Bài 05:
Sử dụng phương thức (Method)

Giảng Viên: ThS. Giang Hào Côn

Đặt vấn đề

Giả sử, cần tính tổng các số từ 1 đến 10, từ 25 đến 35 và từ 37 đến 49.

```java
class Main {
    public static void main(String[] args) {
        int sum = 0;
        for(int i=1; i<=10; i++){
            sum += i;
        }
        System.out.println("Tong tu 1 den 10 la " + sum);

        sum = 0;
        for(int i=25; i<=35; i++){
            sum += i;
        }
        System.out.println("Tong tu 25 den 35 la " + sum);

        sum = 0;
        for(int i=37; i<=49; i++){
            sum += i;
        }
        System.out.println("Tong tu 37 den 49 la " + sum);
    }
}
```

```java
class Main {

    public static int sum(int a, int b){
        int result = 0;
        for(int i=a; i<=b; i++){
            result += i;
        }
        return result;
    }

    public static void main(String[] args) {
        System.out.println("Tong tu 1 den 10 la " + sum(1, 10));
        System.out.println("Tong tu 25 den 35 la " + sum(25, 35));
        System.out.println("Tong tu 37 den 49 la " + sum(37, 49));
    }
}
```

5.1/ Sự cần thiết của Method

- Khi độ phức tạp của chương trình tăng lên, việc sử dụng quá nhiều mã lệnh trong 1 chương trình mang lại nhiều nhược điểm:
  - Rối rắm, Khó kiểm soát
  - Khó chỉnh sửa (*Khi bảo trì, nâng cấp phần mềm*)
  - Có những công việc bị lặp lại một cách không cần thiết …
- Phương thức (cách gọi cũ là hàm _ Function) là một kỹ thuật khá phổ biến trong các ngôn ngữ lập trình bậc cao dùng để cho phép người lập trình tự tạo ra các công cụ để sử dụng trong chương trình của mình.

5.1/ Sự cần thiết của Method

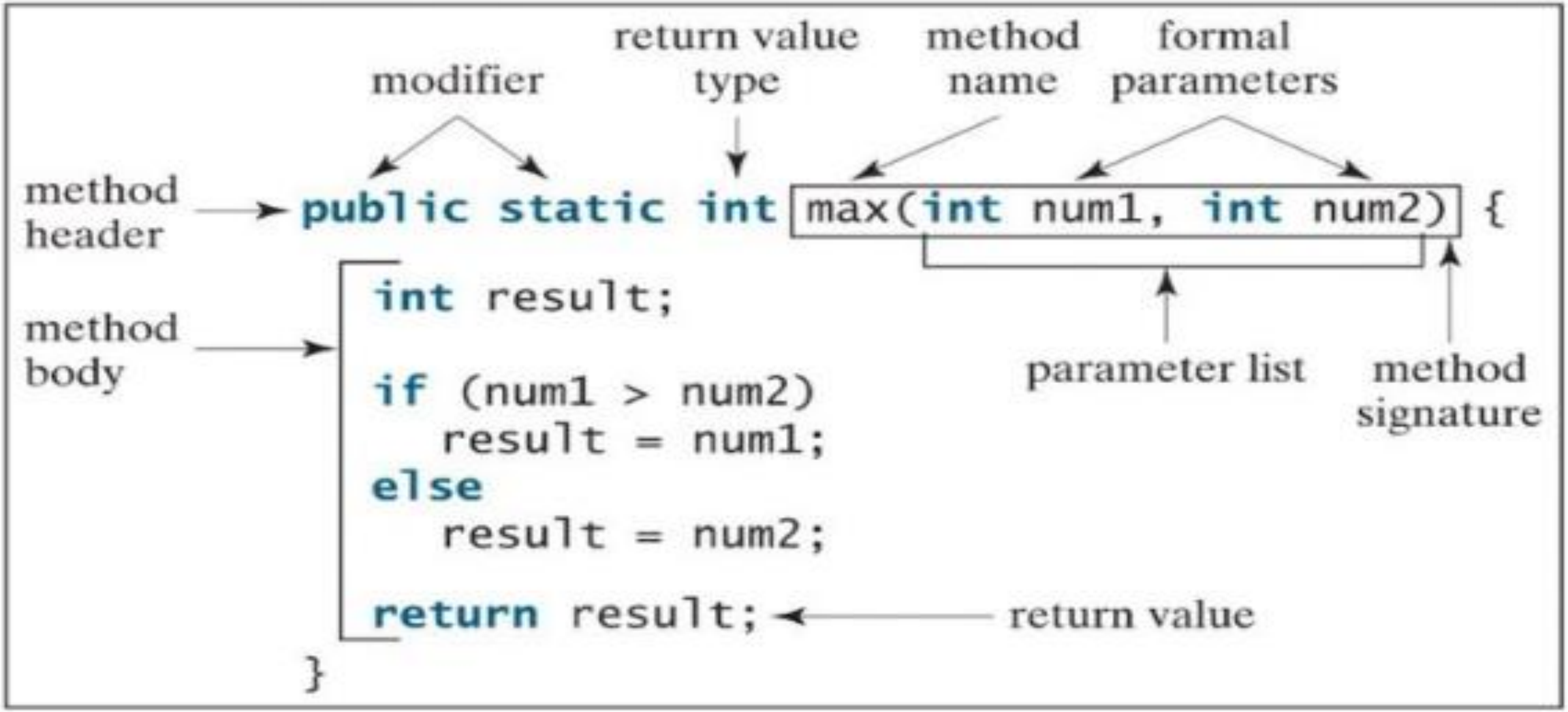
- Việc tổ chức chương trình với các phương thức thành phần đem lại một số ưu điểm như sau:

- **Kiểm soát mã nguồn khoa học**
- **Giảm kích thước**
- **Nhất quán về mặt giải thuật**
- Cú pháp khai báo phương thức:

Một phương thức bao gồm method header và method body.

```
modifier returnValueType methodName(list of parameters)
{
    //method body
}
```

5.1/ Sự cần thiết của Method



5.1/ Sự cần thiết của Method

Ví dụ 01 : Xây dựng phương thức tính n giai thừa

```
long giaiThua(int n)
{
        long kq = 1;
        for(int i, i<=n;i++)
                kq *=kq;
        return kq;
}
```

Ví dụ 02 : Xây dựng phương thức tìm số nhỏ nhất của hai số

```
/** Tra ve gia tri nho nhat cua hai so */
public static int minFunction(int n1, int n2)
{
    int min;
    if (n1 > n2)
      min = n2;
    else
      min = n1;

    return min;
}
```

5.2/ Gọi phương thức trong Java

- Để sử dụng một phương thức, nó nên được gọi. Có hai cách để gọi một phương thức, ví dụ: phương thức trả về một giá trị hoặc phương thức không trả về giá trị nào.
- Tiến trình gọi phương thức là đơn giản. Khi một chương trình gọi phương thức, điều khiển chương trình truyền tín hiệu tới phương thức được gọi. Phương thức được gọi này sau đó trả về điều khiển tới nơi gọi trong hai điều kiện, khi:
  - Lệnh **return** được thực thi.
  - Tiến tới **dấu ngoặc đóng "}"** ở cuối phương thức.

5.2/ Gọi phương thức trong Java

Sau khi đã khai báo và định nghĩa phương thức, trong chương trình chính và các phương thức khác, ta có thể gọi sử dụng phương thức để phục vụ cho mục tiêu xử lý dữ liệu. Một "***lời gọi phương thức***" như vậy có thể được thực hiện như là 1 lệnh độc lập trong chương trình, hoặc cũng có thể sử dụng chúng như là 1 thành phần của 1 biểu thức. Cú pháp:

```
<tên_phương thức>([tham_số], …);
```

Hoặc

```
<kiểu_DL> <biến> = <tên_phương thức>([tham_số], …);
```

5.2/ Gọi phương thức trong Java

- Phương thức trả về **void** được xem như là gọi tới một lệnh.

```
System.out.println("Day la phương thức in nội dung !");
```

- Phương thức trả về giá trị:

```
int result = sum(6, 9);
```

```
public class ExampleMinNumber {
    public static void main(String[] args) {
        int a = 11;
        int b = 6;
        int c = minFunction(a, b);
        System.out.println("Gia tri nho nhat = " + c);
    }
}
```

```
/** Tra ve gia tri nho nhat cua hai so */
public static int minFunction(int n1, int n2) {
    int min;
    if (n1 > n2)
        min = n2;
    else
        min = n1;

    return min;
}
```

5.3.1/ Phương thức không trả về giá trị

Trong trường hợp cần xây dựng phương thức để thực hiện cho một mục tiêu nào đó trong chương trình và sau khi thi hành, chương trình không cần nhận giá trị trả về của phương thức ta dùng từ khoá **void** trong Java. Cú pháp:

```
void <tên_phương thức>([tham_số], [tham_số], …)
{
        . . .
        . . .
        return;
}
```

5.3.1/ Phương thức không trả về giá trị

Cú pháp đầy đủ:

Hình thức thứ nhất
Khai báo và định nghĩa phương thức trước vị trí của main

```
import <thư_viện>
public class <tenLop>{
    <kiểu_DL>  <tên_phương thức>([Tham_số], [Tham_số], …)
    {
        . . . //--- Thân phương thức
        . . .
        return <giá_trị>;
    }
    . . . . . .//--- Các khai báo p.thức khác trong chương trình
    public static void main(String args[])
    {
        . . . //--- Chương trình chính
        . . .
        . . . //--- Gọi phương thức trong chương trình chính
        . . .
    }
}
```

5.3.1/ Phương thức không trả về giá trị

ví dụ 01: *Phương thức methodRankPoints(255.7);*

Nó là một lệnh Java mà kết thúc với dấu chấm phảy như dưới đây.

```java
public class ExampleVoid {

    public static void main(String[] args) {
        methodRankPoints(255.7);
    }

    public static void methodRankPoints(double points) {
        if (points >= 202.5) {
            System.out.println("Rank:A1");
        }
        else if (points >= 122.4) {
            System.out.println("Rank:A2");
        }
        else {
            System.out.println("Rank:A3");
        }
    }
}
```

5.3.1/ Phương thức không trả về giá trị

Ví dụ 02: *Phương thức* phương thức phục vụ cho mục đích in ra bảng cửu chương thứ n.

```
void bangCuuChuong(int n)
{
        for (int i=1; i<=10;i++)
                System.out.printf(" %2d x %2d = %3d \n", n, i, n*i);
}
```

```
public static void main(String[] args) {
    bangCuuChuong(9);
}
```

5.3.2/ Phương thức trả về giá trị

- Đối với hàm có kết quả trả về, chúng ta cần dùng từ khoá return để trả về kết qủa mà nó đã tính toán được. Khai báo biến có kiểu dữ kiệu tương ứng với kết quả trả về của hàm để nhận giá trị trả về.
- Như ví dụ tìm tongHaiSo() của chúng ta, nhiệm vụ của nó là tính tổng của 2 số nguyên a, b nhập vào thì kết quả sẽ ra một số nguyên. Tức là kiểu trả về (returnType) là một số nguyên (int) và kết quả của return cũng là một số nguyên (int). Chúng ta sẽ dùng biến có kiểu dữ liệu tương ứng để nhận kết quả trả về từ hàm.

5.3.2/ Phương thức trả về giá trị

- Ví dụ:

```
public class Thaycacac {

    public static int tongHaiSo(int a, int b) {
        return a + b;
    }

    public static void main(String[] args) {
        int sum = tongHaiSo(2,5);
        System.out.println(sum);
    }
}
```

5.4/ Truyền tham số cho phương thức

Tham số của phương thức là những biến được khai báo trong khai báo phương thức. Tham số đóng vai trò là giá trị đầu vào cho phương thức. Trong Java, có 2 cách để truyền tham số cho phương thức:

- Tham trị (pass by value)
- Tham chiếu (pass by reference)

Lưu ý: Ngôn ngữ Java không có khái niệm con trỏ (pointer) như trong C++ để tránh các trường hợp không kiểm soát tốt việc truy xuất trực tiếp đến bộ nhớ dẫn đến gây lỗi chương trình. Điều này cũng giúp lập trình viên cảm thấy Java dễ hiểu, dễ học hơn C++. Và do Java không hỗ trợ con trỏ nên không có cách truyền địa chỉ cho phương thức như trong C++.

5.4.1/ Truyền tham trị

- Truyền tham trị nghĩa là các tham số truyền cho phương thức không bị thay đổi giá trị sau khi phương thức thi hành. Mặc nhiên, các tham số của hàm đều được xem là "***tham trị***".
- Trong Java, khi gọi một phương thức và truyền một giá trị cho phương thức, được gọi là truyền tham trị. Việc thay đổi giá trị chỉ có hiệu lực trong phương thức được gọi, không có hiệu lực bên ngoài phương thức.
- Trong Java, truyền tham trị dành cho các tham số có kiểu dữ liệu nguyên thủy là byte, short, int, long, float, double, boolean, char.

5.4.1/ Truyền tham trị

- Ví dụ:

```
class Main {
    /*Hoán đổi 2 biến*/
    public static void swap(int n1, int n2) {
        System.out.println("\tGia tri cac bien ben trong ham swap");
        System.out.println("\t\tTruoc khi swap, n1 = " + n1 + " va n2 = " + n2);
        // Swap n1 with n2
        int temp = n1;
        n1 = n2;
        n2 = temp;
        System.out.println("\t\tSau khi swap, n1 = " + n1 + " va n2 = " + n2);
    }
    public static void main(String[] args) {
        int num1 = 1;
        int num2 = 2;
        System.out.println("Truoc khi goi ham swap, num1 = " + num1 + " va num2 = " + num2);
        swap(num1, num2);
        System.out.println("Sau khi goi ham swap, num1 = " + num1 + " va num2 = " + num2);
    }
}
```

5.4.2/ Truyền tham chiếu

- Trong Java, khi gọi một phương thức và truyền một tham chiếu cho phương thức, được gọi là truyền tham chiếu. Việc thay đổi giá trị của biến tham chiếu bên trong phương thức làm thay đổi giá trị của nó.
- Trong Java, tất các phương thức có tham số là biến có kiểu là các lớp (class) đều là kiểu tham chiếu.
- Trong những tình huống nhất định, người lập trình mong muốn, sau khi phương thức xử lý thì các tham số được truyền cho phương thức dưới dạng các biến có thể sẽ bị thay đổi giá trị ở nơi thực hiện "*lời gọi* phương thức". Trong trường hợp này, tham số phải là đối tượng của một class. Kỹ thuật tham chiếu trong java, không áp dụng cho các biến thuộc loại **primitive data types**

5.4.2/ Truyền tham chiếu

- Ví dụ:

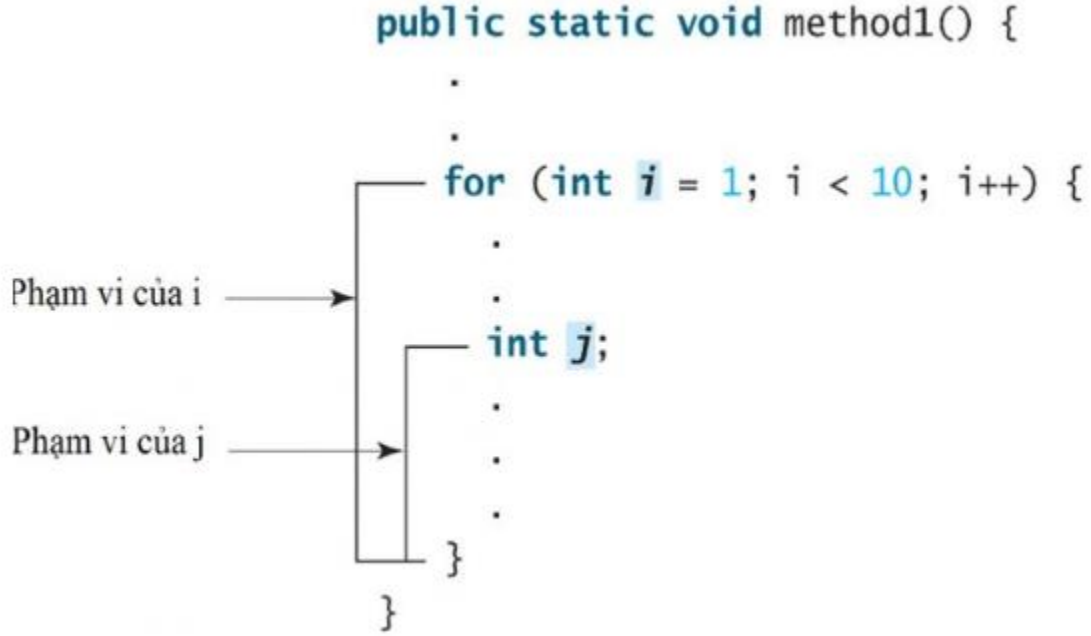
```
static class Main {
    int data;
    Main(int dataValue){
        data = dataValue;
    }
    /*Hoán đổi 2 biến*/
    public static void swap(Main n1, Main n2) {
        System.out.println("\tGia tri cac bien ben trong ham swap");
        System.out.println("\t\tTruoc khi swap, n1.data = " + n1.data + " va n2.data = " + n2.data);
        // Swap n1.data with n2.data
        int temp = n1.data;
        n1.data = n2.data;
        n2.data = temp;
        System.out.println("\t\tSau khi swap, n1.data = " + n1.data + " va n2.data = " + n2.data);
    }
    public static void main(String[] args) {
        Main n1 = new Main(1);
        Main n2 = new Main(2);
        System.out.println("Truoc khi goi ham swap, n1.data = " + n1.data + " va n2.data = " + n2.data);
        swap(n1, n2);
        System.out.println("Sau khi goi ham swap, n1.data = " + n1.data + " va n2.data = " + n2.data);
    }
}
```

5.5/ Tầm vực (phạm vi) của biến

- Tuỳ vào vị trí khai báo mà các biến trong chương trình có phạm vi nhận biết và sử dụng khác nhau. Có hai hình thức khai báo biến mà tầm ảnh hưởng (*hay khu vực có thể nhận biết*) thường được đề cập đến trong chương trình:
- Biến cục bộ
- Biến toàn cục

5.5.1/ Biến cục bộ (local variable)

- **Biến cục bộ (local variable)** là biến được khai báo trong một phương thức hoặc trong một khối lệnh. Biến cục bộ có phạm vi sử dụng chỉ trong phần chương trình mà biến có thể được tham chiếu.



Có thể khai báo một biến cục bộ trùng tên nhiều lần trong các khối lệnh **không lồng nhau**. *Nhưng không thể khai báo một biến cục bộ 2 lần trong các khối lồng nhau*.

5.5.1/ Biến cục bộ (local variable)

Đúng

```
public static void method1() {
  int x = 1;
  int y = 1;

  for (int i = 1; i < 10; i++) {
    x += i;
  }

  for (int i = 1; i < 10; i++) {
    y += i;
  }
}
```

Sai

```
public static void method2() {

  int i = 1;
  int sum = 0;

  for (int i = 1; i < 10; i++)
    sum += i;
  }

}
```

Biến toàn cục (biến instance)

- Biến instance được khai báo trong một lớp(class), bên ngoài các phương thức, constructor và các block.
- Biến instance được lưu trong bộ nhớ heap.
- Biến instance được tạo khi một đối tượng được tạo bằng việc sử dụng từ khóa new và sẽ bị phá hủy khi đối tượng bị phá hủy.
- Biến instance có thể được sử dụng bởi các phương thức, constructor, block, ... Nhưng nó phải được sử dụng thông qua một đối tượng cụ thể.
- Biến instance có giá trị mặc định phụ thuộc vào kiểu dữ liệu của nó.

Biến toàn cục (biến instance)

- Bên trong class mà ta khai báo biến instance, ta có thể gọi nó trực tiếp bằng tên khi sử dụng ở khắp nới bên trong class đó.
- Chúng ta được phép sử dụng access modifier khi khai báo biến instance, mặc định là default.

```
public class SinhVien {
    // biến instance "ten" kiểu String, có giá trị mặc định là null
    public String ten;

    // biến instance "tuoi" kiểu Integer, có giá trị mặc định là 0
    private int tuoi;

    // sử dụng biến ten trong một constructor
    public Sinhvien(String ten) {
        this.ten = ten;
    }
```

```
    // sử dụng biến tuoi trong phương thức setTuoi
    public void setTuoi(int tuoi) {
        this.tuoi = tuoi;
    }

    public void showStudent() {
        System.out.println("Ten  : " + ten);
        System.out.println("Tuoi : " + tuoi);
    }

    public static void main(String args[]) {
        Sinhvien sv = new Sinhvien("Thaycacac");
        sv.setTuoi(21);
        sv.showStudent();
    }
}
```

Biến static trong Java

- Biến static được khai báo trong một class với từ khóa static, phía bên ngoài các phương thức, constructor và block.
- Sẽ chỉ có duy nhất một bản sao của các biến static được tạo ra, dù bạn tạo bao nhiêu đối tượng từ lớp tương ứng.
- Biến static được lưu trữ trong bộ nhớ static riêng.
- Biến static được tạo khi chương trình bắt đầu chạy và chỉ bị phá hủy khi chương trình dừng.
- Giá trị mặc định của biến static phụ thuộc vào kiểu dữ liệu bạn khai báo tương tự biến instance.

Biến static trong Java

- Biến static được truy cập thông qua tên của class chứa nó, với cú pháp: TenClass.tenBien.
- Trong class, các phương thức sử dụng biến static bằng cách gọi tên của nó khi phương thức đó cũng được khai báo với từ khóa static.

```
public class Sinhvien {
    // biến static 'ten'
    public static String ten = "Thaycacac";

    // biến static 'tuoi'
    public static int tuoi = 21;
```

```
    public static void main(String args[]) {
        // Sử dụng biến static bằng cách gọi trực tiếp
        System.out.println("Ten : " + ten);

        // Sử dụng biến static bằng cách gọi thông qua tên class
        System.out.println("Ten : " + Sinhvien.tuoi);
    }
}
```

5.6/ Nạp chồng (overloading) phương thức

- *Nếu 2 method có cùng tên nhưng khác tham số đầu vào thì đó gọi là method overloading.*
- Ví dụ, định nghĩa phương thức ***max*** với các tham số kiểu ***int***. Nhưng nếu cần phải tìm ***max*** của hai số ***double***? Giải pháp là tạo ra một phương thức khác có cùng tên nhưng với các tham số có kiểu ***double***. Đó gọi là **nạp chồng phương thức**.

```java
/* Return the max of two int values */
public static int max(int num1, int num2) {
    if (num1 > num2){
        return num1;
    }
    return num2;
}
/* Find the max of two double values */
public static double max(double num1, double num2) {
    if (num1 > num2){
        return num1;
    }
    return num2;
}
```

5.6/ Nạp chồng (overloading) phương thức

- *Nếu 2 method có cùng tên nhưng khác tham số đầu vào thì đó gọi là method overloading.*
- Ví dụ, định nghĩa phương thức ***max*** với các tham số kiểu ***int***. Nhưng nếu cần phải tìm ***max*** của hai số ***double***? Giải pháp là tạo ra một phương thức khác có cùng tên nhưng với các tham số có kiểu ***double***. Đó gọi là **nạp chồng phương thức**.

```
    /* Return the max of three double values */
    public static double max(double num1,double num2,double num3) {
        return max(max(num1, num2), num3);
    }
    public static void main(String[] args) {
        System.out.println("The maximum of 3 and 4 is " + max(3, 4));
        System.out.println("The maximum of 3.0 and 5.4 is " + max(3.0, 5.4));
        System.out.println("The maximum of 3.0, 5.4, and 10.14 is " + max(3.0, 5.4, 10.14));
    }
}
```

5.6/ Nạp chồng (overloading) phương thức

- *Ví dụ.*

```java
// cho integer
public static int minFunction(int n1, int n2) {
    int min;
    if (n1 > n2)
        min = n2;
    else
        min = n1;

    return min;
}
// cho double
public static double minFunction(double n1, double n2) {
    double min;
    if (n1 > n2)
        min = n2;
    else
        min = n1;

    return min;
}
```

```java
public static void main(String[] args) {
    int a = 11;
    int b = 6;
    double c = 7.3;
    double d = 9.4;
    int result1 = minFunction(a, b);
    // cung ten ham voi tham so khac nhau
    double result2 = minFunction(c, d);
    System.out.println("Gia tri nho nhat = " + result1);
    System.out.println("Gia tri nho nhat = " + result2);
}
```

5.6/ Nạp chồng (overloading) phương thức

- *Ví dụ.*

```
public static int min(int a, int b) {
  if (a <= b){
    return a;
  } else {
    return b;
  }
}

public static int min(int a, int b, int c) {
  if (a <= b && a <= c){
    return a;
  } else if (b <= a && b <= c){
    return b;
  } else {
    return c;
  }
}
```

```
public static void main(String[] args) {
  int a = 5, b = 3;
  System.out.println(min(a, b, -1));
  System.out.println(min(a, b));
}
```

5.7/ Lời gọi hàm mơ hồ (ambiguous)

- Khi nạp chồng phương thức, có thể có nhiều hơn 1 kết quả trùng khớp cho 1 lời gọi hàm. Hiện tượng này gọi là lời gọi hàm mở hồ và làm chương trình sẽ gặp lỗi biên dịch.

```
public static double max(int num1, double num2) {
    if (num1 > num2){
        return num1;
    }
    return num2;
}
public static double max(double num1, int num2) {
    if (num1 > num2){
        return num1;
    }
    return num2;
}
public static void main(String[] args) {
    //ambiguous both method max
    System.out.println("The maximum of 3 and 4 is " + max(3, 4));
}
```

Câu báo lỗi khi biên dịch

```
Exception in thread "main" java.lang.RuntimeException: Uncompilable source code -
reference to max is ambiguous both method max(int,double)
```

Exception in thread "main" java.lang.RuntimeException: Uncompilable source code - reference to max is ambiguous both method max(int,double)

5.8/ Phương thức khởi tạo (Constructor)

- Một constructor khởi tạo một đối tượng khi nó được tạo. Nó có cùng tên với lớp của nó và cú pháp tương tự như một phương thức. Tuy nhiên, các constructor không có kiểu trả về rõ ràng.
- Một nét đặc trưng là, chúng ta sẽ sử dụng một constructor để cung cấp các giá trị khởi tạo tới các biến instance được định nghĩa bởi lớp, hoặc để thực thi bất kỳ thủ tục khởi đầu nào khác được yêu cầu để tạo một đối tượng theo mẫu.
- Tất cả các lớp đều có các constructor, dù chúng ta có hay không định nghĩa nó, bởi vì Java tự động cung cấp một constructor mặc định mà khởi tạo tất cả biến thành viên về zero. Tuy nhiên, một khi ta định nghĩa constructor cho riêng mình, thì constructor mặc định sẽ không còn được sử dụng nữa.

5.8/ Phương thức khởi tạo (Constructor)

- Ví dụ.

```
class MyClass {
    int x;

    // Sau day la constructor
    MyClass() {
        x = 10;
    }
}
```

```
public class ConsDemo {

    public static void main(String args[]) {
        MyClass t1 = new MyClass();
        MyClass t2 = new MyClass();
        System.out.println(t1.x + " " + t2.x);
    }
}
```

5.8/ Phương thức khởi tạo (Constructor)

- Ví dụ.

```
class MyClass {
    int x;

    // Sau day la constructor
    MyClass(int i ) {
        x = i;
    }
}
```

```
public class ConsDemo {

    public static void main(String args[]) {
        MyClass t1 = new MyClass( 10 );
        MyClass t2 = new MyClass( 20 );
        System.out.println(t1.x + " " + t2.x);
    }
}
```

5.9/ Đệ Quy (Recursion)

Đệ quy là 1 khái niệm thường dùng trong toán học để chỉ việc định nghĩa một biểu thức mới dựa trên chính nó nhưng với giá trị thay đổi. Hãy xét tình huống tính tổng các số từ 1 đến n bằng phương pháp quy nạp

```
Ta có: s(n) = 1 + 2 + 3 + … + n
và công thức trên có thể biểu diễn ở dạng sau
s(n)   = s(n-1) + n
s(n-1) = s(n-2) + n-1
. . . . .
s(2)   = s(1) + 2
s(1)   = 1
```

5.9/ Đệ Quy (Recursion)

Trong ngôn ngữ Java, cơ chế đệ quy được trang bị cho các hàm được định nghĩa trong chương trình để cho phép 1 hàm có thể gọi lại chính nó một cách gián tiếp (hay trực tiếp)

Ví dụ 1: vòng lặp vô tận

```
public class RecursionExample1 {
    static void p() {
        System.out.println("hello");
        p();
    }

    public static void main(String[] args) {
        p();
    }
}
```

Kết quả

```
hello
hello
...
Exception in thread "main" java.lang.StackOverflowError
```

5.9/ Đệ Quy (Recursion)

Ví dụ 2: vòng lặp có hạn

```
public class RecursionExample2 {
    static int count = 0;

    static void p() {
        count++;
        if (count <= 5) {
            System.out.println("hello " + count);
            p();
        }
    }

    public static void main(String[] args) {
        p();
    }
}
```

Kết quả

```
hello 1
hello 2
hello 3
hello 4
hello 5
```

5.9/ Đệ Quy (Recursion)

Ví dụ 3: tính giai thừa

```
public class RecursionExample3 {
    static int factorial(int n) {
        if (n == 1)
            return 1;
        else
            return (n * factorial(n - 1));
    }

    public static void main(String[] args) {
        System.out.println("Giai thừa của 5 là: " + factorial(5));
    }
}
```

Kết quả:

```
Giai thừa của 5 là: 120
```

5.9/ Đệ Quy (Recursion)

Ví dụ 3: tính giai thừa

Chương trình trên hoạt động như sau:

```
1  factorial(5)
2     factorial(4)
3        factorial(3)
4           factorial(2)
5              factorial(1)
6                 return 1
7              return 2*1 = 2
8           return 3*2 = 6
9        return 4*6 = 24
10    return 5*24 = 120
```

5.9/ Đệ Quy (Recursion)

Ví dụ 4: dãy số Fibonacci

```
public class RecursionExample4 {
    static int n1 = 0, n2 = 1, n3 = 0;

    static void printFibo(int count) {
        if (count > 0) {
            n3 = n1 + n2;
            n1 = n2;
            n2 = n3;
            System.out.print(" " + n3);
            printFibo(count - 1);
        }
    }

    public static void main(String[] args) {
        int count = 15;
        System.out.print(n1 + " " + n2); // in 0 và 1
        printFibo(count - 2);// n-2 vì 2 số 0 và 1 đã được in ra
    }
}
```

Kết quả:

```
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

Ôn tập

- Hàm là gì, tại sao phải cần xây dựng hàm. Các lợi ích của việc sử dụng hàm trong chương trình
- Cú pháp khai báo, định nghĩa một hàm trong chương trình
- Có bao nhiêu loại hàm có thể định nghĩa trong chương trình ?
- Các hình thức truyền tham số
- Hiểu thế nào về tầm vực của biến ?
- Vấn đề đệ quy trong chương trình

Tham Khảo

- **Recursion**
  (https://introcs.cs.princeton.edu/java/23recursion/)
- **Recursion in Java**
  (https://www.javatpoint.com/recursion-in-java)