LẬP TRÌNH CƠ BẢN

GIỚI THIỆU

Nguyễn Thị Tâm
nguyenthitam.hus@gmail.com

Ngày 14 tháng 10 năm 2024

Nội dung

Giới thiệu

Mục tiêu môn học

- Lập trình cơ bản
- Ngôn ngữ lập trình Java

Tổ chức môn học

- Lý thuyết: 22 tiết
- Thực hành: 46 tiết
- Trao đổi thông tin, tương tác liên quan đến học phần qua Google class room (sử dụng email HUS).
  - MAT2505 1,2,3 - Mã classroom: **n2yezfw**
  - MAT2505 4,5,6 - Mã classroom: **v6pgvtp**
  - MAT2505 7,8,9 - Mã classroom: **4crt4ki**

Giới thiệu

Yêu cầu đối với sinh viên

- Thực hiện đầy đủ các yêu cầu của giảng viên trong giờ học
- Đi học đầy đủ:
  - Nghỉ nhiều hơn 02 buổi học sẽ không được thi.
  - Trường hợp đặc biệt phải xin phép và được sự đồng ý của giảng viên phụ trách.

Tổ chức kiểm tra, đánh giá

- Điểm thường xuyên 20%.
- Điểm giữa kỳ 20%.
- Điểm cuối kỳ 60%.
  - Bài kiểm tra trắc nghiệm
  - Bài lập trình

Giới thiệu

Nội dung môn học

- Kiểu dữ liệu, biến, toán tử, và biểu thức
- Tổ chức chương trình và cấu trúc điều khiển.
- Mảng
- Lớp và đối tượng
- Giao diện lớp và kế thừa.
- Xử lý ngoại lệ.
- Thư viện vào/ra.

Giới thiệu

Tài liệu

- Tài liệu, bài giảng do giảng viên cung cấp.
- Jones, Evan, Adam Marcus, and Eugene Wu, Introduction to Programming in Java, 2010 (Massachusetts Institute of Technology).
- Allen B. Downey, Think Java – How to think like a computer Scientist.
- David J. Eck, Introduction to Programming using Java.

Tham khảo thêm

- Đoàn Văn Ban (2005), Lập trình hướng đối tượng với Java, NXB Khoa học và Kỹ thuật.
- Bruce Eckel (2006), Thinking in Java, 4th Edition, Prentice Hall.
- The Java Tutorial, http://docs.oracle.com/javase/tutorial/.
- Cay S. Horstmann Garry Cornell, Core Java, Volume I and II, 8th Edition, Prentice Hall, 2007.
- C.S. Horstmann, Java for Everyone , John Wiley and Sons, 2003.

Nội dung

Chương trình máy tính

Chương trình máy tính:

- Chương trình máy tính (hoặc phần mềm) là một tập hợp các hướng dẫn và lệnh mà máy tính hoặc thiết bị điện tử có khả năng xử lý thông tin có thể thực hiện. Chúng được viết bằng ngôn ngữ lập trình để thực hiện một loạt các nhiệm vụ hoặc tính toán cụ thể.
- Cần ngôn ngữ để truyền (thể hiện) lệnh cho máy tính.

Lập trình máy tính

- Xây dựng và cung cấp cho máy tính một dãy các lệnh (thông qua ngôn ngữ lập trình).
- Các bước xây dựng phần mềm máy tính:
  - Đặc tả yêu cầu.
  - Phân tích dữ liệu, thuật toán.
  - Thiết kế.
  - Cài đặt.
  - Kiểm thử.
- **Lý thuyết:**
  - Thuật toán (khái niệm, sơ đồ khối, tựa mã - mã giả)
  - Ngôn ngữ lập trình, chương trình dịch
- **Thực hành:**
  - Sử dụng chế độ dòng lệnh - Command line của HĐH
  - Cài đặt các công cụ cần thiết

Ngôn ngữ lập trình

Ngôn ngữ tự nhiên

- Đa nghĩa (nhập nhằng), dư thừa, tiến hóa theo thời gian.
- Phụ thuộc chặt chẽ vào ngữ cảnh.

Ngôn ngữ lập trình máy tính

- **Ngôn ngữ bậc thấp:**
  - VD: Ngôn ngữ máy (mã nhị phân), Hợp ngữ (Assembly)
  - Chạy nhanh; Phụ thuộc kiến trúc máy tính & HĐH; Khó viết, khó đọc & hiểu, khó kiểm soát, khó mang đi; Không phù hợp để xây dựng chương trình lớn xử lý dữ liệu phức tạp
- **Ngôn ngữ bậc cao:**
  - VD: Pascal, C/C++, Java, Python...
  - Thường chạy chậm hơn
  - Gần với ngôn ngữ tự nhiên hơn; Dễ viết - đọc hiểu và kiểm soát; Dễ mang đi...

Biên dịch và thông dịch

Biên dịch

- Trình biên dịch dịch mã nguồn lập trình hoàn toàn thành mã máy hoặc mã trung gian trước khi chạy chương trình.
- Kiểm tra toàn bộ chương trình để tìm lỗi ngữ pháp và logic trước khi tạo mã máy.
- Sau khi mã máy hoặc mã trung gian được tạo, chương trình có thể chạy nhiều lần mà không cần biên dịch lại, trừ khi có sự thay đổi trong mã nguồn.
- Ví dụ: C, C++, Rust.

Biên dịch và thông dịch

Thông dịch

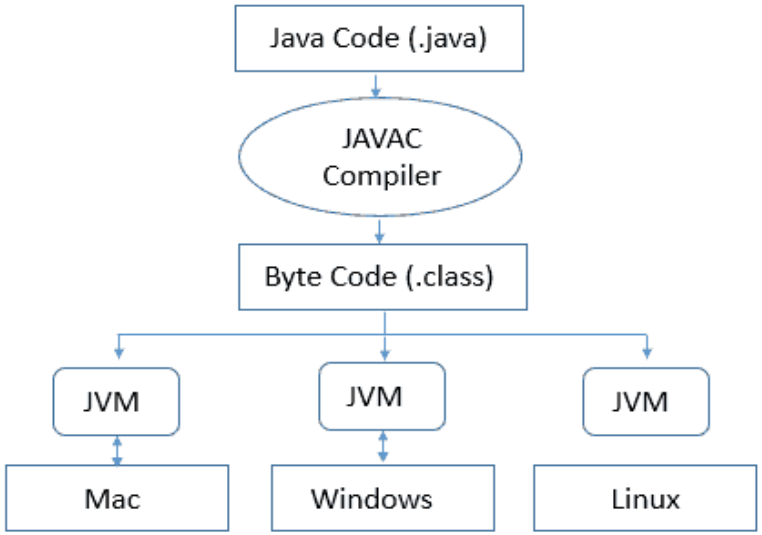
- Trình thông dịch thực thi mã nguồn lập trình từng dòng lệnh hoặc khối lệnh một cách tuần tự.
- Thực thi mã nguồn ngay lập tức, không cần tạo mã máy hoặc mã trung gian.
- Nó kiểm tra lỗi tại thời điểm thực thi, vì vậy nếu có lỗi trong mã nguồn, chương trình sẽ dừng ngay lập tức và báo lỗi.
- Ví dụ: Python, JavaScript, Ruby.

Ngôn ngữ lập trình Java

- Ngôn ngữ lập trình bậc cao hiện đại, mạnh, phổ biến
- Là ngôn ngữ vừa biên dịch vừa thông dịch
  - Biên dịch (Compilation): Mã nguồn Java (.java) được biên dịch thành mã trung gian gọi là bytecode bởi trình biên dịch Java (javac). Bytecode là một mã trung gian độc lập với nền tảng, và nó không thể thực thi trực tiếp trên máy tính. Kết quả của quá trình biên dịch là một tệp .class chứa bytecode.
  - Thực thi: Bytecode được thực thi trên máy ảo Java (Java Virtual Machine - JVM) bằng cách sử dụng trình thông dịch Java. JVM đọc bytecode và thực thi nó trên máy tính. JVM là môi trường thực thi đa nền tảng, cho phép chạy mã Java trên nhiều hệ điều hành khác nhau mà không cần biên dịch lại.

Ngôn ngữ lập trình Java

- Độc lập với nền tảng: Bytecode là độc lập với nền tảng, vì vậy mã bytecode có thể chạy trên bất kỳ máy tính nào có JVM.
- Kiểm tra lỗi trước khi thực thi: Mã nguồn Java được kiểm tra lỗi tại thời điểm biên dịch, giúp phát hiện lỗi sớm và giúp bạn viết mã an toàn hơn.
- Tích hợp sâu với thư viện: Java có một loạt thư viện và framework mạnh mẽ, giúp xây dựng ứng dụng phức tạp. Bytecode sử dụng các thư viện này một cách hiệu quả.
- Bảo mật: JVM cung cấp các cơ chế bảo mật để kiểm soát quyền truy cập và thực thi của bytecode.
- Ngôn ngữ chặt chẽ; Cú pháp rất gần với $C/C++$ nhưng đơn giản hơn



Nền tảng lập trình Java

- Một số "nền tảng Java" được định nghĩa và xây dựng hướng đến các môi trường ứng dụng khác nhau
  - **Java Platform, Micro Edition (Java ME):** dành cho các môi trường hệ thống nhúng, VD ĐTDĐ.
  - **Java Platform, Standard Edition (Java SE):** là nền tảng tiêu chuẩn, thường được dùng để phát triển Java application, Java applet.
  - **Java Platform, Enterprise Edition (Java EE):** dành cho môi trường lớn và phân tán của doanh nghiệp hoặc Internet.
  - **Java FX:** là một framework bao gồm các gói đồ họa, công cụ hỗ trợ cho người lập trình có thể tạo, kiểm tra, gỡ lỗi và triển khai ứng dụng trên nhiều loại thiết bị như: Điện thoại di động, máy tính để bàn, TV,...
- Môi trường lập trình:
  - Có thể sử dụng trình soạn thảo bất kỳ. Soạn và dịch chương trình bằng chế độ dòng lệnh - Command line của HĐH
  - Cài đặt các môi trường lập trình tích hợp (IDE), VD Eclipse, NetBean, JBuilder...

Chuẩn bị thực hành lập trình Java

Tải miễn phí bộ công cụ dịch JDK (Java Developmet Kit) tại `http://www.oracle.com/technetwork/java/javase/downloads/index.html`

Cài đặt công cụ dịch

- Trình compiler javac : Dịch từ file nguồn (∗.java) ⇒ byte code (∗.class)
- Máy ảo java : Chạy chương trình dạng byte code.

Dịch và chạy chương trình

- Dòng lệnh biên dịch file nguồn (∗.java):
  `javac <source_file.java>`
- Dòng lệnh chạy chương trình byte code bằng máy ảo:
  `java <byte_code file.class>`

Lập trình với Java

Thử nghiệm chương trình Java đầu tiên

```
class HelloWorld
{
   public static void main(String []args)
   {
      System.out.println("Hello World");
   }
}
```

Thử nghiệm chương trình Java đầu tiên

Chuẩn bị

- Tải và cài đặt bộ công cụ dịch (JDK). Chú ý đặt đường dẫn (PATH) trong Environment Variables
- Dùng trình soạn thảo bất kỳ, soạn tệp tin văn bản nội dung trên. Lưu tệp với tên "HelloWorld.java".
- **Chú ý:**
  - Tên tệp tin phải trùng với tên lớp (CLASS).
  - VD: Class **HelloWorld** thì tên file là **HelloWorld.java**.

Dịch và chạy thử chương trình

- Dòng lệnh biên dịch tệp nguồn thành tệp bytecode (java $\Rightarrow$ Class):
  `javac HelloWorld.java`
- Chạy chương trình đã dịch trên máy ảo JAVA:
  `java HelloWorld`

Các thành phần trong chương trình

```
class HelloWorld
{
   public static void main(String []args)
   {
      System.out.println("Hello World");
   }
}
```

- Khai báo lớp `public class CLASS_NAME { /*content */}`
- Khai báo phương thức main `public static void main(String[] args)`
- Phần chú thích (tương tự C/C++)
  - Chú thích trên một dòng: `// Comment line`
  - Chú thích theo khối `/* Comment block */`

Tham số của phương thức main()

- Phương thức main `public static void main(String[] args)`
- Đối dòng lệnh (command-line parameters) `String[] args`
- VD minh họa sử dụng tham số truyền qua dòng lệnh

```
public class UseArgument {
    public static void main(String[] args) {
        System.out.print("Hello ");
        System.out.print(args[0]);
        System.out.println(". How are you?");
    }
}
```

Truyền tham số qua đối dòng lệnh

- Lưu chương trình vào tệp UseArgument.java
- Dịch chương trình `javac UseArgument.java`
- Thực thi với các tham số đầu vào khác nhau và xem kết quả.
  VD:

```
java UseArgument VIET
java UseArgument NAM
java UseArgument VIET NAM
java UseArgument "VIET NAM"
```

Kiểu dữ liệu và biến

Kiểu dữ liệu

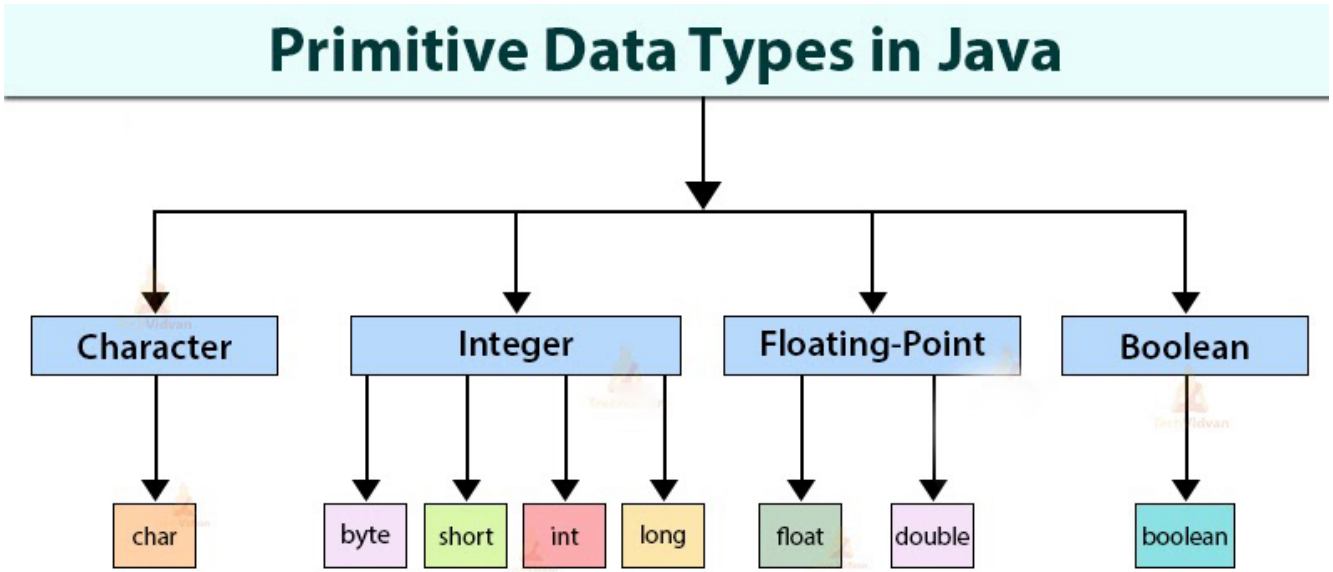
- Kiểu dữ liệu: là tập các giá trị và các phép toán hay thao tác định nghĩa trên nó.
- Khi thực thi chương trình, mỗi giá trị dữ liệu thuộc cùng một kiểu sẽ được lưu trữ trong một vùng nhớ có kích thước xác định cho kiểu dữ liệu đó.
- Java có hai nhóm kiểu dữ liệu
  - Kiểu dữ liệu cơ bản (nguyên thủy): **boolean, byte, char, short, int, long, float, double**
  - Kiểu dữ liệu tham chiếu

Kiểu dữ liệu

Kiểu dữ liệu nguyên thuỷ

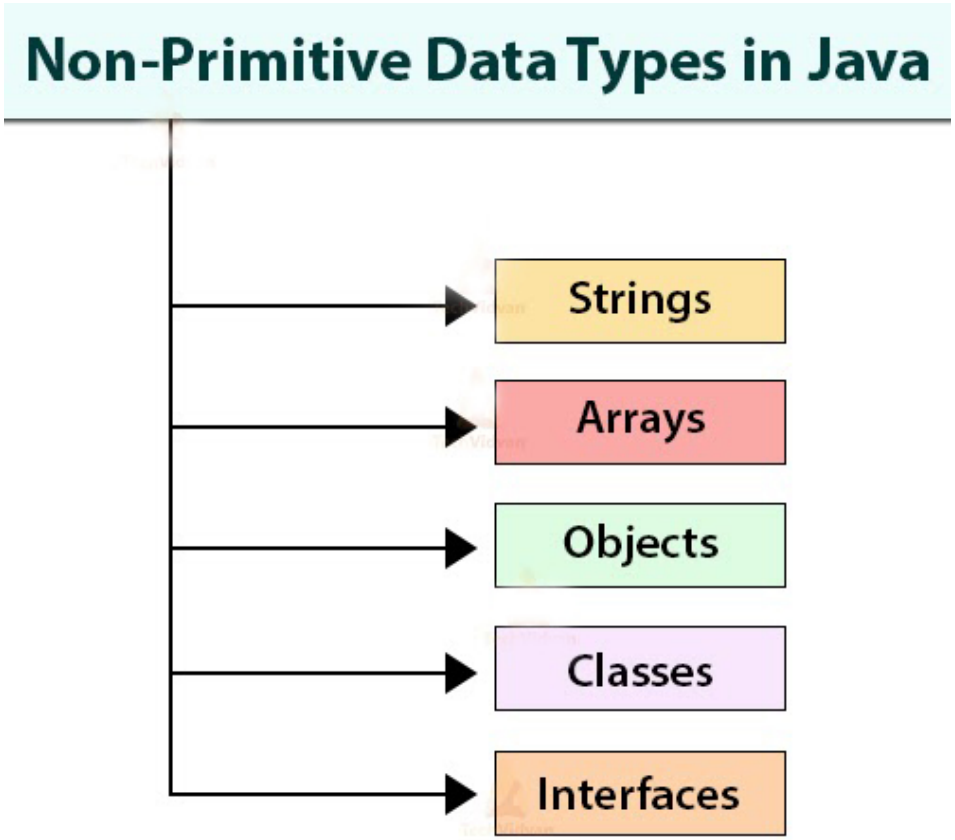
Bảng 1: Các kiểu dữ liệu cơ bản (nguyên thủy)

Kiểu	Gt. mặc định	Kích thước	Khoảng giá trị
char	0	2 bytes	0 .. 65535 (Unicode)
boolean	false	1 bit	false/true
short	0	2 bytes	-32767 .. 32767
int	0	4 bytes	-2.147.483.648 .. 2.147.483.647
long	0	8 bytes	$-2^{63} + 1..2^{63}$
float	0.0	4 bytes	$+/- 1.4023 \times 10^{-45}..3.4028 \times 10^{38}$
double	0.0	8 bytes	$+/- 4.9406 \times 10^{-324}..1.7977 \times 10^{308}$



Kiểu dữ liệu

Kiểu dữ liệu tham chiếu



Hình 2: Các kiểu dữ liệu tham chiếu

- **Kiểu String**: Chuỗi (xâu) ký tự, VD: "Hello", "Good morning", "Vietnam" ...

Biến

- Tên của vùng bộ nhớ lưu trữ dữ liệu thuộc kiểu xác định.
- Biến cần được khai báo trước khi sử dụng.
- Cú pháp khai báo biến:
  TYPE_NAME VARIABLE_NAME [=**VALUE**];.
  VD: `char ch; int a; float b = 0.5;`
- Tên biến
  - Chỉ gồm chữ cái, chữ số và dấu gạch dưới;
  - Không được bắt đầu bằng chữ số;
  - Phân biệt chữ hoa và chữ thường. VD: variable, Variable, vaRiable ... là các biến khác nhau.
- Có thể khai báo nhiều biến cùng kiểu cách nhau bởi dấu phẩy ","
  TYPE_NAME VAR1, VAR2, VAR3;.
  VD: `float a, b, c; String firstname, lastname;`

Toán tử

Phép gán

- Gán (đặt) một giá trị cho một biến qua phép gán. Cú pháp:
  VAR_NAME = VALUE;
  VD: `float a, b, c; a = 3.13; b = 0.5; c = a * b;`
- Có thể kết hợp phép gán và khai báo biến trên cùng dòng lệnh. VD đoạn lệnh trên có thể viết lại

```
float a = 3.13, b = 0.5, c; c = a * b;
```

- Về trái phép gán phải là một biến, về phải có thể là biến, giá trị hoặc biểu thức ...
  VAR_NAME = VALUE | VAR | EXP;
- Thứ tự thực hiện phép gán: Tính toán giá trị ở về phải trước, sau đó gán cho biến ở về trái. VD

```
float a = 4.0, b = 0.5, c; //declaration of variables
c = a*b; //set value for variables c = 4.0 x 0.5 = 2.0
// compute (c + 1)= 2 + 1 = 3 first, then set new value
c = c + 1;
```

Toán tử

- Xác định thao tác trên biến, giá trị; cho phép tạo nên các biểu thức.
- Một số nhóm toán tử:
  - Nhóm toán tử số học (thao tác trên kiểu số) hai ngôi: +, -, *, /, % (chia lấy dư mod).
  - Nhóm toán tử quan hệ: <, <=, >, >=
  - Nhóm toán tử so sánh bằng hoặc khác nhau:
    == (bằng), != (khác)
  - Nhóm toán tử logic: && (AND), || (OR), !(NOT). Chú ý: Biểu thức logic cần đặt trong cặp dấu ().
    VD: (a > b) && (b >=c); !(a <= b); (a < c) || (b < c)...
  - Toán tử ghép nối các xâu ký tự: + . VD:

```
String str1, str2;
str1 = "Hello";
str2 = str1 + "Vietnam";//str2 == "Hello Vietnam"
```

Một số toán tử khác: ++ và −−

- Tăng/giảm giá trị ++/−−
  - Toán tử 1 ngôi, mẫu cú pháp
    `Var_name ++` / `Var_name --` hoặc
    `++Var_name` / `--Var_name`
  - Chỉ áp dụng cho biến, phép toán (++ hoặc −−) có thể đặt trước hoặc sau tên biến.
  - Var_name++ (hoặc ++Var_name): Var_name tăng thêm 1
    Var_name−− (hoặc −−Var_name): Var_name giảm đi 1.
- Thứ tự thực hiện thao tác:
  - **Prefix - phép toán đứng trước** `++Var_name`: Tăng giá trị biến Var_name trước, sau đó trả lại giá trị (đã tăng) trong Var_name.
  - **Postfix - phép toán đứng sau** `Var_name ++`: Trả lại giá trị (ban đầu) trước, sau đó (kết thúc lệnh) tăng giá trị biến Var_name.
  - Tương tự với toán tử giảm −−.
- **Thứ tự ưu tiên các phép toán :** Phép toán 1 ngôi được xét trước. Các phép toán số học +, -, *, /, % theo thứ tự số học.
- Sử dụng cặp dấu () để nhóm và thay đổi thứ tự ưu tiên của nhóm biểu thức.

Một số toán tử khác: ++ và −−

Ví dụ minh họa toán tử ++:

```
public class DataTypesOps {
    public static void main(String[] args) {
        int counter = 5;
        int x, y;
        x = counter++; // output: x = 5
        y = ++counter; // output y = 7
        System.out.println(x);
        System.out.println(y);
    }
}
```

Luyện tập: Học viên xây dựng các chương trình tương tự để minh họa toán tử −−.

Một số toán tử khác: phép toán kết hợp gán

- Nếu **phép gán** có dạng:

  var = var opt (expression) ;

  ở đây, **var** là biến vế trái; **opt** là một toán tử số học; **(expression)** là biểu thức cùng kiểu. Tức là
  - vế phải là biểu thức chứa toán tử 2 ngôi **opt**;
  - biến vế trái **var** cũng là toán hạng của toán tử ở vế phải (chiếm một ngôi)
- thì có thể viết gộp, cú pháp

  var opt= (expression) ;

  opt có thể là +, -, *, /, % .
- Ví dụ
  - **x += y;** tương đương **x = x + y;**
  - **a *= b + 1;** tương đương **a = a *(b + 1);** (không phải a = a * b + 1)
- Cho phép viết gọn chương trình. Tuy nhiên cần sử dụng hợp lý để đảm bảo chương trình vẫn dễ hiểu.

Tổ chức chương trình
Cấu trúc điều khiển

Tổ chức chương trình -Phương thức

Phần thực thi tổ chức thành các chương trình con: Phương thức (**method** - tương tự hàm (function) và thủ tục (procedure)). Xét VD

```
//method||function definition
public static void main(String[] args)
{ // Beginning of method||function
    STATEMENTS
} // End of method||function
```

- **public** (hoặc **private**):Thuộc tính mức độ truy cập;
- **static**: Từ khóa (tùy chọn) cho phép gọi hàm mà không cần khởi tạo đối tượng của lớp. VD: printf(), println(), main() ... (giải thích tại sao cần?)
- **void**, **main**, **String[] args**: tên kiểu trả về, tên hàm/phương thức, tham biến/tham số.

Tổ chức chương trình - Phương thức

- Tổng quát, **định nghĩa** một phương thức mới:

```
[public|private] [static] TYPE NAME([Arguments])
{
        STATMENTS;
        [return EXPRESSION;]
}
```

- Nếu kiểu trả về **TYPE** là
  - **void** (không có giá trị trả về), ta có **Phương thức (method)** và không dùng dòng lệnh **return EXPRESSION;**
  - Trả về giá trị có kiểu khác **void** (tương tự khái niệm HÀM (function) của các ngôn ngữ lập trình đã học), ta sử dụng dòng lệnh **return EXPRESSION;**. **EXPRESSION** phải cùng kiểu **TYPE**.
- Đối số **[Arguments]**, nếu có, được viết theo cú pháp:

  Type_1 Arg_1, Type_2 Arg_2, ..., Type_n Arg_n

  (n là số đối của hàm/phương thức).

Tổ chức chương trình - Phương thức

Với phương thức có **n** đối đã được định nghĩa:

```
public [static] TYPE NAME(Type_1 Arg_1,..., Type_n Arg_n)
{
        STATMENTS;
}
```

- Nếu **TYPE** $\equiv$ **void**, cú pháp gọi phương thức:

```
NAME(Par_1,..., Par_n);
```

- Nếu **TYPE** $\not\equiv$ **void**, cú pháp gọi (tương tự gọi hàm):

```
[Var =] NAME(Par_1,..., Par_n);
```

- **Var** là biến có kiểu **TYPE**;
- **Par_1,..., Par_n** là **n** biến hoặc giá trị có kiểu **Type_1 , ..., Type_n** , tương ứng.

Phương thức - Ví dụ minh họa

Biên dịch, thực hiện và đánh giá kết quả chương trình dưới đây

```
public class NewLine {
    public static void newLine(){
        System.out.println("");
    }
    public static void twoLines(){
        newLine();
        newLine();
    }
    public static void main(String args[]){
        System.out.println("Line 1");
        twoLines();
        System.out.println("Line 2");
    }
}
```

Phương thức - Ví dụ minh họa

```
public class Square {
    public static void printSquare(int x){
        System.out.println(x*x);
    }
    public static void main(String args[]){
        int value = 2;
        printSquare(value); // 4
        printSquare(3); // 9
        printSquare(value*2); //16
    }
}
```

Phương thức - Ví dụ minh họa

Luyện tập: Soạn và dịch chương trình Square2.java dưới đây.

- Chương trình có thực thi được không?
- Có những thông báo gì? Hãy thử giải thích.

```
public class Square2 {
    public static void printSquare(int x){
        System.out.println(x*x);
    }
    public static void main(String args[]){
        printSquare(3.2);
        printSquare("hello");
    }
}
```

Tại sao cần Phương thức

- Cho phép phân rã công việc tổng thể (hệ thống chương trình) thành tập hợp các phần việc đơn giản hơn (chương trình con).
  - Cho phép tổ chức chương trình thành các khối
  - Chia chương trình lớn thành các chương trình nhỏ, đơn giản
- Các chương trình con (phương thức) được phát triển, kiểm thử và tái sử dụng một cách độc lập.
- Người sử dụng các chương trình con không nhất thiết phải biết nó hoạt động cụ thể như thế nào: tính **"Trừu tượng hóa"**.

Tài liệu tham khảo I

- Giáo viên, *Tài liệu, bài giảng do giảng viên cung cấp.*
- Đoàn Văn Ban (Tham khảo thêm), *Lập trình hướng đối tượng với Java*, NXB Khoa học và Kỹ thuật (2005).
- Jones, Evan, Adam Marcus, and Eugene Wu, *Introduction to Programming in Java*, Massachusetts Institute of Technology, 2010.
- Allen B. Downey, *Think Java – How to think like a computer Scientist*, 2004.
- David J. Eck, *Introduction to Programming using Java*, 2009.
- Bruce Eckel, *Thinking in Java*, 4th Edition, Prentice Hall, 2006.
- website, *The Java Tutorial*, http://docs.oracle.com/javase/tutorial/.
- Cay S. Horstmann Garry Cornell, *Core Java*, Vol I and II, 8th Edition, Prentice Hall, 2007.
- Cay .S. Horstmann, *Java for Everyone*, John Wiley and Sons, 2003.

Kết thúc buổi thứ nhất!