Lập trình JAVA cơ bản

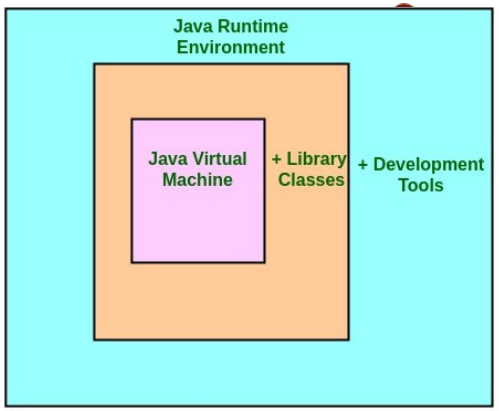
Tuần 1

Giảng viên: Trần Đức Minh

Nội dung bài giảng

- Giới thiệu về Java
- Chương trình Java đầu tiên
- Giới thiệu về biến, các kiểu dữ liệu.
- Các cấu trúc điều khiển.
- Nhập dữ liệu.
- Dữ liệu kiểu mảng

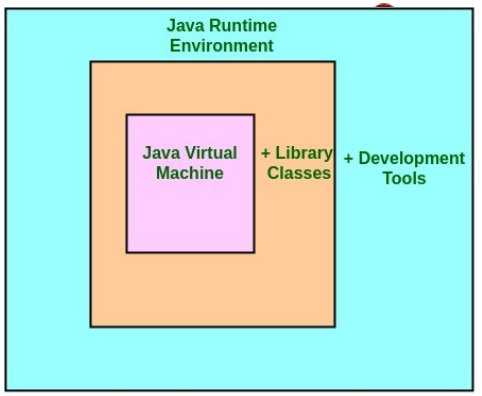
Giới thiệu Java



JDK = JRE + Development Tool
JRE = JVM + Library Classes

- JDK, JRE và JVM
  - **JVM (Java Virtual Machine)**: Máy ảo Java
    - Quản lý bộ nhớ và cung cấp môi trường thực thi cho ứng dụng Java
    - Cụ thể, được dùng để chạy mã **Java bytecode** (là mã được biên dịch từ mã nguồn lập trình)
    - Mỗi hệ điều hành cần một JVM khác nhau
  - **JRE (Java Runtime Environment)** bao gồm **JVM** và tập hợp **các thư viện hỗ trợ cho JVM** thực thi ứng dụng Java.

Giới thiệu Java

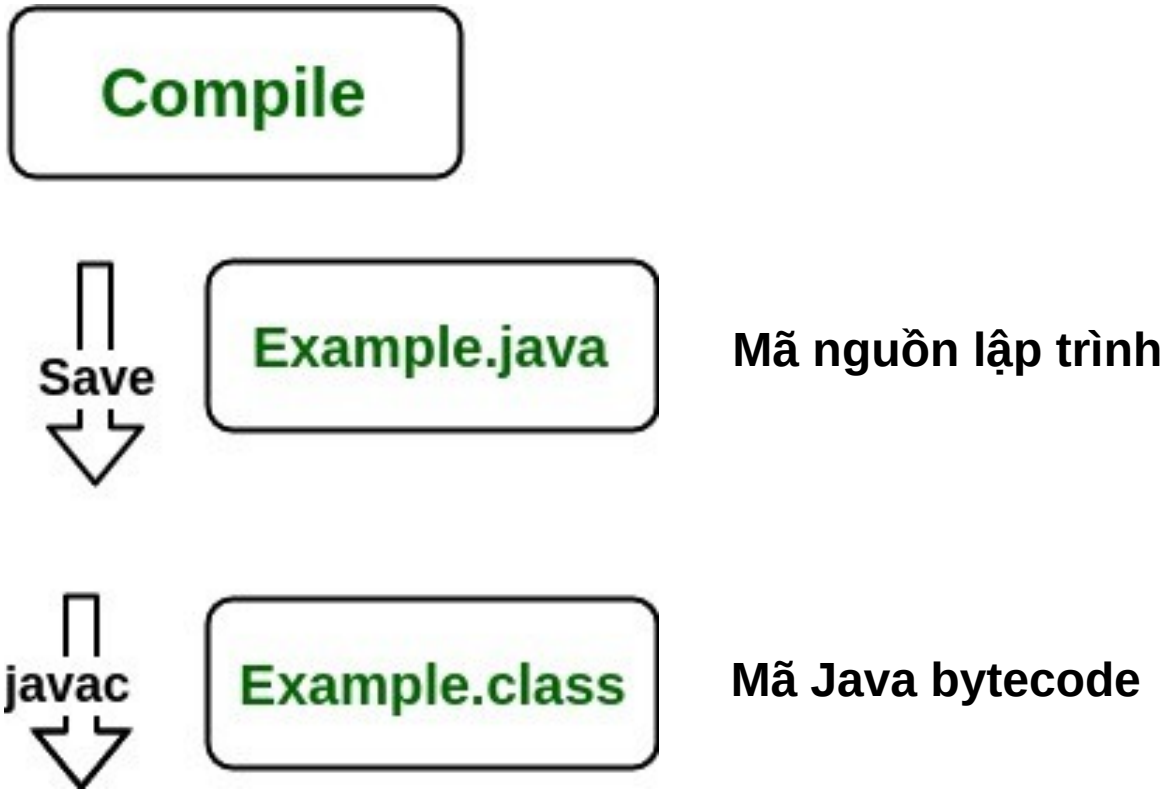


JDK = JRE + Development Tool
JRE = JVM + Library Classes

- JDK, JRE và JVM
  - **JDK (Java Development Kit)** là bộ phần mềm cung cấp **môi trường phát triển ứng dụng** cho ngôn ngữ Java.
    - JDK bao gồm **JRE** và **các công cụ để hỗ trợ phát triển ứng dụng** viết bằng ngôn ngữ Java.
    - Trong JDK có công cụ **Java compiler** để biên dịch các **file mã nguồn Java (.java)** thành các **file Java bytecode (.class)**

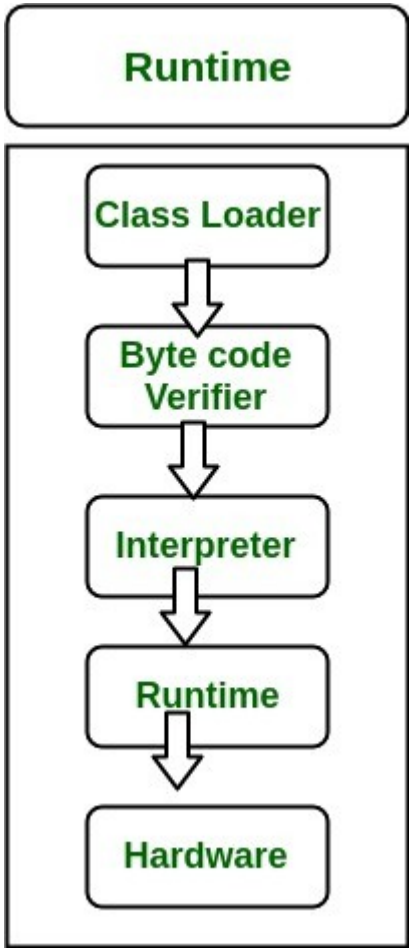
Giới thiệu Java

- Biên dịch chương trình trong Java



Giới thiệu Java

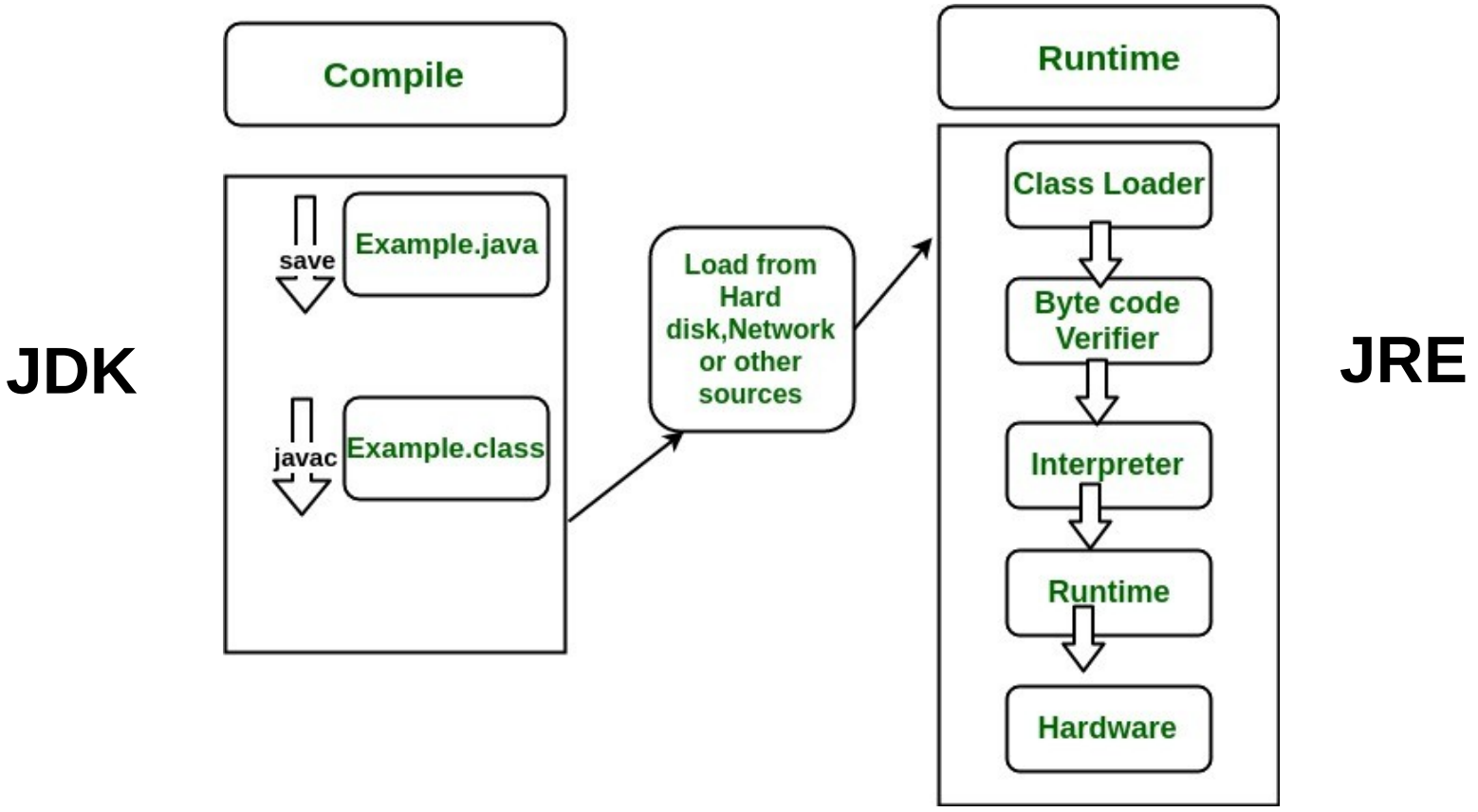
- Chạy mã bytecode trong Java



JRE

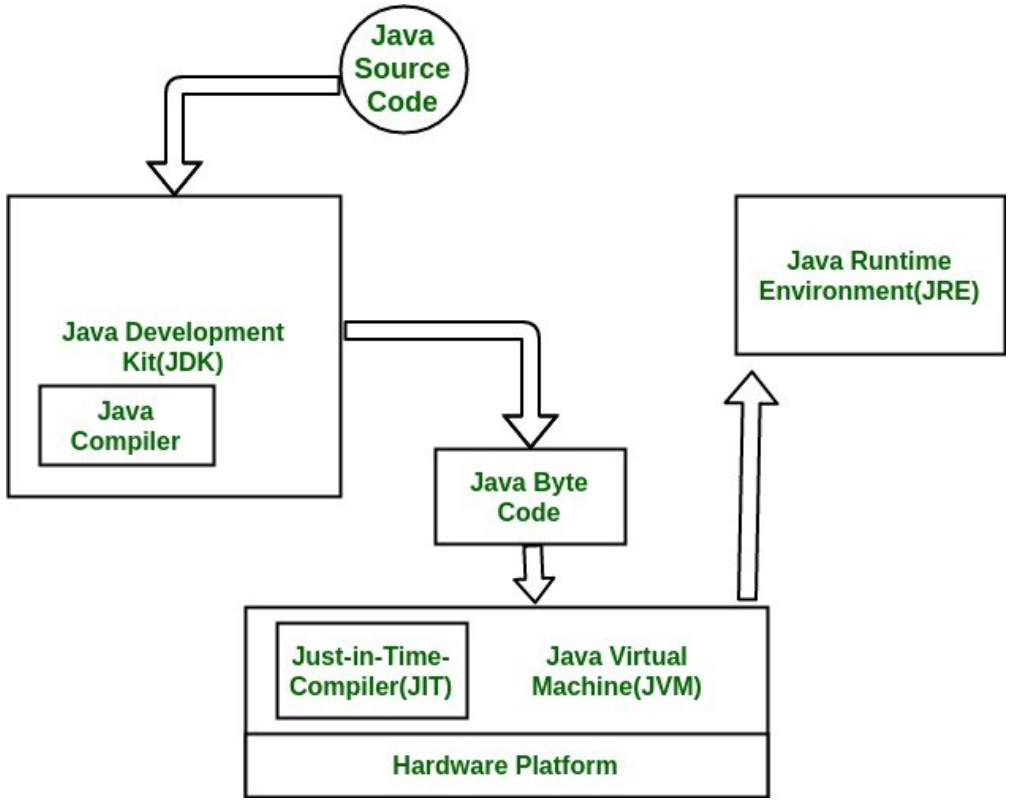
Giới thiệu Java

- Mô hình tổng quát

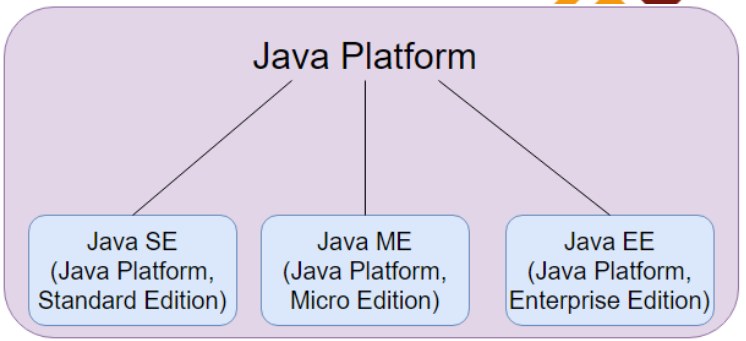


Giới thiệu tổng quát

- Mô hình tương tác giữa JDK và JRE
  - **JIT (Just In Time)** : Thông dịch mã bytecode thành mã máy.



Các nền tảng Java



- **Java SE (Standard Edition)**
  - Đây là Java Core, là phiên bản chuẩn được dùng làm nền tảng cho các phiên bản Java khác.
  - Được dùng để xây dựng với các ứng dụng trên Desktop.
- **Java ME (Micro Edition)**
  - Phiên bản này được dùng để tạo ứng dụng chạy trên các hệ thống nhúng (điện thoại, thiết bị điện tử)
  - Ngoài việc sử dụng các thư viện của Java SE, Java ME cũng có nhiều thư viện của riêng nó.
- **Java EE (Enterprise Edition)**
  - Phiên bản này được dùng để phát triển các ứng dụng Web
  - Ngoài việc sử dụng các công nghệ của Java SE, Java EE cũng có nhiều công nghệ của riêng nó như JSP/Servlet, JavaBeans.

Các phiên bản

Version	Release date	End of Free Public Updates[1][4][5][6]	Extended Support Until
JDK Beta	1995	?	?
JDK 1.0	January 1996	?	?
JDK 1.1	February 1997	?	?
J2SE 1.2	December 1998	?	?
J2SE 1.3	May 2000	?	?
J2SE 1.4	February 2002	October 2008	February 2013
J2SE 5.0	September 2004	November 2009	April 2015
Java SE 6	December 2006	April 2013	December 2018
Java SE 7	July 2011	April 2015	July 2022
Java SE 8 (LTS)	March 2014	**January 2019 for Oracle (commercial)** Indefinitely for Oracle (personal use) December 2030 for Zulu At least May 2026 for AdoptOpenJDK At least May 2026 for Amazon Corretto	December 2030
Java SE 9	September 2017	March 2018 for OpenJDK	N/A
Java SE 10	March 2018	September 2018 for OpenJDK	N/A
Java SE 11 (LTS)	September 2018	September 2027 for Zulu At least October 2024 for AdoptOpenJDK At least September 2027 for Amazon Corretto	September 2026
Java SE 12	March 2019	September 2019 for OpenJDK	N/A
Java SE 13	September 2019	March 2020 for OpenJDK	N/A
Java SE 14	March 2020	September 2020 for OpenJDK	N/A
Java SE 15	September 2020	March 2021 for OpenJDK	N/A
Java SE 16	March 2021	September 2021 for OpenJDK	N/A
Java SE 17 (LTS)	September 2021	September 2030 for Zulu	TBA

Legend: Old version | Older version, still maintained | **Latest version** | Future release

Chương trình đầu tiên

```java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Lớp lập trình Java cơ bản !!!");
    }
}
```

- Các từ khóa trong Java là **case-sensitive**.
- **public** là từ khóa **xác định phạm vi truy cập**.
- **class** là từ khóa dùng để định nghĩa một lớp.
- Tên lớp (**HelloWorld**) phải trùng với tên của file chứa lớp (**HelloWorld.java**).
- Phần thân của lớp chứa các thuộc tính và phương thức đều nằm giữa cặp dấu ngoặc nhọn **{ ... }**.

Chương trình đầu tiên

```java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Lớp lập trình Java cơ bản !!!");
    }
}
```

- Điểm bắt đầu của chương trình là phương thức **main**.
- **public** là từ khóa **xác định phạm vi truy cập** của phương thức main.
- **static** là từ khóa cho phép truy cập phương thức main từ tên lớp mà không cần thiết tạo đối tượng lớp.
- **void** là từ khóa để xác định phương thức không có giá trị trả về.
- Tham số **String[] args** chứa thông tin được gửi từ bên ngoài vào chương trình.
- Phần thân của phương thức main được chứa bên trong **code block** (cặp dấu ngoặc nhọn **{ … }**).
- Lệnh **System.out.println(…)** được sử dụng để đưa thông tin ra màn hình.

Biến trong Java

- **Biến** (variable) là một cách để lưu trữ thông tin trong khi lập trình.
- Có nhiều **kiểu dữ liệu** (Data types) khác nhau được dùng để định nghĩa cho biến.
- Để **định nghĩa một biến**, ta cần xác định kiểu dữ liệu của biến, tên biến và có thể một giá trị khởi tạo nào đó cho biến (nếu cần).

```
int nonFirstValue;

int myFirstVariable = 10;

System.out.println(myFirstVariable);
```

Biểu thức

- Toán tử cơ bản: + - * / %
- Phân tích chương trình sau:

```java
public class HelloWorld {
    public static void main(String[] args) {
        int myFirstVariable = 5;
        int mySecondVariable = 5 * 2;
        int myThirdVariable = (10 + myFirstVariable) / (mySecondVariable - 1);
        System.out.println(myThirdVariable);
    }
}
```

Các kiểu dữ liệu nguyên thủy

- Java có 8 kiểu dữ liệu nguyên thủy (primitives)
  - **byte**: số nguyên 1 byte, gồm cả số âm.
  - **short**: số nguyên 2 bytes, gồm cả số âm.
  - **int**: số nguyên 4 bytes, gồm cả số âm.
  - **long**: số nguyên 8 bytes, gồm cả số âm.
  - **float**: số thực 4 bytes.
    - Khoảng giá trị từ 1.4E-45 đến 3.4028235E+38
  - **double**: số thực 8 bytes.
    - Khoảng giá trị từ 4.9E-324 đến 1.7976931348623157E+308
  - **char**: chiếm 2 bytes trong bộ nhớ, lưu trữ ký tự Unicode và cũng có thể lưu trữ một số nguyên không âm với khoảng giá trị từ 0 đến 65535.
  - **boolean:** chiếm 1 byte trong bộ nhớ.

Số nguyên kiểu int

- Java gán mặc định kiểu dữ liệu **int** cho một chữ số nguyên bên trong biểu thức.
  - Ví dụ: a = 9 + b;
    - Số 9 ở đây là một chữ số nguyên và được hệ thống coi có kiểu dữ liệu là **int**.
- Java cho phép sử dung dấu gạch dưới (_) để phân chia dãy chữ số cho dễ nhìn.
  - Ví dụ: 1_234_567 tương đương với số 1234567.
- Số nguyên kiểu **int** trong Java chiếm 4 **bytes** bên trong bộ nhớ.

```
public class HelloWorld {
    public static void main(String[] args) {
        int minIntValue = Integer.MIN_VALUE;
        int maxIntValue = Integer.MAX_VALUE;

        System.out.println(minIntValue);
        System.out.println(maxIntValue);
    }
}
```

- Java sử dụng **Wrapper class** (lớp bọc) để giúp ta thực hiện các thao tác trên các kiểu dữ liệu cơ bản.
  - Ví dụ: Lớp bọc có tên **Integer** được dùng cho **kiểu dữ liệu int**.

Tràn số trên và tràn số dưới

- **Tràn số trên** (Overflow) nếu ta cố đưa vào một giá trị **lớn hơn giá trị lớn nhất** của kiểu dữ liệu.

```
int overflowNumber1 = Integer.MAX_VALUE + 1;
int overflowNumber2 = 2147483648;
System.out.println(overflowNumber1);
```

- **Tràn số dưới** (Underflow) nếu ta cố đưa vào một giá trị **nhỏ hơn giá trị nhỏ nhất** của kiểu dữ liệu.

```
int underflowNumber1 = Integer.MIN_VALUE – 1;
int myIntNumber = -2_147_483_648;
System.out.println(underflowNumber);
```

Số nguyên kiểu byte, short, long

```
byte myMaxByte = Byte.MAX_VALUE;
byte myMinByte = Byte.MIN_VALUE;

short myMaxShort = Short.MAX_VALUE;
short myMinShort = Short.MIN_VALUE;

long myLongNumber = 100L; // Tạo vùng nhớ 8 bytes
long myLongError = 2_147_483_647_234;
long myMaxLong = Long.MAX_VALUE;
long myMinLong = Long.MIN_VALUE;
```

Ép kiểu

- Phân tích đoạn code sau:

```
1. int myIntNumber = 52;
2. byte myByteValue = myIntNumber;
3. short myShortValue = myIntNumber;
4. long myLongValue = myIntNumber;
```

  - **Dòng 2 và 3 bị lỗi** do kiểu không tương thích (có thể bị mất giá trị sau khi tính toán) do kích cỡ của kiểu dữ liệu byte và short nhỏ hơn int.
- **Ép kiểu** (casting) được dùng để chuyển đổi kiểu dữ liệu của một số sang một kiểu dữ liệu khác.

```
byte myByteValue = (byte) (myIntNumber);
```

Số thực

- Java gán mặc định kiểu dữ liệu **double** cho một chữ số thực bên trong biểu thức.
  - Ví dụ: a = 9.2 + b;
    - Số 9.2 ở đây là một chữ số thực và được hệ thống coi có kiểu dữ liệu là **double**.
- Phân tích đoạn code:

  1. float myFloatValue = 7.5;

  2. double myDoubleValue = 6.7;

  - **Dòng 1 bị lỗi** do kiểu không tương thích (có thể bị mất giá trị sau khi tính toán) do kích cỡ của kiểu dữ liệu float nhỏ hơn double.
- Khởi tạo giá trị

  float myFloatValue1 = 7.5**f**;

  float myFloatValue2 = (float) 7.5;

  double myDoubleValue = 6.7**d**;

Số thực

- Chạy thử và phân tích kết quả đoạn code

```
int myIntValue = 7 / 3;
float myFloatValue = 7f / 3f;
double myDoubleValue = 7d / 3d;
```

- Chạy thử và phân tích từng câu lệnh bên dưới

```
float myFloatValue1 = 7 / 3;
float myFloatValue2 = 7f / 3;
float myFloatValue3 = 7f / 3.0;
double myDoubleValue = 7d / 3f;
```

Kiểu ký tự và kiểu logic

- Kiểu ký tự có thể lưu trữ các ký tự Unicode

  char myChar = **‘ộ’**;

  char myUnicodeChar = **‘\uCB04’**;

- **Unicode** là một chuẩn mã hóa ký tự quốc tế dành cho nhiều ngôn ngữ khác nhau trên thế giới. Do Unicode có thể mã hóa lên đến 65536 ký tự.
- Kiểu logic chỉ có 2 giá trị **true** hoặc **false**

  boolean myBooleanValue = **true**;

Kiểu dữ liệu String

- **String** là một kiểu dữ liệu trong Java, nhưng không phải kiểu dữ liệu cơ bản, nó là một lớp.

  String myString = “Chào các bạn sinh viên !!!”;

- String là một dãy các ký tự có độ dài tối đa lên đến **Integer.MAX_VALUE** (hơn 2 tỷ) ký tự.
- Java sẽ thực hiện việc **ghép 2 dãy ký tự String** vào với nhau bằng toán tử cộng (+).
  - Ví dụ:

```
String myString = “Chào các bạn sinh viên ”;
String myClass = “lớp lập trình Java cơ bản.”;
String myWholeString = myString + myClass;
System.out.println(myWholeString);
```

Kiểu dữ liệu String

- Ta có thể đưa một ký tự Unicode vào bên trong dãy ký tự String bằng cách **đặt mã của ký tự ngay sau từ khóa \u**
  - Ví dụ:

```
String strBanQuyen = “\u00A9 Java Tutorials 2020”;
System.out.println(strBanQuyen);
```

- Java sẽ tự động chuyển đổi kiểu số sang kiểu String trước khi thực hiện phép cộng với kiểu dữ liệu String.
  - Ví dụ:

```
int mySoNguyen = 7 / 3;
float mySoThuc = 7f / 3f;
System.out.println(“Phần nguyên là: “ + mySoNguyen);
System.out.println(“Số thực là: “ + mySoThuc);
```

Chú thích

- Chú thích sẽ không tham gia vào quá trình biên dịch chương trình.
- Chú thích trên một dòng, Java sử dụng ký hiệu ***//***

```
int myPhanNguyen = 7 / 4; // Lấy phần nguyên
```

- Chú thích trên nhiều dòng, Java sử dụng ký hiệu ***/* ... */*** với các dòng được đặt giữa các ký hiệu này.

Các toán tử rút gọn

- ++ : Tăng giá trị biến số nguyên thêm 1 đơn vị.
  - ++a : Tăng giá trị a trước khi thực hiện biểu thức.
  - a++: Tăng giá trị a sau khi thực hiện biểu thức.
- -- : Giảm giá trị biến số nguyên đi 1 đơn vị.
  - --a : Giảm giá trị a trước khi thực hiện biểu thức.
  - a--: Giảm giá trị a sau khi thực hiện biểu thức.
- a += b: Tương đương với a = a + b
- a -= b: Tương đương với a = a - b
- a *= b: Tương đương với a = a * b
- a /= b: Tương đương với a = a / b
- a %= b: Tương đương với a = a % b

Các toán tử dùng với biểu thức logic

- Các toán tử so sánh: **>**, **<, >=**, **<=**, **==**, **!=**
- Các toán tử logic: **&&** (phép and), **||** (phép or)
- Toán tử phủ định: **!**
- Ví dụ:

```
boolean myValue1 = 6 > 7;
boolean myValue2 = (6 >= 7) && (9.0 <= 11.0);
boolean myValue3 = !myValue2;
```

Câu lệnh if

if (<biểu thức logic>) {

….

} **else** {

…

}

Toán tử Ternary

- Toán tử **ternary** (3 ngôi) trả về giá trị phụ thuộc vào biểu thức logic đứng trước dấu hỏi.

  <biểu thức logic> ? <giá trị 1> : <giá trị 2>

- Toán tử trả về **<giá trị 1>** khi **<biểu thức logic>** đúng, trả về **<giá trị 2>** khi **<biểu thức logic>** sai.
- Ví dụ:

```
int a = 5, b = 6;
int value = a > b ? a - b : a + b;
```

Phương thức

- Phương thức (method) trong Java **bắt buộc** phải được đặt bên trong một lớp nào đó.
- Ví dụ:

```
public class HelloWorld {
    public static int tinhTong(int soHang1, int soHang2){
        return soHang1 + soHang2;
    }
}
```

Phương thức

- Phân tích đoạn code sau:

```
public class HelloWorld {
    public static void main(String[] args) {
        int soNguyen1 = 11, soNguyen2 = 9;

        if (soNguyen1 > soNguyen2) {
            System.out.println("Hiệu của hai số là: " + tinhHieu(soNguyen1, soNguyen2));
        } else {
            System.out.println("Tổng của hai số là: " + tinhTong(soNguyen1, soNguyen2));
        }
    }

    public static int tinhTong(int soHang1, int soHang2) {
        return soHang1 + soHang2;
    }

    public static int tinhHieu(int soBiTru, int soTru) {
        return soBiTru - soTru;
    }
}
```

Quá tải phương thức

- **Quá tải phương thức** (Method overloading) là hành động xác định nhiều phương thức có cùng tên, nhưng các tham số khác nhau về số lượng hoặc kiểu dữ liệu.
- Không nên viết chương trình kiểu như sau:

```
public static int tongHaiSoNguyen(int soThu1, int soThu2) {
    return soThu1 + soThu2;
}

public static int tongBaSoNguyen(int soThu1, int soThu2, int soThu3) {
    return soThu1 + soThu2 + soThu3;
}

public static int tongBonSoNguyen(int soThu1, int soThu2, int soThu3, int soThu4) {
    return soThu1 + soThu2 + soThu3 + soThu4;
}
```

- Sử dụng quá tải phương thức giúp ta dễ nhớ tên phương thức và có tính mềm dẻo đối với các phương thức có tính chất tương tự nhưng chỉ khác kiểu dữ liệu.

Quá tải phương thức

- Phân tích đoạn code sau:

```
public class HelloWorld {
    public static void main(String[] args) {
        int soNguyen1 = 11, soNguyen2 = 9, soNguyen3 = 7;
        hienThiTong(soNguyen1, soNguyen2, soNguyen3);
    }

    public static void hienThiTong(int soThu1, int soThu2) {
        System.out.println("Tổng của 2 số nguyên: " + (soThu1 + soThu2));
    }

    public static void hienThiTong(int soThu1, int soThu2, int soThu3) {
        System.out.println("Tổng của 3 số nguyên: " + (soThu1 + soThu2 + soThu3));
    }

    public static void hienThiTong(double soThu1, double soThu2) {
        System.out.println("Tổng của 2 số thực: " + (soThu1 + soThu2));
    }
}
```

Câu lệnh switch

```
switch (<biểu thức>) {
    case <giá trị 1>:
        …
        break;
    case <giá trị 2>:
        …
        break;
    ….
    case <giá trị n>:
        …
        break;
    default:
        …
        break;
}
```

Câu lệnh switch

- Phân tích đoạn chương trình sau:

```
public class HelloWorld {
    public static void main(String[] args) {
        int thang = 11;
        System.out.println("Thang " + thang + " co " + getNgay(thang) + " ngay.");
    }

    public static int getNgay(int thang) {
        switch(thang) {
            case 1: case 3: case 5: case 7: case 8: case 10: case 12:
                return 31;

            case 4: case 6: case 9: case 11:
                return 30;

            case 2:
                return 29;
        }
        return 0;
    }
}
```

Vòng lặp for

for (<khởi tạo giá trị>; <biểu thức logic>; <biến điều khiển>) {

    <tăng/giảm biến điều khiển>

}

- Ví dụ:

```
public class HelloWorld {
    public static void main(String[] args) {
        int tong = 0;
        int n = 9;

        for(int i = 1; i <= 9; i++)
            tong += i;

        System.out.println("Tổng của " + n + " số tự nhiên đầu tiên là: " + tong);
    }
}
```

Vòng lặp while

while (<biểu thức logic>) {

…

}

- Ví dụ:

```
public class HelloWorld {
    public static void main(String[] args) {
        int tong = 0, n = 9, i = 1;

        while (i <= n) {
            tong += i;
            i++;
        }

        System.out.println("Tổng của " + n + " số tự nhiên đầu tiên là: " + tong);
    }
}
```

Vòng lặp do … while

do {

…

} **while** (<biểu thức logic>);

- Ví dụ:

```
public class HelloWorld {
    public static void main(String[] args) {
        int tong = 0, n = 1, i = 0;

        do {
            tong += i;
            i++;
        }
        while (i <= n);

        System.out.println("Tổng của " + n + " số tự nhiên đầu tiên là: " + tong);
    }
}
```

Sử dụng các vòng lặp

- **for** nên được sử dụng với các vòng lặp **biết trước số lần lặp**.
- **while** và **do … while** nên sử dụng với các vòng lặp **không biết trước số lần lặp**. Tuy nhiên:
  - **while** sử dụng với vòng lặp mà cần phải **kiểm tra điều kiện lặp trước**, sau đó mới thực hiện các câu lệnh trong code block.
  - **do … while** sử dụng khi cần thực hiện các câu lệnh trong code block 1 lần trước khi kiểm tra điều kiện.

Chuyển đổi dữ liệu từ chuỗi sang số

- Chúng ta sử dụng một số phương thức trong **Wrapper class** để biến đổi dữ liệu từ chuỗi sang số.
  - Ví dụ:

```
String chuoiSoNguyen = “155”;
int soNguyen = Integer.parseInt(chuoiSoNguyen);

String chuoiSoThuc = “123.45”;
double soThuc = Double.parseDouble(chuoiSoThuc);
```

Sử dụng thư viện trong Java

- Java có rất nhiều thư viện và phương thức hỗ trợ người lập trình.
- Một số thư viện cơ bản có sẵn trong Java có tên gọi chung là **gói** (**package**)
  - **java.lang** chứa hầu hết những thành phần cơ bản nhất được dùng trong lập trình như các Wrapper class, kiểu String, ...
  - **java.util** là gói tiện ích chứa các cấu trúc dữ liệu, xử lý ngày tháng, số ngẫu nhiên, …
  - **java.io** là gói làm việc với nhập xuất dữ liệu (file, màn hình, …)
  - **java.net** là gói hỗ trợ xây dựng các ứng dụng mạng.
  - …

Sử dụng thư viện trong Java

- Để sử dụng một gói nào đó trong Java ta sử dụng cú pháp:

import <tên gói>;

  - Ví dụ:

```
import java.util.Scanner;
import java.io.*;
```

- Gói **java.lang.*** mặc định **luôn được import**, do đó ta không cần phải khai báo để sử dụng

Đọc dữ liệu từ bàn phím

- Để đọc dữ liệu từ bàn phím ta sử dụng lớp **Scanner** trong gói thư viện **java.util**
- Ví dụ:

```java
import java.util.Scanner;

public class HelloWorld {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Nhập họ và tên: ");
        String strHoVaTen = scanner.nextLine();

        System.out.println("Nhập năm sinh: ");
        int iNamSinh = scanner.nextInt();

        System.out.println("Nhập điểm: ");
        double dDiem = scanner.nextDouble();

        scanner.close();
    }
}
```

Mảng

- **Mảng** (Array) là một tập hợp các phần tử có cùng kiểu dữ liệu và có một số đặc điểm sau:
  - Địa chỉ các phần tử của mảng nằm liên tiếp nhau bên trong bộ nhớ.
  - Số lượng phần tử của mảng là cố định.
  - Chỉ số phần tử đầu tiên của mảng bắt đầu từ 0.
- Khai báo mảng
  - <kiểu dữ liệu>[] <tên mảng> = **new** <kiểu dữ liệu>[<số lượng phần tử>];
- Ví dụ:

```
int[] arrMang1;
arrMang1 = new int[10];

double[] arrMang2 = new double[20];

String[] arrMang3 = {“Java”, “Trí tuệ nhân tạo”, “Máy học” }; // Mảng 3 phần tử
```

Mảng

- Đọc hiểu đoạn code sau:
  - Hàm **nextInt()** của lớp **Random** dùng để sinh ra ngẫu nhiên một số nguyên có dấu **kiểu int**.

```
int[] arrIntDaySo = new int[10];
Random generator = new Random();

for(int i = 0; i < arrIntDaySo.length; i++) {
    arrIntDaySo[i] = generator.nextInt();
}
```

Mảng là đối số của phương thức

- Phân tích phương thức sau:
  - Phương thức **nhận đối số là một mảng**, dữ liệu của mảng được sử dụng để tính toán

```
public static long tinhTong(int[] arrInt) {
    long lTong = 0;
    for(int i = 0; i < arrInt.length; i++) {
        lTong += arrInt[i];
    }
    return lTong;
}
```

Mảng là đối số của phương thức

- Phân tích các phương thức sau:
  - Phương thức **trả về giá trị là một mảng**

```
private static Scanner scanner = new Scanner(System.in);

public static int[] nhapDuLieu(int number) {
    System.out.println("Nhập vào " + number + " số nguyên: ");

    int[] values = new int[number];
    for (int i = 0; i < values.length; i++) {
        values[i] = scanner.nextInt();
    }
    return values;
}
```

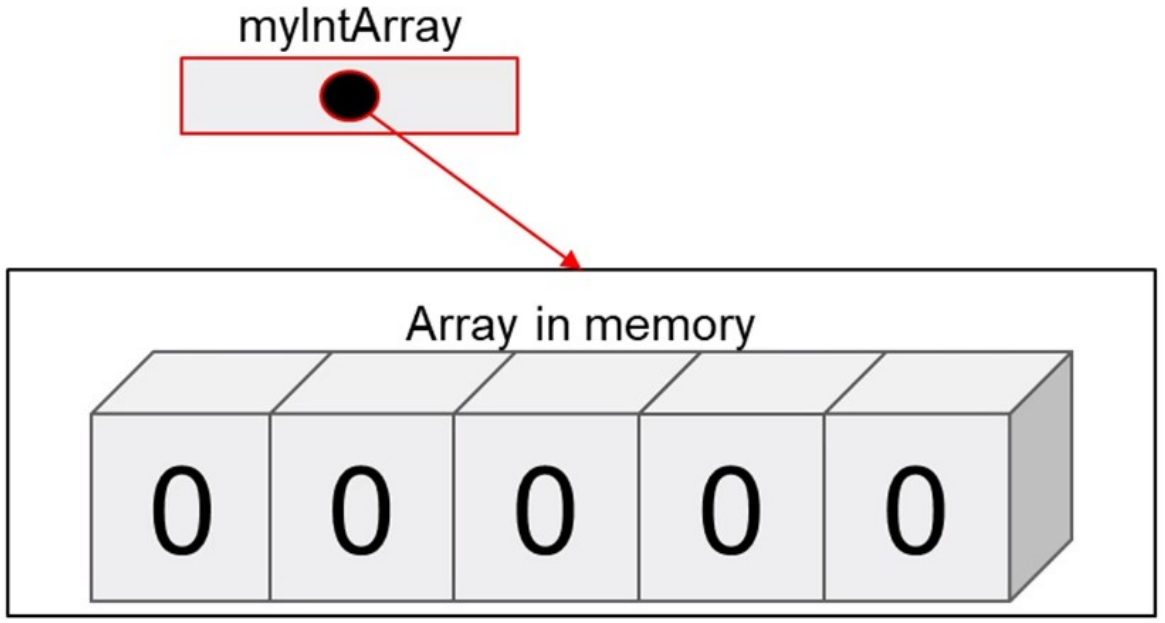
Mảng là đối số của phương thức

- Ví dụ sử dụng các hàm trên

```
int[] arrData = nhapDuLieu(5);
System.out.println("Tổng mảng là: " + tinhTong(arrData));
```

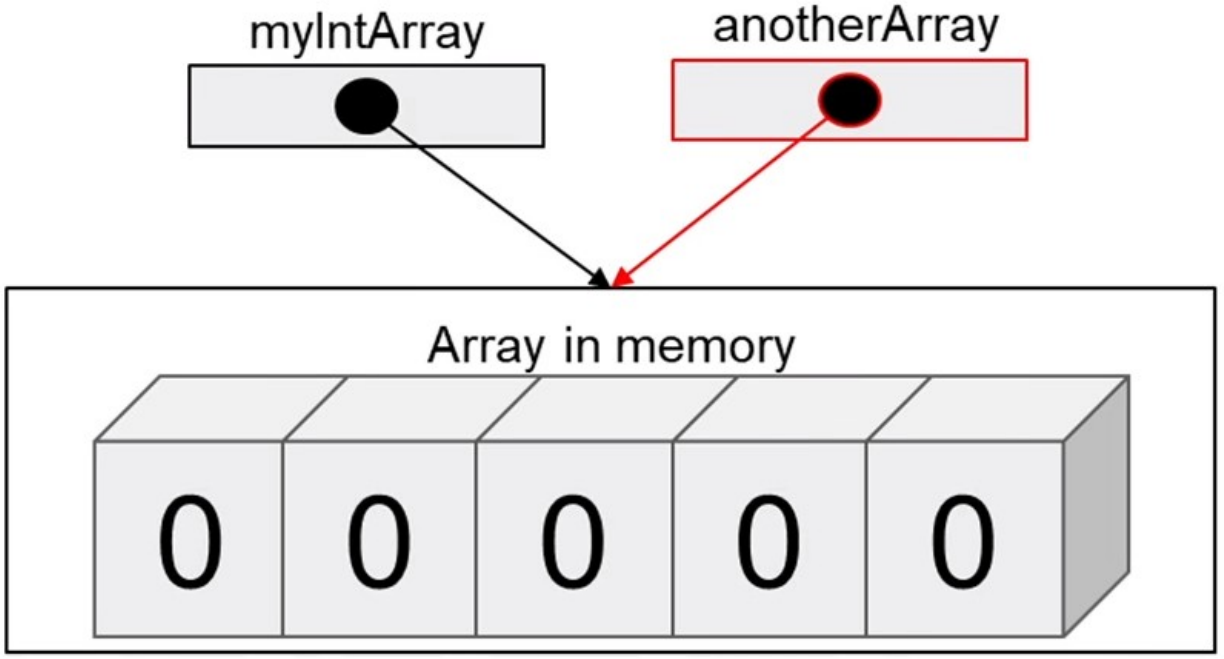
Mảng là một kiểu tham chiếu

```
int[] myIntArray = new int[5];
```



Mảng là một kiểu tham chiếu

```
int[] myIntArray = new int[5];
int[] anotherArray = myIntArray;
```



Mảng là đối số của phương thức

- Phương thức đảo ngược mảng số
  - **Thay đổi dữ liệu của mảng ở bên trong hàm**

```
public static void reverse(int[] array) {
    int maxIndex = array.length – 1;
    int halfLength = array.length / 2;
    for (int i = 0; i < halfLength; i++) {
        int temp = array[i];
        array[i] = array[maxIndex – i];
        array[maxIndex – i] = temp;
    }
}
```

Mảng nhiều chiều

- Ta có thể áp dụng toàn bộ các kiến thức xử lý trên mảng 1 chiều đối với mảng nhiều chiều.
- Ví dụ một số xử lý trên mảng 2 chiều

```
int[][] m = new int[][] { {1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };


for (int i = 0; i < m.length; i++) {
    for (int j = 0; j < m[i].length; j++) {
        System.out.print(m[i][j] + " ");
    }
    System.out.println("");
}
```

Hết Tuần 1

Cảm ơn các bạn đã chú ý lắng nghe !!!