Source tiếng Anh: https://www.freecodecamp.org/news/the-java-handbook/
Link cài đặt Java cho Vscode: https://www.youtube.com/watch?v=BB0gZFpukJU

Java Course Note

Các phần trong bài:

1. Variable
2. Operator
3. Conditional Statement
4. Array
5. Hashmap
6. Vòng lặp
7. Class và Object

Những phần chưa đề cập đến trong bài này:

1. Chưa đi sâu về 4 tính chất OOP trong Java
2. Design Pattern trong Java
3. Các loại vòng lặp khác: Do-While, For-each
4. Switch-case
5. Các từ khóa như: static, String,

1. Variable

a. Làm việc với biến

```
public class Main {

	public static void main(String[] args) {
		// <type> <name>
		int age;

		// <name> = <value>
		age = 27;

		// prints the age on the terminal
		System.out.println("I am " + age + " years old.");

	}

}
```

Khi định dạng biến trong Java, ta cần khai báo loại dữ liệu biến và tên biến.

Ví dụ ở dòng int age, ta khai báo biến age có kiểu dữ liệu là int. Sau đó gán giá trị 27 vào biến age, và in ra màn hình.

Tuy nhiên, nếu ta comment dòng gán giá trị: age=27. Thì chương trình sẽ in lỗi

```
Exception in thread "main" java.lang.Error: Unresolved compilation
problem:
	The local variable age may not have been initialized

	at variables.Main.main(Main.java:13)
```

Cho nên nếu như ta cần phải gán biến liền sau khi khởi tạo, ta có thể gán ngay khi khởi tạo

```
public class Main {

	public static void main(String[] args) {
		// <type> <name> = <value>
		int age = 27;

		// prints the age on the terminal
		System.out.println("I am " + age + " years old.");

	}

}
```

Nhưng lưu ý rằng, ta không thể khởi tạo một biến hai lần trong Java

```
public class Main {

	public static void main(String[] args) {
		// <type> <name> = <value>
		int age = 27;

		int age = 28;

		// prints the age on the terminal
		System.out.println("I am " + age + " years old.");

	}

}
```

```
Exception in thread "main" java.lang.Error: Unresolved compilation
problem:
	Duplicate local variable age
```

```
	at variables.Main.main(Main.java:9)
```

Và khi ta muốn định dạng một hằng số (không cho phép thay đổi giá trị) thì ta dùng từ khóa *final* để khởi tạo

```
public class Main {

	public static void main(String[] args) {
		// final <type> <name> = <value>
		final int age = 27;

		age = 28;

		System.out.println("I am " + age + " years old.");

	}

}
```

```
Exception in thread "main" java.lang.Error: Unresolved compilation
problem:
	The final local variable age cannot be assigned. It must be blank
and not using a compound assignment

	at variables.Main.main(Main.java:9)
```

b. Primitive Data Types trong Java

Primitive Data Types là kiểu dữ liệu cơ bản là những kiểu dữ liệu: có sẵn, được viết thường.

TYPE	EXPLANATION
`byte`	8-bit signed integer within the range of -128 to 127
`short`	16-bit signed integer within the range of -32,768 to 32,767
`int`	32-bit signed integer within the range of -2147483648 to 2147483647
`long`	64-bit signed integer within the range of -9223372036854775808 to 922337203685477
`float`	single-precision 32-bit floating point within the range of 1.4E-45 to 3.4028235E38
`double`	double-precision 64-bit floating point within the range of 4.9E-324 to 1.7976931348623
`boolean`	It can be either `true` or `false`
`char`	single 16-bit Unicode character within the range of `\u0000` (or 0) to `\uffff` (or 65,535

Tổng kết, ta có 4 kiểu dữ liệu cho số nguyên, 2 cho số thực, 1 cho boolean và 1 cho ký tự. Java có nhiều kiểu dữ liệu cho số nguyên và số thực giúp người dùng kiểm soát memory sử dụng cho từng biến qua đó tối ưu bộ nhớ cho chương trình.

```
public class App {
    public static void main(String[] args) throws Exception {
        double gg = 4; // 4.0
        double gpe = 4.8; //4.8
        float gpa = 4.8f; //4.8

        System.out.println(gpe+" " + gpa+" "+gg);
    }
}
```

1. Khi ta khai báo biến *gg* là kiểu dữ liệu *double* thì giá trị sẽ có số thập phân, dù ta không gán
2. Khi ta khai báo biến *gpa* là kiểu dữ liệu *float* thì phải có thêm ký tự *f* hoặc *F*

```
public class Main {

	public static void main(String[] args) {
		boolean isWeekend = false;

		System.out.println(isWeekend); // false

	}
}
```

biến isWeekend có thể mang giá trị *true* hoặc *false.*

```
public class Main {

	public static void main(String[] args) {
		char percentSign = '%';

		System.out.println(percentSign); // %

	}
}
```

biến percentSign có thể mang bất kỳ ký tự Unicode nào.

c. Wrapper Classes (Lớp bao bọc)

PRIMITIVE TYPE	WRAPPER CLASS
`int`	`Integer`
`long`	`Long`
`short`	`Short`
`byte`	`Byte`
`boolean`	`Boolean`
`char`	`Character`
`float`	`Float`
`double`	`Double`

```java
public class Main {
    public static void main (String[] args) {
        Integer age = 27;
        Double gpa = 4.8;

        System.out.println(age); // 27
        System.out.println(gpa); // 4.8
    }
}
```

Lớp bao bọc là lớp class tương ứng với các data type cơ bản.
Khi sử dụng lớp bao bọc, ta sẽ có thể dùng thêm những function xây dựng cho lớp đó. Việc sử dụng lớp bao bọc sẽ được đề cập ở bài Mảng động ở dưới.

2. Operator (Toán tử)

a. **Arithmetic Operator (Toán tử số học)**

Toán tử số học là toán tử như cộng, trừ, nhân, chia giữa các biến hoặc giá trị với nhau.

OPERATOR	OPERATION
`+`	Addition
`-`	Subtraction
`*`	Multiplication
`/`	Division
`%`	Remainder (Modulo/Modulus)

```
public class Main {

	public static void main(String[] args) {
		int number1 = 10;
		int number2 = 5;

		System.out.println(number1 + number2); // 15
		System.out.println(number1 - number2); // 5
		System.out.println(number1 * number2); // 50
		System.out.println(number1 / number2); // 2
		System.out.println(number1 % number2); // 0

	}

}
```

```
public class Main {

	public static void main(String[] args) {
		int number1 = 8;
		int number2 = 5;

		System.out.println(number1 / number2); // 1
	}

}
```

Ta thấy rằng, 8/5 = 1.6 nhưng giá trị output chỉ ra là 1. Vì ta đang định dạng hai biến *number1* và *number2* là dạng int. Để khắc phục, ta cần định dạng hai biến về *double*

```java
public class Main {

	public static void main(String[] args) {
		double number1 = 8;
		double number2 = 5;

		System.out.println(number1 / number2); // 1.6
	}

}
```

b. Assignment Operator (Toán tử gán)

OPERATOR	OPERATION	EQUIVALENT TO
`+=`	`a += b`	`a = a + b`
`-=`	`a -= b`	`a = a - b`
`*=`	`a *= b`	`a = a * b`
`/=`	`a /= b`	`a = a / b`
`%=`	`a %= b`	`a = a % b`

c. Relational Operator (Toán tử quan hệ)

OPERATOR	EXPLANATION	USAGE
`==`	Is Equal To	`5 == 8` returns `false`
`!=`	Is Not Equal To	`5 != 8` returns `true`
`>`	Is Greater Than	`5 > 8` returns `false`
`<`	Is Less Than	`5 < 8` returns `true`
`>=`	Greater Than or Equal To	`5 >= 8` returns `false`
`<=`	Less Than or Equal To	`5 <= 8` returns `true`

```java
public class Main {

	public static void main(String[] args) {
		double number1 = 10;
		double number2 = 5;

		System.out.println(number1 == number2); // false
		System.out.println(number1 != number2); // true
		System.out.println(number1 > number2); // true
		System.out.println(number1 < number2); // false
		System.out.println(number1 >= number2); // true
		System.out.println(number1 <= number2); // false
	}

}
```

d. Logical Operator (Toán tử logic)

OPERATOR	USAGE	EXPLANATION
Logical And (&&)	`age >= 18 && age <= 40`	Evaluates to true, only if both conditions are true
Logical Or (||)	`isSchoolStudent || isLibraryMember`	Evaluates to true if one of the two or both conditions are true
Not (!)	`!isLibraryMember`	Evaluates to false if the inner condition evaluates to true and vise versa

```java
public class Main {

	public static void main(String[] args) {
		int age = 20;

		System.out.println(age >= 18 && age <= 40); // true
	}
```

```
}
```

e. **Unary Operator**

OPERATOR	EXPLANATION
Increment (`++`)	Increments a given value by 1
Decrement (`--`)	Decrements a given value by 1

```
public class Main {

	public static void main(String[] args) {
		int score = 95;
		int turns = 11;

		score++;
		turns--;

		System.out.println(score); // 96
		System.out.println(turns); // 10
	}

}
```

Dấu ++ / - - có thể nằm ở trước hoặc sau giá trị biến. Và vị trí nằm sẽ mang lại kết quả khác nhau

```
public class Main {

	public static void main(String[] args) {
		int score = 95;


		System.out.println(++score); // 96
		System.out.println(score); // 96
	}

}
```

```
public class Main {

	public static void main(String[] args) {
		int score = 95;
```

```
            System.out.println(score++); // 95
            System.out.println(score); // 96
      }

}
```

Với ++ / - - trước thì giá trị của biến sẽ được ưu tiên cập nhất trước hết, rồi mới sử dụng biến.
Với ++/ - - sau thì giá trị của biến sẽ thay đổi khi biến đó được sử dụng xong.

3. Conditional Statement (Điều kiện)

```
public class Main {

      public static void main(String[] args) {
            int age = 20;

            // if (condition) {...}
            if (age >= 18 && age <= 40) {
                  System.out.println("you can use the program");
            }

      }

}
```

Nếu muốn set thêm điều kiện ngược lại, thì ta dùng thêm từ khóa else

```
public class Main {

      public static void main(String[] args) {
            int age = 20;

            if (age >= 18 && age <= 40) {
                  System.out.println("you can use the program");
            } else {
                  System.out.println("you can not use the program");
            }

      }

}
```

Và nếu muốn set nhiều điều kiện hơn, thì với mỗi điều kiện mới, ta dùng từ khóa else if

```
public class Main {
```

```
	public static void main(String[] args) {
		int age = 50;
		boolean isSchoolStudent = true;
		boolean isLibraryMember = false;

		// if (condition) {...}
		if (age >= 18 && age <= 40) {
			System.out.println("you can use the program");
		} else if (isSchoolStudent || isLibraryMember) {
			System.out.println("you can use the program for a short
time");
		} else {
			System.out.println("you can not use the program");
		}

	}

}
```

4. Array (Mảng)

Trong Java, mảng là một biến có thể lưu nhiều giá trị, và sử dụng index để truy xuất từng giá trị trong biến đó.

1. **Mảng tĩnh (Static Array):** Là mảng mà ta không thể tăng dung lượng kích thước chứa của biến lên sau khi đã khởi tạo.

```
public class Main {

	public static void main(String[] args) {
		char vowels[] = new char[5];

		vowels[0] = 'a';
		vowels[1] = 'e';
		vowels[2] = 'i';
		vowels[3] = 'o';
		vowels[4] = 'u';

	}
}
```

2. **Mảng động (Dynamic Array)**: Là mảng mà ta có thể mở rộng kích thước chứa của biến lên.

```
public class Main {
    public static void main (String[] args) {
        ArrayList<Integer> oddNumbers = new ArrayList<>();

        oddNumbers.add(1);
        oddNumbers.add(3);
```

```
        oddNumbers.add(5);
        oddNumbers.add(7);
        oddNumbers.add(9);

        System.out.println(oddNumbers.toString()); // [1, 3, 5, 7, 9]
    }
}
```

Ở đây, ta tạo ra một mảng động thuộc class Integer (là dạng class bao bọc của kiểu dữ liệu int). Sau đó dùng hàm add() để add thêm giá trị vào mảng. Và tương tự như vậy, dùng hàm remove(index muốn bỏ) để bỏ giá trị dựa trên index ra khỏi mảng.
Ngoài ra thì ArrayList hỗ trợ nhiều hàm khác, chi tiết ở link:
https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html

5. Hashmap (dạng Dictionary trong Python)

```
public class Main {
    public static void main (String[] args) {
        HashMap<String, Double> prices = new HashMap<>();

        prices.put("apple", 2.0);
        prices.put("orange", 1.8);
        prices.put("guava", 1.5);
        prices.put("berry", 2.5);
        prices.put("banana", 1.0);

        System.out.printf(prices.toString()); // {orange=1.8, banana=1.0,
apple=2.0, berry=2.5, guava=1.5}
    }
}
```

Cũng giống như tất cả các dạng dữ liệu khác, HashMap cũng phải định dạng kiểu dữ liệu của nó. Và hơn thế nữa, phải định dạng kiểu dữ liệu cho cả key và value. Cũng giống như ArrayList, Hashmap cũng có nhiều hàm hỗ trợ đi kèm. Chi tiết ở link dưới đây:

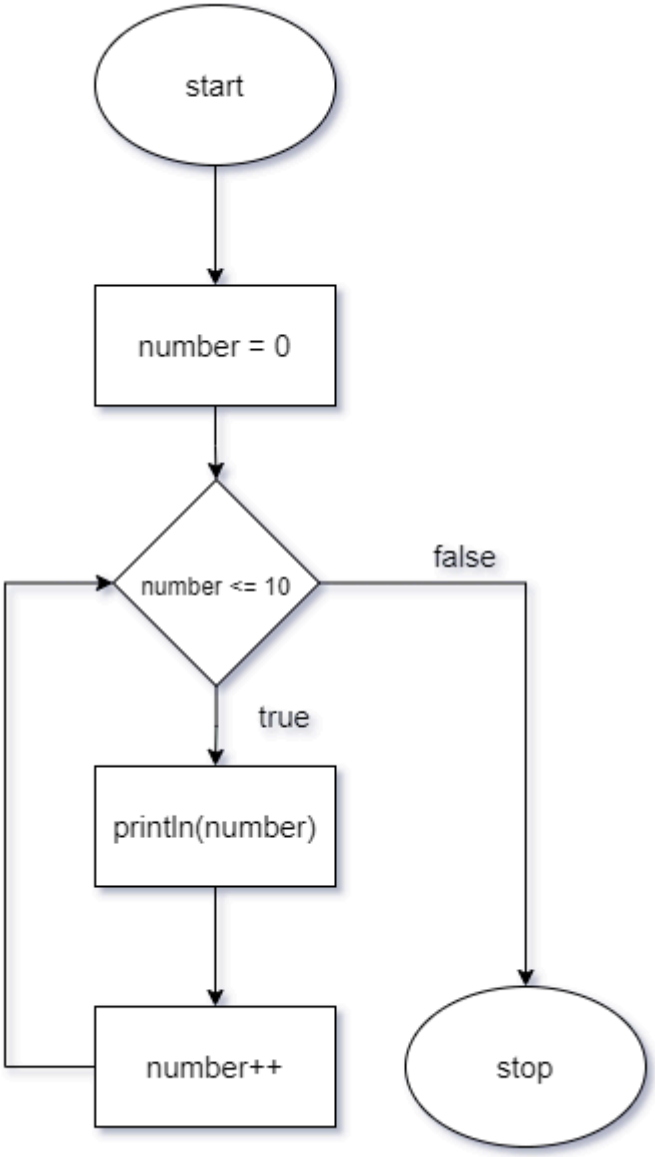
https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html

6. Vòng lặp:

a. **For loop:**

```
public class Main {

	public static void main(String[] args) {
		for (int number = 0; number <= 10; number++) {
			System.out.println(number);
		}
	}
}
```



Flowchart của đoạn code trên được mô tả như trên.

```
public class Main {

	public static void main(String[] args) {
		int fibonacciNumbers[] = {0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55};

		for(int index = 0; index < fibonacciNumbers.length; index++) {
			System.out.println(fibonacciNumbers[index]);
		}
	}
}
```

Ta muốn chạy vòng lặp lên mảng bằng cách dùng từ khóa *length* để lấy số lượng giá trị có trong mảng.

b. While loop:

```
public class Main {

	public static void main(String[] args) {
		int number = 5;
		int multiplier = 1;

		while (multiplier <= 10) {
			System.out.println(String.format("%d x %d = %d",
number, multiplier, number*multiplier));

			multiplier++;
		}
	}
}
```

Đối chiếu với for loop, thì ta có thêm code của while loop.

7. Class và Object (Lớp và đối tượng) trong Java

a. Giới thiệu cơ bản

Một class sẽ là bản khung nền của object. Ví dụ class Car gồm những thành phần A,B,C thì tất cả các đối tượng từ class Car đều có các thành phần A,B,C nhưng có thể sẽ khác giá trị. Các thành phần A,B,C sẽ có 2 loại: attribute và method

```
public class User {
    String name;
    LocalDate birthDay;

    ArrayList<String> borrowedBooks = new ArrayList<String>();

    int age() {
        return Period.between(this.birthDay, LocalDate.now()).getYears();
    }

    void borrow(String bookTitle) {
        this.borrowedBooks.add(bookTitle);
    }
}
```

Ở đây chúng ta định dạng các attribute cho lớp User gồm: name, birthDay, borrowedBooks. Và method gồm: age, borrow. Về mặt bản chất, attribute sẽ tương tự variable (biến) và method sẽ tượng tự hàm.

b. Access Modifier (Quyền truy cập)

Public	Accessible everywhere
Private	Accessible within the class
Protected	Accessible within the class and subclasses

Khi ta xác định các thành phần trong lớp, để kiểm soát phạm vi truy cập, thì ta sẽ sử dụng thêm những 4 từ khóa trên để kiểm soát.

1. Phạm vi Private

```
class A {
    private int data = 40;
```

```
    private void msg() {
        System.out.println("Hello java "+ data);
    }
}

public class Simple {
    public static void main(String args[]) {
        A obj = new A();
        System.out.println(obj.data);// Compile Time Error
        obj.msg();// Compile Time Error
    }
}
```

Với attribute data và method msg được định dạng là private. Thì ta không thể gọi ở ngoài khuôn khổ của class A đó.

2. Phạm vi Protected

```
public class Parent {
    protected int protectedVariable;
    protected void protectedMethod() {
        // code here
    }
}

class Child extends Parent {
    void accessProtected() {
        protectedVariable = 10; // có thể truy cập từ subclass
        protectedMethod();      // có thể gọi từ subclass
    }
}
```

Hai thành phần protectedVariable và protectedMethod đều thuộc dạng protected, cho nên nó chỉ được truy cập khi lớp kế thừa của lớp đó gọi.

3. Phạm vi Public

Là các thành phần được gọi thoải mái ở khắp mọi nơi trong code.

```java
// Lưu file với tên A.java
package vn.viettuts.demo;

public class A {
    public void msg() {
        System.out.println("Hello");
    }
}
```

```java
// Lưu file với tên B.java
package vn.viettuts.mypack;

import vn.viettuts.demo.*;

public class B {
    public static void main(String args[]) {
        A obj = new A();
        obj.msg();
    }
}
```