered(MouseEvent e);

public void mouseExited(MouseEvent e);

}

**Thực thi interface**

* Một lớp chỉ có thể extends 1 lớp cha trực tiếp của nó nhưng có thể implements một hoặc nhiều interface.
* Nếu một lớp thực thi nhiều interface thì các interface sẽ được phân tách nhau bởi dấu phẩy đằng sau keyword implements.
* Chúng ta sử dụng tính đa hình của interface như của lớp cha thông thường.
* Các lớp khi implements interface có 2 lựa chọn:
  + Hoặc là thực thi tất cả các phương thức của interface.
  + Hoặc là thực thi một phần các phương thức của interface và trở thành abstract class.
* Ví dụ implement 1 phần và trở thành abstract class:

package lesson52;

interface Drawable {

void draw(); // vẽ

void erase(); // tẩy xóa

}

abstract class Shape implements Drawable {

@Override

public void draw() {

System.out.println("Drawing a shape...");

}

}

* Ví dụ sau thực thi hoàn chỉnh interface:

package lesson52;

interface Drawable {

void draw(); // vẽ

void erase(); // tẩy xóa

}

class Shape implements Drawable {

@Override

public void draw() {

System.out.println("Drawing a shape...");

}

@Override

public void erase() {

System.out.println("Erasing a shape...");

}

}

* Ví dụ thực thi nhiều interface:

package lesson52;

interface Drawable {

void draw(); // vẽ

void erase(); // tẩy xóa

}

interface Movable {

void up(); // di chuyển lên theo trục y

void down(); // di chuyển xuống theo trục y

void left(); // di chuyển sang trái theo trục x

void right(); // di chuyển sang phải theo trục x

}

// giả định vẽ hình 2D, gốc tọa độ là góc trái trên màn hình

// tọa độ bắt đầu từ (x, y) = (0, 0)

class Shape implements Drawable, Movable {

private int x; // tọa độ x

private int y; // tọa độ y

// ...

@Override

public void draw() {

System.out.println("Drawing a shape...");

}

@Override

public void erase() {

System.out.println("Erasing a shape...");

}

@Override

public void up() {

y--;

}

@Override

public void down() {

y++;

}

@Override

public void left() {

x--;

}

@Override

public void right() {

x++;

}

}

**Kế thừa interface**

* Một interface có thể kế thừa **1 hoặc nhiều interface** khác qua keyword extends.
* Nếu kế thừa nhiều interface, các interface sẽ được liệt kê ở phía bên phải của keyword implements, tách nhau bởi dấu phẩy.
* Ví dụ:

package lesson52;

interface Drawable {

void draw(); // vẽ

void erase(); // tẩy xóa

}

interface Movable {

void up(); // di chuyển lên theo trục y

void down(); // di chuyển xuống theo trục y

void left(); // di chuyển sang trái theo trục x

void right(); // di chuyển sang phải theo trục x

}

interface ShapeDrawer extends Drawable, Movable {

void moveIn(); // di chuyển vào

void moveOut(); // di chuyển ra

void moveDiagonally(); // di chuyển theo đường chéo

}

* Như vậy ta có tổng quát về quan hệ kế thừa giữa lớp và interface:
  + Một lớp có thể extends một lớp khác, implements một hoặc nhiều interface.
  + Một interface có thể extends từ một hoặc nhiều interface khác.

**Ví dụ minh họa**

* Đoạn chương trình sau tạo một đối tượng shape của Shape và thực hiện vẽ hình:

package lesson52;

interface Drawable {

void draw(); // vẽ

void erase(); // tẩy xóa

}

interface Movable {

void up(); // di chuyển lên theo trục y

void down(); // di chuyển xuống theo trục y

void left(); // di chuyển sang trái theo trục x

void right(); // di chuyển sang phải theo trục x

}

// giả định vẽ hình 2D, gốc tọa độ là góc trái trên màn hình

// tọa độ bắt đầu từ (x, y) = (0, 0)

class Shape implements Drawable, Movable {

private int x; // tọa độ x

private int y; // tọa độ y

public Shape() {

}

public Shape(int x, int y) {

this.x = x;

this.y = y;

}

public int getX() {

return x;

}

public void setX(int x) {

this.x = x;

}

public int getY() {

return y;

}

public void setY(int y) {

this.y = y;

}

// ...

@Override

public void draw() {

System.out.println("Drawing a shape...");

}

@Override

public void erase() {

System.out.println("Erasing a shape...");

}

@Override

public void up() {

y--;

}

@Override

public void down() {

y++;

}

@Override

public void left() {

x--;

}

@Override

public void right() {

x++;

}

@Override

public String toString() {

return getClass().getName() + "[" +

"x=" + x +

", y=" + y +

']';

}

}

* Lớp Test.java:

package lesson52;

public class Test {

public static void main(String[] args) {

Shape myShape = new Shape(10, 25);

System.out.println("Trước khi vẽ: " + myShape);

// thực hiện một số hành động di chuyển đơn giản

// trong hệ tọa độ hai chiều

myShape.up();

myShape.right();

myShape.right();

myShape.right();

myShape.up();

System.out.println("Sau khi vẽ: " + myShape);

}

}

Trước khi vẽ: lesson52.Shape[x=10, y=25]

Sau khi vẽ: lesson52.Shape[x=13, y=23]

# Bài 9.3. Các tính năng mới của interface trong Java 8, Java 9+

### Nội dung bài học

1. Các phương thức default
2. Các phương thức static
3. Các phương thức private
4. Bài tập thực hành

Video lý thuyết

Video minh họa

### Các phương thức default

* Các phương thức default được thêm vào interface từ Java 8.
* Trước Java 8 các interface không thể chứa các phương thức được thực thi cụ thể. Tức trong interface thời bấy giờ chỉ có thể chứa các abstract methods.
* Từ phiên bản Java 8 về sau ta có thể thêm các phương thức được thực thi vào interface. Đó là các default methods.
* Việc thêm các phương thức default vào interface cho phép ta bổ sung tính năng của interface. Và việc làm này sẽ không làm ảnh hưởng đến code cũ đã implements interface đó. Tức là nếu ta thêm một phương thức abstract vào interface, toàn bộ các implements cũ trước đó sẽ phải thay đổi để tương thích với những gì được thêm vào. Nhưng với default method thì điều này không xảy ra.
* Các lớp imlements interface lúc này có thêm tùy chọn là override lại phương thức default hoặc không.
* Ví dụ ban đầu ta có interface Calculable chỉ gồm hai phương thức add, sub:

package lesson53;

/\*\*

\* interface chứa các phương thức có thể tính toán số học

\* hai số nguyên a, b

\*/

public interface Calculable {

int add(int a, int b); // cộng hai số

int sub(int a, int b); // trừ hai số

}

/\*\*

\* lớp Calculator implements interface trên

\*/

class Calculator implements Calculable {

@Override

public int add(int a, int b) {

return a + b;

}

@Override

public int sub(int a, int b) {

return a - b;

}

public static void main(String[] args) {

int a = 10;

int b = 25;

Calculable cal = new Calculator();

System.out.println(a + " + " + b + " = " + cal.add(a, b));

System.out.println(a + " - " + b + " = " + cal.sub(a, b));

}

}

10 + 25 = 35

10 - 25 = -15

* Sau đó ta muốn bổ sung tính năng nhân, chia cho interface trên. Nếu ta thêm theo kiểu truyền thống sẽ làm các code cũ phải implements lại interface. Còn nếu thêm theo kiểu default method thì không ảnh hưởng đến code cũ.

package lesson53;

/\*\*

\* interface chứa các phương thức có thể tính toán số học

\* hai số nguyên a, b

\*/

public interface Calculable {

int add(int a, int b); // cộng hai số

int sub(int a, int b); // trừ hai số

default float div(int a, int b) { // chia

return 1.0f \* a / b;

}

default int mul(int a, int b) { // nhân

return a \* b;

}

}

/\*\*

\* lớp Calculator implements interface trên

\*/

class Calculator implements Calculable {

@Override

public int add(int a, int b) {

return a + b;

}

@Override

public int sub(int a, int b) {

return a - b;

}

public static void main(String[] args) {

int a = 10;

int b = 25;

Calculable cal = new Calculator();

System.out.println(a + " + " + b + " = " + cal.add(a, b));

System.out.println(a + " - " + b + " = " + cal.sub(a, b));

// gọi phương thức default:

System.out.println(a + " \* " + b + " = " + cal.mul(a, b));

System.out.println(a + " / " + b + " = " + cal.div(a, b));

}

}

10 + 25 = 35

10 - 25 = -15

10 \* 25 = 250

10 / 25 = 0.4

* Tất nhiên trong lớp implements interface đó có thể override lại phương thức default này:

class Calculator implements Calculable {

@Override

public float div(int a, int b) { // override lại

if (b == 0) {

System.out.println("Mẫu số phải khác 0");

return Float.NaN;

}

return a \* 1.0f / b;

}

// ...

#### Kế thừa interface có phương thức default

Khi kế thừa interface có phương thức default ta có 3 tùy chọn:

* Bỏ qua các phương thức default của interface cha. Điều này làm cho interface con kế thừa lại các phương thức default của interface cha.
* Khai báo lại các phương thức default của interface cha. Điều này làm phương thức default của lớp cha trở thành abstract method trong interface con.
* Định nghĩa lại phương thức default của interface cha. Điều này làm cho interface con override lại phương thức default của interface cha.
* Ví dụ:

package lesson53;

/\*\*

\* interface chứa các phương thức có thể tính toán số học

\* hai số nguyên a, b

\*/

public interface Calculable {

int add(int a, int b); // cộng hai số

int sub(int a, int b); // trừ hai số

default float div(int a, int b) { // chia

return 0;

}

default int mul(int a, int b) { // nhân

return 1;

}

}

interface OtherCalculable extends Calculable {

@Override

default float div(int a, int b) { // override

return a \* 1.0f / b;

}

int mul(int a, int b); // mul trở thành abstract method

}

/\*\*

\* lớp Calculator implements interface trên

\*/

class Calculator implements OtherCalculable {

@Override

public float div(int a, int b) { // override lại

if (b == 0) {

System.out.println("Mẫu số phải khác 0");

return Float.NaN;

}

return a \* 1.0f / b;

}

@Override

public int mul(int a, int b) { // bắt buộc phải implement

return a \* b \* 2;

}

@Override

public int add(int a, int b) {

return a + b;

}

@Override

public int sub(int a, int b) {

return a - b;

}

public static void main(String[] args) {

int a = 10;

int b = 25;

Calculable cal = new Calculator();

System.out.println(a + " + " + b + " = " + cal.add(a, b));

System.out.println(a + " - " + b + " = " + cal.sub(a, b));

// gọi phương thức default:

System.out.println(a + " \* " + b + " = " + cal.mul(a, b));

System.out.println(a + " / " + b + " = " + cal.div(a, b));

}

}

10 + 25 = 35

10 - 25 = -15

10 \* 25 = 500

10 / 25 = 0.4

#### Khi một lớp implements nhiều interface có cùng phương thức default với cùng tham số, kiểu trả về.

* Lớp này khi sử dụng các default method sẽ phải chỉ rõ muốn dùng của interface nào.

package lesson53;

public interface Interface1 {

default void show(String msg) {

System.out.println("Gọi show() của Interface1: " + msg);

}

}

interface Interface2 {

default void show(String msg) {

System.out.println("Gọi show() của Interface2: " + msg);

}

}

class Example implements Interface1, Interface2 {

public static void main(String[] args) {

Example example = new Example();

var msg = "I'm a developer";

example.show(msg);

}

@Override

public void show(String msg) {

Interface1.super.show(msg);

Interface2.super.show(msg);

System.out.println("Gọi show() của lớp Example: " + msg);

}

}

Gọi show() của Interface1: I'm a developer

Gọi show() của Interface2: I'm a developer

Gọi show() của lớp Example: I'm a developer

### Các phương thức static

* Nếu một interface có chứa một hành vi nào đó là cố định không thay đổi ở các interface con của nó hoặc các lớp implement nó. Ta dùng static method.
* Giống như các static method trong các lớp, ta gọi và sử dụng phương thức static trong interface qua tên Interface.
* Static method trong interface thuộc sở hữu của interface. Nó không thể bị override.
* Thường sử dụng static method làm các phương thức bổ trợ hoặc phương thức tiện ích cho interface.
* Ví dụ:

package lesson53;

import java.util.Arrays;

public interface Splitable {

// phương thức tách và trả về mảng các từ trong chuỗi

// với dấu hiệu phân tách là vị trí có dấu cách

static String[] splitByWhitespace(String src) {

if (src == null) {

return null;

}

return src.split("\\s+");

}

// phương thức tách lấy tên của một người từ String cho trước

static String getFirstName(String fullName) {

if (fullName == null) {

return null;

}

var index = fullName.indexOf(" ");

return fullName.substring(0, index);

}

// một phương thức abstract

String[] split(String src, String regex);

}

class Example2 implements Splitable {

@Override

public String[] split(String src, String regex) {

if (src == null) {

return null;

}

return src.split(regex);

}

public static void main(String[] args) {

Example2 example = new Example2();

var result = example.split("I love You", " ");

System.out.println(Arrays.deepToString(result));

// gọi phương thức static:

var fullName = "Trung Trần";

System.out.println("Tên trong \""

+ fullName + "\": " + Splitable.getFirstName(fullName));

}

}

[I, love, You]

Tên trong "Trung Trần": Trung

### Các phương thức private

* Là phương thức được khai báo với keyword private trong interface.
* Mục đích sử dụng: chỉ sử dụng được trong nội bộ interface. Làm tăng tính đóng gói và tái sử dụng code trong interface. Có thể dùng để làm helper methods.
* Quy tắc sử dụng phương thức private trong interface:
  + Phương thức private không thể là phương thức trừu tượng.
  + Phương thức private chỉ có thể được sử dụng trong nội bộ interface bởi các phương thức không trừu tượng.
  + Phương thức private non-static không thể được sử dụng trong phương thức private static.
* Ví dụ:

package lesson53;

public interface Splitable {

// phương thức private static

private static String[] splitByWhitespace(String src) {

if (src == null) {

return null;

}

return src.split("\\s+");

}

// phương thức tách lấy tên của một người từ String cho trước

static String getFirstName(String fullName) {

if (fullName == null) {

return null;

}

// gọi phương thức private static để tách các từ trong tên

var words = splitByWhitespace(fullName);

return words[0]; // giả định tên đứng đầu tiên trong họ tên

}

}

class Example2 implements Splitable {

public static void main(String[] args) {

var fullName = "Tony Stark";

System.out.println("Tên trong \""

+ fullName + "\": " + Splitable.getFirstName(fullName));

}

}

Tên trong "Tony Stark": Tony

**Bài 9.4. Thực hành: Tự định nghĩa Interface**

**Bài học này là phần thực hành tạo interface quản lý tài khoản đơn giản**

**Bài tập thực hành ở cuối bài.**

Code mẫu: [**Click vào đây để xem code đầy đủ.**](https://github.com/thantrieu/java_course1_exercises/tree/master/B%C3%A0i%20th%E1%BB%B1c%20h%C3%A0nh%20t%E1%BA%A1o%20interface/practice_interface)

**Một số lớp chính:**

* Interface: AccountManagerment.java

package practice\_interface;

/\*\*

\* Thiết kế interface cho phép quản lý tài khoản gồm các thao tác:

\* đăng kí

\* đăng nhập

\* đăng xuất

\* xóa tài khoản

\* ->add: vô hiệu hóa TK

\* khôi phục tài khoản

\* đổi mật khẩu

\* ->add: cấp lại mật khẩu

\* chỉnh sửa thông tin tài khoản

\*/

public interface AccountManagerment {

boolean signup(Account account);

boolean login(Account account);

void logout(Account account);

boolean removeAccount(Account account);

boolean recoverAccount(Account account);

boolean changePassword(Account account);

void changeAccountInfo(Account account);

default void deactiveAccount(Account account) {

System.out.println("Tài khoản của bạn đã vô hiệu hóa");

}

default String randomPassword(Account account) {

return "";

}

}

* Lớp thực thi: AccountManagermentImp.java

package practice\_interface;

import java.util.ArrayList;

public class AccountManagermentImp implements AccountManagerment {

private ArrayList<Account> accounts;

public AccountManagermentImp() {

accounts = new ArrayList<>();

}

public AccountManagermentImp(ArrayList<Account> accounts) {

this.accounts = accounts;

}

public ArrayList<Account> getAccounts() {

return accounts;

}

public void setAccounts(ArrayList<Account> accounts) {

this.accounts = accounts;

}

@Override

public boolean signup(Account account) {

if (!isExist(account)) {

accounts.add(account);

return true;

}

return false;

}

private boolean isExist(Account account) {

for (var acc : accounts) {

if (acc.getUsername().compareTo(account.getUsername()) == 0) {

return true;

}

}

return false;

}

@Override

public boolean login(Account account) {

// for (var acc : accounts) {

// if (acc.equals(account)) {

// return true;

// }

// }

// code mới để test:

for (int i = 0; i < accounts.size(); i++) {

if (accounts.get(i).equals(account)

&& accounts.get(i) instanceof ActiveAccount) {

var activeAcc = (ActiveAccount) accounts.get(i);

activeAcc.setOnline(true); // bật trạng thái online về false

return true; // kết thúc

}

}

return false;

}

@Override

public void logout(Account account) {

for (int i = 0; i < accounts.size(); i++) {

if (accounts.get(i).equals(account)

&& accounts.get(i) instanceof ActiveAccount) {

var activeAcc = (ActiveAccount) accounts.get(i);

activeAcc.setOnline(false); // bật trạng thái online về false

break; // kết thúc

}

}

}

@Override

public boolean removeAccount(Account account) {

for (int i = 0; i < accounts.size(); i++) {

if (account.equals(accounts.get(i))) {

accounts.remove(i);

return true;

}

}

return false;

}

/\*\*

\* Phương thức kích hoạt lại tài khoản đang vô hiệu hóa.

\* Điều kiện active tk là nhập đúng mật khẩu và tài khoản

\* chưa bị xóa khỏi hệ thống.

\*

\* @param account

\* @return

\*/

@Override

public boolean recoverAccount(Account account) {

for (int i = 0; i < accounts.size(); i++) {

if (accounts.get(i).equals(account)

&& accounts.get(i) instanceof ActiveAccount) {

var activeAcc = (ActiveAccount) accounts.get(i);

activeAcc.setActive(true); // khôi phục lại tài khoản

return true;

}

}

return false;

}

/\*\*

\* phương thức thiết lập lại mật khẩu

\*

\* @param account

\* @return

\*/

@Override

public boolean changePassword(Account account) {

for (int i = 0; i < accounts.size(); i++) {

var acc = accounts.get(i);

if (acc.equals(account)) {

acc.setPassword(account.getPassword());

// your implement...

return true; // xong thì thoát vì chỉ có một tk thỏa mãn đk

}

}

return false;

}

/\*\*

\* Phương thức dùng để cập nhật thông tin TTK

\* giả sử ta chỉ update thông tin email và số ĐT.

\* Bạn có thể tự bổ sung cập nhật các thông tin khác.

\* Lưu ý không được đổi thông tin username

\*

\* @param account

\*/

@Override

public void changeAccountInfo(Account account) {

for (int i = 0; i < accounts.size(); i++) {

var acc = accounts.get(i);

if (acc.equals(account)) {

acc.setEmail(account.getEmail());

acc.setPhoneNumber(account.getPhoneNumber());

// your implement...

break; // xong thì thoát vì chỉ có một tk thỏa mãn đk

}

}

}

}

* Lớp Account.java:

package practice\_interface;

import java.util.Objects;

/\*\*

\* Thông tin tài khoản gồm mã tài khoản, username,

\* mật khẩu, họ, đệm, tên, địa chỉ, email, số điện thoại

\*/

public class Account {

private String id;

private String username;

private String password;

private FullName fullName;

private String address;

private String email;

private String phoneNumber;

public Account() {

}

public Account(String id, String username, String password) {

this.id = id;

this.username = username;

this.password = password;

}

public Account(String id, String username, String password,

FullName fullName, String address, String email,

String phoneNumber) {

this.id = id;

this.username = username;

this.password = password;

this.fullName = fullName;

this.address = address;

this.email = email;

this.phoneNumber = phoneNumber;

}

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

public FullName getFullName() {

return fullName;

}

public void setFullName(FullName fullName) {

this.fullName = fullName;

}

public String getAddress() {

return address;

}

public void setAddress(String address) {

this.address = address;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public String getPhoneNumber() {

return phoneNumber;

}

public void setPhoneNumber(String phoneNumber) {

this.phoneNumber = phoneNumber;

}

@Override

public boolean equals(Object otherObject) {

if (this == otherObject) return true;

if (otherObject == null

|| getClass() != otherObject.getClass()) return false;

Account other = (Account) otherObject;

return Objects.equals(username, other.username) &&

Objects.equals(password, other.password);

}

@Override

public int hashCode() {

return Objects.hash(username, password);

}

class FullName {

private String first;

private String last;

private String mid;

public FullName(String first, String last, String mid) {

this.first = first;

this.last = last;

this.mid = mid;

}

public String getFirst() {

return first;

}

public void setFirst(String first) {

this.first = first;

}

public String getLast() {

return last;

}

public void setLast(String last) {

this.last = last;

}

public String getMid() {

return mid;

}

public void setMid(String mid) {

this.mid = mid;

}

}

}

[**Click vào đây để xem code đầy đủ.**](https://github.com/thantrieu/java_course1_exercises/tree/master/B%C3%A0i%20th%E1%BB%B1c%20h%C3%A0nh%20t%E1%BA%A1o%20interface/practice_interface)

* Kết quả chạy chương trình:

Sau khi đăng nhập:

Tài khoản "rickkid2005" đang online

Sau khi đăng xuất:

Tài khoản "rickkid2005" đang offline

# Bài 9.5. Lớp vô danh – anonymous class

### Nội dung bài học

1. Khái niệm và đặc điểm
2. Các loại lớp vô danh
3. Lưu ý

Video lý thuyết

Video minh họa

### Khái niệm và đặc điểm

* Lớp vô danh(anonymous class) là một inner class không có tên và chỉ dùng để tạo một đối tượng.
* Lớp vô danh không phải là thành phần của bất kì lớp nào.
* Lớp vô danh hữu ích trong trường hợp ta muốn tạo đối tượng của lớp mà không cần tạo lớp. Tức tạo đồng thời cả lớp và đối tượng tại một thời điểm.
* Thường sử dụng để tạo đối tượng thực thi của abstract class, interface. Ví dụ như tạo các đối tượng thực thi interface lắng nghe sự kiện trong lập trình giao diện.
* Lớp vô danh thường được tạo bằng hai cách:
  + Qua các lớp, gồm cả lớp thông thường và lớp abstract.
  + Qua các interface.
* Cú pháp:

Type objectName = new Type() {

// các dữ liệu cần thiết

// các phương thức

...

};

* Ví dụ:

package lesson54;

/\*\*

\* interface chứa các phương thức có thể tính toán số học

\* hai số nguyên a, b

\*/

public interface Calculable {

int add(int a, int b); // cộng hai số

int sub(int a, int b); // trừ hai số

default float div(int a, int b) { // chia hai so

return 0;

}

default int mul(int a, int b) {

return 1;

}

}

class Example {

public static void main(String[] args) {

Calculable myCalculator = new Calculable() {

@Override

public int add(int a, int b) {

return a + b;

}

@Override

public int sub(int a, int b) {

return a - b;

}

};

// sử dụng đối tượng tạo ra từ lớp vô danh:

System.out.println(myCalculator.add(100, 250));

System.out.println(myCalculator.sub(200, 125));

}

}

350

75

### Các loại lớp vô danh

#### Lớp vô danh kế thừa một lớp khác

* Ta có thể tạo lớp vô danh kế thừa một lớp thông thường hoặc tạo đối tượng của abstract class.
* Ví dụ sau tạo đối tượng kế thừa lớp thông thường:

package lesson54;

public class Father {

public void speak() {

System.out.println("Father speaking English");

}

}

class FatherExample {

public static void main(String[] args) {

// tạo lớp vô danh kế thừa lớp Father

Father son = new Father() {

@Override

public void speak() {

super.speak(); // gọi phương thức của lớp cha

System.out.println("Son is speaking Vietnamese");

}

};

son.speak();

}

}

Father speaking English

Son is speaking Vietnamese

* Ví dụ sau tạo lớp vô danh kế thừa một abstract class:

package lesson54;

public abstract class Animal {

private String name;

public Animal(String name) {

this.name = name;

}

protected abstract void move(); // di chuyển

protected abstract void eat(); // ăn

public String getName() {

return name;

}

}

class AnimalExample {

public static void main(String[] args) {

Animal cat = new Animal("Tom") {

@Override

protected void move() {

System.out.println("Mèo " + getName() + " đang chạy trên nóc nhà");

}

@Override

protected void eat() {

System.out.println("Mèo " + getName() + " đang ăn cá rán giòn");

}

};

System.out.println("Tên mèo là: " + cat.getName());

cat.move();

cat.eat();

}

}

* Kết quả:

Tên mèo là: Tom

Mèo Tom đang chạy trên nóc nhà

Mèo Tom đang ăn cá rán giòn

#### Lớp vô danh thực thi interface

* Ta có thể tạo lớp vô danh thực thi interface. Điều này rất phổ biến và thường thấy khi muốn tạo đối tượng mà không cần tạo lớp cụ thể thực thi interface.
* Ví dụ ta tạo lớp vô danh thực thi một interface:

package lesson54;

public interface Speakable {

void speak(); // nói chuyện

void whisper(); // nói thì thầm

}

class SpeakableExample {

public static void main(String[] args) {

// tạo lớp vô danh từ interface

Speakable speakable = new Speakable() {

@Override

public void speak() {

System.out.println("Hello...");

}

@Override

public void whisper() {

System.out.println("I love you...");

}

};

// sử dụng đối tượng speakable

speakable.speak();

speakable.whisper();

}

}

Hello...

I love you...

#### Lớp vô danh trong tham số của phương thức

* Ta có thể tạo ra lớp anonymous trong tham số của một phương thức.
* Thường sử dụng trong lập trình giao diện, tạo các callback.
* Ví dụ:

package lesson54;

public interface Speakable {

void speak(); // nói chuyện

void whisper(); // nói thì thầm

}

class Speaking {

private Speakable mSpeakable;

public Speaking(Speakable speakable) {

mSpeakable = speakable;

}

public void doSpeak() {

System.out.println("Inside class Speaking...");

mSpeakable.speak();

mSpeakable.whisper();

}

}

class SpeakableExample {

public static void main(String[] args) {

// tạo đối tượng của lớp với tham số là lớp vô danh

Speaking speaking = new Speaking(new Speakable() {

@Override

public void speak() {

System.out.println("Hello...");

}

@Override

public void whisper() {

System.out.println("I love you...");

}

});

// thực hiện hành động

speaking.doSpeak();

}

}

Inside class Speaking...

Hello...

I love you...

### Lưu ý

* Một lớp vô danh chỉ có thể kế thừa 1 lớp tại một thời điểm. Nó không thể vừa kế thừa lớp vừa implements interface.
* Một lớp vô danh chỉ có thể implements một interface. Lớp thông thường có thể implements nhiều interface.
* Anonymous class không thể chứa constructor vì nó không có tên.
* Một lớp vô danh có thể truy cập các thành phần của lớp chứa nó.
* Lớp vô danh có thể truy cập các biến cục bộ trong phạm vi nó được khai báo.

public class Anonymous {

// ...

private void doSomething() {

int age = 20;

// lớp vô danh cso thể truy cập vào biến cục bộ của phương thức chứa nó

((X) () -> System.out.println(age)).print();

}

public interface X {

void print();

}

}

* Không thể khai báo khối khởi tạo static hoặc interface trong lớp vô danh.
* Lớp vô danh có thể chứa các trường, hằng số static, phương thức, các local class khác.

**Bài 9.6. Một số interface thường dùng trong sắp xếp**

**Nội dung bài học**

1. Interface Comparable
2. Interface Comparator
3. Ví dụ minh họa
4. Bài tập thực hành

Video lý thuyết

Video minh họa

**Interface Comparable**

* **Nếu muốn cho phép đối tượng cùng lớp có thể so sánh với nhau theo chỉ một tiêu chí nào đó nhằm mục đích sắp xếp, ta cho lớp đó triển khai interface Comparable.**
* **Ví dụ điển hình các lớp bao của các kiểu nguyên thủy: Integer, Float, Long, Double, Char, …; kiểu String mặc định đã triển khai sẵn Comparable<T> do đó chúng ta có thể so sánh các giá trị kiểu số, các chuỗi kí tự với nhau.**
* Các kiểu collection như SortedSet, TreeSet,… yêu cầu các đối tượng lưu trữ trong nó phải triển khai Comparable<T>.
* Interface này chỉ có duy nhất 1 abstract method cần thực thi là compareTo(T other).
* Giá trị trả về bởi phương thức compareTo() này sẽ có 3 khả năng:
  + Nếu giá trị cần so sánh của thuộc tính gốc nhỏ hơn giá trị của thuộc tính của other, trả về giá trị âm.
  + Nếu hai giá trị của thuộc tính cần so sánh và thuộc tính của other trùng nhau, trả về 0.
  + Nếu giá trị của thuộc tính gốc lớn hơn giá trị của thuộc tính trong other, trả về giá trị dương.
* Khi muốn sắp xếp tăng dần ta lấy thuộc tính của đối tượng hiện thời so sánh với thuộc tính của đối tượng other.
* Khi muốn sắp xếp tăng dần ta lấy thuộc tính của đối tượng other so sánh với thuộc tính của đối tượng hiện thời.
* Sử dụng phương thức sort(list) của Collections để sắp xếp.
* Ví dụ sau sắp xếp theo tên tăng dần a-z:

public class Movie implements Comparable<Movie> {

private String name;

private int year;

private String director;

private float price;

private float star;

public Movie() {

}

public Movie(String name, int year,

String director, float price, float star) {

this.name = name;

this.year = year;

this.director = director;

this.price = price;

this.star = star;

}

//...

@Override

public int compareTo(Movie o) {

return name.compareTo(o.name);

}

}

* Lớp chạy chương trình:

public class Test {

public static void main(String[] args) {

ArrayList<Movie> movies = new ArrayList<>();

movies.add(new Movie("Thor", 2011, "ABC", 59, 4.3f));

movies.add(new Movie("Iron man 3", 2013, "DEF", 79, 4.5f));

movies.add(new Movie("Ant man", 2015, "MNO", 75, 4.5f));

movies.add(new Movie("Captain Marvel", 2019, "GHI", 59, 4.7f));

movies.add(new Movie("Avengers: Infinity War", 2018, "ONL", 50, 4.8f));

movies.add(new Movie("Avengers: End game", 2019, "XYZ", 99, 4.9f));

System.out.println("===> Trước khi sắp xếp: ");

showMovies(movies);

System.out.println("===> Sau khi sắp xếp theo tên: ");

Collections.sort(movies);

showMovies(movies);

}

private static void showMovies(ArrayList<Movie> movies) {

System.out.printf("%-25s%-10s%-10s%-10s%-10s\n",

"Tên phim", "Năm SX", "Đạo diễn", "giá vé", "Số sao");

for (var movie : movies) {

System.out.printf("%-25s%-10d%-10s%-10.2f%-10.2f\n",

movie.getName(), movie.getYear(), movie.getDirector(),

movie.getPrice(), movie.getStar());

}

}

}

===> Trước khi sắp xếp:

Tên phim Năm SX Đạo diễn giá vé Số sao

Thor 2011 ABC 59.00 4.30

Iron man 3 2013 DEF 79.00 4.50

Ant man 2015 MNO 75.00 4.50

Captain Marvel 2019 GHI 59.00 4.70

Avengers: Infinity War 2018 ONL 50.00 4.80

Avengers: End game 2019 XYZ 99.00 4.90

===> Sau khi sắp xếp theo tên:

Tên phim Năm SX Đạo diễn giá vé Số sao

Ant man 2015 MNO 75.00 4.50

Avengers: End game 2019 XYZ 99.00 4.90

Avengers: Infinity War 2018 ONL 50.00 4.80

Captain Marvel 2019 GHI 59.00 4.70

Iron man 3 2013 DEF 79.00 4.50

Thor 2011 ABC 59.00 4.30

**Interface Comparator**

* **Để cho phép thuận tiện sắp xếp danh sách đối tượng theo nhiều tiêu chí khác nhau, ta dùng interface Comparator.**
* Interface này cho phép ta tạo lớp thực thi độc lập bên ngoài lớp chứa đối tượng cần sắp xếp.
* Ta override lại phương thức compare để tiến hành thiết lập quy chuẩn thứ tự sắp đặt đối tượng.
* Phương thức compare(T o1, T o2) nhận vào hai tham số cùng kiểu T và trả về một số nguyên: âm, bằng 0 hoặc dương.
  + Âm trong trường hợp thuộc tính gốc có giá trị nhỏ hơn giá trị của thuộc tính cần so sánh.
  + Bằng 0 nếu hai thuộc tính có giá trị trùng khớp.
  + Dương nếu giá trị thuộc tính gốc lớn hơn giá trị thuộc tính cần so sánh.
* Nếu muốn sắp xếp theo thứ tự tăng dần, ta cho thuộc tính của tham số thứ nhất so sánh với thuộc tính của tham số thứ hai.
* Nếu muốn sắp xếp theo thứ tự giảm dần, ta cho thuộc tính của tham số thứ hai so sánh với thuộc tính của tham số thứ nhất.
* Khi tiến hành sắp xếp ta sử dụng phương thức sort(list, comparator) của Collections. Phương thức này nhận vào hai tham số, tham số thứ nhất là danh sách cần sắp xếp. Tham số thứ hai là đối tượng của lớp thực thi interface Comparator sử dụng kiểu đang cần sắp xếp trong danh sách của tham số thứ nhất.
* Ví dụ: file Movie.java.

public class Movie {

private String name;

private int year;

private String director;

private float price;

private float star;

public Movie() {

}

public Movie(String name, int year,

String director, float price, float star) {

this.name = name;

this.year = year;

this.director = director;

this.price = price;

this.star = star;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public int getYear() {

return year;

}

public void setYear(int year) {

this.year = year;

}

//...

}

* Lớp thực thi interface Comparator và lớp Test:

package lesson55;

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

public class Test {

public static void main(String[] args) {

ArrayList<Movie> movies = new ArrayList<>();

movies.add(new Movie("Thor", 2011, "ABC", 59, 4.3f));

movies.add(new Movie("Iron man 3", 2013, "DEF", 79, 4.5f));

movies.add(new Movie("Ant man", 2015, "MNO", 75, 4.5f));

movies.add(new Movie("Captain Marvel", 2019, "GHI", 59, 4.7f));

movies.add(new Movie("Avengers: Infinity War", 2018, "ONL", 50, 4.8f));

movies.add(new Movie("Avengers: End game", 2019, "XYZ", 99, 4.9f));

System.out.println("===> Trước khi sắp xếp: ");

showMovies(movies);

System.out.println("===> Sau khi sắp xếp theo tên: ");

Collections.sort(movies, new SortByName());

showMovies(movies);

System.out.println("===> Sau khi sắp xếp theo năm sản xuất: ");

Collections.sort(movies, new SortByYear());

showMovies(movies);

}

private static void showMovies(ArrayList<Movie> movies) {

System.out.printf("%-25s%-10s%-10s%-10s%-10s\n",

"Tên phim", "Năm SX", "Đạo diễn", "giá vé", "Số sao");

for (var movie : movies) {

System.out.printf("%-25s%-10d%-10s%-10.2f%-10.2f\n",

movie.getName(), movie.getYear(), movie.getDirector(),

movie.getPrice(), movie.getStar());

}

}

}

/\*\*

\* Lớp này thực thi việc sắp xếp theo tên phim

\*/

class SortByName implements Comparator<Movie> {

@Override

public int compare(Movie o1, Movie o2) {

return o1.getName().compareTo(o2.getName());

}

}

/\*\*

\* lớp này thực thi việc sắp xếp theo năm sản xuất

\*/

class SortByYear implements Comparator<Movie> {

@Override

public int compare(Movie o1, Movie o2) {

return o1.getYear() - o2.getYear();

}

}

* Kết quả:

===> Trước khi sắp xếp:

Tên phim Năm SX Đạo diễn giá vé Số sao

Thor 2011 ABC 59.00 4.30

Iron man 3 2013 DEF 79.00 4.50

Ant man 2015 MNO 75.00 4.50

Captain Marvel 2019 GHI 59.00 4.70

Avengers: Infinity War 2018 ONL 50.00 4.80

Avengers: End game 2019 XYZ 99.00 4.90

===> Sau khi sắp xếp theo tên:

Tên phim Năm SX Đạo diễn giá vé Số sao

Ant man 2015 MNO 75.00 4.50

Avengers: End game 2019 XYZ 99.00 4.90

Avengers: Infinity War 2018 ONL 50.00 4.80

Captain Marvel 2019 GHI 59.00 4.70

Iron man 3 2013 DEF 79.00 4.50

Thor 2011 ABC 59.00 4.30

===> Sau khi sắp xếp theo năm sản xuất:

Tên phim Năm SX Đạo diễn giá vé Số sao

Thor 2011 ABC 59.00 4.30

Iron man 3 2013 DEF 79.00 4.50

Ant man 2015 MNO 75.00 4.50

Avengers: Infinity War 2018 ONL 50.00 4.80

Avengers: End game 2019 XYZ 99.00 4.90

Captain Marvel 2019 GHI 59.00 4.70

**Ví dụ minh họa**

* File Movie.java đầy đủ của ví dụ phần Comparable:

package lesson55;

public class Movie implements Comparable<Movie> {

private String name;

private int year;

private String director;

private float price;

private float star;

public Movie() {

}

public Movie(String name, int year,

String director, float price, float star) {

this.name = name;

this.year = year;

this.director = director;

this.price = price;

this.star = star;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public int getYear() {

return year;

}

public void setYear(int year) {

this.year = year;

}

public String getDirector() {

return director;

}

public void setDirector(String director) {

this.director = director;

}

public float getPrice() {

return price;

}

public void setPrice(float price) {

this.price = price;

}

public float getStar() {

return star;

}

public void setStar(float star) {

this.star = star;

}

@Override

public int compareTo(Movie o) {

return name.compareTo(o.name);

}

}

* File Movie.java đầy đủ của ví dụ phần Comparator:

package lesson55;

public class Movie {

private String name;

private int year;

private String director;

private float price;

private float star;

public Movie() {

}

public Movie(String name, int year,

String director, float price, float star) {

this.name = name;

this.year = year;

this.director = director;

this.price = price;

this.star = star;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public int getYear() {

return year;

}

public void setYear(int year) {

this.year = year;

}

public String getDirector() {

return director;

}

public void setDirector(String director) {

this.director = director;

}

public float getPrice() {

return price;

}

public void setPrice(float price) {

this.price = price;

}

public float getStar() {

return star;

}

public void setStar(float star) {

this.star = star;

}

}

**Bài 9.7. So sánh abstract class với interface**

**Nội dung bài học**

**\*\* Đây là câu hỏi phỏng vấn kinh điển về lập trình OOP các bạn lưu ý nắm chắc nhé!**

1. Điểm chung của abstract class vs interface
2. Sự khác nhau của abstract class và interface
3. Khi nào sử dụng abstract class?
4. Khi nào sử dụng interface?

Cả abstract class và interface đều liên quan đến tính chất trừu tượng trong Java. Tính trừu tượng có thể hiểu là việc ẩn giấu các thực thi chi tiết của các tính năng và chỉ cung cấp cho nơi sử dụng nó các chức năng được nhìn ở góc độ hoàn chỉnh. Tức là chỉ public ra ngoài các tính năng đó làm gì, không public cách thực hiện cụ thể.

**Điểm chung của abstract class vs interface**

* Cả hai đều là các kiểu tham chiếu của Java.
* Cả hai đều có thể dùng để đạt được tính trừu tượng trong Java.
* Cả hai có thể chứa dữ liệu là hằng số: các trường public static final.
* Cả hai đều có thể được sử dụng trong kế thừa.
* Cả hai có thể chứa các phương thức đã thực thi chi tiết.
* Cả hai không hỗ trợ default constructor.
* Cả hai đều không thể khởi tạo. Tức không thể tạo đối tượng của abstract class hay interface với keyword new.

**Sự khác nhau của abstract class và interface**

|  |  |  |
| --- | --- | --- |
| **Tiêu chí** | **Abstract class** | **interface** |
| Keyword abstract | Phải khai báo tường minh | Ngầm định là abstract nên ta không cần khai báo abstract trong interface |
| Loại phương thức | Có thể chứa tất cả các loại phương thức: từ constructors, static, non-static, abstract | Chỉ có thể chứa các phương thức abstract, default, static, private |
| Dữ liệu | Có thể chứa tất cả các loại dữ liệu: hằng số, không phải hằng số, static, non-static. | Chỉ có thể chứa hằng số public static final. |
| Sự triển khai lẫn nhau | Abstract class có thể cung cấp các triển khai cho interface | Interface không thể cung cấp các triển khai cho abstract class |
| Đa kế thừa | Abstract class chỉ có thể kế thừa 1 lớp cha trực tiếp duy nhất | Inteface có thể kế thừa nhiều interface khác nhau |
| Đa thực thi | Abstract class có thể extends class khác và implements nhiều interface tùy ý | Interface chỉ có thể extends các interface |
| Access modifier | Có thể chứa các thành phần public, protected, private, private package access | Mặc định các thành phần của interface là public. Với phương thức thì từ Java 9 có thêm thành phần private |
| Constructors | Có thể chứa các constructor có hoặc không có tham số nhưng không chứa default constructor | Không hỗ trợ bất kì kiểu constructor nào |
| Mục đích sử dụng | Được thiết kế để làm lớp cha chung trong một chuỗi kế thừa của các lớp có liên quan đến nhau nào đó. | Được thiết kế để cho phép các lớp không liên quan gì đến nhau nhưng muốn sử dụng chung một số chức năng nào đó. |

* Ví dụ về abstract class:

package lesson56;

public abstract class Shape {

// hằng số:

public static final double MAX\_AREA = 500;

// các thuộc tính khác:

private String name; // tên

private int numOfEdge; // số cạnh

// phương thức abstract

public abstract void draw();

// các phương thức thông thường:

public final String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

}

// các lớp kế thừa lớp abstract

class Triangle extends Shape {

//...

@Override

public void draw() {

System.out.println("Vẽ hình tròn....");

}

}

class Circle extends Shape {

//...

@Override

public void draw() {

System.out.println("Vẽ hình chữ nhật...");

}

}

* Ví dụ về interface và abstract class implements nhiều interface(đa kế thừa):

package lesson56;

// chứa các hành động có thể để vẽ

interface Drawable {

void draw(); // vẽ

void erase(); // tẩy xóa

}

// chứa các hành động có thể để di chuyển trên màn hình

interface Movable {

void up(); // di chuyển lên theo trục y

void down(); // di chuyển xuống theo trục y

void left(); // di chuyển sang trái theo trục x

void right(); // di chuyển sang phải theo trục x

}

// bổ sung thêm những hành động di chuyển mới

interface ShapeDrawer extends Drawable, Movable {

void moveIn(); // di chuyển vào

void moveOut(); // di chuyển ra

void moveDiagonally(); // di chuyển theo đường chéo

}

// lớp abtract kế thừa Shape và implement ShapeDrawer:

abstract class MyShape extends Shape implements ShapeDrawer {

private int x;

private int y;

public MyShape(int x, int y) {

this.x = x;

this.y = y;

}

@Override

public void draw() { // thực thi phương thức của inteface

up();

up();

left();

down();

}

@Override

public void up() {

y--;

}

@Override

public void down() {

y++;

}

@Override

public void left() {

x--;

}

}

**Khi nào sử dụng abstract class?**

* Sử dụng abstract class nếu bạn có 1 trong các tiêu chí sau:
  + Nếu bạn có một nhóm các lớp liên quan cần chia sẻ chung một đoạn code, tính năng nào đó. Bạn có thể đưa đoạn code đó hay tính năng đó vào lớp abstract và các lớp con có liên quan sẽ extends lớp abstract này.
  + Bạn mong đợi rằng các lớp kế thừa lớp abstract của bạn có chung nhiều trường, phương thức hoặc mong muốn có thể sử dụng các access modifier khác chứ không chỉ là public.
  + Bạn muốn khai báo các trường non-static, non-final. Nhờ đó bạn có thể định nghĩa các phương thức có thể truy cập và sửa đổi các trạng thái của từng đối tượng.

**Khi nào sử dụng interface?**

* Sử dụng interface trong các trường hợp sau:
  + Đạt được tính trừu tượng hoàn toàn. Tất cả các phương thức được định ra chưa có phần thân thực thi mà cần được thực thi ở các lớp con implements nó.
  + Muốn đạt được tính chất đa kế thừa.
  + Muốn cho các lớp không liên quan đến nhau cũng có thể implements interface.
  + Bạn muốn chỉ định các hành vi cần thực hiện nhưng không quan tâm các hành vi đó sẽ được thực hiện như nào, do lớp nào.

**Bài 9.8. Callback methods**

**Nội dung bài học**

1. Giới thiệu về callback
2. Ví dụ minh họa về callback

**Giới thiệu về callback**

* Trong C và C++, quá trình gọi một hàm từ trong một hàm khác được gọi là callback.
* Ở đây, địa chỉ của hàm cần gọi được truyền vào làm đối số của hàm khác qua khái niệm gọi là con trỏ hàm. Sau đó trong hàm này sẽ gọi callback qua con trỏ hàm.
* Như vậy, callback trong C/C++ đạt được bằng cách truyền con trỏ hàm vào hàm khác.
* Đáng tiếc trong Java ta không còn khái niệm con trỏ, con trỏ hàm do đó cũng không có khái niệm hàm callback.
* Tuy nhiên ta có thể tạo callback method trong Java bằng cách truyền interface tham chiếu đến địa chỉ phương thức thay vì truyền địa chỉ của phương thức vào phương thức cần gọi.
* Vì Java thuần OOP nên ta không gọi callback function, mà chuyển sang gọi callback method cho đúng chuẩn. Ta gọi callback function theo thói quen mượn từ C/C++.
* Quá trình tạo và sử dụng callback method vắn tắt:
  + B1: tạo interface X với một phương thức m().
  + B2: tạo lớp hoặc đối tượng triển khai interface X và định nghĩa ghi đè phương thức m().
  + B3: tạo phương thức m1() nhận interface X làm tham số.
  + B4: gọi phương thức m từ trong phương thức m1().
  + B5: để gọi phương thức m1(), ta tạo đối tượng của lớp triển khai interface X hoặc tạo đối tượng vô danh triển khai interface X sau đó truyền vào làm đối số của m1(). Có thể dùng biểu thức lambda để rút gọn code.

**Ví dụ minh họa về callback**

* Ví dụ 1: mỗi loài vật có một cách nói lời chào khác nhau. Ta sẽ thể hiện điều này qua callback.

// interface cho phép bất kì đối tượng nào muốn thể hiện lời chào đều có thể triển khai

public interface Greetable {

void greet();

}

public class LearnCallback {

public static void main(String[] args) {

CatGreeting catGreeting = new CatGreeting();

DuckGreeting duckGreeting = new DuckGreeting();

PigGreeting pigGreeting = new PigGreeting();

var test = new LearnCallback();

// hiển thị lời chào qua callback

System.out.println("Lời chào của loài mèo: ");

test.sayGreeting(catGreeting);

System.out.println("Lời chào của loài vịt: ");

test.sayGreeting(duckGreeting);

System.out.println("Lời chào của loài heo: ");

test.sayGreeting(pigGreeting);

}

/\*\*

\* Phương thức thực hiện việc gọi callback thể hiện lời chào theo từng

\* loại đối tượng triển khai interface Greetable được truyền vào.

\*

\* @param greetable tham số đóng vai trò là callback

\*/

public void sayGreeting(Greetable greetable) {

greetable.greet(); // gọi callback method

}

}

/\*\*

\* Lớp mô tả cách chào của loài mèo

\*/

class CatGreeting implements Greetable {

@Override

public void greet() {

System.out.println("Méo mèo meo");

}

}

/\*\*

\* Lớp mô tả cách chào của loài heo

\*/

class PigGreeting implements Greetable {

@Override

public void greet() {

System.out.println("Ụt ụt ịt ịt");

}

}

/\*\*

\* Lớp mô tả cách chào của loài vịt

\*/

class DuckGreeting implements Greetable {

@Override

public void greet() {

System.out.println("Quạc quạc quạc quạc");

}

}

Lời chào của loài mèo:

Méo mèo meo

Lời chào của loài vịt:

Quạc quạc quạc quạc

Lời chào của loài heo:

Ụt ụt ịt ịt

* Ví dụ 2: gọi callback bằng cách truyền vào đối tượng vô danh.

public class CallbackExample2 {

public static void main(String[] args) {

Button myButton = new Button();

Handler handler = new Handler();

myButton.onClick(handler);

// gọi callback bằng đối tượng vô danh

new Button().onClick(new EventHandler() {

@Override

public void clickEventHandler() {

System.out.println("Một nút khác đang được nhấn.");

}

});

}

}

/\*\*

\* Lớp mô tả nút bấm

\*/

class Button {

public void onClick(EventHandler handler) {

handler.clickEventHandler();

}

}

/\*\*

\* Interface mô tả sự kiện click

\*/

interface EventHandler {

void clickEventHandler();

}

/\*\*

\* Lớp triển khai hành động click/nhấn vào một nút

\*/

class Handler implements EventHandler {

@Override

public void clickEventHandler() {

System.out.println("Nút bấm đang được nhấn.");

}

}

Nút bấm đang được nhấn.

Một nút khác đang được nhấn.

* Ví dụ 3: sử dụng lambda để thay thế callback method.
* Chi tiết về biểu thức lambda bạn xem trong bài học kế tiếp nhé.

public class CallbackExample3 {

public static void main(String[] args) {

ButtonClass button = new ButtonClass();

button.pressEvent(() -> System.out.println("Nút Đăng nhập đang được nhấn."));

button.pressEvent(() -> System.out.println("Nút Đăng xuất đang được nhấn."));

button.pressEvent(() -> System.out.println("Nút Đăng ký đang được nhấn."));

}

}

/\*\*

\* Nút mô tả thông tin về một nút nào đó

\*/

class ButtonClass {

public void pressEvent(PressHandler handler) {

handler.pressEventHandler();

}

}

/\*\*

\* interface mô tả hành động nhấn nút.

\*/

interface PressHandler {

void pressEventHandler();

}

Nút Đăng nhập đang được nhấn.

Nút Đăng xuất đang được nhấn.

Nút Đăng ký đang được nhấn.

**Bài 9.9. Lambda expression**

**Nội dung bài học**

1. Khái niệm và mục đích sử dụng
2. Cú pháp tổng quát
3. Ví dụ minh họa
4. Biểu thức lambda vs lập trình giao diện

**Khái niệm và mục đích sử dụng**

* Biểu thức lambda là một tính năng quan trọng được bổ sung vào Java từ phiên bản 1.8.
* Nó cung cấp cách thức ngắn gọn và rõ ràng để biểu thị một interface có 1 phương thức qua một biểu thức.
* Đây là tính năng rất hữu dụng khi ta làm việc với tập hợp. Nó giúp ta thực hiện các thao tác: lặp, lọc, trích xuất dữ liệu từ tập hợp.
* Biểu thức lambda chủ yếu sử dụng để cung cấp các triển khai cho các functional interface(interface có duy nhất 1 phương thức).
* Khi sử dụng lambda expression ta không cần định nghĩa lại phương thức cung cấp các triển khai mà chỉ cần thay thế nó bằng biểu thức lambda ngắn gọn.
* Cần nói thêm rằng biểu thức lambda được coi như một phương thức do vậy mà trình biên dịch không cần tạo file .class để chạy chương trình.
* Tóm lại, lý do để sử dụng biểu thức lambda là:
  + Cung cấp triển khai cho các functional interface.
  + Đơn giản hóa, rút gọn code.
* Đây là một số ví dụ về functional interface: Callable<T>, Comparable<T>, Runnable, Closable, ActionListener, …
* Một ví dụ về functional interface ta tự định nghĩa trong bài học trước:

interface PressHandler {

void pressEventHandler();

}

**Cú pháp tổng quát**

* Cú pháp tổng quát của biểu thức lambda có dạng:

**(argument\_list) -> {expressions}**

* Trong đó gồm 3 thành phần chính:
  + argument\_list: có thể chứa 0, 1 hoặc nhiều đối số phân tách nhau bởi dấu phẩy đặt trong dấu ngoặc tròn bên trái mũi tên. Lưu ý rằng vì chúng là đối số nên không chứa kiểu dữ liệu như tham số của phương thức.
  + Kí hiệu mũi tên: -> dùng để liên kết danh sách đối số và phần thân của biểu thức lambda.
  + Phần thân bên phải mũi tên. Ở đây chứa các lệnh cần thực hiện như phần thân của phương thức.
* Lưu ý rằng nếu danh sách đối số chỉ có duy nhất 01 đối số thì ta có thể bỏ qua cặp () bao bên ngoài.
* Lưu ý thứ hai là nếu phần thân của biểu thức lamda chỉ có 01 lệnh ta cũng có thể bỏ cặp {} bao bên ngoài đi.
* Giá trị trả về của biểu thức lamda phụ thuộc vào lệnh trong phần thân của nó. Nếu lệnh thực hiện xong có kết quả là một giá trị cụ thể của một kiểu nào đó trong Java thì biểu thức lambda sẽ trả về giá trị đó theo mặc định.
* Nếu lệnh cần thực hiện là các lời gọi tới các phương thức kiểu trả về là void thì biểu thức lambda sẽ không trả về và nó sẽ tương đương phương thức có kiểu trả về là void.
* Câu lệnh return là tùy chọn và không được coi là một biểu thức trong lambda expression.
* Sau đây là một số cú pháp tổng quát của biểu thức lambda:
  + Không đối số:

() -> { // phần thân của biểu thức lambda }

* + Một đối số:

p -> { // phần thân của biểu thức lambda }

hoặc

(p) -> { // phần thân của biểu thức lambda }

* + Nhiều đối số:

(a, b, c) -> { // phần thân của biểu thức lambda }

**Ví dụ minh họa**

* Ví dụ 1: dùng biểu thức lambda trong callback method.

public class LambdaExample1 {

public static void main(String[] args) {

ButtonClass button = new ButtonClass();

button.pressEvent(() -> System.out.println("Nút Đăng nhập đang được nhấn."));

button.pressEvent(() -> System.out.println("Nút Đăng xuất đang được nhấn."));

button.pressEvent(() -> System.out.println("Nút Đăng ký đang được nhấn."));

}

}

/\*\*

\* Nút mô tả thông tin về một nút nào đó

\*/

class ButtonClass {

public void pressEvent(PressHandler handler) {

handler.pressEventHandler();

}

}

/\*\*

\* interface mô tả hành động nhấn nút.

\*/

interface PressHandler {

void pressEventHandler();

}

Nút Đăng nhập đang được nhấn.

Nút Đăng xuất đang được nhấn.

Nút Đăng ký đang được nhấn.

* Ví dụ 2: dùng biểu thức lambda trong sắp xếp.
* Lớp Person:

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

public class Person {

private String firstName;

private String lastName;

private String phoneNumber;

private Date birthDate;

public Person() {

birthDate = new Date();

}

public Person(String firstName, String lastName, String phoneNumber, String birthDate) {

this.birthDate = createDate(birthDate);

this.firstName = firstName;

this.lastName = lastName;

this.phoneNumber = phoneNumber;

}

private Date createDate(String birthDate) {

SimpleDateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy");

if (birthDate != null && !birthDate.isEmpty()) {

try {

return dateFormat.parse(birthDate);

} catch (ParseException e) {

e.printStackTrace();

}

}

return new Date();

}

public String getFirstName() {

return firstName;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public String getLastName() {

return lastName;

}

public void setLastName(String lastName) {

this.lastName = lastName;

}

public String getPhoneNumber() {

return phoneNumber;

}

public void setPhoneNumber(String phoneNumber) {

this.phoneNumber = phoneNumber;

}

public Date getBirthDate() {

return birthDate;

}

public void setBirthDate(Date birthDate) {

this.birthDate = birthDate;

}

@Override

public String toString() {

return "Person{" +

"firstName='" + firstName + '\'' +

", lastName='" + lastName + '\'' +

", phoneNumber='" + phoneNumber + '\'' +

", birthDate=" + birthDate +

'}';

}

public static int compareFirstname(Person person, Person other) {

return person.getFirstName().compareTo(other.getFirstName());

}

}

package net.braniumacademy.lambdaexpression;

import java.util.ArrayList;

import java.util.List;

public class LambdaExample2 {

public static void main(String[] args) {

List<Person> people = new ArrayList<>();

people.add(new Person("Nhung", "Phạm", "0912345687", "12/12/2000"));

people.add(new Person("Hồng", "Nguyễn", "0912345684", "10/12/2002"));

people.add(new Person("Mai", "Trần", "0912345685", "12/10/2003"));

people.add(new Person("Thanh", "Hoàng", "0912345686", "10/10/2001"));

people.add(new Person("Khánh", "Ngô", "0912345688", "05/06/2000"));

people.add(new Person("Minh", "Lê", "0912345689", "08/04/2001"));

// tiến hành sắp xếp

people.sort((p1, p2) -> p1.getFirstName().compareTo(p2.getFirstName()));

showPeopleInfo(people);

}

private static void showPeopleInfo(List<Person> people) {

for (var p : people) {

System.out.println(p);

}

}

}

Person{firstName='Hồng', lastName='Nguyễn', phoneNumber='0912345684', birthDate=Tue Dec 10 00:00:00 ICT 2002}

Person{firstName='Khánh', lastName='Ngô', phoneNumber='0912345688', birthDate=Mon Jun 05 00:00:00 ICT 2000}

Person{firstName='Mai', lastName='Trần', phoneNumber='0912345685', birthDate=Sun Oct 12 00:00:00 ICT 2003}

Person{firstName='Minh', lastName='Lê', phoneNumber='0912345689', birthDate=Sun Apr 08 00:00:00 ICT 2001}

Person{firstName='Nhung', lastName='Phạm', phoneNumber='0912345687', birthDate=Tue Dec 12 00:00:00 ICT 2000}

Person{firstName='Thanh', lastName='Hoàng', phoneNumber='0912345686', birthDate=Wed Oct 10 00:00:00 ICT 2001}

* Ví dụ 3: dùng biểu thức lambda trong liệt kê trích xuất phần tử.

package net.braniumacademy.lambdaexpression;

import java.util.ArrayList;

import java.util.List;

public class LambdaExample3 {

public static void main(String[] args) {

List<Person> people = new ArrayList<>();

people.add(new Person("Nhung", "Phạm", "0912345687", "12/12/2000"));

people.add(new Person("Hồng", "Nguyễn", "0912345684", "10/12/2002"));

people.add(new Person("Mai", "Trần", "0912345685", "12/10/2003"));

people.add(new Person("Thanh", "Hoàng", "0912345686", "10/10/2001"));

people.add(new Person("Khánh", "Ngô", "0912345688", "05/06/2000"));

people.add(new Person("Minh", "Lê", "0912345689", "08/04/2001"));

// tiến hành sắp xếp

people.sort((p1, p2) -> p1.getFirstName().compareTo(p2.getFirstName()));

showPeopleInfo(people);

}

private static void showPeopleInfo(List<Person> people) {

people.forEach(p -> System.out.println(p.getFirstName() + " " + p.getLastName()));

}

}

Hồng Nguyễn

Khánh Ngô

Mai Trần

Minh Lê

Nhung Phạm

Thanh Hoàng

* Ví dụ 4: Sắp xếp, hiển thị các phần tử mảng.

public class LambdaExample4 {

public static void main(String[] args) {

String[] friends = new String[]{"Mai", "Nam", "Nhân", "Tuấn", "Hải", "An"};

Arrays.sort(friends, (f1, f2) -> f2.compareTo(f1));

for (var friend : friends) {

System.out.print(friend + " ");

}

}

}

Tuấn Nhân Nam Mai Hải An

* Ví dụ 5: biểu thức lambda có return.

interface Operator {

double operator(double a, double b);

}

public class LambdaExample5 {

public static void main(String[] args) {

Operator addition = (a, b) -> a + b; // không có return

Operator subtraction = (a, b) -> {

return a - b; // có return

};

System.out.println("10 + 20" + " = " + addition.operator(10, 20));

System.out.println("60 - 15" + " = " + subtraction.operator(60, 15));

}

}

10 + 20 = 30.0

60 - 15 = 45.0

**Thay thế biểu thức lambda bằng tham chiếu phương thức**

* Khi một biểu thức lambda không làm gì khác ngoài việc gọi một phương thức đã có của đối tượng, lớp thì ta có thể thay thế biểu thức lambda bằng tham chiếu tới phương thức cho ngắn gọn và rõ ràng hơn.
* Về phạm vi truy cập các biến, biểu thức lambda không tạo một vùng truy cập mới nào. Ta có thể truy cập vào các biến cục bộ, thuộc tính, hằng số, biến của lớp, khối chứa nó như lớp vô danh, lớp inner.
* Ví dụ phương thức sort ở ví dụ 4 có thể sửa lại thành:

import java.util.Arrays;

import java.util.Comparator;

public class LambdaExample4 {

public static void main(String[] args) {

String[] friends = new String[]{"Mai", "Nam", "Nhân", "Tuấn", "Hải", "An"};

Arrays.sort(friends, Comparator.reverseOrder());

for (var friend : friends) {

System.out.print(friend + " ");

}

}

}

* Ví dụ 3 trên có thể thay thế nội dung biểu thức lambda thành:

import java.util.ArrayList;

import java.util.List;

public class LambdaExample2 {

public static void main(String[] args) {

List<Person> people = new ArrayList<>();

people.add(new Person("Nhung", "Phạm", "0912345687", "12/12/2000"));

people.add(new Person("Hồng", "Nguyễn", "0912345684", "10/12/2002"));

people.add(new Person("Mai", "Trần", "0912345685", "12/10/2003"));

people.add(new Person("Thanh", "Hoàng", "0912345686", "10/10/2001"));

people.add(new Person("Khánh", "Ngô", "0912345688", "05/06/2000"));

people.add(new Person("Minh", "Lê", "0912345689", "08/04/2001"));

// tiến hành sắp xếp

people.sort(Person::compareFirstname); // thay thế bằng tham chiếu phương thức ta tự so sánh

showPeopleInfo(people);

}

private static void showPeopleInfo(List<Person> people) {

people.forEach(p -> System.out.println(p.getFirstName() + " " + p.getLastName()));

}

}

* Hoặc:

import java.util.ArrayList;

import java.util.Comparator;

import java.util.List;

public class LambdaExample2 {

public static void main(String[] args) {

List<Person> people = new ArrayList<>();

people.add(new Person("Nhung", "Phạm", "0912345687", "12/12/2000"));

people.add(new Person("Hồng", "Nguyễn", "0912345684", "10/12/2002"));

people.add(new Person("Mai", "Trần", "0912345685", "12/10/2003"));

people.add(new Person("Thanh", "Hoàng", "0912345686", "10/10/2001"));

people.add(new Person("Khánh", "Ngô", "0912345688", "05/06/2000"));

people.add(new Person("Minh", "Lê", "0912345689", "08/04/2001"));

// tiến hành sắp xếp

people.sort(Comparator.comparing(Person::getFirstName)); // thay thế bằng tham chiếu method

showPeopleInfo(people);

}

private static void showPeopleInfo(List<Person> people) {

people.forEach(p -> System.out.println(p.getFirstName() + " " + p.getLastName()));

}

}

**Biểu thức lambda vs lập trình giao diện**

* Trong xử lý sự kiện của lập trình giao diện ta thường sử dụng đến biểu thức lambda, callback.
* Ví dụ sau bắt sự kiện nút được click và thực hiện hành động tương ứng qua biểu thức lambda.
* Chi tiết về lập trình giao diện bạn xem trong chương 14 của khóa học.

import javax.swing.\*;

import java.awt.\*;

public class LambdaExample6 extends JFrame {

private final int width;

private final int height;

public LambdaExample6(String title, int width, int height) {

this.height = height;

this.width = width;

this.setTitle(title);

this.setBackground(Color.YELLOW);

this.setSize(this.width, this.height);

this.setPreferredSize(new Dimension(this.width, this.height));

this.setDefaultCloseOperation(EXIT\_ON\_CLOSE);

this.setLocationRelativeTo(null);

this.setLayout(null);

addComponents();

this.setVisible(true);

}

private void addComponents() {

JLabel label = new JLabel("Nhập họ và tên của bạn: ", JLabel.CENTER);

label.setFont(new Font("Arial", Font.PLAIN, 14));

label.setBounds(this.width / 2 - 100, this.height / 5 - 20, 200, 40);

this.add(label);

JButton btnClickMe = new JButton("Xác nhận");

var width = 120;

var height = 40;

btnClickMe.setBounds(this.width / 2 - width / 2,

3 \* this.height / 5 - height / 2, width, height);

btnClickMe.setBackground(Color.RED);

btnClickMe.setForeground(Color.WHITE);

this.add(btnClickMe);

JTextField txtMessage = new JTextField();

txtMessage.setFont(new Font("Arial", Font.PLAIN, 14));

txtMessage.setBounds(this.width / 2 - 90, 2 \* this.height / 5 - 20, 180, 40);

this.add(txtMessage);

// đăng ký sự kiện sử dụng biểu thức lambda

btnClickMe.addActionListener(e -> {

var msg = "Xin chào " + txtMessage.getText() + "!";

JOptionPane.showMessageDialog(this, msg);

});

}

public static void main(String[] args) {

var title = "Ví dụ minh họa biểu thức lambda";

var width = 400;

var height = 250;

new LambdaExample6(title, width, height);

}

}

* Sau khi ứng dụng hiện lên bạn gõ họ tên mình hoặc họ tên gấu, ny cũ, kẻ thù, thần tượng của bạn… vào và nhấn Xác nhận để xem kết quả nhé.

Ảnh có chứa văn bản, ảnh chụp màn hình, màn hình, phần mềm

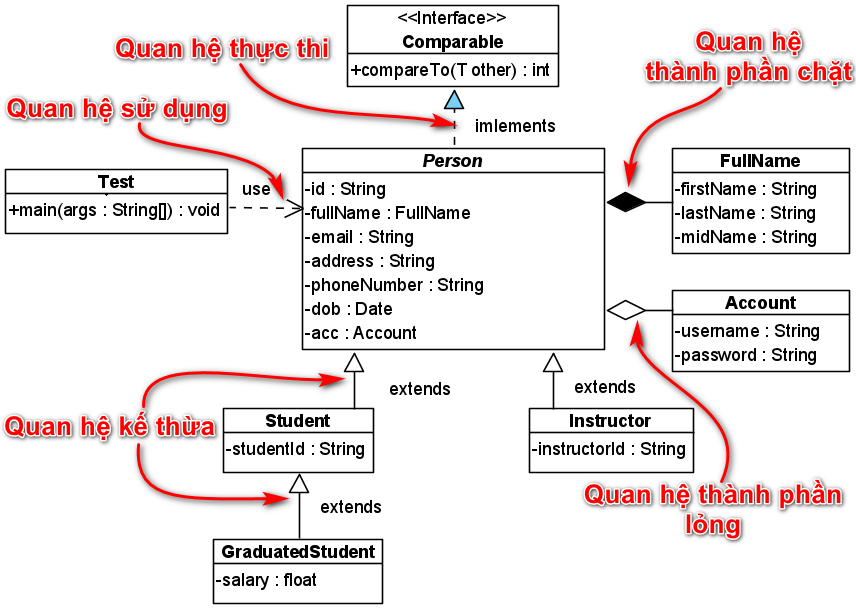
Mô tả được tạo tự động

# Bài 9.12. Biểu diễn quan hệ lớp trong UML

### Nội dung bài học

1. Biểu diễn quan hệ kế thừa giữa hai lớp
2. Biểu diễn quan hệ implements giữa lớp và interface
3. Biểu diễn quan hệ thành phần lỏng giữa hai lớp
4. Biểu diễn quan hệ thành phần chặt giữa hai lớp
5. Biểu diễn quan hệ sử dụng giữa hai lớp

Hình ảnh minh họa quan hệ:

[](https://braniumacademy.net/wp-content/uploads/2020/08/lesson57.png)

#### Trong đó:

* Quan hệ thành phần lỏng là quan hệ lớp này là thành phần của lớp kia. Nhưng sự tồn tại hoặc bị hủy đi của đối tượng lớp này không ảnh hưởng đến lớp kia.
* Quan hệ thành phần chặt là quan hệ lớp này là thành phần bắt buộc của lớp kia. Nếu không có đối tượng của lớp này thì không thể tạo được đối tượng của lớp kia.
* Quan hệ sử dụng: Lớp này sử dụng lớp kia để vận hành. Ví dụ lớp chứa phương thức main phải sử dụng các lớp khác để tạo đối tượng, vận hành…

Bạn có thể [**click vào đây**](https://github.com/thantrieu/java_course1_exercises/tree/master/VP%20files) để tải Class Diagram hoàn chỉnh trong bài học về tham khảo. Lưu ý rằng bạn phải cài Visual Paradigm phiên bản lớn hơn hoặc bằng 16.2 thì mới sử dụng được file mẫu nhé.

# Bài 9.13. Sắp xếp chuỗi kí tự tiếng Việt có dấu

### Nội dung bài học

1. [Sắp xếp họ và tên tiếng Việt theo thứ tự từ điển](https://braniumacademy.net/courses/java/lessons/java-vni-sorting/#s1)

Vấn đề: khi sắp xếp thông thường ta không thể đưa tên người hoặc chuỗi kí tự tiếng Việt về đúng thứ tự của nó trong bảng chữ cái. Vậy phải làm sao?

### Sắp xếp họ và tên tiếng Việt theo thứ tự từ điển

* Sắp xếp thông thường:

names.sort(Comparator.comparing((FullName o) -> o.first)

.thenComparing(o -> o.last));

* Kết quả sắp xếp thông thường:

Trần Cảnh

Trần Duy

Hoàng Hiên

Lê Hiên

Ma Hiên

Nguyễn Hiên

Trần Hiên

Trần Hoa

Trần Hà

Trần Hân

Trần Hòa

Trần Hưng

Trần Hạnh

Trần Hồng

Trần Trang

Trần Đoan

Trần Đức

* Giải pháp: sử dụng đối tượng Collator.
* Cú pháp: ta tạo đối tượng Collator xử lý chuỗi kí tự ở tiếng Việt sau đó trả về giá trị được so sánh trong phương thức compare.
* Code mẫu:

private static int compareName(String s1, String s2) {

Collator collator = Collator.getInstance(Locale.forLanguageTag("vi"));

collator.setStrength(Collator.PRIMARY);

return collator.compare(s1, s2);

}

* Triển khai hoàn chỉnh:

import java.text.Collator;

import java.util.ArrayList;

import java.util.List;

import java.util.Locale;

public class CompareVNi {

public static void main(String[] args) {

List<FullName> names = new ArrayList<>();

createFakeData(names);

names.sort((o1, o2) -> {

var compareFirstName = compareName(o1.first, o2.first);

if (compareFirstName != 0) {

return compareFirstName;

}

return compareName(o1.last, o2.last);

});

for (var item : names) {

System.out.println(item);

}

}

private static int compareName(String s1, String s2) {

Collator collator = Collator.getInstance(Locale.forLanguageTag("vi"));

collator.setStrength(Collator.PRIMARY);

return collator.compare(s1, s2);

}

private static void createFakeData(List<FullName> names) {

names.add(new FullName("Hoa", "Trần"));

names.add(new FullName("Hòa", "Trần"));

names.add(new FullName("Hiên", "Hoàng"));

names.add(new FullName("Hiên", "Lê"));

names.add(new FullName("Hiên", "Trần"));

names.add(new FullName("Hiên", "Nguyễn"));

names.add(new FullName("Hiên", "Ma"));

names.add(new FullName("Hưng", "Trần"));

names.add(new FullName("Hồng", "Trần"));

names.add(new FullName("Hạnh", "Trần"));

names.add(new FullName("Hân", "Trần"));

names.add(new FullName("Hà", "Trần"));

names.add(new FullName("Duy", "Trần"));

names.add(new FullName("Đức", "Trần"));

names.add(new FullName("Đoan", "Trần"));

names.add(new FullName("Cảnh", "Trần"));

names.add(new FullName("Trang", "Trần"));

}

static class FullName {

private final String first;

private final String last;

public FullName(String first, String last) {

this.first = first;

this.last = last;

}

@Override

public String toString() {

return last + " " + first;

}

}

}

* Kết quả:

Trần Cảnh

Trần Duy

Trần Đoan

Trần Đức

Trần Hà

Trần Hạnh

Trần Hân

Hoàng Hiên

Lê Hiên

Ma Hiên

Nguyễn Hiên

Trần Hiên

Trần Hoa

Trần Hòa

Trần Hồng

Trần Hưng

Trần Trang

**Bài 10.1. Tổng quan về xử lý ngoại lệ**

**Nội dung bài học**

1. Khái niệm, nguyên nhân
2. Phân biệt ngoại lệ và lỗi
3. Xử lý ngoại lệ khi nó xảy ra
4. Minh họa trong công cụ lập trình

Video lý thuyết

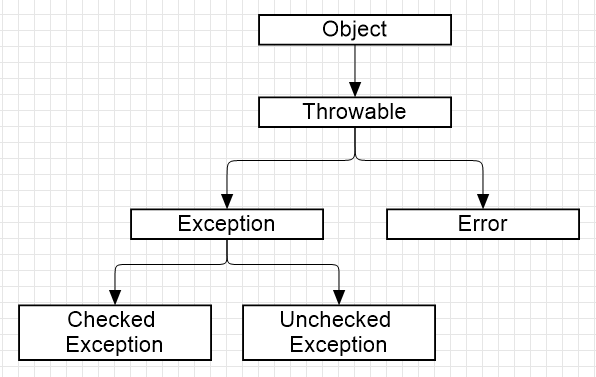
Video minh họa

**Khái niệm, nguyên nhân**

* Một ngoại lệ(exception) là sự kiện không mong muốn nảy sinh trong quá trình chạy của chương trình.
* Khi ngoại lệ xảy ra sẽ làm gián đoạn luồng thực thi bình thường của các tập lệnh và làm chương trình kết thúc bất thường.
* Đó là điều không ai mong muốn với cả người sử dụng ứng dụng và các lập trình viên. Để khắc phục sự bất thường khi ngoại lệ xảy ra, ta phải xử lý các ngoại lệ.
* Nguyên nhân dẫn đến ngoại lệ có thể là do chủ quan hoặc khách quan:
  + Người dùng nhập dữ liệu không đúng định dạng
  + Một file cần đọc không tồn tại
  + Kết nối bị mất khi đang thực hiện phiên giao dịch giữa máy chủ và máy khách
  + Bộ nhớ đầy trong quá trình thực thi
  + Lỗi logic chương trình của lập trình viên
* Ví dụ: ClassCastException, NullPointerException, ArrayIndexOutOfBoundsExeption…

**Phân biệt ngoại lệ và lỗi**

* Sơ đồ phân cấp tổng quát:

[](https://braniumacademy.net/wp-content/uploads/2020/08/Java-exception.png)

* Ngoại lệ(exception): những vấn đề nảy sinh và có thể xử lý bởi chương trình.
* Ví dụ ngoại lệ ArrayIndexOutOfBoundsException, ta có thể phòng ngừa nó bằng cách kiểm tra chỉ số mảng có nằm trong đoạn hợp lệ hay không trước khi thực hiện bất kì hành động nào.

package lesson58;

public class Lesson58 {

public static void main(String[] args) {

int[] numbers = {1, 2, 3, 6, 5, 4, 7, 8, 9};

int index = 100;

System.out.println("Phần tử tại vị trí " +

index + " là: " + numbers[index]);

}

}

* Kết quả:

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 100 out of bounds for length 9

at lesson58.Lesson58.main(Lesson58.java:7)

* Có thể khắc phục bằng cách thêm đoạn kiểm tra chỉ số trước khi sử dụng:

package lesson58;

public class Lesson58 {

public static void main(String[] args) {

int[] numbers = {1, 2, 3, 6, 5, 4, 7, 8, 9};

int index = 100;

if (index >= 0 && index < numbers.length) {

System.out.println("Phần tử tại vị trí " +

index + " là: " + numbers[index]);

} else {

System.out.println("Chỉ số " + index + " không hợp lệ");

System.out.println("Chỉ số tối đa của mảng là: " +

numbers.length);

}

}

}

* Kết quả:

Chỉ số 100 không hợp lệ

Chỉ số tối đa của mảng là: 9

* Lỗi(error): những vấn đề nghiêm trọng xảy ra không thể xử lý bởi chương trình.
* Ví dụ lỗi hệ thống, lỗi phần cứng. Những vấn đề này chương trình chấp nhận số phận và hoàn toàn bất lực khi nó xảy đến.

**Xử lý ngoại lệ khi nó xảy ra**

* Để phòng ngừa ngoại lệ xảy ra, lập trình viên sẽ chủ động kiểm soát logic chương trình. Sử dụng các cấu trúc rẽ nhánh như if để lọc điều kiện trước khi thực hiện một hành động nào đó.
* Khi ngoại lệ xảy ra ta có hai hướng xử lý:
  + Một là xử lý triệt để bằng try-catch: gói đoạn code có thể xảy ra ngoại lệ trong khối try và bắt ngoại lệ tương ứng trong khối catch.
  + Hai là trì hoãn việc xử lý ngoại lệ ở nơi nó xảy ra, chuyển việc xử lý ngoại lệ đó cho nơi đã thực hiện lời gọi phương thức thông qua throws.
* Cụ thể từng cách:
  + Xử lý chủ động với try-catch:

package lesson58;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

public class Lesson58 {

public static void main(String[] args) {

SimpleDateFormat dateFormat =

new SimpleDateFormat("dd/MM/yyyy");

var dateString = "27-05-2025";

// chuyển đổi string sang đối tượng date:

try {

// dòng code dưới đây có thể xảy ra ngoại lệ:

Date myBirthday = dateFormat.parse(dateString);

System.out.println("Đối tượng date hoàn chỉnh: " + myBirthday);

} catch (ParseException e) {

System.out.println("Không thể chuyển đổi String sang Date");

e.printStackTrace();

}

// các câu lệnh kế tiếp phía sau

System.out.println("Thực hiện các câu lệnh tiếp theo...");

}

}

* Kết quả:

Không thể chuyển đổi String sang Date

Thực hiện các câu lệnh tiếp theo...

java.text.ParseException: Unparseable date: "27-05-2025"

at java.base/java.text.DateFormat.parse(DateFormat.java:396)

at lesson58.Lesson58.main(Lesson58.java:15)

* Trong ví dụ này nếu ta không để dòng lệnh có khả năng xảy ra ngoại lệ vào bên trong khối try thì chương trình không chạy. Bởi vì đây là checked exception-ngoại lệ bắt buộc phải xử lý tại thời điểm biên dịch chương trình.

package lesson58;

import java.text.SimpleDateFormat;

import java.util.Date;

public class Lesson58 {

public static void main(String[] args) {

SimpleDateFormat dateFormat =

new SimpleDateFormat("dd/MM/yyyy");

var dateString = "27-05-2025";

// chuyển đổi string sang đối tượng date:

// dòng code dưới đây có thể xảy ra ngoại lệ:

Date myBirthday = dateFormat.parse(dateString);

System.out.println("Đối tượng date hoàn chỉnh: " + myBirthday);

// các câu lệnh kế tiếp phía sau

System.out.println("Thực hiện các câu lệnh tiếp theo...");

}

}

* Thông báo lỗi:

java: unreported exception java.text.ParseException; must be caught or declared to be thrown

* Trì hoãn việc xử lý ngoại lệ cho nơi thực hiện lời gọi:

package lesson58;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

public class Lesson58 {

public static void main(String[] args) throws ParseException {

SimpleDateFormat dateFormat =

new SimpleDateFormat("dd/MM/yyyy");

var dateString = "27-05-2025";

// chuyển đổi string sang đối tượng date:

// dòng code dưới đây có thể xảy ra ngoại lệ:

Date myBirthday = dateFormat.parse(dateString);

System.out.println("Đối tượng date hoàn chỉnh: " + myBirthday);

// các câu lệnh kế tiếp phía sau

System.out.println("Thực hiện các câu lệnh tiếp theo...");

}

}

* Kết quả:

Exception in thread "main" java.text.ParseException: Unparseable date: "27-05-2025"

at java.base/java.text.DateFormat.parse(DateFormat.java:396)

at lesson58.Lesson58.main(Lesson58.java:14)

* Nếu ta không biết cách xử lý ngoại lệ tại nơi nó xảy ra ta có thể chuyển nhượng nhiệm vụ đó cho nơi thực hiện lời gọi phương thức chứa ngoại lệ này như trên. Ưu điểm của phương pháp này là nhanh nhưng không triệt để và không được khuyến khích. Khi chạy chương trình gặp ngoại lệ thì chương trình sẽ kết thúc ngay. Không tiếp tục thực hiện các lệnh phía sau(nếu có).
* Chi tiết về cách thức hoạt động của từng phương pháp ta sẽ tìm hiểu ở các bài học sau.

**Bài 10.2. Checked vs Unchecked exceptions**

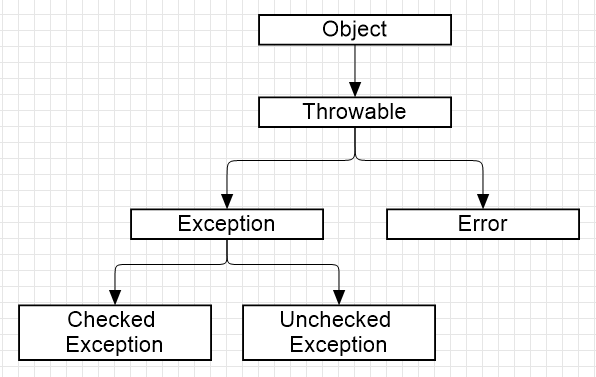
**Nội dung bài học**

1. Checked exception
2. Unchecked exception
3. Tìm hiểu lớp Exception
4. Minh họa trong công cụ lập trình

Video lý thuyết

Video minh họa

Sơ đồ phân cấp của ngoại lệ:

[](https://braniumacademy.net/wp-content/uploads/2020/08/Java-exception.png)

Ngoại lệ trong Java chia làm hai loại:

**Checked exception**

* Checked exception: là một ngoại lệ được kiểm tra và cảnh báo bởi trình biên dịch(compiler) tại thời điểm biên dịch chương trình, còn được gọi là ngoại lệ tại thời điểm biên dịch(compile time exception).
* Ngoại lệ loại này không thể bị bỏ qua và người lập trình bắt buộc phải xử lý nó.
* Để xử lý ngoại lệ loại này ta có hai cách đã biết: xử lý trực tiếp với try-catch hoặc hoãn xử lý với throws.
* Một số ngoại lệ loại này:
  + FileNotFoundException
  + ParseException
  + ClassNotFoundException
  + NoSuchMethodException
  + UnsuportedEncodingException
  + InterruptedIOException
  + InvalidClassException
  + SQLException…
* Ví dụ 1: Xử lý ngoại lệ ParseException.

package lesson59;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

public class Lesson59 {

public static void main(String[] args) {

SimpleDateFormat dateFormat =

new SimpleDateFormat("dd/MM/yyyy");

var dateString = "27-05-2025";

// chuyển đổi string sang đối tượng date:

try {

// dòng code dưới đây có thể xảy ra ngoại lệ:

Date myBirthday = dateFormat.parse(dateString);

System.out.println("Đối tượng date hoàn chỉnh: " + myBirthday);

} catch (ParseException e) {

System.out.println("Không thể chuyển đổi String sang Date");

e.printStackTrace();

}

// các câu lệnh kế tiếp phía sau

System.out.println("Thực hiện các câu lệnh tiếp theo...");

}

}

* Kết quả:

Không thể chuyển đổi String sang Date

Thực hiện các câu lệnh tiếp theo...

java.text.ParseException: Unparseable date: "27-05-2025"

     at java.base/java.text.DateFormat.parse(DateFormat.java:396)

     at lesson58.Lesson58.main(Lesson58.java:15)

* Ví dụ 2: Xử lý ngoại lệ FileNotFoundException.

package lesson59;

import java.io.File;

import java.io.FileNotFoundException;

import java.util.Scanner;

public class Lesson59 {

public static void main(String[] args) {

File input = new File("input.txt");

try {

Scanner readFile = new Scanner(input);

System.out.println("Mở file thành công");

// thực hiện đọc dữ liệu trong file

} catch (FileNotFoundException e) {

e.printStackTrace();

}

// các lệnh phía sau nơi xảy ra ngoại lệ

System.out.println("Thực hiện các lệnh kế tiếp...");

}

}

* Kết quả:

java.io.FileNotFoundException: input.txt (The system cannot find the file specified)

at java.base/java.io.FileInputStream.open0(Native Method)

at java.base/java.io.FileInputStream.open(FileInputStream.java:212)

at java.base/java.io.FileInputStream.<init>(FileInputStream.java:154)

at java.base/java.util.Scanner.<init>(Scanner.java:639)

at lesson59.Lesson59.main(Lesson59.java:11)

Thực hiện các lệnh kế tiếp...

**Unchecked exception**

* Unchecked exception: là ngoại lệ xảy ra tại thời điểm chương trình đang chạy.
* Nó còn có tên là runtime exception. Ngoại lệ loại này thường được bỏ qua tại thời điểm biên dịch chương trình.
* Nguyên nhân chủ yếu dẫn đến ngoại lệ này là do lỗi logic của lập trình viên(bug).
* Đây là các ngoại lệ có thể tránh được bằng cách kiểm soát code tốt với các cấu trúc điều kiện. Chúng ta cũng có thể xử lý các ngoại lệ này với try-catch.
* Một số ngoại lệ loại này là:
  + NullPointerException
  + ArrayIndexOutOfBoundsException
  + ArithmeticException
  + ClassCastException
  + IlligalArgumentException
  + SecurityException
  + …
* Ví dụ 1: Ngoại lệ NullPointerException.

package lesson59;

public class Lesson59 {

public static void main(String[] args) {

int[] numbers = null;

int index = 100;

// dòng code có ngoại lệ

System.out.println("Số phần tử của mảng numbers là: " +

numbers.length);

// các lệnh phía sau nơi xảy ra ngoại lệ

System.out.println("Thực hiện các lệnh kế tiếp...");

}

}

* Kết quả: chương trình die ngay khi gặp ngoại lệ.

Exception in thread "main" java.lang.NullPointerException

at lesson59.Lesson59.main(Lesson59.java:8)

* Ngăn ngừa ngoại lệ trên bằng if-else:

package lesson59;

public class Lesson59 {

public static void main(String[] args) {

int[] numbers = null;

int index = 100;

if (numbers != null) {

// dòng code có ngoại lệ

System.out.println("Số phần tử của mảng numbers là: " +

numbers.length);

} else {

System.out.println("Đối tượng numbers là null");

}

// các lệnh phía sau nơi xảy ra ngoại lệ

System.out.println("Thực hiện các lệnh kế tiếp...");

}

}

Đối tượng numbers là null

Thực hiện các lệnh kế tiếp...

* Xử lý ngoại lệ này bằng try-catch:

package lesson59;

public class Lesson59 {

public static void main(String[] args) {

int[] numbers = null;

int index = 100;

try {

// dòng code có ngoại lệ

System.out.println("Số phần tử của mảng numbers là: " +

numbers.length);

} catch (NullPointerException e) {

System.out.println(e.getMessage());

}

// các lệnh phía sau nơi xảy ra ngoại lệ

System.out.println("Thực hiện các lệnh kế tiếp...");

}

}

* Kết quả: chương trình chạy bình thường khi ngoại lệ xảy ra.

null

Thực hiện các lệnh kế tiếp...

* Ví dụ 2: Ngoại lệ ArrayIndexOutOfBoundsException.

package lesson59;

public class Lesson59 {

public static void main(String[] args) {

int[] numbers = {1, 2, 3, 6, 5, 4, 7, 8, 9};

int index = 100;

System.out.println("Phần tử thứ " + index +

" của mảng numbers là: " + numbers[index]);

// các lệnh phía sau nơi xảy ra ngoại lệ

System.out.println("Thực hiện các lệnh kế tiếp...");

}

}

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 100 out of bounds for length 9

at lesson59.Lesson59.main(Lesson59.java:11)

* Ngăn ngừa ngoại lệ bằng if-else:

package lesson59;

public class Lesson59 {

public static void main(String[] args) {

int[] numbers = {1, 2, 3, 6, 5, 4, 7, 8, 9};

int index = 100;

if (index >= 0 && index < numbers.length) {

System.out.println("Phần tử thứ " + index +

" của mảng numbers là: " + numbers[index]);

} else {

System.out.println("Chỉ số mảng " + index + " không hợp lệ");

System.out.println("Chỉ số phần tử tối đa của mảng: " +

(numbers.length - 1));

}

// các lệnh phía sau nơi xảy ra ngoại lệ

System.out.println("Thực hiện các lệnh kế tiếp...");

}

}

Chỉ số mảng 100 không hợp lệ

Chỉ số phần tử tối đa của mảng: 8

Thực hiện các lệnh kế tiếp...

* Xử lý ngoại lệ này bằng try-catch:

package lesson59;

public class Lesson59 {

public static void main(String[] args) {

int[] numbers = {1, 2, 3, 6, 5, 4, 7, 8, 9};

int index = 100;

try {

System.out.println("Phần tử thứ " + index +

" của mảng numbers là: " + numbers[index]);

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println(e.getMessage());

}

// các lệnh phía sau nơi xảy ra ngoại lệ

System.out.println("Thực hiện các lệnh kế tiếp...");

}

}

* Kết quả: chương trình chạy bình thường khi ngoại lệ xảy ra.

Index 100 out of bounds for length 9

Thực hiện các lệnh kế tiếp...

**Tìm hiểu lớp Exception**

* Sau đây là một số phương thức của lớp Exception và mô tả:

|  |  |
| --- | --- |
| **Phương thức** | **Mô tả** |
| String getMessage() | Trả về thông điệp chi tiết về ngoại lệ đã xảy ra. Thông điệp này được  khởi tạo trong constructor của Throwable. |
| Throwable getCause() | Trả về nguyên nhân gây ra ngoại lệ trong đối tượng của Throwable. |
| void printStackTray() | In ra màn hình loại, thông điệp của ngoại lệ xảy ra và thứ tự các lời gọi lưu trong stack đến khi gặp ngoại lệ. |
| StackTraceElement [] getStackTrace() | Trả về mảng chứa các lời gọi trong khay stack. Phần tử tại vị trí 0 là phần tử trên top của stack. Phần tử cuối là lời gọi đầu tiên trong khay stack dẫn đến ngoại lệ. |

* Ví dụ:

package lesson59;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

public class Lesson59 {

public static void main(String[] args) {

method1(); // gọi phương thức 1

}

private static void method1() {

method2(); // gọi phương thức 2

}

// phương thức chứa các dòng lệnh xảy ra ngoại lệ

private static void method2() {

SimpleDateFormat dateFormat =

new SimpleDateFormat("dd/MM/yyyy");

var dateString = "27-05-2025";

// chuyển đổi string sang đối tượng date:

try {

// dòng code dưới đây có thể xảy ra ngoại lệ:

Date myBirthday = dateFormat.parse(dateString);

System.out.println("Đối tượng date hoàn chỉnh: " + myBirthday);

} catch (ParseException e) {

System.out.println("Không thể chuyển đổi String sang Date");

for (var item : e.getStackTrace()) {

System.out.println(item);

}

}

// các câu lệnh kế tiếp phía sau

System.out.println("Thực hiện các câu lệnh tiếp theo...");

}

}

Không thể chuyển đổi String sang Date

java.base/java.text.DateFormat.parse(DateFormat.java:396)

lesson59.Lesson59.method2(Lesson59.java:24)

lesson59.Lesson59.method1(Lesson59.java:13)

lesson59.Lesson59.main(Lesson59.java:9)

Thực hiện các câu lệnh tiếp theo...

# Bài 10.3. Sử dụng try-catch-finally

### Nội dung bài học

1. Mục đích sử dụng của try, catch, finally
2. Sử dụng try-catch
3. Sử dụng try-catch-finally
4. Minh họa trong công cụ lập trình

Video lý thuyết

Video minh họa

### Mục đích sử dụng của try, catch, finally

* Các keyword trong xử lý ngoại lệ của Java gồm: try, catch, finally, throw, throws.
* Nội dung bài này ta sẽ tìm hiểu ba keyword đầu. Nội dung bài kế tiếp tìm hiểu hai keyword cuối.
* Cụ thể ý nghĩa sử dụng của chúng như sau:
  + Đoạn code có thể xảy ra ngoại lệ sẽ được đặt trong khối try. Như vậy khối try làm nhiệm vụ giám sát ngoại lệ có xảy ra hay không. Mỗi khối try thường đi liền với 1 vài khối catch/finally tương ứng.
  + Khi ngoại lệ xảy ra trong khối try, các khối catch đi sau try sẽ được kiểm tra. Nếu kiểu của ngoại lệ xảy ra trong try trùng với kiểu ngoại lệ trong khối catch nào, khối catch đó sẽ được gọi ra để xử lý ngoại lệ mà nó đảm nhận. Một try có thể có 1 hoặc nhiều khối catch đi kèm.
  + Khối finally thường là khối tùy chọn, tức có thể có hoặc không. Nếu có, khối này luôn được thực hiện cho dù ngoại lệ có xảy ra hay không. Khối finally thường dùng để dọn dẹp, đóng kết nối, thực hiện các hành động luôn xảy ra trong bất kì điều kiện nào.
* Ta thường sử dụng cặp try-catch, try-catch-finally. Chi tiết trình bày ở phần kế tiếp.

### Sử dụng try-catch

* Để bắt và xử lý ngoại lệ ta dùng cặp try-catch, cú pháp tổng quát như sau:

try {

// đoạn code có thể xảy ra ngoại lệ

} catch (ExceptionType e) {

// đoạn code xử lý ngoại lệ

}

* Trong đó:
  + Thân của khối try sẽ chứa các đoạn code có thể xảy ra ngoại lệ
  + ExceptionType là kiểu ngoại lệ có thể xảy ra trong khối try mà chương trình muốn xử lý.
  + Đoạn code xử lý ngoại lệ dùng để nêu hướng xử lý, nêu vấn đề đã nảy sinh để người dùng khắc phục. Ví dụ nhập đúng định dạng ngày/tháng/năm chẳng hạn. Nếu ta không biết cách xử lý hoặc muốn trì hoãn xử lý ngoại lệ, ta có thể ném nó ra ngoài để xử lý tiếp với throw(học trong bài kế tiếp).
* Sau đây là một ví dụ:

package lesson60;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

public class Lesson60 {

public static void main(String[] args) {

SimpleDateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy");

var dateString = "27-05-2025";

try {

// dòng code dưới đây có thể xảy ra ngoại lệ:

Date myBirthday = dateFormat.parse(dateString);

System.out.println("Đối tượng date hoàn chỉnh: " + myBirthday);

} catch (ParseException e) {

System.out.println("Không thể chuyển đổi String sang Date");

System.out.println("Ngoại lệ: " + e.getMessage());

}

// các câu lệnh kế tiếp phía sau

System.out.println("Thực hiện các câu lệnh tiếp theo...");

}

}

* Kết quả: khi ngoại lệ xảy ra chương trình sẽ không kết thúc đột ngột mà ta sẽ nhận được cách thông báo về điều gì đã xảy ra. Từ đó khắc phục sự cố. Các đoạn chương trình khác hoặc tính năng khác vẫn hoạt động bình thường.

Không thể chuyển đổi String sang Date

Ngoại lệ: Unparseable date: "27-05-2025"

Thực hiện các câu lệnh tiếp theo...

#### Một khối try có thể có nhiều khối catch đi kèm.

* Cú pháp tổng quát:

**try {**

**// đoạn code có thể xảy ra ngoại lệ**

**} catch (ExceptionType1 e1) {**

**// đoạn code xử lý ngoại lệ kiểu ExceptionType1**

**} catch (ExceptionType2 e2) {**

**// đoạn code xử lý ngoại lệ kiểu ExceptionType2**

**} catch (ExceptionType3 e3) {**

**// đoạn code xử lý ngoại lệ kiểu ExceptionType3**

**} catch (...) {**

**// đoạn code xử lý ngoại lệ kiểu ...**

**}**

* Trong đó:
  + Mỗi ExceptionTypei là kiểu ngoại lệ i nào đó cần bắt để xử lý.
  + Khi ta đặt các khối catch liên tiếp như trên thì lưu ý không để các lớp Exception cha ở phía trên, mà nếu có thì nên để ở sau cùng. Lý do là tính đa hình trong Java cho phép kiểu cha tham chiếu được đến các kiểu con nên làm như vậy sẽ không có nhiều ý nghĩa.
* Ta cũng có thể gộp nhóm các ngoại lệ vào cùng một catch như sau:

<strong>try {

// đoạn code có thể xảy ra ngoại lệ

} catch (ExceptionType1 | ExceptionType2 | ExceptionType3 ex) {

// đoạn code xử lý ngoại lệ chung

}</strong>

* Trong đó ta có thể thấy các ngoại lệ được viết chung vào trong ngoặc tròn của catch và phân tách nhau bởi dấu |. Cũng lưu ý rằng chỉ dùng một biến ex như trên cho nhiều kiểu ngoại lệ vì chỉ ngoại lệ được phát hiện đầu tiên được gọi trong catch.
* Ví dụ:

package lesson60;

import java.io.File;

import java.io.FileNotFoundException;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.Scanner;

public class Lesson60 {

public static void main(String[] args) {

SimpleDateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy");

var dateString = "";

try {

// dòng code dưới đây có thể xảy ra ngoại lệ:

Scanner fileReader = new Scanner(new File("input.txt"));

dateString = fileReader.nextLine(); // đọc dữ liệu từ file

Date myBirthday = dateFormat.parse(dateString);

System.out.println("Đối tượng date hoàn chỉnh: " + myBirthday);

} catch (ParseException e) {

System.out.println("Đã xảy ra ngoại lệ: " + e.getMessage());

System.out.println("Vui lòng kiểm tra lại!");

} catch (FileNotFoundException e) {

System.out.println("Đã xảy ra ngoại lệ: " + e.getMessage());

System.out.println("Vui lòng kiểm tra lại!");

}

// các câu lệnh kế tiếp phía sau

System.out.println("Thực hiện các câu lệnh tiếp theo...");

}

}

* Tương đương với:

package lesson60;

package lesson60;

import java.io.File;

import java.io.FileNotFoundException;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.Scanner;

public class Lesson60 {

public static void main(String[] args) {

SimpleDateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy");

var dateString = "";

try {

// dòng code dưới đây có thể xảy ra ngoại lệ:

Scanner fileReader = new Scanner(new File("input.txt"));

dateString = fileReader.nextLine(); // đọc dữ liệu từ file

Date myBirthday = dateFormat.parse(dateString);

System.out.println("Đối tượng date hoàn chỉnh: " + myBirthday);

} catch (ParseException | FileNotFoundException e) {

System.out.println("Đã xảy ra ngoại lệ: " + e.getMessage());

System.out.println("Vui lòng kiểm tra lại!");

}

// các câu lệnh kế tiếp phía sau

System.out.println("Thực hiện các câu lệnh tiếp theo...");

}

}

Đã xảy ra ngoại lệ: input.txt (The system cannot find the file specified)

Vui lòng kiểm tra lại!

Thực hiện các câu lệnh tiếp theo...

### Sử dụng try-catch-finally

* Ta có thể thêm tùy chọn khối finally vào sau cùng của try-catch để tạo thành try-catch-finally:
* Cú pháp tổng quát:

**try {**

**// đoạn code có thể xảy ra ngoại lệ**

**} catch (ExceptionType e) {**

**// đoạn code xử lý ngoại lệ**

**} finally {**

**// đoạn code luôn được thực thi, vd: đóng kết nối, dọn dẹp**

**}**

* Trong đó:
  + Đoạn chương trình trong khối finally luôn chạy cho dù có xảy ra ngoại lệ hay không.
  + Finally luôn ở cuối của chuỗi try-catch-finally.
* Lưu ý:
  + Khối catch không thể tồn tại nếu không có khối try.
  + Khối try không thể tồn tại một mình nếu không có catch hay finally đi sau nó.
  + Khối finally không có ý nghĩa nếu không có try-catch phía trước nó.
  + Không thể tồn tại bất kì dòng code nào xen giữa khối try-catch-finally.
* Ví dụ:

package lesson60;

import java.io.File;

import java.io.FileNotFoundException;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.Scanner;

public class Lesson60 {

public static void main(String[] args) {

SimpleDateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy");

Date myBirthday = null;

var dateString = "";

try {

// dòng code dưới đây có thể xảy ra ngoại lệ:

Scanner fileReader = new Scanner(new File("input.txt"));

dateString = fileReader.nextLine(); // đọc dữ liệu từ file

myBirthday = dateFormat.parse(dateString);

} catch (ParseException | FileNotFoundException e) {

System.out.println("Đã xảy ra ngoại lệ: " + e.getMessage());

System.out.println("Vui lòng kiểm tra lại!");

} finally { // đoạn chương trình sau luôn chạy

if (myBirthday == null) {

myBirthday = new Date();

}

System.out.println("==> Sinh nhật của tôi là: " +

dateFormat.format(myBirthday) + " <==");

}

// các câu lệnh kế tiếp phía sau

System.out.println("Thực hiện các câu lệnh tiếp theo...");

}

}

* Kết quả:

Đã xảy ra ngoại lệ: input.txt (The system cannot find the file specified)

Vui lòng kiểm tra lại!

==> Sinh nhật của tôi là: 15/10/2020 <==

Thực hiện các câu lệnh tiếp theo...

**Bài 10.4. Sử dụng throw và throws**

**Nội dung bài học**

1. Sử dụng throw
2. Sử dụng throws
3. Sự khác nhau giữa throw vs throws
4. Minh họa trong công cụ lập trình

Video lý thuyết

Video minh họa

**Sử dụng throw**

* Keyword throw dùng để ném một ngoại lệ một cách tường minh.
* Hữu ích khi muốn chủ động tạo ra ngoại lệ trong một điều kiện nào đó. Ví dụ nếu rút tiền quá giới hạn cho phép ta ném một ngoại lệ E nào đó, khi nhập điểm cho sinh viên mà nhập giá trị < 0 hoặc > 10 ta chủ động ném một ngoại lệ nào đó: IlligalGradeException chẳng hạn. Cụ thể ta sẽ cùng ứng dụng trong bài học kế tiếp.
* Ngoại lệ bị ném có thể là ngoại lệ mới hoặc là ngoại lệ bạn đã bắt và xử lý trong khối catch.
* Throw được sử dụng bên trong thân phương thức thường trong khối catch.
* Ví dụ:

package Lesson61;

public class Lesson61 {

public static void main(String[] args) {

method1(); // gọi đến method1

// các câu lệnh kế tiếp sau nơi bắt ngoại lệ:

System.out.println("Thực hiện các câu lệnh kế tiếp...");

}

private static void method1() {

String str = null;

if (str == null) {

var msg = "str là null, vui lòng kiểm tra lại";

throw new NullPointerException(msg);

}

System.out.println("Các lệnh phía sau nơi ném ngoại lệ");

}

}

* Kết quả:

Exception in thread "main" java.lang.NullPointerException: str là null, vui lòng kiểm tra lại

at Lesson61.Lesson61.method1(Lesson61.java:14)

at Lesson61.Lesson61.main(Lesson61.java:5)

* Một ví dụ khác:

package Lesson61;

public class Lesson61 {

public static void main(String[] args) {

try {

method1(); // gọi đến method1

} catch (NullPointerException e) {

System.out.println("Đã xảy ra ngoại lệ " + e.getMessage());

System.out.println("Vui lòng kiểm tra lại!");

}

// các câu lệnh kế tiếp sau nơi bắt ngoại lệ:

System.out.println("Thực hiện các câu lệnh kế tiếp...");

}

private static void method1() {

String str = null;

try {

System.out.println("Số kí tự trong chuỗi str là: " + str.length());

} catch (NullPointerException e) {

System.out.println("Chuỗi kí tự vô định");

throw e; // chủ động ném ngoại lệ này ra

}

System.out.println("Các lệnh phía sau nơi ném ngoại lệ");

}

}

Chuỗi kí tự vô định

Đã xảy ra ngoại lệ null

Vui lòng kiểm tra lại!

Thực hiện các câu lệnh kế tiếp...

**Sử dụng throws**

* Khi bạn muốn trì hoãn việc xử lý ngoại lệ hoặc đẩy việc xử lý ngoại lệ cho một phương thức khác, sử dụng throws.
* Keyword throws nằm ở khai báo tên phương thức, sau dấu ngoặc ).
* Bạn có thể khai báo 1 hoặc nhiều kiểu ngoại lệ khác nhau với throws.
* Nếu khai báo throws với các ngoại lệ thuộc loại checked exception, nơi gọi phương thức có throws phải:
  + Hoặc là xử lý ngoại lệ được đẩy về hoặc là
  + Tiếp tục throws các checked exception này.
* Ví dụ:

package Lesson61;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

public class Lesson61 {

public static void main(String[] args) {

try {

method1();

System.out.println("Xử lý ngoại lệ có thể xảy ra trong main");

} catch (ParseException e) {

System.out.println("Đã xảy ra ngoại lệ: " + e.getMessage());

System.out.println("Vui lòng kiểm tra lại!");

}

}

private static void method1() throws ParseException {

method2(); // không xử lý ngoại lệ xảy ra -> tiếp tục throws

}

private static void method2() throws ParseException {

SimpleDateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy");

Date myBirthday = null;

var dateString = "";

// dòng code dưới đây có thể xảy ra ngoại lệ:

dateString = "22/10/2020"; // đọc dữ liệu từ file

myBirthday = dateFormat.parse(dateString);

System.out.println("Sinh nhật của tôi là: " + dateFormat.format(myBirthday));

}

}

* Kết quả:

Sinh nhật của tôi là: 22/10/2020

Xử lý ngoại lệ có thể xảy ra trong main

Mô tả thứ tự thực hiện của phương thức nếu có throws một checked exception:

Trong main() gọi method1, method1 gọi method2. method2 xảy ra ngoại lệ nhưng nó trì hoãn và khai báo lại với throws. method1 không xử lí ngoại lệ do method2 chuyển giao nên nó tiếp tục trì hoãn và đẩy nhiệm vụ này cho main. Trong main xử lý ngoại lệ nó được bàn giao từ method1 bằng khối try-catch nên main không cần throws nữa.

[Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, biểu đồ

Mô tả được tạo tự động](https://braniumacademy.net/wp-content/uploads/2020/08/throw-vs-throws.png)

* Ta có thể throws cả các ngoại lệ thuộc loại unchecked:

package Lesson61;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

public class Lesson61 {

public static void main(String[] args) {

try {

method1();

System.out.println("Xử lý ngoại lệ có thể xảy ra trong main");

} catch (ParseException | NullPointerException e) {

System.out.println("Đã xảy ra ngoại lệ: " + e.getMessage());

System.out.println("Vui lòng kiểm tra lại!");

}

// các câu lệnh kế tiếp sau nơi bắt ngoại lệ:

System.out.println("Thực hiện các câu lệnh kế tiếp..."); }

private static void method1() throws ParseException, NullPointerException {

method2(); // không xử lý ngoại lệ xảy ra -> tiếp tục throws

}

private static void method2() throws ParseException, NullPointerException {

SimpleDateFormat dateFormat = null;

Date myBirthday = null;

var dateString = "";

// dòng code dưới đây có thể xảy ra ngoại lệ:

dateString = "22/10/2020"; // đọc dữ liệu từ file

myBirthday = dateFormat.parse(dateString);

System.out.println("Sinh nhật của tôi là: " + dateFormat.format(myBirthday));

}

}

* Kết quả:

Đã xảy ra ngoại lệ: null

Vui lòng kiểm tra lại!

Thực hiện các câu lệnh kế tiếp...

**Sự khác nhau giữa throw vs throws**

|  |  |
| --- | --- |
| **Throw** | **throws** |
| Dùng để trực tiếp/chủ động ném một ngoại lệ ra khỏi khối/phương thức | Dùng để khai báo sự trì hoãn của một hoặc một số ngoại lệ nào đó |
| Không dùng để ném ngoại lệ thuộc loại checked exception có sẵn của Java | Có thể khai bảo cả checked exception |
| Throw sử dụng để ném một đối tượng | Throws dùng để khai báo các kiểu ngoại lệ |
| Bạn chỉ có thể ném một ngoại lệ với throw | Bạn có thể khai báo nhiều kiểu ngoại lệ với throws |
| Xuất hiện ở trong thân phương thức | Xuất hiện ở khai báo tên phương thức |

**Bài 10.5. Ngoại lệ do người dùng tự định nghĩa**

**Nội dung bài học**

1. Quy tắc tạo ngoại lệ
2. Tạo ngoại lệ loại checked exception
3. Tạo ngoại lệ loại unchecked exception
4. Minh họa trong công cụ lập trình
5. Bài tập thực hành

Video lý thuyết

Video minh họa

**Quy tắc tạo ngoại lệ**

Sau đây là các quy tắc khi bạn muốn tạo ngoại lệ:

* Tất cả các kiểu ngoại lệ đều phải là con của Throwable.
* Tên ngoại lệ có đuôi Exception.
* Nếu bạn muốn tạo một checked exception, tức ngoại lệ sẽ được cảnh báo bởi trình biên dịch khi code, người lập trình bắt buộc phải xử lý nó thì cho ngoại lệ của bạn kế thừa Exception.
* Nếu bạn muốn tạo một runtime exception, cho lớp ngoại lệ của bạn kế thừa RuntimeException.
* Một lớp ngoại lệ cũng giống như các lớp thông thường khác có đầy đủ constructor, các trường, các phương thức.

**Tạo ngoại lệ loại checked exception**

Phần này ta sẽ tạo một ngoại lệ thuộc loại checked exception. Kịch bản như sau:

* Biết sinh viên có các thông tin về họ và tên, mã sinh viên, điểm trung bình. Giả sử điểm là giá trị ở thang 10 điểm, tức là từ 0.0 đến 10.0. Nếu người dùng nhập các giá trị điểm ngoài phạm vi trên mà gán vào làm điểm trung bình cho sinh viên, hệ thống sẽ phát sinh ngoại lệ. Ngoại lệ này phải được xử lý tại thời điểm biên dịch chương trình để đảm bảo người dùng nhập đúng điểm cho sinh viên. Nếu ngoại lệ xảy ra cần đảm bảo rằng điểm trung bình của sinh viên là 0.0 và chương trình tiếp tục thực hiện các chức năng khác một cách bình thường.

Các bước để giải quyết vấn đề trên:

* Tạo lớp InvalidGradeException kế thừa Exception vì đây là checked exception.
* Tạo một trường float invalidGrade để lưu giá trị điểm không hợp lệ đã được cung cấp.
* Tạo các constructor, các getter/setter cần thiết cho lớp ngoại lệ vừa tạo.
* Tại nơi sử dụng, trong phương thức thiết lập điểm trung bình cho sinh viên, ta kiểm tra nếu điểm nhận được không hợp lệ thì ta gán điểm tb là 0.0 và chủ động văng ngoại lệ InvalidGradeException.
* Tại nơi sử dụng đoạn code có khai báo ngoại lệ bị trì hoãn ta viết code xử lý ngoại lệ tương ứng.

Cụ thể:

* Lớp InvalidGradeException:

package lesson62;

/\*\*

\* Lớp này mô tả ngoại lệ điểm số không hợp lệ

\*/

public class InvalidGradeException extends Exception {

private float invalidGrade;

public InvalidGradeException() {

}

public InvalidGradeException(String message, float invalidGrade) {

super(message);

this.invalidGrade = invalidGrade;

}

public float getInvalidGrade() {

return invalidGrade;

}

}

* Lớp Student kiểm tra và quyết định có văng ngoại lệ không:

package lesson62;

public class Student {

private String id;

private String fullName;

private float avgGrade;

public Student() {

}

public Student(String id) {

this.id = id;

}

public Student(String id, String fullName) {

setId(id);

setFullName(fullName);

}

public Student(String id, String fullName, float avgGrade)

throws InvalidGradeException {

this.id = id;

this.fullName = fullName;

setAvgGrade(avgGrade);

}

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public String getFullName() {

return fullName;

}

public void setFullName(String fullName) {

this.fullName = fullName;

}

public float getAvgGrade() {

return avgGrade;

}

public void setAvgGrade(float avgGrade) throws InvalidGradeException {

if (avgGrade >= 0 && avgGrade <= 10) {

this.avgGrade = avgGrade;

} else {

this.avgGrade = 0;

var msg = "Bạn đã nhập điểm không hợp lệ";

throw new InvalidGradeException(msg, avgGrade);

}

}

}

* Tạo lớp xử lý ngoại lệ:

package lesson62;

import java.util.Scanner;

public class Test {

public static void main(String[] args) {

Student student = new Student();

var input = new Scanner(System.in);

System.out.println("Tên sinh viên: ");

var fullName = input.nextLine();

System.out.println("Nhập điểm TB(0.0->10.0) cho sinh viên: ");

var grade = input.nextFloat();

student.setFullName(fullName);

try { // giám sát ngoại lệ có thể xảy ra

student.setAvgGrade(grade);

} catch (InvalidGradeException e) { // xử lý ngoại lệ

e.printStackTrace(); // in ra thứ tự lời gọi gây ra ngoại lệ

}

showStudentInfo(student);

}

/\*\*

\* phương thức hiển thị tên và điểm trung bình của sinh viên

\*

\* @param student sinh viên cần hiển thị tên

\*/

private static void showStudentInfo(Student student) {

System.out.println("Tên: " + student.getFullName());

System.out.printf("Điểm TB: %5.2f\n", student.getAvgGrade());

}

}

* Kết quả:

Tên sinh viên:

Ngô Văn Hoàng

Nhập điểm TB(0.0->10.0) cho sinh viên:

11.25

lesson62.InvalidGradeException: Bạn đã nhập điểm không hợp lệ

at lesson62.Student.setAvgGrade(Student.java:53)

at lesson62.Test.main(Test.java:15)

Tên: Ngô Văn Hoàng

Điểm TB: 0.00

**Tạo ngoại lệ loại unchecked exception**

* Giả sử vẫn là kịch bản trong phần 2 nhưng thay bằng unchecked exception.
* Các bước để giải quyết vấn đề trên:
  + Tạo lớp InvalidGradeException kế thừa RuntimeException vì đây là unchecked exception.
  + Tạo một trường float invalidGrade để lưu giá trị điểm không hợp lệ đã được cung cấp.
  + Tạo các constructor, các getter/setter cần thiết cho lớp ngoại lệ vừa tạo.
  + Tại nơi sử dụng, trong phương thức thiết lập điểm trung bình cho sinh viên, ta kiểm tra nếu điểm nhận được không hợp lệ thì ta gán điểm tb là 0.0 và chủ động văng ngoại lệ InvalidGradeException.
  + Tại nơi sử dụng đoạn code có khai báo ngoại lệ bị trì hoãn ta viết code xử lý ngoại lệ tương ứng.

Code mẫu:

* Lớp ngoại lệ:

package lesson62;

/\*\*

\* Lớp này mô tả ngoại lệ điểm số không hợp lệ

\*/

public class InvalidGradeException extends RuntimeException {

private float invalidGrade;

public InvalidGradeException() {

}

public InvalidGradeException(String message, float invalidGrade) {

super(message);

this.invalidGrade = invalidGrade;

}

public float getInvalidGrade() {

return invalidGrade;

}

}

* Lớp sinh viên kiểm tra và văng ngoại lệ:

package lesson62;

public class Student {

private String id;

private String fullName;

private float avgGrade;

public Student() {

}

public Student(String id) {

this.id = id;

}

public Student(String id, String fullName) {

setId(id);

setFullName(fullName);

}

public Student(String id, String fullName, float avgGrade) {

this.id = id;

this.fullName = fullName;

setAvgGrade(avgGrade);

}

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public String getFullName() {

return fullName;

}

public void setFullName(String fullName) {

this.fullName = fullName;

}

public float getAvgGrade() {

return avgGrade;

}

// với ngoại lệ unchecked không bắt buộc phải throws

public void setAvgGrade(float avgGrade) {

if (avgGrade >= 0 && avgGrade <= 10) {

this.avgGrade = avgGrade;

} else {

this.avgGrade = 0;

var msg = "Bạn đã nhập giá trị điểm không hợp lệ";

throw new InvalidGradeException(msg, avgGrade);

}

}

}

* Lớp thực thi nơi xử lý ngoại lệ:

package lesson62;

import java.util.Scanner;

public class Test {

public static void main(String[] args) {

Student student = new Student();

var input = new Scanner(System.in);

System.out.println("Tên sinh viên: ");

var fullName = input.nextLine();

System.out.println("Nhập điểm TB(0.0->10.0) cho sinh viên: ");

var grade = input.nextFloat();

student.setFullName(fullName);

try { // giám sát ngoại lệ có thể xảy ra

student.setAvgGrade(grade);

} catch (InvalidGradeException e) { // xử lý ngoại lệ

e.printStackTrace(); // in ra thứ tự lời gọi gây ra ngoại lệ

}

showStudentInfo(student);

}

/\*\*

\* phương thức hiển thị tên và điểm trung bình của sinh viên

\*

\* @param student sinh viên cần hiển thị tên

\*/

private static void showStudentInfo(Student student) {

System.out.println("Tên: " + student.getFullName());

System.out.printf("Điểm TB: %5.2f\n", student.getAvgGrade());

}

}

* Kết quả:

Tên sinh viên:

Nguyễn Mai Loan

Nhập điểm TB(0.0->10.0) cho sinh viên:

-6

lesson62.InvalidGradeException: Bạn đã nhập giá trị điểm không hợp lệ

at lesson62.Student.setAvgGrade(Student.java:53)

at lesson62.Test.main(Test.java:15)

Tên: Nguyễn Mai Loan

Điểm TB: 0.00

# Bài 10.6. Triển khai project theo mô hình MVC

### Nội dung bài học

1. [Giới thiệu về mô hình MVC](https://braniumacademy.net/courses/java/lessons/java-mvc-implement/#s1)
2. [Triển khai](https://braniumacademy.net/courses/java/lessons/java-mvc-implement/#s2)
3. [Project mẫu](https://braniumacademy.net/courses/java/lessons/java-mvc-implement/#s3)

### Giới thiệu về mô hình MVC

* MVC là mô hình kiến trúc ứng dụng phân tầng, trong đó chia ứng dụng thành 3 tầng riêng rẽ:
  + **M**: tầng model, chứa các lớp thực thể của project, ứng dụng, ví dụ các Student, Subject, Account, Order, Cart…
  + **V**: tầng view, chứa các lớp UI hiển thị dữ liệu cho người dùng và nhận tương tác của người dùng. Ví dụ các JFrame trong JavaSwing, các lớp chứa phương thức main, các Activity, Fragment trong lập trình Android, …
  + **C**: các lớp controller, thường bao gồm các interface và các lớp triển khai của nó. Các lớp này có nhiệm vụ thực thi các logic nghiệp vụ và các chức năng chính của hệ thống: tìm kiếm, sắp xếp, tính toán, xử lý logic..
* Thường thì mỗi lớp trong tầng model sẽ có 1 interface và một lớp triển khai tương ứng trong tầng controller. Ví dụ ta có lớp Student trong tầng model thì sẽ có StudentController interface và lớp triển khai StudentControllerImp trong tầng controller.

### Triển khai

* Sau đây ta sẽ triển khai project mẫu theo mô hình MVC có cấu trúc project như sau:

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, số

Mô tả được tạo tự động

* Lớp Operator mô tả các thành phần cấu thành của phép toán:

package net.braniumacademy.model;

public class Operator {

private double first; // toán hạng 1

private double second; // toán hạng 2

private double result; // kết quả phép toán

private ArithmeticException error; // ngoại lệ

public Operator() {

}

public Operator(double first, double second, double result, ArithmeticException error) {

this.first = first;

this.second = second;

this.result = result;

this.error = error;

}

public double getFirst() {

return first;

}

public void setFirst(double first) {

this.first = first;

}

public double getSecond() {

return second;

}

public void setSecond(double second) {

this.second = second;

}

public double getResult() {

return result;

}

public void setResult(double result) {

this.result = result;

}

public ArithmeticException getError() {

return error;

}

public void setError(ArithmeticException error) {

this.error = error;

}

}

* Interface CalculatorController:

package net.braniumacademy.controller;

import net.braniumacademy.model.Operator;

public interface CalculatorController {

Operator add(double a, double b);

Operator sub(double a, double b);

Operator mul(double a, double b);

Operator div(double a, double b);

Operator mod(double a, double b);

}

* Lớp triển khai controller CalculatorControllerImp:

package net.braniumacademy.controller;

import net.braniumacademy.model.Operator;

public class CalculatorControllerImp implements CalculatorController {

@Override

public Operator add(double a, double b) {

return new Operator(a, b, a + b, null);

}

@Override

public Operator sub(double a, double b) {

return new Operator(a, b, a - b, null);

}

@Override

public Operator mul(double a, double b) {

return new Operator(a, b, a \* b, null);

}

@Override

public Operator div(double a, double b) {

if (b == 0) {

return new Operator(a, b, 0, new ArithmeticException("Dive by zero exception"));

}

return new Operator(a, b, a / b, null);

}

@Override

public Operator mod(double a, double b) {

if (b == 0) {

return new Operator(a, b, 0, new ArithmeticException("Dive by zero exception"));

}

return new Operator(a, b, a % b, null);

}

}

* Lớp Main chứa giao diện ứng dụng hiển thị và nhận tương tác người dùng:

package net.braniumacademy.view;

import net.braniumacademy.controller.CalculatorControllerImp;

import net.braniumacademy.model.Operator;

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

var scanner = new Scanner(System.in);

System.out.println("Toán hạng a: ");

var a = scanner.nextDouble();

System.out.println("Toán hạng b: ");

var b = scanner.nextDouble();

var calculator = new CalculatorControllerImp();

var resultAddition = calculator.add(a, b);

var resultSubtraction = calculator.sub(a, b);

var resultMultiply = calculator.mul(a, b);

var resultDivision = calculator.div(a, b);

var resultMode = calculator.mod(a, b);

showResult(resultAddition, " + ");

showResult(resultSubtraction, " - ");

showResult(resultMultiply, " \* ");

showResult(resultDivision, " / ");

showResult(resultMode, " % ");

}

private static void showResult(Operator operator, String option) {

if (operator.getError() == null) {

System.out.println(operator.getFirst()

+ option + operator.getSecond() + " = " + operator.getResult());

} else {

System.out.print("Error at operator: " + option + ". ");

System.out.println(operator.getError().getMessage());

}

}

}

* Kết quả:

// kết quả phép toán thường:

Toán hạng a:

200

Toán hạng b:

300

200.0 + 300.0 = 500.0

200.0 - 300.0 = -100.0

200.0 \* 300.0 = 60000.0

200.0 / 300.0 = 0.6666666666666666

200.0 % 300.0 = 200.0

// kết quả phép toán có ngoại lệ:

Toán hạng a:

500

Toán hạng b:

0

500.0 + 0.0 = 500.0

500.0 - 0.0 = 500.0

500.0 \* 0.0 = 0.0

Error at operator: / . Dive by zero exception

Error at operator: % . Dive by zero exception

**Bài 11.1. Phương thức generic**

**Nội dung bài học**

1. Khái niệm và đặc điểm
2. Cú pháp tổng quát
3. Giới hạn các kiểu đã tham số hóa
4. Bài tập thực hành

Video lý thuyết

Video minh họa

**Khái niệm và đặc điểm**

* Phương thức generic là phương thức có kiểu được tham số hóa.
* Ý tưởng của phương thức generic là cho phép sử dụng các kiểu(Integer, class, interface…) làm tham số của phương thức.
* Mục đích của phương thức generic: tái sử dụng cùng một chức năng cho nhiều kiểu dữ liệu khác nhau.
* Khác biệt giữa đối số thông thường và đối số generic: đối số thông thường là các giá trị. Trong khi đối số của generic là các kiểu dữ liệu.
* Lợi ích của generic:
  + Kiểm soát được kiểu đang sử dụng tại thời điểm biên dịch chương trình: khi kiểu được đưa vào làm đối số generic thì kiểu sẽ được kiểm tra nên code sẽ đảm bảo tính đúng đắn hơn. Mặt khác lỗi được phát hiện và sửa đổi tại thời điểm biên dịch dễ và thuận tiện hơn so với việc phát hiện và sửa lỗi tại thời điểm chạy chương trình.
  + Giảm thao tác ép kiểu các thành phần trong generic: chỉ những kiểu hợp lệ được chỉ rõ mới được phép đưa vào phương thức generic.
  + Cho phép thực hiện các thuật toán chung áp dụng cho tập hợp của nhiều kiểu dữ liệu khác nhau. Hơn nữa code generic còn dễ đọc, an toàn về kiểu.
* Ví dụ về phương thức generic:

package lesson63;

import java.util.ArrayList;

public class Lesson63 {

public static void main(String[] args) {

// tạo danh sách bạn bè kiểu String

ArrayList<String> friends = new ArrayList<>();

friends.add("Linh");

friends.add("Hương");

friends.add("Phương");

friends.add("Loan");

// danh sách các con số kiểu nguyên Integer

ArrayList<Integer> numbers = new ArrayList<>();

numbers.add(1);

numbers.add(2);

numbers.add(3);

numbers.add(4);

// hiển thị hai danh sách trên bằng 1 phương thức

System.out.println("Danh sách những người bạn: ");

showList(friends);

System.out.println("Danh sách những con số: ");

showList(numbers);

}

/\*\*

\* phương thức dùng để hiển thị danh sách các phần tử

\* của một kiểu bất kì.

\*

\* @param list danh sách cần hiển thị

\* @param <T> kiểu cần thực hiện thao tác

\*/

public static <T> void showList(ArrayList<T> list) {

for (var e : list) {

System.out.print(e + " ");

}

System.out.println();

}

}

Danh sách những người bạn:

Linh Hương Phương Loan

Danh sách những con số:

1 2 3 4

* Ở ví dụ trên ta thấy rằng ta có thể viết một phương thức duy nhất nhưng có thể áp dụng hành động đó cho nhiều kiểu dữ liệu khác nhau. Vậy thuận tiện và tiết kiệm thời gian hơn rất nhiều so với việc mỗi kiểu dữ liệu lại phải viết một lần cùng phương thức đó(nạp chồng).

**Cú pháp tổng quát**

* Sau đây là cú pháp tổng quát của phương thức generic:

access none-access <Types> methodType methodName(params) {

// method body

// do something here

}

* Trong đó:
  + Các thành phần đặc trưng của phương thức được giữ nguyên và bổ sung thành phần generic.
  + Phần ***Types***đặt trong cặp ngoặc **<>** là danh sách các kiểu có thể được sử dụng làm kiểu trả về của phương thức, kiểu của danh sách các tham số. Trong phần này có thể khai báo nhiều kiểu, mỗi kiểu thường dùng 1 chữ cái hoa biểu thị, phân tách nhau bởi dấu phẩy. Ví dụ: <T>, <T, V>, <U, V, T>.
  + Cặp ngoặc <> là bắt buộc và là đặc trưng của generic. Cặp ngoặc <> luôn bao bên ngoài danh sách các kiểu tham số hóa được sử dụng trong phương thức. Vị trí của <> luôn nằm sau access modifier và kiểu trả về của phương thức.
  + Danh sách tham số có thể dùng các kiểu đã được khai báo trong <Types> để làm kiểu.
* Ví dụ:

public static <T> void showList(ArrayList<T> list) {

for (var e : list) {

System.out.print(e + " ");

}

System.out.println();

}

* Một ví dụ khác:

/\*\*

\* Lấy phần tử chính giữa của một List

\*

\* @param list danh sách các phần tử

\* @param <T> kiểu của danh sách đang thao tác

\* @return giá trị chính giữa tìm được

\*/

public <T> T getMid(ArrayList<T> list) {

int mid = list.size() / 2;

return list.get(mid);

}

**Giới hạn các kiểu đã tham số hóa**

* Việc giới hạn các kiểu được tham số hóa trong <> của phương thức generic là cách thức chính giúp ta sử dụng phương thức generic để triển khai các thuật toán phổ biến dùng chung cho nhiều kiểu dữ liệu khác nhau như sắp xếp, tìm kiếm, đếm…
* Để giới hạn 1 kiểu đã tham số hóa ta thực hiện theo cú pháp:

<SubType extends SuperType>

* Ví dụ:

package lesson63;

import java.util.ArrayList;

public class Lesson63 {

public static void main(String[] args) {

// tạo danh sách bạn bè kiểu String

ArrayList<String> friends = new ArrayList<>();

friends.add("Linh");

friends.add("Hương");

friends.add("Phương");

friends.add("Loan");

// danh sách các con số kiểu nguyên Integer

ArrayList<Integer> numbers = new ArrayList<>();

numbers.add(1);

numbers.add(2);

numbers.add(3);

numbers.add(4);

Lesson63 myObj = new Lesson63();

// tìm phần tử lớn nhất trong danh sách số nguyên

var maxNumber = myObj.findMax(numbers);

// tìm người có tên đứng sau cùng trong danh sách bạn bè

var lastInFriendList = myObj.findMax(friends);

System.out.println("Giá trị số nguyên lớn nhất trong danh sách: "

+ maxNumber);

System.out.println("Người có tên sau cùng trong friend list: "

+ lastInFriendList);

}

/\*\*

\* Phương thức tìm giá trị lớn nhất trong danh sách cho trước

\*

\* @param list danh sách cần tìm giá trị lớn nhất

\* @param <T> kiểu của các phần tử trong danh sách

\* @return giá trị lớn nhất tìm được hoặc null nếu danh sách null hoặc rỗng

\*/

public <T extends Comparable> T findMax(ArrayList<T> list) {

if (list == null || list.size() == 0) {

return null;

}

var max = list.get(0);

for (var e : list) {

if (e.compareTo(max) > 0) {

max = e;

}

}

return max;

}

}

Giá trị số nguyên lớn nhất trong danh sách: 4

Người có tên sau cùng trong friend list: Phương

# Bài 11.2. Lớp generic

### Nội dung bài học

1. Khái niệm
2. Cú pháp tổng quát
3. Giới hạn trên, giới hạn dưới
4. Ví dụ minh họa
5. Bài tập thực hành

Video lý thuyết

Video minh họa

### Khái niệm

* Một kiểu generic là một lớp hoặc interface chứa các kiểu đã được tham số hóa.
* Mục đích của generic:
  + Tái sử dụng lại cùng chức năng cho nhiều kiểu dữ liệu khác nhau.
  + Cho phép kiểm soát các kiểu đang được sử dụng.
  + Cho code dễ đọc.
* Ví dụ về lớp generic: ArrayList<T>, ta có thể dùng đối tượng của lớp này để thao tác với rất nhiều kiểu dữ liệu: các kiểu lớp bao của kiểu nguyên thủy, các lớp, interface người dùng tự định nghĩa ra…

package lesson64;

import java.util.ArrayList;

public class GenericExample {

public static void main(String[] args) {

// dùng kiểu generic ArrayList<T> để lưu các giá trị kiểu String

ArrayList<String> friends = new ArrayList<>();

friends.add("Nhung");

friends.add("Linh");

friends.add("Khánh");

friends.add("Loan");

// dùng kiểu generic ArrayList<T> để lưu giá trị số thực:

ArrayList<Double> doubles = new ArrayList<>();

doubles.add(1.25);

doubles.add(3.12456);

doubles.add(6.1454);

doubles.add(99.899);

// dùng kiểu generic ArrayList<T> lưu các đối tượng của lớp Student

ArrayList<Student> students = new ArrayList<>();

students.add(new Student("B25DCCN100", "Ngô Thùy Linh"));

students.add(new Student("B25DCCN101", "Nguyễn Phương Hoa"));

students.add(new Student("B25DCCN102", "Trần Xuân Nhật"));

students.add(new Student("B25DCCN103", "Lê Hoài Nam"));

//...

}

}

class Student {

private String studentId;

private String fullName;

private String email;

private float avgGrade;

public Student() {

}

public Student(String studentId, String fullName) {

this.studentId = studentId;

this.fullName = fullName;

}

// ...

}

### Cú pháp tổng quát

* Cú pháp tổng quát:

access non-acess class name<T1, T2, T..., Tn> {

// do something here...

}

access non-acess interface name<T1, T2, T..., Tn> {

// do something here...

}

Trong đó:

* Các thành phần cơ bản của class, interface đều giữ nguyên.
* Bổ sung thêm các kiểu đã được tham số hóa trong cặp <>.
* Cặp <> phải nằm sau tên class/interface.
* Trong <> là các kí tự đại diện cho các kiểu được tham số hóa.
* Tên các kiểu đã tham số hóa được đặt theo quy ước:
  + Dùng 1 chữ cái hoa làm tham số đại diện cho kiểu tương ứng.
  + Các kí tự phổ biến và ý nghĩa:
    - E: element, hay phần tử
    - K: key, hay từ khóa, keyword
    - N: number, hay các con số
    - T: Type, hay kiểu
    - V: value, hay giá trị
    - S, U, V… các kiểu thứ 2, 3…
  + Các kiểu được cho ở trong cặp <> bắt buộc khác các kiểu nguyên thủy và chúng có thể là bất kì class, interface nào.
  + Khi khai báo kiểu generic ta cung cấp đầy đủ tên kiểu ở vế trái dấu = và có thể bỏ qua tên kiểu, chỉ cần <> ở vế bên phải.
  + Ví dụ sau tạo một lớp generic nhận vào hai giá trị ở hai kiểu K, V nào đó:

package lesson64;

public class MyMap<K, V> {

private K key;

private V value;

public MyMap() {

}

public MyMap(K key, V value) {

this.key = key;

this.value = value;

}

public K getKey() {

return key;

}

public void setKey(K key) {

this.key = key;

}

public V getValue() {

return value;

}

public void setValue(V value) {

this.value = value;

}

}

* Tạo và sử dụng đối tượng của MyMap:

package lesson64;

public class Lesson64 {

public static void main(String[] args) {

// tạo đối tượng thao tác với giá trị kiểu Integer, String

MyMap<Integer, String> map = new MyMap<>(1, "One");

System.out.println("Key: " + map.getKey()

+ " - Value: " + map.getValue());

// tạo đối tượng thao tác với giá trị kiểu String, String

MyMap<String, String> hello = new MyMap<>("Hello", "Xin chào");

System.out.println("Key: " + hello.getKey()

+ " - Value: " + hello.getValue());

}

}

Key: 1 - Value: One

Key: Hello - Value: Xin chào

### Giới hạn trên, giới hạn dưới

* Kí tự đại diện: dấu ? trong generic được gọi là kí tự đại diện. Kí tự này được sử dụng trong nhiều trường hợp khác nhau:
  + Làm kiểu của tham số
  + Làm kiểu của trường dữ liệu
  + Làm kiểu của biến cục bộ
  + Đôi khi là kiểu trả về

#### Giới hạn trên:

* Ta có thể sử dụng kí tự đại diện giới hạn trên để nới lỏng các hạn chế đối với một biến.
* Ví dụ: Ta muốn tạo ra một lớp nhận vào một danh sách giá trị kiểu Float, Integer, Long, … ta có thể đạt được điều này bằng sử dụng kí tự đại diện ? với extends.
* Lưu ý rằng extends ở đây hàm ý chung cho extends với class và implements cho interface.
* Khi sử dụng giới hạn trên, ta giới hạn các kiểu chưa biết lại trong phạm vi kiểu con hoặc bản thân một kiểu cụ thể nào đó đứng sau keyword extends.
* Ví dụ:

package lesson64;

import java.util.ArrayList;

public class UperBoundWildcard {

public static double averageValue(ArrayList<? extends Number> list) {

if (list == null || list.size() == 0) {

return 0;

}

double sum = 0;

for (var item : list) {

sum += item.doubleValue();

}

return sum / list.size();

}

public static void main(String[] args) {

// danh sách các số kiểu int

ArrayList<Integer> integers = new ArrayList<>();

integers.add(1);

integers.add(5);

integers.add(8);

// danh sách các số kiểu double

ArrayList<Double> doubles = new ArrayList<>();

doubles.add(1.25);

doubles.add(5.98);

doubles.add(78.15);

// danh sách các String object

ArrayList<String> names = new ArrayList<>();

names.add("Hoa");

names.add("Hương");

names.add("Phương");

names.add("Linh");

// lấy giá trị trung bình của mỗi danh sách:

System.out.printf("Giá trị trung bình các số nguyên: %5.2f\n",

averageValue(integers)); // 4.67

System.out.printf("Giá trị trung bình các số thực: %5.2f\n",

averageValue(doubles)); // 28.46

System.out.println("Giá trị trung bình các String object: %5.2f",

averageValue(names)); // error

}

}

#### Giới hạn dưới:

* Khi muốn tạo giới hạn dưới cho các kiểu được tham số hóa trong generic của Java, ta dùng kí tự đại diện kết hợp với keyword super.
* Khi sử dụng giới hạn dưới, ta thu hẹp các kiểu cần sử dụng chưa biết lại trong phạm vi các kiểu cha hoặc bản thân của một kiểu cụ thể nào đó đứng sau keyword super.
* Ví dụ: sử dụng giới hạn dưới để nhập liệu các phần tử cho danh sách.

package lesson64;

import java.util.ArrayList;

public class LowerBoundWildcard {

public static void addNewObject(ArrayList<? super Grandchildren> list) {

list.add(new Grandchildren("Anh"));

list.add(new Grandchildren("Trang"));

list.add(new Grandchildren("Thanh"));

}

public static void main(String[] args) {

ArrayList<Father> fathers = new ArrayList<>();

ArrayList<Child> children = new ArrayList<>();

ArrayList<Grandchildren> grandchildren = new ArrayList<>();

addNewObject(fathers); // ok

addNewObject(children); // ok

addNewObject(grandchildren); // ok

}

}

### Ví dụ minh họa

* Ví dụ 1: Sử dụng kiểu ArrayList<T>.

package lesson64;

import java.util.ArrayList;

public class GenericExample {

public static void main(String[] args) {

// dùng kiểu generic ArrayList<T> để lưu các giá trị kiểu String

ArrayList<String> friends = new ArrayList<>();

friends.add("Nhung");

friends.add("Linh");

friends.add("Khánh");

friends.add("Loan");

// dùng kiểu generic ArrayList<T> để lưu giá trị số thực:

ArrayList<Double> doubles = new ArrayList<>();

doubles.add(1.25);

doubles.add(3.12456);

doubles.add(6.1454);

doubles.add(99.899);

// dùng kiểu generic ArrayList<T> lưu các đối tượng của lớp Student

ArrayList<Student> students = new ArrayList<>();

students.add(new Student("B25DCCN100", "Ngô Thùy Linh"));

students.add(new Student("B25DCCN101", "Nguyễn Phương Hoa"));

students.add(new Student("B25DCCN102", "Trần Xuân Nhật"));

students.add(new Student("B25DCCN103", "Lê Hoài Nam"));

// hiển thị danh sách các kiểu trên

showInfo(friends);

showInfo(doubles);

showInfo(students);

}

/\*\*

\* phương thức dùng để hiển thị danh sách các phần tử kiểu T nào đó

\*

\* @param list danh sách cần hiển thị ra màn hình

\* @param <T> kiểu của các phần tử trong danh sách

\*/

private static <T> void showInfo(ArrayList<T> list) {

System.out.println("==========================================");

for (var e : list) {

System.out.println(e);

}

}

}

class Student {

private String studentId;

private String fullName;

private String email;

private float avgGrade;

public Student() {

}

public Student(String studentId, String fullName) {

this.studentId = studentId;

this.fullName = fullName;

}

public Student(String studentId, String fullName,

String email, float avgGrade) {

this.studentId = studentId;

this.fullName = fullName;

this.email = email;

this.avgGrade = avgGrade;

}

public String getStudentId() {

return studentId;

}

public void setStudentId(String studentId) {

this.studentId = studentId;

}

public String getFullName() {

return fullName;

}

public void setFullName(String fullName) {

this.fullName = fullName;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public float getAvgGrade() {

return avgGrade;

}

public void setAvgGrade(float avgGrade) {

this.avgGrade = avgGrade;

}

@Override

public String toString() {

return "Student{" +

"studentId='" + studentId + '\'' +

", fullName='" + fullName + '\'' +

", email='" + email + '\'' +

", avgGrade=" + avgGrade +

'}';

}

}

==========================================

Nhung

Linh

Khánh

Loan

==========================================

1.25

3.12456

6.1454

99.899

==========================================

Student{studentId='B25DCCN100', fullName='Ngô Thùy Linh', email='null', avgGrade=0.0}

Student{studentId='B25DCCN101', fullName='Nguyễn Phương Hoa', email='null', avgGrade=0.0}

Student{studentId='B25DCCN102', fullName='Trần Xuân Nhật', email='null', avgGrade=0.0}

Student{studentId='B25DCCN103', fullName='Lê Hoài Nam', email='null', avgGrade=0.0}

* Ví dụ 2: Tự viết lớp generic ghép cặp sinh viên và lớp học. Yêu cầu tạo danh sách lớp học sao cho mỗi sinh viên với một lớp học chỉ xuất hiện 1 lần trong danh sách.

Lớp Student, Subject: [**Click vào đây để xem code.**](https://github.com/thantrieu/java_course1_exercises/tree/master/Example%20lesson64)

package lesson64;

import java.util.ArrayList;

import java.util.Objects;

public class Example2 {

public static void main(String[] args) {

ArrayList<Pair<Student, Subject>> pairs = new ArrayList<>();

Pair<Student, Subject>[] pairArr = new Pair[6];

pairArr[0] = new Pair(new Student("ST1"), new Subject("SJ1"));

pairArr[1] = new Pair(new Student("ST1"), new Subject("SJ2"));

pairArr[2] = new Pair(new Student("ST2"), new Subject("SJ1"));

pairArr[3] = new Pair(new Student("ST2"), new Subject("SJ1"));

pairArr[4] = new Pair(new Student("ST2"), new Subject("SJ2"));

pairArr[5] = new Pair(new Student("ST1"), new Subject("SJ2"));

// thêm vào danh sách các cặp sv-môn học chưa tồn tại

addToList(pairs, pairArr);

// hiển thị kết quả

showPairs(pairs);

}

private static void addToList(ArrayList<Pair<Student, Subject>> pairs,

Pair<Student, Subject>[] pairArr) {

for (int i = 0; i < pairArr.length; i++) {

if (!pairs.contains(pairArr[i])) {

pairs.add(pairArr[i]);

}

}

}

private static void showPairs(ArrayList<Pair<Student, Subject>> pairs) {

for (var e : pairs) {

System.out.println("Mã SV: " + e.getStudent().getStudentId()

+ " - Mã môn: " + e.getSubject().getId());

}

}

}

class Pair<Student, Subject> {

private Student student;

private Subject subject;

public Pair() {

}

public Pair(Student student, Subject subject) {

this.student = student;

this.subject = subject;

}

public Student getStudent() {

return student;

}

public void setStudent(Student student) {

this.student = student;

}

public Subject getSubject() {

return subject;

}

public void setSubject(Subject subject) {

this.subject = subject;

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

Pair<?, ?> pair = (Pair<?, ?>) o;

return Objects.equals(student, pair.student) &&

Objects.equals(subject, pair.subject);

}

@Override

public int hashCode() {

return Objects.hash(student, subject);

}

}

Mã SV: ST1 - Mã môn: SJ1

Mã SV: ST1 - Mã môn: SJ2

Mã SV: ST2 - Mã môn: SJ1

Mã SV: ST2 - Mã môn: SJ2

**Bài 11.3. Giới thiệu Collections Framework**

**Nội dung bài học**

1. Mục đích sử dụng và đặc điểm
2. Các interface trong framework
3. Interface Collection
4. Interface Iterator
5. Một số lớp thực thi chi tiết collections framework

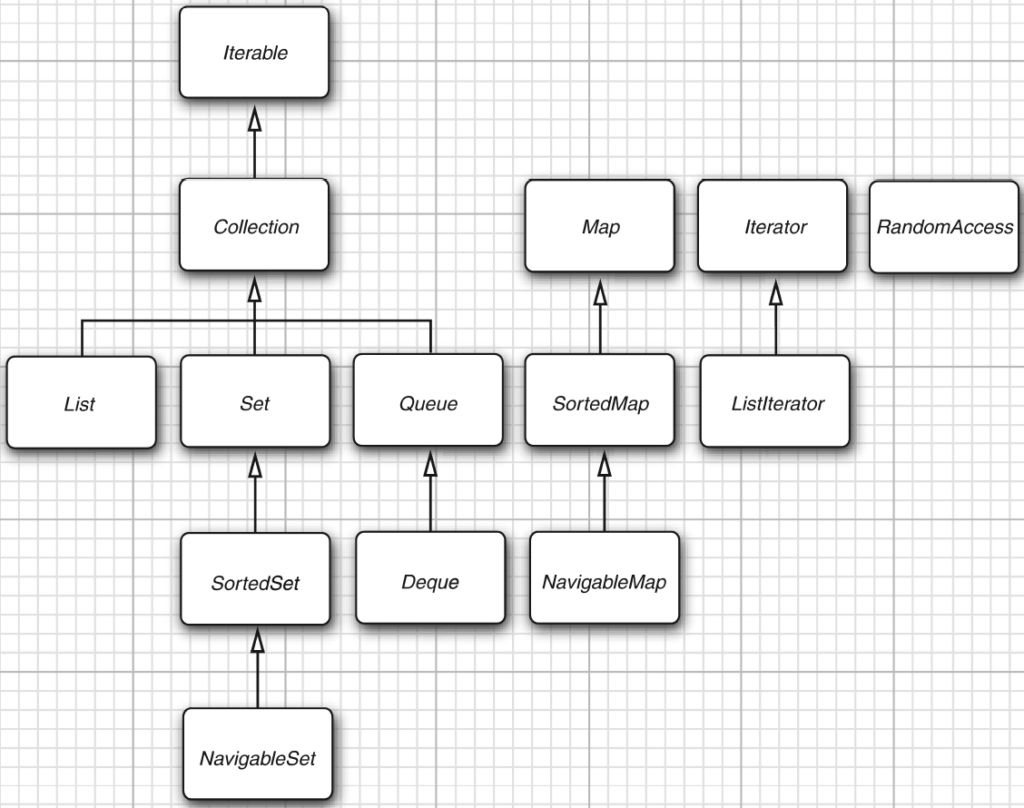
**Nội dung này chỉ mang tính chất giới thiệu các collection và mục đích sử dụng. Để hiểu sâu và chuyên nghiệp bạn vui lòng dành thời gian học môn “Cấu trúc dữ liệu và giải thuật” nhé.**

**Mục đích sử dụng và đặc điểm**

* Ngôn ngữ lập trình Java cung cấp collections framework. Một collection(tập hợp) là một đối tượng đại diện cho một nhóm đối tượng.
* Các mục đích sử dụng chính của collections framework:
  + Giảm gánh nặng về yêu cầu kĩ năng lập trình bằng cách cung cấp các cấu trúc dữ liệu và giải thuật để lập trình viên không phải tự viết chúng.
  + Tăng hiệu năng bằng cách cung cấp các thực thi các cấu trúc dữ liệu và thuật toán hiệu năng cao. Vì các triển khai của interface có thể hoán đổi cho nhau nên các chương trình có thể chuyển đổi qua lại.
  + Cung cấp khả năng tương tác giữa các API không liên quan bằng cách thiết lập một ngôn ngữ chung để chuyển đổi các collection qua lại lẫn nhau.
  + Giảm nỗ lực cần thiết để thiết kế và triển khai API bằng cách không yêu cầu bạn tạo các collection API chuyên biệt.
  + Khuyến khích tái sử dụng phần mềm bằng cách cung cấp các interface chuẩn cho các tập hợp và các thuật toán để thao tác với chúng.
* Hướng đến mục tiêu này, toàn bộ framework được thiết kế xoay quanh một tập hợp các interface chuẩn. Một số lớp đã thực thi sẵn cho bạn sử dụng như LinkedList, HashSet, TreeSet. Tất nhiên bạn cũng có thể tự triển khai tập hợp của riêng mình nếu muốn.
* Một collections framework là một kiến trúc thống nhất để đại diện và thao tác với các tập hợp. Framework này gồm:
  + Các interface: đây là các kiểu dữ liệu trừu tượng đại diện cho các tập hợp. Các interface cho phép các tập hợp được thao tác độc lập với các biểu diễn chi tiết của chúng. Trong hướng đối tượng, các interface tạo nên một hệ thống phân cấp.
  + Các lớp thực thi interface: đây là các lớp triển khai cụ thể của các interface trong tập hợp. Về bản chất chúng là các cấu trúc dữ liệu có thể tái sử dụng.
  + Các thuật toán: đây là các phương thức thực hiện các phép tính toán hữu ích, ví dụ tìm kiếm, sắp xếp các đối tượng. Các thuật toán được coi là đa hình, tức là cùng một phương pháp có thể sử dụng nhiều cách triển khai khác nhau dựa trên việc thực thi các interface phù hợp.

**Các interface trong framework**

Sau đây là hình minh họa các interface trong collections framework và mối quan hệ của chúng:

[](https://braniumacademy.net/wp-content/uploads/2020/10/interface-in-collection-framework.png)

**Ý nghĩa sử dụng:**

* **Collection**: cung cấp các cách thức cho phép bạn làm việc với một nhóm các đối tượng. Là loại top level interface của chuỗi phân cấp kế thừa.
* **List**: kế thừa Collection và đối tượng của List dùng để lưu trữ danh sách các phần tử có thứ tự.
* **Set**: kế thừa Collection, dùng để chứa tập hợp các phần tử không trùng nhau.
* **SortedSet**: kế thừa Set, cung cấp một tập hợp các phần tử đơn nhất và được sắp xếp theo thứ tự nhất định.
* **NavigableSet**: kế thừa SortedSet, bổ sung các phương thức điều hướng để cung cấp các kết quả tìm kiếm phù hợp nhất cho các mục tiêu tìm kiếm nhất định.
* **Queue**: một tập hợp hàng đợi dùng để lưu trữ các phần tử trước khi xử lý theo thứ tự vào trước ra trước.
* **DeQueue**: kế thừa Queue, hàng đợi này cho phép thực hiện các thao tác thêm mới và loại bỏ phần tử từ cả hai chiều(đầu và cuối).
* **Map**: biểu thị dữ liệu dạng cặp key-value.
* **SortedMap**: kế thừa từ Map, cung cấp một map với các key được sắp xếp theo thứ tự tăng dần.
* **Iterator**: cung cấp khả năng duyệt, loại bỏ các phần tử trong tập hợp.
* **ListIterator**: kế thừa Iterator, cho phép duyệt và thực hiện các thao tác sửa đổi tập hợp theo cả hai hướng: đầu-cuối hoặc ngược lại.
* **RandomAccess**: là một interface đánh dấu nhằm chỉ ra rằng một tập hợp thực thi nó có hỗ trợ truy cập ngẫu nhiên(truy cập bằng chỉ số phần tử).

**Interface Collection**

* Là interface gốc trong phân cấp của tập hợp. Một tập hợp đại diện cho một nhóm các đối tượng được gọi là các phần tử của nó. Một số tập hợp cho phép các phần tử được trùng lặp, một số khác thì không. Một số có thứ tự và số khác lại không có thứ tự.
* Một số phương thức và mô tả:

|  |  |
| --- | --- |
| **Phương thức** | **Mô tả** |
| boolean add(E e) | Thêm một phần tử vào tập hợp hiện thời. Nếu thêm thành công return true và ngược lại return false |
| boolean addAll(Collection<? extends E> c) | Thêm tất cả các phần tử trong một tập hợp cho trước vào tập hợp hiện thời |
| void clear() | Xóa bỏ toàn bộ các phần tử trong tập hợp hiện thời |
| boolean contains(Object o) | Kiểm tra xem tập hợp hiện thời chứa đối tượng cho trong tham số hay không |
| boolean containsAll(Collection<?> c) | Trả về true nếu tập hợp hiện thời chứa tất cả các phần tử trong  tập hợp được chỉ ra trong tham số |
| boolean equals(Object o) | So sánh hai đối tượng về tính tương đương |
| int hasCode() | Trả về mã băm của đối tượng tập hợp hiện thời |
| boolean isEmpty() | Trả về true nếu tập hợp hiện thời không có phần tử nào cả |
| Iterator<E> iterator() | Trả về một iterator của các phần tử trong tập hợp hiện thời |
| boolean remove(Object o) | Loại bỏ một phần tử được chỉ định nếu nó tồn tại |
| boolean removeAll(Collection<?>c) | Xóa toàn bộ các phần tử chứa trong tham số nếu nó xuất hiện trong tập hợp hiện thời |
| int size() | Trả về số phần tử hiện có của tập hợp |
| Object[] toArray() | Trả về một mảng chứa tất cả các phần tử trong tập hợp hiện thời |

**Interface Iterator**

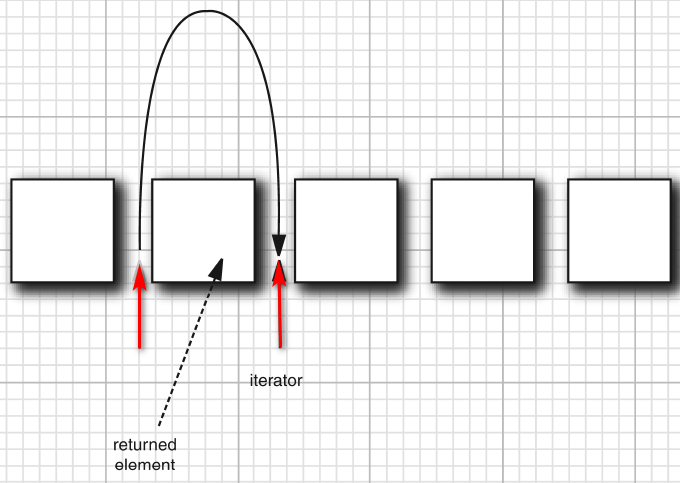
* Là một trình lặp trên một tập hợp, tức có thể sử dụng đối tượng của Iterator để thực hiện thao tác lặp.
* Các phương thức và mô tả:

|  |  |
| --- | --- |
| **Phương thức** | **Mô tả** |
| default void forEachRemaining(Consumer<? super E> action) | Thực hiện hành động cho trước với từng phần tử còn lại của tập hợp đến khi tất cả các phần tử đã được xét duyệt hoặc hành động văng một ngoại lệ |
| boolean hasNext() | Trả về true nếu có phần tử kế tiếp để duyệt |
| E next() | Trả về phần tử tiếp theo |
| default void remove() | Xóa bỏ phần tử trả về bởi iterator hiện tại |

Interface Iterable chứa Iterator nên ta có thể áp dụng vòng lặp for each cho mọi tập hợp là con, implements của Collection.

Lưu ý rằng Iterator trong Java thực hiện bước nhảy giữa các phần tử chứ không phải nhảy từ phần tử này qua phần tử kia. Khi gọi next(), nó nhảy qua phần tử kế tiếp và trả về tham chiếu đến phần tử vừa nhảy qua.

Minh họa:

[](https://braniumacademy.net/wp-content/uploads/2020/10/iterator.png)

**Một số lớp thực thi chi tiết collections framework**

Sau đây là một số lớp thực thi của collections framework:

[Ảnh có chứa biểu đồ, Kế hoạch, Hình chữ nhật, Bản vẽ kỹ thuật

Mô tả được tạo tự động](https://braniumacademy.net/wp-content/uploads/2020/10/class-in-collection-framework.png)

* Một số lớp và mục đích sử dụng:
  + **ArrayList**: tập hợp các phần tử dạng mảng có thể tự thay đổi kích thước để phù hợp với số phần tử hiện tại đang chứa. Thường truy cập bằng chỉ số phần tử.
  + **LinkedList**: một tập hợp có thứ tự được sử dụng hiệu quả trong trường hợp chèn, xóa phần tử tại bất kì vị trí nào.
  + **ArrayDeque**: một hàng đợi hai chiều cho phép thực hiện các hành động thêm, xóa phần tử ở hai đầu của nó.
  + **HashSet**: tập hợp không có thứ tự của các phần tử và không cho phép trùng lặp phần tử.
  + **TreeSet**: một tập hợp có thứ tự của các phần tử.
  + **EnumSet**: một tập hợp của các giá trị kiểu liệt kê được.
  + **LinkedHashSet**: một tập hợp ghi nhớ thứ tự mà các phần tử đã được thêm vào.
  + **PriorityQueue**: hàng đợi ưu tiên cho phép loại bỏ các phần tử theo mức độ ưu tiên.
  + **HashMap**: một cấu trúc dữ liệu lưu trữ dữ liệu dạng key-value.
  + **TreeMap**: là một map trong đó các key được sắp xếp theo thứ tự.
  + **EnumMap**: là một map trong đó các key thuộc về một kiểu liệt kê.
  + **LinkedHashMap**: một map ghi nhớ thứ tự các phần tử đã được thêm vào.
  + **WeakHashMap**: một map với các giá trị có thể bị thu hồi bởi trình dọn dẹp nếu nó không được sử dụng ở đâu cả.
  + **IdentityHashMap**: một map với các key được so sánh bởi == không dùng equals().

**Bài 11.4. Sử dụng Stack, Queue**

**Nội dung bài học**

1. Sử dụng stack
2. Sử dụng queue
3. Ví dụ minh họa

**Sử dụng stack**

* Khái niệm: stack hay còn gọi ngăn xếp, là một cấu trúc dữ liệu theo tiêu chí FILO, tức vào sau ra trước
* Có thể hình dung một stack như cái thùng có 1 đáy. Phần tử vào đầu tiên ở dưới cùng, phần tử vào sau ở trên phần tử vào trước
* Các hành động chính của stack gồm:
  + push(): thêm phần tử vào đầu stack
  + pop(): lấy và xóa phần tử ở đầu stack
  + peek(): lấy phần tử ở đầu và không xóa
  + empty(): kiểm tra stack có rỗng không
  + search(): tìm kiếm phần tử trong stack
* Lưu các lời gọi của chương trình, thực hiện việc tính toán các biểu thức…
* Hình sau minh họa thứ tự lời gọi được lưu trong stack.[Ảnh có chứa văn bản, ảnh chụp màn hình, hàng, Phông chữ

  Mô tả được tạo tự động](https://braniumacademy.net/wp-content/uploads/2020/10/stack-application.png)

**Sử dụng queue**

* Queue là cấu trúc dữ liệu hàng đợi theo tiêu chí vào trước ra trước(FIFO)
* Có thể hình dung queue như một cái ống rỗng hai đầu, ta thả một hòn đá từ đầu này thì nó sẽ ra ở đầu bên kia vậy
* Ứng dụng của queue:
  + Phục vụ yêu cầu trong hệ thống dùng chung tài nguyên, ví dụ máy in, lập lịch CPU
  + Trong thực tế, tổng đài sử dụng queue để xếp thứ tự cuộc gọi đến và phục vụ lần lượt
  + Xử lý lỗi trong hệ thống, lỗi nào xảy ra trước xử lý trước
* Một số phương thức chính của Queue:
  + add(): Thêm phần tử vào cuối queue
  + remove(): xóa bỏ phần tử ở đầu queue
  + peek(): lấy phần tử đầu queue nhưng không xóa
* Một số phương thức của ArrayDequeue:
  + addFirst(): thêm vào đầu
  + addLast(): thêm vào cuối
  + peekFirst(): lấy ở đầu
  + peekLast(): lấy ở cuối
  + removeFirst(): lấy và xóa phần tử ở đầu
  + removeLast(): lấy và xóa phần tử ở cuối

**Ví dụ minh họa**

* Ví dụ minh họa stack:

package lesson66;

import java.util.Stack;

public class Lesson66 {

public static void main(String[] args) {

Stack<String> friends = new Stack<>();

friends.push("Trung");

friends.push("Nam");

friends.push("Hoa");

friends.push("Nhung");

System.out.println("Top: " + friends.peek());

friends.pop();

System.out.println("Top: " + friends.peek());

friends.pop();

friends.pop();

friends.pop();

if(!friends.empty()) {

friends.pop();

System.out.println("Top: " + friends.peek());

}

}

}

Top: Nhung

Top: Hoa

* Ví dụ minh họa queue:

package lesson66;

import java.util.ArrayDeque;

import java.util.Queue;

public class QueueEx {

public static void main(String[] args) {

Queue<String> queue = new ArrayDeque<>();

queue.add("Nam");

queue.add("Trung");

queue.add("Nhung");

queue.add("Hoa");

System.out.println(queue.remove());

System.out.println(queue.remove());

System.out.println(queue.remove());

System.out.println(queue.remove());

if(!queue.isEmpty()) {

System.out.println(queue.remove());

}

}

}

Nam

Trung

Nhung

Hoa

* Ví dụ minh họa ArrayDequeue:

package lesson66;

import java.util.ArrayDeque;

public class QueueEx2 {

public static void main(String[] args) {

ArrayDeque<String> deque = new ArrayDeque<>();

deque.addFirst("Nhung");

deque.addLast("Hoa");

deque.add("Nam");

deque.add("Long");

deque.addFirst("Hiên");

System.out.println("cuối: " + deque.removeLast());

System.out.println("?: " + deque.remove());

System.out.println("Số phần tử: " + deque.size());

}

}

cuối: Long

?: Hiên

Số phần tử: 3

**Bài 11.5. Sử dụng LinkedList, HashSet, TreeSet**

**Nội dung bài học**

1. Sử dụng LinkedList
2. Sử dụng HashSet
3. Sử dụng TreeSet

**Sử dụng LinkedList**

* Ví dụ trong video:

Lớp Student:

package lesson66b;

public class Student implements Comparable<Student> {

private String id;

private String fullName;

public Student() {

}

public Student(String id, String fullName) {

this.id = id;

this.fullName = fullName;

}

public Student(String id) {

this.id = id;

}

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public String getFullName() {

return fullName;

}

public void setFullName(String fullName) {

this.fullName = fullName;

}

@Override

public String toString() {

return "Student{" +

"id='" + id + '\'' +

", fullName='" + fullName + '\'' +

'}';

}

@Override

public int compareTo(Student other) {

return other.id.compareTo(this.id);

}

}

package lesson66b;

import java.util.LinkedList;

/\*\*

\* Minh họa sử dụng danh sách liên kết đôi

\*/

public class LinkedListEx {

public static void main(String[] args) {

Student s1 = new Student("S1", "Nam");

Student s2 = new Student("S2", "Nhung");

Student s3 = new Student("S3", "Hoa");

Student s4 = new Student("S4", "Linh");

LinkedList<Student> studentLinkedList = new LinkedList<>();

int index = 2;

studentLinkedList.add(s1);

studentLinkedList.addLast(s1);

studentLinkedList.add(2, s1);

studentLinkedList.addLast(s2);

studentLinkedList.addFirst(s3);

if (index < studentLinkedList.size()) {

studentLinkedList.add(index, s4);

}

// sử dụng iterator

var iterator = studentLinkedList.iterator();

var listIterator

= studentLinkedList.listIterator(studentLinkedList.size());

System.out.println("====================");

while (iterator.hasNext()) {

System.out.println(iterator.next() + " ==>");

}

System.out.println("====================");

// duyệt từ cuối lên đầu

while (listIterator.hasPrevious()) {

System.out.println(listIterator.previous() + " <==");

}

// System.out.println("====================");

// System.out.println(studentLinkedList.getFirst());

// System.out.println("====================");

// System.out.println(studentLinkedList.getLast());

}

}

====================

Student{id='S3', fullName='Hoa'} ==>

Student{id='S1', fullName='Nam'} ==>

Student{id='S4', fullName='Linh'} ==>

Student{id='S1', fullName='Nam'} ==>

Student{id='S1', fullName='Nam'} ==>

Student{id='S2', fullName='Nhung'} ==>

====================

Student{id='S2', fullName='Nhung'} <==

Student{id='S1', fullName='Nam'} <==

Student{id='S1', fullName='Nam'} <==

Student{id='S4', fullName='Linh'} <==

Student{id='S1', fullName='Nam'} <==

Student{id='S3', fullName='Hoa'} <==

**Sử dụng HashSet**

* Code mẫu:

package lesson66b;

import java.util.HashSet;

import java.util.TreeSet;

public class SetEx {

public static void main(String[] args) {

Student s1 = new Student("S1", "Nam");

Student s2 = new Student("S2", "Nhung");

Student s3 = new Student("S3", "Hoa");

Student s4 = new Student("S4", "Linh");

HashSet<Student> hashSet = new HashSet<>();

hashSet.add(s1);

hashSet.add(s2);

hashSet.add(s3);

hashSet.add(s4);

hashSet.add(s1);

hashSet.add(s2);

hashSet.add(s3);

for (var e : hashSet) {

System.out.println(e);

}

}

}

Student{id='S3', fullName='Hoa'}

Student{id='S4', fullName='Linh'}

Student{id='S2', fullName='Nhung'}

Student{id='S1', fullName='Nam'}

**Sử dụng TreeSet**

* Code mẫu:

package lesson66b;

import java.util.HashSet;

import java.util.TreeSet;

public class SetEx {

public static void main(String[] args) {

Student s1 = new Student("S1", "Nam");

Student s2 = new Student("S2", "Nhung");

Student s3 = new Student("S3", "Hoa");

Student s4 = new Student("S4", "Linh");

System.out.println("=========================");

TreeSet<Student> treeSet = new TreeSet<>();

treeSet.add(s1);

treeSet.add(s2);

treeSet.add(s3);

treeSet.add(s4);

treeSet.add(s1);

treeSet.add(s2);

treeSet.add(s3);

treeSet.add(s4);

for (var e : treeSet) {

System.out.println(e);

}

}

}

=========================

Student{id='S4', fullName='Linh'}

Student{id='S3', fullName='Hoa'}

Student{id='S2', fullName='Nhung'}

Student{id='S1', fullName='Nam'}

**Bài 11.6. Sử dụng HashMap, TreeMap**

**Nội dung bài học**

1. Sử dụng HashMap
2. Sử dụng TreeMap

**Sử dụng HashMap**

* Code mẫu:

Lớp Student:

package lesson66c;

public class Student implements Comparable<Student> {

private String id;

private String fullName;

public Student() {

}

public Student(String id, String fullName) {

this.id = id;

this.fullName = fullName;

}

public Student(String id) {

this.id = id;

}

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public String getFullName() {

return fullName;

}

public void setFullName(String fullName) {

this.fullName = fullName;

}

@Override

public String toString() {

return "Student{" +

"id='" + id + '\'' +

", fullName='" + fullName + '\'' +

'}';

}

@Override

public int compareTo(Student other) {

return other.id.compareTo(this.id);

}

}

package lesson66c;

import java.util.HashMap;

/\*\*

\* Sử dụng Map

\*/

public class MapExample {

public static void main(String[] args) {

HashMap<String, Student> hashMapStudents = new HashMap<>();

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

Student s1 = new Student("S1", "Nam");

Student s2 = new Student("S2", "Nhung");

Student s3 = new Student("S3", "Hoa");

Student s4 = new Student("S4", "Linh");

Student s5 = new Student("S5", "Trung");

hashMapStudents.put(s5.getId(), s5);

hashMapStudents.put(s2.getId(), s2);

hashMapStudents.put(s1.getId(), s1);

hashMapStudents.put(s4.getId(), s4);

hashMapStudents.put("S3", s3);

System.out.println(hashMapStudents.keySet());

System.out.println(hashMapStudents.values());

// System.out.println("Đối tượng ứng với key S1: " + hashMapStudents.get("S1"));

// System.out.println("Đối tượng ứng với key S10: " + hashMapStudents.getOrDefault("S10", s5));

// System.out.println("Số lượng phần tử của map: " + hashMapStudents.size());

}

}

[S3, S4, S5, S1, S2]

[Student{id='S3', fullName='Hoa'}, Student{id='S4', fullName='Linh'},

Student{id='S5', fullName='Trung'}, Student{id='S1', fullName='Nam'},

Student{id='S2', fullName='Nhung'}]

**Sử dụng TreeMap**

* Code mẫu:

package lesson66c;

import java.util.TreeMap;

public class TreeMapEx {

public static void main(String[] args) {

Student s1 = new Student("S1", "Nam");

Student s2 = new Student("S2", "Nhung");

Student s3 = new Student("S3", "Hoa");

Student s4 = new Student("S4", "Linh");

Student s5 = new Student("S5", "Trung");

TreeMap<String, Student> studentTreeMap = new TreeMap<>();

studentTreeMap.put(s5.getId(), s5);

studentTreeMap.put(s2.getId(), s2);

studentTreeMap.put(s1.getId(), s1);

studentTreeMap.put(s4.getId(), s4);

studentTreeMap.put(s3.getId(), s3);

System.out.println(studentTreeMap.get("S3"));

// System.out.println(studentTreeMap.keySet());

// System.out.println(studentTreeMap.values());

}

}

Student{id='S3', fullName='Hoa'}

**Bài 12.1. Tổng quan về regular expression**

**Nội dung bài học**

1. Giới thiệu về regular experession
2. Các siêu kí tự dùng trong regular expression
3. Một số phương thức thường dùng của lớp Pattern
4. Một số phương thức thường dùng của lớp Matcher
5. Ví dụ minh họa
6. Bài tập thực hành

Video lý thuyết

Video minh họa

**Giới thiệu về regular experession**

* Regular expression dịch ra là biểu thức chính quy. Là một chuỗi kí tự được sắp xếp theo thứ tự đặc biệt dùng để so khớp, tìm kiếm chuỗi kí tự sử dụng cú pháp đặc trưng được thiết lập trong các pattern(mẫu so khớp).
* Chúng chủ yếu dùng để tìm kiếm, chỉnh sửa hoặc thao tác với dữ liệu dạng văn bản.
* Các lớp chức năng và mô tả:
  + Lớp Pattern: đối tượng của lớp này là một đại diện đã biên dịch của một biểu thức chính quy. Lớp pattern không cung cấp constructor công khai mà muốn tạo đối tượng của lớp này ta phải dùng phương thức compile và truyền cho nó một mẫu so khớp.
  + Lớp Matcher: đối tượng của lớp này thực hiện việc so khớp chuỗi đầu vào với mẫu so khớp(pattern). Lớp Matcher không có public constructor. Ta thường tạo đối tượng của nó thông qua lời gọi tới phương thức matcher() từ đối tượng của lớp Pattern.
  + Lớp PatternSyntaxException: lớp ngoại lệ của biểu thức chính quy dùng để chỉ ra ngoại lệ đã gặp phải. Đây là một runtime exception.

**Các siêu kí tự dùng trong regular expression**

Sau đây là một số các siêu kí tự dùng trong biểu thức của regular expression:

|  |  |
| --- | --- |
| **Kí tự** | **Ý nghĩa** |
| ^ | So khớp đầu dòng |
| $ | So khớp cuối dòng |
| . | Khớp với bất kì kí tự nào |
| [abc] | Khớp với bất kì kí tự nào trong ngoặc vuông |
| [^abc] | Khớp với bất kì kí tự nào không nằm trong ngoặc vuông sau dấu ^ |
| abc\* | Khớp với 0 hoặc nhiều biểu thức abc nào đó |
| abc+ | Khớp với 1 hoặc nhiều biểu thức abc nào đó |
| abc? | Khớp xem biểu thức abc có xuất hiện hay không |
| abc{ n } | Khớp với chính xác n lần xuất hiện của giá trị abc |
| abc{ n, } | Khớp với ít nhất n lần xuất hiện của biểu thức abc |
| abc{ m, n} | Khớp với ít nhất m, nhiều nhất n lần xuất hiện của biểu thức abc |
| a | b | Khớp với a hoặc b |
| (abc) | Gom nhóm biểu thức abc lại |
| \w | Khớp với các kí tự chữ cái hoặc chữ số |
| \W | Khớp với bất kì kí tự nào không phải chữ cái hoặc chữ số |
| \s | Khớp với khoảng trắng(\t, \n, \r, \f, dấu cách) |
| \S | Khớp với các kí tự không phải khoảng trắng |
| \d | Khớp với kí tự số (0-9) |
| \D | Khớp với kí tự không phải số |
| \A | Khớp với vị trí bắt đầu chuỗi kí tự |
| \Z | Khớp với vị trí cuối chuỗi kí tự, nếu cuối chuỗi là kí tự \n thì nó khớp với vị trí trước \n |
| \z | Khớp với vị trí cuối chuỗi kí tự |
| \G | Khớp với điểm mà lần so khớp trước đó kết thúc |

**Một số phương thức thường dùng của lớp Pattern**

Sau đây là một số phương thức và mô tả:

|  |  |
| --- | --- |
| **Phương thức** | **Mô tả** |
| static Pattern compile(String regex) | Sử dụng để compile một biểu thức chính quy thành một mẫu so khớp thường dùng |
| static Pattern compile(String regex, int flags) | Sử dụng để compile một biểu thức chính quy thành một mẫu so khớp thường dùng với một cờ cho trước |
| int flags() | Lấy flag đang được sử dụng trong pattern hiện thời |
| Matcher matcher(CharSequence input) | Được sử dụng để tạo ra đối tượng matcher đem so khớp mẫu pattern đã có với chuỗi input vừa nhận được |
| Static boolean matches(String regex, CharSequence input) | Sử dụng để biên dịch biểu thức chính quy đã cho và thực hiện so khớp với input nhận được |
| String[] split(CharSequence input) | Sử dụng để tách chuỗi đầu vào tại các vị trí có mẫu so khớp. Trả về mảng các chuỗi con sau khi tách. |

**Một số phương thức thường dùng của lớp Matcher**

Sau đây là một số phương thức và mô tả:

|  |  |
| --- | --- |
| **Phương thức** | **Mô tả** |
| int start() | Trả về vị trí bắt đầu khớp với mẫu pattern trong chuỗi input của lần so khớp đang xét |
| int end() | Trả về vị trí kế tiếp sau vị trí khớp với mẫu so khớp trong chuỗi đầu vào đang xét |
| boolean find() | Chủ yếu dùng để tìm sự xuất hiện nhiều lần của pattern trong chuỗi gốc |
| boolean find(int start) | Dùng để tìm sự xuất hiện của pattern trong chuỗi gốc từ vị trí start cho trước |
| int groupCount() | Sử dụng để trả về số nhóm trong pattern |
| boolean matches() | Dùng để test xem liệu pattern có khớp với input không |
| String replaceAll(String replacement) | dùng để thay thế toàn bộ chuỗi con khớp với pattern trong input bởi tham số được cung cấp. Trả về String kết quả |
| String replaceFirst(String replacement) | Thay thế chuỗi con đầu tiên trong chuỗi input khớp với pattern bởi tham số cho bởi phương thức. Trả về String kết quả |

**Ví dụ minh họa**

* Ví dụ sau minh họa sử dụng mẫu pattern đơn giản:

package lesson67;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class Test {

public static void main(String[] args) {

String regex = "\\s+"; // vị trí có 1 hoặc nhiều dấu cách

String str = "Hi there! I love learning Java programming " +

"language. I hope you enjoy it too.";

Pattern pattern = Pattern.compile(regex);

Matcher matcher = pattern.matcher(str);

// kiểm tra xem mẫu pattern có khớp với chuỗi con nào trong

// str hay không

System.out.println("Chuỗi str chỉ chứa dấu cách? "

+ matcher.matches());

// tìm vị trí bắt đầu, kết thúc của regex trong str

while (matcher.find()) {

System.out.println("Bắt đầu: " + matcher.start()

+ " - kết thúc: " + (matcher.end()));

}

// tách str tại vị trí có regex

var result = pattern.split(str);

showWords(result);

// thay thế regex bởi một substring nào đó

var newStr = matcher.replaceAll("\_");

System.out.println("Chuỗi str sau khi thay thế: \n" + newStr);

// thay thế vị trí đầu tiên mà regex xuất hiện trong str

var newStr2 = matcher.replaceFirst("->");

System.out.println("Chuỗi str sau khi thay thế: \n" + newStr2);

}

// phương thức dùng để hiển thị mảng kết quả

private static void showWords(String[] result) {

for (var word : result) {

System.out.println(word);

}

}

}

* Kết quả:

Chuỗi str chỉ chứa dấu cách? false

Bắt đầu: 2 - kết thúc: 3

Bắt đầu: 9 - kết thúc: 10

Bắt đầu: 11 - kết thúc: 12

Bắt đầu: 16 - kết thúc: 17

Bắt đầu: 25 - kết thúc: 26

Bắt đầu: 30 - kết thúc: 31

Bắt đầu: 42 - kết thúc: 43

Bắt đầu: 52 - kết thúc: 53

Bắt đầu: 54 - kết thúc: 55

Bắt đầu: 59 - kết thúc: 60

Bắt đầu: 63 - kết thúc: 64

Bắt đầu: 69 - kết thúc: 70

Bắt đầu: 72 - kết thúc: 73

Hi

there!

I

love

learning

Java

programming

language.

I

hope

you

enjoy

it

too.

Chuỗi str sau khi thay thế:

Hi\_there!\_I\_love\_learning\_Java\_programming\_language.\_I\_hope\_you\_enjoy\_it\_too.

Chuỗi str sau khi thay thế:

Hi->there! I love learning Java programming language. I hope you enjoy it too.

**Bài 12.2. Kiểm tra định dạng mã sinh viên**

**Thưc hành áp dụng regular expresssion**

Yêu cầu:

**Ví dụ 1: Kiểm tra định dạng mã sinh viên.**

* Mã sinh viên là một String gồm cả chữ cái và số không phân biệt chữ hoa chữ thường. Định dạng chuẩn của mã sinh viên như sau:
  + Bắt đầu bởi các chữ cái a, b, hoặc c
  + Sau đó là 2 số
  + Tiếp đó là 4 chữ cái
  + Kết thúc bởi 3 chữ số
* Ví dụ mã sinh viên hợp lệ:
  + B25DCCN123
  + B20DCVT111
  + A21DCKD515
* Nếu mã sinh viên nhập vào là chuẩn thì ta viết hoa tất cả các chữ cái của mã này rồi lưu vào đối tượng sinh viên. Nếu mã sinh viên không đúng thì văng ngoại lệ compile time exception: InvalidStudentIdException thông báo mã sinh viên không hợp lệ.
* Code mẫu:
* Lớp Student.

package lesson68;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class Student {

private String id;

private String fullName;

private String major;

public Student() {

}

public Student(String id) throws InvalidStudentIdException {

setId(id);

}

public Student(String id, String fullName, String major)

throws InvalidStudentIdException {

this(id);

this.fullName = fullName;

this.major = major;

}

public String getId() {

return id;

}

public void setId(String id) throws InvalidStudentIdException {

var regex = "^(a|b|c){1}\\d{2}[a-z]{4}\\d{3}$"; // regular expression

Pattern pattern = Pattern.compile(regex, Pattern.CASE\_INSENSITIVE);

Matcher matcher = pattern.matcher(id);

if (matcher.matches()) {

this.id = id.toUpperCase();

} else {

this.id = null;

var msg = "Mã sinh viên không hợp lệ, vui lòng kiểm tra lại";

throw new InvalidStudentIdException(msg, id);

}

}

public String getFullName() { return fullName; }

public void setFullName(String fullName) { this.fullName = fullName; }

public String getMajor() { return major; }

public void setMajor(String major) { this.major = major; }

}

* Lớp InvalidStudentIdException.

package lesson68;

public class InvalidStudentIdException extends Exception {

private String invalidId;

public InvalidStudentIdException() {

}

public InvalidStudentIdException(String invalidId) {

this.invalidId = invalidId;

}

public InvalidStudentIdException(String message, String invalidId) {

super(message);

this.invalidId = invalidId;

}

public String getInvalidId() {

return invalidId;

}

}

* Lớp Test.

package lesson68;

import java.util.Scanner;

public class Test {

public static void main(String[] args) {

Student student = new Student();

System.out.println("Nhập mã sinh viên theo định dạng đã cho: ");

var input = new Scanner(System.in);

var id = input.nextLine();

try {

student.setId(id);

} catch (InvalidStudentIdException e) {

System.out.println(e.getMessage());

System.out.println("Ví dụ mã hợp lệ: B25DCCN123");

}

System.out.println("Mã sinh viên là: " + student.getId());

}

}

* Kết quả test:

Nhập mã sinh viên theo định dạng đã cho:

B25DCCN278

Mã sinh viên là: B25DCCN278

Nhập mã sinh viên theo định dạng đã cho:

B25DCCN369

Mã sinh viên không hợp lệ, vui lòng kiểm tra lại

Ví dụ mã hợp lệ: B25DCCN123

Mã sinh viên là: null

**Bài 12.3.Kiểm tra định dạng mã nhân viên**

**Thực hành kiểm tra định dạng mã nhân viên**

* Yêu cầu:
* Định dạng mã nhân viên của một công ty không phân biệt chữ hoa chữ thường và được cho như sau:
  + Mã nhân viên bắt đầu với NV hoặc EMP
  + Tiếp đó là 6 số
  + Kết thúc bởi mã phòng ban có 2 đến 5 chữ cái
* Ví dụ mã nhân viên hợp lệ:
  + NV100000IT
  + NV100001GV
  + NV100002GDNS
  + NV100003CVKD
* Nếu mã nhân viên không hợp lệ thì văng ngoại lệ InvalidEmpIdException thông báo mã nhân viên không hợp lệ. Đây là một compile time exception. Nếu mã nhập vào hợp lệ, viết hoa toàn bộ mã nhân viên và gán vào cho đối tượng nhân viên.
* Code mẫu:
* Lớp Employee.java

package lesson69;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class Employee {

private String id;

private String fullName;

private String email;

public Employee() {

}

public Employee(String id) throws InvalidEmpIdException {

setId(id);

}

public Employee(String id, String fullName, String email)

throws InvalidEmpIdException {

this(id);

this.fullName = fullName;

this.email = email;

}

public String getId() {

return id;

}

public void setId(String id) throws InvalidEmpIdException {

var regex = "^(NV|EMP)\\d{6}[a-z]{2,5}$";

Pattern pattern = Pattern.compile(regex, Pattern.CASE\_INSENSITIVE);

Matcher matcher = pattern.matcher(id);

if (matcher.matches()) {

this.id = id.toUpperCase();

} else {

var msg = "Mã nhân viên không hợp lệ, vui lòng kiểm tra lại: " + id;

this.id = null;

throw new InvalidEmpIdException(msg, id);

}

}

public String getFullName() {

return fullName;

}

public void setFullName(String fullName) {

this.fullName = fullName;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

}

* Lớp InvalidEmpIdException.java

package lesson69;

public class InvalidEmpIdException extends Exception {

private String invalidId;

public InvalidEmpIdException() {

}

public InvalidEmpIdException(String invalidId) {

this.invalidId = invalidId;

}

public InvalidEmpIdException(String message, String invalidId) {

super(message);

this.invalidId = invalidId;

}

public String getInvalidId() {

return invalidId;

}

}

* Lớp Test.java

package lesson69;

import java.util.Scanner;

public class Test {

public static void main(String[] args) {

var emp = new Employee();

var input = new Scanner(System.in);

System.out.println("Nhập mã nhân viên, ví dụ: NV100000IT");

var id = input.nextLine().trim();

try {

emp.setId(id);

} catch (InvalidEmpIdException e) {

System.out.println(e.getMessage());

System.out.println("Ví dụ mã hợp lệ: NV100000IT");

}

System.out.println("Mã nhân viên vừa nhập là: " + emp.getId());

}

}

* Kết quả:

Nhập mã nhân viên, ví dụ: NV100000IT

NV100000CVKD

Mã nhân viên vừa nhập là: NV100000CVKD

Nhập mã nhân viên, ví dụ: NV100000IT

EMP10000CVKD

Mã nhân viên không hợp lệ, vui lòng kiểm tra lại: EMP10000CVKD

Ví dụ mã hợp lệ: NV100000IT

Mã nhân viên vừa nhập là: null

# Bài 12.4. kiểm tra định dạng email

### Sử dụng regular expression kiểm tra định dạng email

* Yêu cầu:
* Giả sử định dạng email của Google được cho như sau:
  + Bắt đầu bởi chữ cái
  + Tiếp đó là các kí tự có thể là chữ cái, chữ số, dấu chấm, dấu gạch dưới
  + Tiếp theo là @gmail
  + Kết thúc bởi đuôi .com
  + Không phân biệt chữ hoa chữ thường
* Ví dụ email hợp lệ:
  + [thantrieu@gmail.com](mailto:thantrieu@gmail.com)
  + [hoanganh123\_jinx@gmail.com](mailto:hoanganh123_jinx@gmail.com)
* Ví dụ email không hợp lệ:
  + than [trieu@gmail.com](mailto:trieu@gmail.com)
  + [Hoang^o^oanh###@gmail.co](mailto:Hoang%5eo%5eoanh)
* Nếu email không hợp lệ thì gán email bằng null, văng ngoại lệ InvalidEmailException thông báo cho người dùng biết và xử lý. Nếu email hợp lệ thì thông báo ra màn hình email hợp lệ.
* Code mẫu:
* Lớp User.java

package lesson70;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class User {

private String email;

private String username;

private String password;

public User() {

}

public User(String email, String username, String password)

throws InvalidEmailException {

this.setEmail(email);

this.username = username;

this.password = password;

}

public String getEmail() {

return email;

}

public void setEmail(String email) throws InvalidEmailException {

var regex = "^[a-z]+[\\w.\_]\*@gmail.com$";

Pattern pattern = Pattern.compile(regex, Pattern.CASE\_INSENSITIVE);

Matcher matcher = pattern.matcher(email);

if (matcher.matches()) {

this.email = email.toLowerCase();

} else {

this.email = null;

var msg = "Email không hợp lệ: " + email;

throw new InvalidEmailException(msg, email);

}

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

}

* Lớp InvalidEmailException.java

package lesson70;

public class InvalidEmailException extends Exception {

private String invalidEmail;

public InvalidEmailException() {

}

public InvalidEmailException(String message, String invalidEmail) {

super(message);

this.invalidEmail = invalidEmail;

}

public String getInvalidEmail() {

return invalidEmail;

}

}

* Lớp Test.java

package lesson70;

import java.util.Scanner;

public class Test {

public static void main(String[] args) {

var input = new Scanner(System.in);

System.out.println("Nhập email: ");

var email = input.nextLine().trim();

User user = new User();

try {

user.setEmail(email);

System.out.println("Email hợp lệ!");

} catch (InvalidEmailException e) {

System.out.println(e.getMessage());

System.out.println("Email bạn vừa nhập không đúng.");

}

}

}

* Kết quả:

Nhập email:

thantrieu123@gmail.com

Email hợp lệ!

Nhập email:

123than@gmail.com

Email không hợp lệ: 123than@gmail.com

Email bạn vừa nhập không đúng.

# Bài 12.5. Kiểm tra định dạng mật khẩu

### Áp dụng regular expression kiểm tra định dạng mật khẩu

* Yêu cầu:
* Định dạng mật khẩu của tài khoản người dùng trong một hệ thống nào đó được cho như sau:
  + Dài tối thiểu 8 kí tự
  + Bao gồm cả kí tự chữ cái thường, chữ cái hoa
  + Chứa tối thiểu 1 kí tự số và 1 kí tự đặc biệt(~!#$^&)
  + Mật khẩu không được trùng với địa chỉ email của người dùng
* Ví dụ email của người dùng là [thantrieu@gmail.com](mailto:thantrieu@gmail.com) thì mật khẩu hợp lệ là:
  + thanTH27@blabla
  + momyGO111!
* Mật khẩu không hợp lệ:
  + [thantrieu@gmail.com](mailto:thantrieu@gmail.com)
  + 123456789
  + Abcblablabla
  + Abcblabla123
* Nếu mật khẩu không hợp lệ thì gán mật khẩu bằng null, văng ngoại lệ InvalidPasswordException thông báo cho người dùng biết. Nếu mật khẩu hợp lệ thông báo ra màn hình mật khẩu hợp lệ.
* Code mẫu:
* Lớp User.java

package lesson70;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class User {

private String email;

private String username;

private String password;

public User() {

}

public User(String email, String username, String password)

throws InvalidEmailException, InvalidPasswordException {

this.setEmail(email);

this.username = username;

this.setPassword(password);

}

public String getEmail() {

return email;

}

public void setEmail(String email) throws InvalidEmailException {

var regex = "^[a-z]+[\\w.\_]\*@gmail.com$";

Pattern pattern = Pattern.compile(regex, Pattern.CASE\_INSENSITIVE);

Matcher matcher = pattern.matcher(email);

if (matcher.matches()) {

this.email = email.toLowerCase();

} else {

this.email = null;

var msg = "Email không hợp lệ: " + email;

throw new InvalidEmailException(msg, email);

}

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getPassword() {

return password;

}

public void setPassword(String password) throws InvalidPasswordException {

var regex = "^(.){8,}$";

var regex2 = "^.\*[a-z]+.\*$";

var regex3 = "^.\*[A-Z]+.\*$";

var regex4 = "^.\*[0-9]+.\*$";

var regex5 = "^.\*[~!@#$%^&\*]+.\*$";

var regex6 = email;

Pattern pattern = Pattern.compile(regex);

Matcher matcher = pattern.matcher(password);

Pattern pattern2 = Pattern.compile(regex2);

Matcher matcher2 = pattern2.matcher(password);

Pattern pattern3 = Pattern.compile(regex3);

Matcher matcher3 = pattern3.matcher(password);

Pattern pattern4 = Pattern.compile(regex4);

Matcher matcher4 = pattern4.matcher(password);

Pattern pattern5 = Pattern.compile(regex5);

Matcher matcher5 = pattern5.matcher(password);

Pattern pattern6 = Pattern.compile(regex6, Pattern.CASE\_INSENSITIVE);

Matcher matcher6 = pattern6.matcher(password);

if (matcher.matches() && matcher3.matches() && matcher2.matches()

&& matcher4.matches() && matcher5.matches()

&& !matcher6.matches()) {

this.password = password;

} else {

this.password = null;

var msg = "Mật khẩu không hợp lệ: " + password;

throw new InvalidPasswordException(msg, password);

}

}

}

* Lớp Ngoại lệ: InvalidPasswordException .java

package lesson70;

public class InvalidPasswordException extends Exception {

private String invalidPassword;

public InvalidPasswordException() {

}

public InvalidPasswordException(String invalidPassword) {

this.invalidPassword = invalidPassword;

}

public InvalidPasswordException(String message, String invalidPassword) {

super(message);

this.invalidPassword = invalidPassword;

}

public String getInvalidPassword() {

return invalidPassword;

}

}

* Lớp Test.java

package lesson70;

import java.util.Scanner;

public class Test {

public static void main(String[] args) {

var input = new Scanner(System.in);

// System.out.println("Nhập email: ");

var email = "thanTRieu1232@gmail.com";

User user = new User();

try {

user.setEmail(email);

System.out.println("Email hợp lệ: " + email);

} catch (InvalidEmailException e) {

System.out.println(e.getMessage());

System.out.println("Email bạn vừa nhập không đúng.");

}

// nhập mật khẩu

System.out.println("Nhập mật khẩu(gồm chữ số, chữ cái thường," +

"\n chữ cái hoa, kí tự đặc biệt, không trùng với email," +

"\n tối thiểu 8 kí tự.");

var password = input.nextLine();

try {

user.setPassword(password);

System.out.println("Mật khẩu hợp lệ!");

} catch (InvalidPasswordException e) {

e.printStackTrace();

System.out.println("Hãy nhập đúng định dạng mật khẩu!");

}

}

}

* Kết quả:

Email hợp lệ: thanTRieu1232@gmail.com

Nhập mật khẩu(gồm chữ số, chữ cái thường,

chữ cái hoa, kí tự đặc biệt, không trùng với email,

tối thiểu 8 kí tự.

THAN123trieuHAHAHA@@

Mật khẩu hợp lệ!

Email hợp lệ: thanTRieu1232@gmail.com

Nhập mật khẩu(gồm chữ số, chữ cái thường,

chữ cái hoa, kí tự đặc biệt, không trùng với email,

tối thiểu 8 kí tự.

thanTRieu1232@gmail.com

lesson70.InvalidPasswordException: Mật khẩu không hợp lệ: thanTRieu1232@gmail.com

at lesson70.User.setPassword(User.java:81)

at lesson70.Test.main(Test.java:25)

Hãy nhập đúng định dạng mật khẩu!

**Bài 12.6. Tổng quan về annotation**

**Nội dung bài học**

1. Khái niệm và mục đích sử dụng
2. Định dạng của annotation
3. Annotation được sử dụng ở đâu

**Khái niệm và mục đích sử dụng**

* Annotation dịch ra là chú giải, mô tả, chú thích, là một dạng siêu dữ liệu(metadata) cung cấp dữ liệu về chương trình không phải một phần của chính bản thân chương trình đó.
* Annotation không gây ảnh hưởng trực tiếp lên các thao tác trong mã nguồn mà nó chú thích.
* Annotation có nhiều dạng sử dụng khác nhau:
  + Annotation có thể được sử dụng bởi trình biên dịch để phát hiện lỗi, ngăn chặn các cảnh báo.
  + Các công cụ phần mềm có thể xử lý thông tin từ annotation để sinh code, file XML, …
  + Một số annotation sẵn sàng để được xử lý tại thời điểm chạy chương trình(runtime).

**Định dạng của annotation**

* Annotation có dạng @X. Ví dụ:
  + @Entity
  + @Override
  + @Author(name=”Susan”)…
* Trong đó một annotation luôn bắt đầu với kí tự @ sau đó là tên của annotation. Kí tự @ báo với trình biên dịch rằng ngay sau đó là một annotation.
* Ví dụ:

@Override

public void paint(Graphics g) {

ar g2d = (Graphics2D) g;

g2d.setStroke(new BasicStroke(2));

drawCircle(g);

drawRoundRect(g);

}

* Một annotation có thể chứa các phần tử bên trong, ví dụ:

@ClassPreamble(

date = "2022/09/28",

lastModify = "2022/09/28",

lastModifyBy = "Branium Academy",

reviewers = {"Than Trieu", "Branium"}

)

public class MyFrame extends JFrame { ...

* Nếu annotation chỉ chứa 1 phần tử bên trong thì ta có thể bỏ tên phần tử đi và chỉ cung cấp giá trị của nó thôi.

@SuppressWarnings(value = "unchecked")

void myMethod() { ... } chuyển thành:

@SuppressWarnings("unchecked")

void myMethod() { ... }

* Nếu annotation không chứa phần tử nào bên trong ta có thể lược bỏ cặp dấu ngoặc tròn như @Override.
* Một khai báo có thể chứa nhiều annotation khác nhau:

@Author(name = "Branium Academy")

@EBook

class MyClass { ... }

* Nếu một annotation được sử dụng lặp đi lặp lại nhiều lần thì ta gọi nó là annotation lặp. Ví dụ:

@Author(name = "Janny")

@Author(name = "Huong")

class MyClass { ... }

* Kiểu của một annotation có thể là kiểu bất kì nào trong gói java.lang hoặc java.lang.annotation của Java chuẩn SE.
* Ngoài ra bạn có thể tự định nghĩa annotation của riêng mình. Chi tiết trong các bài học kế tiếp.

**Annotation có thể được sử dụng ở đâu?**

* Annotation có thể áp dụng cho các khai báo:
  + Khai báo lớp.
  + Khai báo phương thức.
  + Khai báo trường dữ liệu.
  + Các phần tử khác của chương trình.
* Theo nguyên tắc mỗi annotation được khai báo trên một dòng.
* Từ Java 8, annotation có thể được sử dụng trong:
  + Biểu thức tạo đối tượng: new @Interned SomeObject();
  + Ép kiểu: str = (@NoneNull String)otherStr;
  + Trong mệnh đề implements: class SomeClass<T> implements @Readonly List<@Readonly T> {…}
  + Trong khai báo ngoại lệ: void method() throws @Critical SomeException {…}
* Các dạng annotation vừa liệt kê ở trên gọi là một type annotation(chú thích kiểu dữ liệu).

# Bài 12.7. Các annotation có sẵn trong Java

**Nội dung bài học**

1. Annotation @Deprecated
2. Anotation @Override
3. Annotation @SuppressWarnings
4. Một số annotation dùng để quản lý các annotation khác

**Annotation @Deprecated**

* Trong gói java.lang có nhiều annotation được định nghĩa sẵn. Ta sẽ lần lượt tìm hiểu về chúng
* **@Deprecated** annotation được sử dụng để chỉ ra rằng một phần tử nào đó đã lỗi thời và sẽ bị loại bỏ trong tương lai.
* Do đó người dùng không nên sử dụng chúng nữa. Khi người dùng sử dụng những thành phần được đánh dấu với @Deprecated thì trình biên dịch sẽ sinh ra một cảnh báo.
* Các thành phần được đánh dấu với annotation này nên được chú thích tài liệu với thẻ Javadoc @deprecated.
* Lưu ý phải dùng @ ở cả hai nơi. Trong đó chú thích tài liệu bắt đầu với chữ thường còn annotation bắt đầu với chữ hoa sau @.
* Ví dụ:

public class LearnAnnotation {

/\*\*

\* @deprecated

\* This method is no longer supported next year.

\* @param message the message to print on screen

\*/

@Deprecated

public void printMessage(String message) {

System.out.println("Message: " + message);

}

public static void main(String[] args) {

var message = "Hello Java";

var obj = new LearnAnnotation();

obj.printMessage(message);

}

}

**Anotation @Override**

* **@Override** annotation được sử dụng để thông báo cho trình biên dịch rằng thành phần đang được đánh dấu là được ghi đè từ lớp/interface cha.
* Việc ghi đè sẽ tái định nghĩa lại cùng một hành động đã có ở lớp cha theo cách của lớp con. Chi tiết bạn xem trong phần bài học về kế thừa và tính trừu tượng.
* Annotation này không bắt buộc phải dùng khi ta ghi đè phương thức của lớp cha. Tuy nhiên việc sử dụng annotation sẽ giúp ta ngăn chặn lỗi. Tức là khi phương thức đánh dấu với @Override ghi đè thất bại thì trình biên dịch sẽ sinh lỗi.

interface Calculable {

int add(int a, int b);

}

class Calculator implements Calculable {

@Override

public int add(int a, int b) { // triển khai interface Calculable

return a + b;

}

}

class ChildCalculator extends Calculator {

@Override

public int add(int a, int b) { // ghi đè phương thức của lớp cha

return super.add(a, b) + 1;

}

}

**Annotation @SuppressWarnings**

* **@SuppressWarnings** annotation sử dụng để báo với trình biên dịch rằng cần ngăn chặn những cảnh báo cụ thể nào đó nếu được sinh ra.
* Ví dụ khi ta sử dụng phương thức đã lỗi thời, tức bị đánh dấu với @Deprecated thì trình biên dịch thường sinh cảnh báo. Để chặn cảnh báo được sinh ra ta dùng @SuppressWarnings.

@SuppressWarnings("all")

public static void main(String[] args) {

Thread thread = new Thread("My Thread");

thread.start();

thread.suspend(); // call to deprecated method of thread object

}

* Các giá trị có thể đưa vào trong làm đối số của @SuppressWarnings gồm:
  + **all**: chặn mọi loại cảnh báo.
  + **cast**: chặn cảnh báo khi ép kiểu từ kiểu generic sang một kiểu không đạt chuẩn hoặc ngược lại.
  + **deprecation**: phớt lờ cảnh báo khi sử dụng các kiểu, phương thức đã lỗi thời(không còn quan trọng).
  + **divzero**: chặn cảnh báo chia cho 0.
  + **empty**: chặn cảnh báo một câu lệnh rỗng phần thân.
  + **unchecked**: bỏ qua việc kiểm tra kiểu là đối tượng hay nguyên thủy.
  + **fallthrough**: bỏ qua việc trượt lệnh trong cấu trúc switch nếu thiếu break trong một case nào đó.
  + **hiding**: chặn cảnh báo khi biến cục bộ trùng tên các biến bên ngoài, thuộc tính làm chúng bị ẩn khỏi phạm vi cục bộ trong khối nào đó.
  + **serial**: chặn việc cảnh báo thiếu/sai sót giá trị serialVersionUID.
  + **finally**: tránh cảnh báo liên quan đến khối finally không return.
  + **unused**: chặn các cảnh báo liên quan đến các đoạn code không được sử dụng
* Ví dụ 2: khi một lớp, một phương thức,… được tạo ra mà không sử dụng, trình biên dịch có thể sinh cảnh báo và gợi ý xóa bỏ các phần không sử dụng. Ta có thể sử dụng SuppressWarnings(“unused”) để chặn các cảnh báo này.

interface Calculable {

@SuppressWarnings("unused")

int add(int a, int b);

}

class Calculator implements Calculable {

@Override

public int add(int a, int b) { // triển khai interface Calculable

return a + b;

}

}

@SuppressWarnings("unused")

class ChildCalculator extends Calculator {

@Override

public int add(int a, int b) { // ghi đè phương thức của lớp cha

return super.add(a, b) + 1;

}

}

**Một số annotation quản lý các annotation khác**

* @FunctionalInterface là annotation được sử dụng để chỉ định rằng một interface được sử dụng với mục đích làm functional interface.
* Các annotation có thể sử dụng để áp đặt ngữ nghĩa lên các annotation khác. Annotation loại này được gọi là siêu chú thích(meta-annotations).
* Ví dụ cụ thể sẽ được liệt kê ngay dưới đây.
* @Retention annotation chỉ định cách thức lưu trữ của các annotation:
  + *RetentionPolicy.SOURCE*: annotation chỉ được lưu trữ ở cấp độ tài nguyên và bị lờ đi bởi trình biên dịch.
  + *RetentionPolicy.CLASS*: annotation được lưu trữ bởi trình biên dịch tại thời điểm biên dịch và bị bỏ qua bởi JVM.
  + *RetentionPolicy.RUNTIME*: annotation được lưu trữ bởi JVM do đó nó có thể được sử dụng bởi môi trường đang hoạt động.
* @Documented annotation chỉ ra rằng khi nào một annotation đang chỉ định được sử dụng thì các phần tử đó phải được mô tả trong tài liệu sử dụng công cụ Javadoc. Theo mặc định thì các annotation không được bao gồm trong Javadoc.
* @Target annotation dùng để đánh dấu một annotation khác nhằm giới hạn những kiểu phần tử Java mà annotation đó có thể khả dụng.
  + *ElementType.ANNOTATION\_TYPE*: có thể áp dụng cho một kiểu annotation.
  + *ElementType.CONSTRUCTOR*: có thể áp dụng cho phương thức khởi tạo.
  + *ElementType.FIELD*: có thể áp dụng cho các trường hoặc thuộc tính.
  + *ElementType.LOCAL\_VARIABLE*: có thể áp dụng cho các biến cục bộ.
  + *ElementType.METHOD*: có thể áp dụng cho các phương thức.
  + *ElementType.PARAMETER*: có thể áp dụng cho các tham số của một phương thức.
  + *ElementType.TYPE*: có thể áp dụng cho bất kì phần tử nào của một lớp.
* @Inherited annotation chỉ ra rằng một annotation nào đó có thể được kế thừa từ lớp cha. Chỉ áp dụng cho khai báo lớp. Khi cần truy vấn tới một annotation trong một lớp mà không có kiểu annotation đó, lớp cha của lớp đó sẽ thực hiện việc tìm kiếm annotation cần dùng.
* @Repeatable annotation dùng để đánh dấu một annotation có thể được áp dụng nhiều lần lặp đi lặp lại.

# Bài 12.8. Các annotation user tự định nghĩa

### Nội dung bài học

1. Tạo annotation cấp độ class
2. Tạo annotation cấp độ field
3. Tạo annotation cấp độ method
4. Ứng dụng annotation vừa tạo
5. Xử lý annotations

* Annotation là kĩ thuật bổ sung siêu dữ liệu vào mã nguồn chương trình.
* Đây là phần đầy thú vị và quyền năng được thêm vào java từ phiên bản java 1.5.
* Mặc dù ta có thể gắn annotation vào package, các lớp, interface, phương thức,… nhưng bản thân annotation không gây ảnh hưởng đến việc hoạt động của chương trình.
* Nội dung này ta sẽ tạo và sử dụng các annotation ở các level khác nhau.

### Tạo annotation cấp độ class

* Để tạo annotation ta bắt đầu với @interface sau đó là tên của annotation. Quy tắc đặt tên annotation như tên lớp, interface:

public @interface JsonSerializable {

}

* Tiếp theo ta thêm meta-annotation để chỉ định phạm vi và mục tiêu của từng annotation:

@Retention(RetentionPolicy.RUNTIME)

@Target(ElementType.TYPE)

public @interface JsonSerializable {

}

* Ta thấy rằng annotation trên hiệu lực với thời gian runtime, có thể áp dụng với các kiểu(class), không có phương thức do đó phục vụ như một marker với các lớp có thể được tuần tự hóa sang JSON.
* Ví dụ 2:

import java.lang.annotation.\*;

@Retention(RetentionPolicy.CLASS)

@Target(ElementType.TYPE)

@Documented

public @interface ClassPreamble {

String author() default "Branium Academy";

String date();

int currentRevision() default 1;

String lastModify() default "N/A";

String lastModifyBy() default "N/A";

String[] reviewers();

}

* Khi ta khai báo annotation ta có thể tạo 0, 1 hoặc nhiều phần tử chứa trong annotation đó để bổ sung thông tin.
* Mỗi phần tử sẽ có kiểu là các kiểu hợp lệ trong Java như String, int, float, mảng…
* Các phần tử có tên bắt đầu với chữ thường như tên phương thức và không có tham số.
* Mỗi phần tử có thể chứa giá trị mặc định đi liền ngay sau keyword default.
* Mỗi phần tử đặt trên 1 dòng và kết thúc với dấu chấm phẩy như khai báo trường, thuộc tính.

### Tạo annotation cấp độ field

* Tiếp tục từ ví dụ đầu tiên của bài học, ta tạo annotation để đánh dấu các trường sẽ bao gồm trong quá trình sinh tài liệu JSON:

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

@Retention(RetentionPolicy.RUNTIME)

@Target(ElementType.FIELD)

public @interface JsonElement {

public String key() default "";

}

* Trong ví dụ trên ta khai báo phần tử tên key với giá trị mặc định là string rỗng “”.
* Lưu ý rằng trong ngoặc tròn của phần tử trong anotation không được phép chứa tham số và không throws bất kì ngoại lệ nào.
* Giá trị mặc định gắn cho phần tử trong annotation không thể null.

### Tạo annotation cấp độ method

* Giả định rằng trước khi tuần tự hóa một đối tượng sang JSON string ta muốn thực hiện vài thao tác để khởi tạo một đối tượng.
* Lúc này ta cần tạo một annotation để hiện thực hóa mục đích này.

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

@Retention(RetentionPolicy.RUNTIME)

@Target(ElementType.METHOD)

public @interface Init {

}

### Ứng dụng annotation vừa tạo

* Bây giờ ta sẽ tạo một lớp mô tả thông tin nhân viên của một công ty với các thông tin: mã nhân viên, họ tên, email, mức lương, số điện thoại, địa chỉ.

@JsonSerializable

public class Employee {

@JsonElement

private String EmpId; // mã nhân viên

@JsonElement

private String firstName; // tên

@JsonElement

private String lastName; // họ

@JsonElement

private String email; // email

@JsonElement

private float salary; // mức lương

private String address; // địa chỉ

/\*\*

\* Phương thức khởi tạo họ và tên trở thành cụm string có chữ cái đầu từ được viết hoa

\*/

@Init

private void initNames() {

this.firstName = this.firstName.substring(0, 1).toUpperCase() +

this.firstName.substring(1);

this.lastName = this.lastName.substring(0, 1).toUpperCase() +

this.lastName.substring(1);

}

}

* Theo mô tả của các annotation ta tự định nghĩa thì ta muốn rằng có thể tuần tự hóa một nhân viên sang JSON string. Trong đó dữ liệu đầu ra chỉ cần mã nhân viên, họ, tên, email và mức lương.
* Ta cũng chỉ định rằng muốn gọi initNames() trước khi tiến hành tuần tự hóa.
* Bằng cách đặt @JsonElement trước một thuộc tính của lớp ta ám chỉ rằng muốn dùng thành phần đó trong kết quả đầu ra của JSON string.

### Xử lý annotations

* Bây giờ ta sẽ đưa nội dung vừa có ở trên vào vận hành sử dụng Java Reflection API.
* Bước đầu tiên là kiểm tra xem đối tượng cần tuần tự hóa có null không và tiếp đó kiểm tra xem đối tượng đó có được gắn annotation @JsonSerializable không:

private void checkifSerializable(Object obj) {

if (Objects.isNull(obj)) {

throw new JSonSerializationException("The object to serialize is null");

}

Class<?> c = obj.getClass();

if (!c.isAnnotationPresent(JsonSerializable.class)) {

throw new JSonSerializationException("The class " + c.getSimpleName() +

" is not annotated with JsonSerializable");

}

}

* Tiếp theo ta tìm phương thức có nhãn annotation @Init và thực thi nó:

private void initializeObject(Object obj) throws Exception {

Class<?> c = obj.getClass();

for (Method method : c.getDeclaredMethods()) {

if (method.isAnnotationPresent(Init.class)) {

method.setAccessible(true);

method.invoke(obj);

}

}

}

* Việc gọi tới method.setAccessible(true); đảm bảo rằng ta có thể thực thi phương thức private initNames() từ trong lớp bên ngoài hoàn toàn tách biệt với lớp Employee.
* Sau đó ta duyệt các trường của đối tượng và truy cập key, value của phần tử JSON. Tiếp đó ném chúng vào một bản đồ và tạo JSON string từ bản đồ đó.

private String getJsonString(Object obj) throws Exception {

Class<?> c = obj.getClass();

Map<String, String> jsonElementsMap = new HashMap<>();

for (Field field : c.getDeclaredFields()) {

field.setAccessible(true);

if (field.isAnnotationPresent(JsonElement.class)) {

jsonElementsMap.put(getKey(field), field.get(obj) + "");

}

}

String jsonString = jsonElementsMap.entrySet()

.stream()

.map(entry -> "\"" + entry.getKey() + "\":\"" + entry.getValue() + "\"")

.collect(Collectors.joining(","));

return "{" + jsonString + "}";

}

* Ở đây ta cũng phải gọi field.setAccessible(true); để có thể truy cập tới các trường private của đối tượng.
* Sau đó ta tạo phương thức để tuần tự hóa kết hợp các bước trên nhằm tạo ra một JSON string từ Java object:

public String convertToJson(Object obj) throws JSonSerializationException {

try {

checkifSerializable(obj);

initializeObject(obj);

return getJsonString(obj);

} catch (Exception e) {

throw new JSonSerializationException(e.getMessage());

}

}

* Cuối cùng ta tạo một lớp tên Main chứa phương thức main() để thực thi xem kết quả có ok không:

public class Main {

public static void main(String[] args) {

Employee employee = new Employee("EMP1001", "hương",

"nguyễn", "huongcute@xmail.com", "Hà Nội", 12.5f);

var converter = new ObjectToJsonConverter();

var result = converter.convertToJson(employee);

System.out.println(result);

}

}

{"empId":"EMP1001","firstName":"Hương","lastName":"Nguyễn","salary":"12.5","email":"huongcute@xmail.com"}

* Để ý rằng họ tên khi tạo đối tượng và trong kết quả có sự khác biệt từ chữ thường => chữ hoa(chữ cái đầu từ).
* Thật thú vị và ảo diệu phải không bạn? Vậy là ta đã tạo được một chương trình tương đối phức tạp ứng dụng annotation trong Java rồi đó.
* Chi tiết các lớp, anotation của bài học bạn [click vào đây](https://github.com/thantrieu/JavaBasicToAdvanced/tree/master/src/net/braniumacademy/customanotations) để xem nhé.

# Bài 13.1. Đọc file text với BufferedReader

### Nội dung bài học

1. [Giới thiệu về BufferedReader](https://braniumacademy.net/courses/java/lessons/java-read-text-file-bufferedreader/#s1)
2. [Các phương thức và mô tả](https://braniumacademy.net/courses/java/lessons/java-read-text-file-bufferedreader/#s2)
3. [Ví dụ minh họa](https://braniumacademy.net/courses/java/lessons/java-read-text-file-bufferedreader/#s4)
4. [Bài tập thực hành](https://braniumacademy.net/courses/java/lessons/java-read-text-file-bufferedreader/#s3)

### Giới thiệu về BufferedReader

* Lớp BufferedReader nằm trong gói java.io và có định nghĩa như sau:

public class BufferedReader extends Reader { // kế thừa từ lớp Reader

private Reader in;

private char[] cb;

private int nChars, nextChar;

private static final int INVALIDATED = -2;

private static final int UNMARKED = -1;

private int markedChar = UNMARKED;

private int readAheadLimit = 0; /\* Valid only when markedChar > 0 \*/

/\*\* If the next character is a line feed, skip it \*/

private boolean skipLF = false;

/\*\* The skipLF flag when the mark was set \*/

private boolean markedSkipLF = false;

private static int defaultCharBufferSize = 8192; // kích thước bộ đệm mặc định chứa đc 8192 kí tự

private static int defaultExpectedLineLength = 80; // giả định độ dài mặc định của một dòng

// ...

}

* Lớp này dùng để đọc dữ liệu dạng text từ luồng nhập liệu kí tự, lưu vào bộ đệm từ đó tăng hiệu quả đọc các kí tự, mảng và các dòng dữ liệu văn bản.
* Ta có thể chỉ định kích thước bộ đệm theo mong muốn hoặc sử dụng kích thước bộ đệm mặc định của lớp này. Giá trị mặc định đủ lớn để ta thực hiện hầu hết các mục đích khác nhau.
* Nhìn chung ta nên gói các yêu cầu đọc dữ liệu với phương thức read() trong đối tượng BufferedReader bởi vì việc đọc từng ký tự hoặc luồng các byte cơ bản thường gây tốn kém chi phí để thực hiện. Lớp BufferedReader sẽ giúp tăng hiệu quả đọc dữ liệu.
* Khuyến nghị: ta nên thay những đoạn code sử dụng DataInputStream dùng để đọc dữ liệu dạng văn bản bởi BufferedReader.
* Khi hoạt động, đối tượng của lớp BufferedReader sẽ đọc cả một tập kí tự và lưu tạm vào trong bộ đệm của nó. Sau đó từ trong bộ đệm nội tại của nó, từng kí tự sẽ được đọc ra.
* Từ đó số lần phải giao tiếp giữa chương trình với ổ cứng sẽ giảm đi đáng kể dẫn tới tính hiệu quả của BufferedReader.

### Các phương thức và mô tả

* Sau đây là một số phương thức phổ biến thường sử dụng của lớp:
  + BufferedReader(@NotNull Reader reader): phương thức khởi tạo dùng để tạo một đối tượng BufferedReader sử dụng bộ đệm với kích thước mặc định.
  + BufferedReader(@NotNull Reader reader, int size): phương thức khởi tạo dùng để tạo đối tượng BufferedReader sử dụng kích thước bộ đệm được chỉ định. Nếu kích thước chỉ định <= 0 ta sẽ nhận được một ngoại lệ.
  + int read(): phương thức dùng để đọc một kí tự đơn, trả về mã định danh của kí tự đó trong bảng mã.
  + int read(char[] cbuf): đọc vào một tập kí tự và lưu nó vào mảng đã chỉ định.
  + int read(char[] cbuf, int offset, int len): đọc tối đa len kí tự lưu vào mảng cbuf từ vị trí offset. Trả về số lượng kí tự đọc được hoặc -1 nếu đã đến cuối luồng. Ngoại lệ IndexOutOfBoundException nếu offset âm hoặc len âm hoặc len lớn hơn cbuf.length – offset.
  + String readLine(boolean ignoreLF, boolean[] term): đọc một dòng văn bản. Một dòng có thể kết thúc bằng kí tự \n hoặc \r hoặc \r\n hoặc kết thúc file EOF. Trả về một chuỗi kí tự là nội dung của dòng đọc được, không bao gồm kí tự kết thúc dòng ở cuối. Nếu đọc đến cuối luồng và không có kí tự nào thì ta sẽ nhận được giá trị null.
  + String readLine(): đọc cả dòng, tương tự như phương thức trên nhưng không cần tham số đầu vào.
  + long skip(long n): bỏ qua n kí tự.
  + boolean ready(): kiểm tra xem liệu rằng luồng này đã sẵn sàng để đọc dữ liệu chưa. Một bộ đệm đã sẵn sàng khi nó không trống hoặc các kí tự ở trong luồng đã sẵn sàng.
  + void mark(int readAheadLimit): đánh dấu vị trí hiện tại trong luồng. Các lời gọi tới reset() sau đó sẽ đưa luồng về vị trí được chỉ định này.
  + void reset(): reset luồng về vị trí gần nhất đã đánh dấu.
  + Stream<String> lines(): trả về một đối tượng Stream của các dòng đã đọc được bởi đối tượng BufferedReader.
  + void close(): đóng luồng của đối tượng BufferedReader hiện tại. Ta không thể thực hiện các thao tác khác sau khi luồng đã đóng với phương thức này. Ta cũng không thể đóng một luồng đã đóng trước đó.

### Ví dụ minh họa

* Sau đây ta sẽ minh họa việc đọc file với đối tượng của lớp BufferedInputStream.
* File input.txt:

S1001,Lê Bá Nam,Quảng Ninh,CNTT,3.26

S1002,Hoàng Quốc Tuấn,Hồ Chí Minh,CNTT,3.54

S1003,Mai Thị Diệu Linh,Vũng Tàu,CNTT,3.71

S1004,Ngô Quang Trường,Bình Dương,CNTT,3.33

S1005,Hồ Hoài Anh,Quảng Bình,CNTT,3.12

S1006,Nguyễn Ngọc Mai,Đà Nẵng,CNTT,3.95

S1007,Nguyễn Thị Ngọc Linh,Hà Nội,CNTT,3.18

S1008,Trần Quang Khải,Bắc Giang,CNTT,3.29

S1009,Ma Thùy Dung,Thái Nguyên,CNTT,3.37

S1010,Lương Đình Vinh,Nghệ An,CNTT,3.59

* File Student.java: mô tả thông tin sinh viên.

package net.braniumacademy.chapter13.lesson1;

import java.util.Objects;

public class Student {

private String studentId; // mã sv

private String fullName; // họ tên

private String major; // chuyên ngành

private String address; // địa chỉ(tỉnh/thành phố)

private float gpa; // điểm gpa

public Student() {

}

public Student(String studentId) {

this.studentId = studentId;

}

public Student(String studentId, String fullName, String address, String major, float gpa) {

this.studentId = studentId;

this.fullName = fullName;

this.major = major;

this.address = address;

this.gpa = gpa;

}

public String getStudentId() {

return studentId;

}

public String getFullName() {

return fullName;

}

public String getMajor() {

return major;

}

public String getAddress() {

return address;

}

public float getGpa() {

return gpa;

}

@Override

public boolean equals(Object o) { // hai sinh viên được coi là một nếu có cùng mã sinh viên

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

Student student = (Student) o;

return Objects.equals(studentId, student.studentId);

}

@Override

public int hashCode() {

return Objects.hash(studentId);

}

@Override

public String toString() {

return "Student{" +

"studentId='" + studentId + '\'' +

", fullName='" + fullName + '\'' +

", major='" + major + '\'' +

", address='" + address + '\'' +

", gpa=" + gpa +

'}';

}

}

* File Lesson131.java: triển khai các phương thức phụ trợ và chạy chương trình.

package net.braniumacademy.chapter13.lesson1;

import java.io.\*;

import java.util.ArrayList;

import java.util.List;

public class Lesson131 {

public static void main(String[] args) {

var students = new ArrayList<Student>();

try {

var inputFile = new File("input.txt");

// sử dụng try-with-resource để không phải đóng các stream auto closable theo cách thủ công

try (var fileReader = new FileReader(inputFile);

var reader = new BufferedReader(fileReader)) {

// cách 1: sử dụng phương thức readLine() đọc cả dòng văn bản

var data = reader.readLine();

while (data != null) {

var student = createStudentFromData(data);

if (student != null) {

students.add(student);

}

data = reader.readLine();

}

}

} catch (IOException ex) {

System.out.println(ex.getMessage());

}

showStudentsInfo(students);

}

private static Student createStudentFromData(String data) {

if (data == null) {

return null;

}

var info = data.split(","); // các thuộc tính phân tách bằng dấu phẩy

if (info.length == 5) {

var gpa = Float.parseFloat(info[4]);

return new Student(info[0], info[1], info[2], info[3], gpa);

}

return null;

}

private static void showStudentsInfo(List<Student> students) {

for (var student : students) {

System.out.println(student);

}

}

}

* Cách 2: chỉ định bộ đệm riêng theo ý của ta:

package net.braniumacademy.chapter13.lesson1;

import java.io.\*;

import java.util.ArrayList;

import java.util.List;

public class Lesson131 {

public static void main(String[] args) {

var students = new ArrayList<Student>();

var dataBuilder = new StringBuilder();

try {

var inputFile = new File("input.txt");

try (var fileReader = new FileReader(inputFile);

var reader = new BufferedReader(fileReader)) {

var buff = new char[100]; // tạo mảng lưu bộ đệm 100 kí tự

while (reader.read(buff) != -1) {

dataBuilder.append(String.valueOf(buff));

}

}

} catch (IOException ex) {

System.out.println(ex.getMessage());

}

createStudents(students, dataBuilder.toString());

showStudentsInfo(students);

}

private static void createStudents(List<Student> students, String data) {

var lines = data.split("\r\n");

for (var line : lines) {

var student = createStudentFromData(line);

if (student != null) {

students.add(student);

}

}

}

private static Student createStudentFromData(String data) {

if (data == null) {

return null;

}

var info = data.split(","); // các thuộc tính phân tách bằng dấu phẩy

if (info.length == 5) {

var gpa = Float.parseFloat(info[4]);

return new Student(info[0], info[1], info[2], info[3], gpa);

}

return null;

}

private static void showStudentsInfo(List<Student> students) {

for (var student : students) {

System.out.println(student);

}

}

}

* Cách 3: chỉ định số lượng kí tự tối đa mỗi lần đọc:

try (var fileReader = new FileReader(inputFile);

var reader = new BufferedReader(fileReader)) {

var buff = new char[124]; // tạo mảng lưu bộ đệm 124 kí tự

// số kí tự tối đa mỗi lần đọc phải bằng số kí tự có thể chứa trong buff

// để không bị thừa dữ liệu rỗng

while (reader.read(buff, 0, 124) != -1) {

dataBuilder.append(String.valueOf(buff));

}

}

* Kết quả:

Student{studentId='S1001', fullName='Lê Bá Nam', major='CNTT', address='Quảng Ninh', gpa=3.26}

Student{studentId='S1002', fullName='Hoàng Quốc Tuấn', major='CNTT', address='Hồ Chí Minh', gpa=3.54}

Student{studentId='S1003', fullName='Mai Thị Diệu Linh', major='CNTT', address='Vũng Tàu', gpa=3.71}

Student{studentId='S1004', fullName='Ngô Quang Trường', major='CNTT', address='Bình Dương', gpa=3.33}

Student{studentId='S1005', fullName='Hồ Hoài Anh', major='CNTT', address='Quảng Bình', gpa=3.12}

Student{studentId='S1006', fullName='Nguyễn Ngọc Mai', major='CNTT', address='Đà Nẵng', gpa=3.95}

Student{studentId='S1007', fullName='Nguyễn Thị Ngọc Linh', major='CNTT', address='Hà Nội', gpa=3.18}

Student{studentId='S1008', fullName='Trần Quang Khải', major='CNTT', address='Bắc Giang', gpa=3.29}

Student{studentId='S1009', fullName='Ma Thùy Dung', major='CNTT', address='Thái Nguyên', gpa=3.37}

# Bài 13.2. Ghi file text với BufferedWriter

### Nội dung bài học

1. [Giới thiệu về BufferedWriter](https://braniumacademy.net/courses/java/lessons/java-bufferedwriter/#s1)
2. [Một số phương thức thường dùng](https://braniumacademy.net/courses/java/lessons/java-bufferedwriter/#s2)
3. [Ví dụ minh họa](https://braniumacademy.net/courses/java/lessons/java-bufferedwriter/#s3)
4. [Bài tập thực hành](https://braniumacademy.net/courses/java/lessons/java-bufferedwriter/#s4)

### Giới thiệu về BufferedWriter

* Lớp BufferedWriter của gói java.io có thể được sử dụng với các trình ghi dữ liệu ra file khác để ghi dữ liệu dạng văn bản ra file được hiệu quả.
* Lớp BufferedWriter kế thừa từ lớp Writer, sử dụng để ghi các kí tự đơn, mảng kí tự và chuỗi string ra một luồng đầu ra dạng văn bản, tạo bộ đệm các kí tự do đó nó cung cấp khả năng ghi dữ liệu văn bản hiệu quả.
* Triển khai của lớp BufferedWriter có dạng như sau:

public class BufferedWriter extends Writer {

private static final int DEFAULT\_INITIAL\_BUFFER\_SIZE = 512;

private static final int DEFAULT\_MAX\_BUFFER\_SIZE = 8192;

private Writer out;

private char cb[];

private int nChars, nextChar;

private final int maxChars; // maximum number of buffers chars

// ...

}

* Ta có thể sử dụng phương thức newLine() để tạo kí tự xuống dòng đặc thù nền tảng mà không cần trực tiếp sử dụng kí tự ‘\n’ để xuống dòng.
* Ví dụ:

private static void writeDataToFile(List<Student> students, String fileName) {

try (FileWriter fileWriter = new FileWriter(fileName);

BufferedWriter writer = new BufferedWriter(fileWriter)) {

for (var student : students) {

writer.write(student.toString()); // ghi cả dòng string ra file

writer.newLine(); // tạo dòng mới trong file ngay dưới dòng vừa ghi

}

} catch (IOException ex) {

System.out.println(ex.getMessage());

}

}

* Thông thường đối tượng Writer ngay lập tức gửi đầu ra của nó tới các kí tự hoặc luồng các byte phía dưới trừ khi một lời nhắc đầu ra được yêu cầu.
* Ta được khuyến nghị gói đối tượng BufferedWriter xung quanh bất kỳ đối tượng Writer nào sử dụng các biến thể của phương thức write() khi vận hành mà có thể gây tốn kém, ví dụ FileWriter, OutputStreamWriter.
* Nếu không sử dụng bộ đệm, mỗi lời gọi của phương thức write() sẽ làm cho các kí tự được chuyển đổi sang các byte và có thể được ghi ngay lập tức vào file. Do đó rất không hiệu quả.
* Lớp BufferedWriter sử dụng một bộ đệm nội tại có độ lớn **8192** kí tự.
* Khi thực hiện thao tác ghi, các kí tự cần ghi sẽ được ghi vào bộ đệm nội tại thay vì ghi trực tiếp vào ổ cứng. Một khi bộ đệm đã được ghi xong hoặc đối tượng writer được đóng lại, toàn bộ kí tự trong bộ đệm sẽ được ghi vào ổ cứng.
* Do đó số lần cần giao tiếp giữa chương trình và phần cứng sẽ giảm đáng kể -> dẫn tới tính hiệu quả và tốc độ cao khi sử dụng BufferedWriter.

### Một số phương thức thường dùng

* BufferedWriter(Writer out, int initSize): phương thức khởi tạo để tạo đối tượng BufferedWriter từ đối tượng writer, kích thước bộ đệm đầu ra.
* BufferedWriter(Writer out): tạo một bộ đệm kí tự đầu ra sử dụng kích thước bộ đệm mặc định với đối tượng Writer cho trước.
* flushBuffer(): đẩy dữ liệu bộ đệm đầu ra xuống luồng kí tự phía dưới mà không đẩy dữ liệu bên trong bản thân luồng dữ liệu đó ra file.
* write(int c): ghi một kí tự đơn ra file.
* write(char[] cbuff, int off, int len): ghi một phần mảng các kí tự ra file. Phương thức này lưu trữ các kí tự từ mảng cho trước vào luồng bộ đệm, đẩy bộ đệm xuống luồng phía dưới nếu cần. Khi độ dài yêu cầu >= độ dài bộ đệm thì phương thức này sẽ đẩy bộ đệm và ghi các kí tự trực tiếp vào luồng phía dưới.
  + cbuff: mảng bộ đệm được cung cấp.
  + off: vị trí bắt đầu đọc các kí tự.
  + len: số lượng các kí tự cần ghi.
  + Ngoại lệ IndexOutOfBoundException nếu tham số off âm hoặc len âm hoặc off + len âm hoặc lớn hơn độ dài của mảng cho trước.
  + Ngoại lệ IOException nếu thao tác I/O xảy ra lỗi.
* write(String s, int off, int len): ghi một phần chuỗi kí tự vào file.
  + s: chuỗi kí tự cần ghi.
  + off: vị trí từ đó bắt đầu đọc các kí tự.
  + len: số lượng các kí tự cần ghi.
  + Ngoại lệ có thể xảy ra: IndexOutOfBoundException, IOException.
* newLine(): ghi dòng trống vào file. Kí tự dòng trống được chỉ định theo hệ điều hành đang vận hành.
* flush(): đẩy dữ liệu ra khỏi luồng/làm sạch luồng.
* close(): đóng luồng.

### Ví dụ minh họa

* Sau đây ta sẽ minh họa việc nhập liệu thông tin sinh viên từ bàn phím và lưu trữ dữ liệu vào file sử dụng BufferedWriter.
* Lớp Student:

package net.braniumacademy.chapter13.lesson1;

import java.util.Objects;

public class Student {

private String studentId; // mã sinh viên

private String fullName; // họ và tên

private String address; // địa chỉ

private String major; // chuyên ngành

private float gpa; // điểm TB tích lũy gpa

public Student() {

}

public Student(String studentId) {

this.studentId = studentId;

}

public Student(String studentId, String fullName, String address, String major, float gpa) {

this.studentId = studentId;

this.fullName = fullName;

this.address = address;

this.major = major;

this.gpa = gpa;

}

public String getStudentId() {

return studentId;

}

public String getFullName() {

return fullName;

}

public String getAddress() {

return address;

}

public String getMajor() {

return major;

}

public float getGpa() {

return gpa;

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

Student student = (Student) o;

return Objects.equals(studentId, student.studentId);

}

@Override

public int hashCode() {

return Objects.hash(studentId);

}

@Override

public String toString() {

return studentId + "," + fullName + "," + address + "," + major + "," + gpa + '\n';

}

}

* Lớp Lesson132:

package net.braniumacademy.chapter13.lesson1;

import java.io.BufferedWriter;

import java.io.FileWriter;

import java.io.IOException;

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

public class Lesson132 {

public static void main(String[] args) {

int choice;

Scanner scanner = new Scanner(System.in);

List<Student> students = new ArrayList<>();

do {

System.out.println("================ MENU ================");

System.out.println("1. Thêm mới sinh viên.");

System.out.println("2. Hiển thị danh sách sinh viên.");

System.out.println("3. Ghi danh sách sinh viên vào file.");

System.out.println("0. Thoát chương trình.");

choice = scanner.nextInt();

scanner.nextLine(); // đọc bỏ dòng thừa

switch (choice) {

case 0:

System.out.println("==> Xin cảm ơn! <==");

break;

case 1:

var student = createStudent();

students.add(student);

break;

case 2:

showStudents(students);

break;

case 3:

System.out.println("Nhập tên file để lưu dữ liệu: ");

var fileName = scanner.nextLine();

writeDataToFile(students, fileName);

break;

default:

System.out.println("==> Sai chức năng, vui lòng chọn lại! <==");

break;

}

} while (choice != 0);

}

private static Student createStudent() {

Scanner scanner = new Scanner(System.in);

System.out.println("Nhập mã sinh viên: ");

var id = scanner.nextLine();

System.out.println("Nhập họ và tên sinh viên: ");

var fullName = scanner.nextLine();

System.out.println("Nhập địa chỉ của sinh viên: ");

var address = scanner.nextLine();

System.out.println("Nhập chuyên ngành của sinh viên: ");

var major = scanner.nextLine();

System.out.println("Nhập điểm gpa của sinh viên: ");

var gpa = scanner.nextFloat();

return new Student(id, fullName, address, major, gpa);

}

private static void showStudents(List<Student> students) {

System.out.println("==> DANH SÁCH SINH VIÊN <==");

System.out.printf("%-10s%-30s%-20s%-20s%-10s\n",

"Mã SV", "Họ tên", "Địa chỉ", "Chuyên ngành", "Điểm GPA");

for (var student : students) {

System.out.printf("%-10s%-30s%-20s%-20s%-10.2f\n",

student.getStudentId(), student.getFullName(),

student.getAddress(), student.getMajor(), student.getGpa());

}

}

private static void writeDataToFile(List<Student> students, String fileName) {

try (FileWriter fileWriter = new FileWriter(fileName);

BufferedWriter writer = new BufferedWriter(fileWriter)) {

for (var student : students) {

writer.write(student.toString());

}

} catch (IOException ex) {

System.out.println(ex.getMessage());

}

}

}

* Input nhập từ bàn phím:

================ MENU ================

1. Thêm mới sinh viên.

2. Hiển thị danh sách sinh viên.

3. Ghi danh sách sinh viên vào file.

0. Thoát chương trình.

1

Nhập mã sinh viên:

SV1001

Nhập họ và tên sinh viên:

Hoàng Văn Tuấn

Nhập địa chỉ của sinh viên:

Hà Nội

Nhập chuyên ngành của sinh viên:

CNTT

Nhập điểm gpa của sinh viên:

3.66

================ MENU ================

1. Thêm mới sinh viên.

2. Hiển thị danh sách sinh viên.

3. Ghi danh sách sinh viên vào file.

0. Thoát chương trình.

1

Nhập mã sinh viên:

SV1002

Nhập họ và tên sinh viên:

Lê Trần Công Minh

Nhập địa chỉ của sinh viên:

Hồ Chí Minh

Nhập chuyên ngành của sinh viên:

CNTT

Nhập điểm gpa của sinh viên:

3.54

================ MENU ================

1. Thêm mới sinh viên.

2. Hiển thị danh sách sinh viên.

3. Ghi danh sách sinh viên vào file.

0. Thoát chương trình.

1

Nhập mã sinh viên:

SV1003

Nhập họ và tên sinh viên:

Nguyễn Thị Ngọc Linh

Nhập địa chỉ của sinh viên:

Đà Nẵng

Nhập chuyên ngành của sinh viên:

CNTT

Nhập điểm gpa của sinh viên:

3.56

================ MENU ================

1. Thêm mới sinh viên.

2. Hiển thị danh sách sinh viên.

3. Ghi danh sách sinh viên vào file.

0. Thoát chương trình.

2

==> DANH SÁCH SINH VIÊN <==

Mã SV Họ tên Địa chỉ Chuyên ngành Điểm GPA

SV1001 Hoàng Văn Tuấn Hà Nội CNTT 3.66

SV1002 Lê Trần Công Minh Hồ Chí Minh CNTT 3.54

SV1003 Nguyễn Thị Ngọc Linh Đà Nẵng CNTT 3.56

================ MENU ================

1. Thêm mới sinh viên.

2. Hiển thị danh sách sinh viên.

3. Ghi danh sách sinh viên vào file.

0. Thoát chương trình.

3

Nhập tên file để lưu dữ liệu:

test.txt

================ MENU ================

1. Thêm mới sinh viên.

2. Hiển thị danh sách sinh viên.

3. Ghi danh sách sinh viên vào file.

0. Thoát chương trình.

0

==> Xin cảm ơn! <==

* File kết quả:

SV1001,Hoàng Văn Tuấn,Hà Nội,CNTT,3.66

SV1002,Lê Trần Công Minh,Hồ Chí Minh,CNTT,3.54

SV1003,Nguyễn Thị Ngọc Linh,Đà Nẵng,CNTT,3.56