



# **Môn: Lập trình Hướng đối tượng (Object Oriented Programming)**

## **Chương 2. Những khái niệm cơ bản của Lập trình HĐT**

# Nội dung

- 2.1. Khái niệm đối tượng
- 2.2. So sánh classes và structures
- 2.3. Mô tả thành phần Private và Public của classes
- 2.4. Định nghĩa các hàm của classes
- 2.5. Phương pháp sử dụng các đối tượng và các hàm thành viên của classes
- 2.6. Các ngôn ngữ lập trình hướng đối tượng thông dụng hiện nay
- 2.7. Cách viết class trong Java

## 2.7. Cách viết class trong Java



2.7.1. Lớp trong Java

2.7.2. Khai báo định nghĩa lớp

2.7.3. Thuộc tính của lớp

2.7.4. Phương thức của lớp

2.7.5. Tạo đối tượng của lớp

2.7.6. this

2.7.7. Phương thức chồng overloading

2.7.8. Encapsulation (che dấu thông tin trong lớp)

## 2.7.1. Lớp trong Java

- Có thể xem lớp (class) như một khuôn mẫu (template) của đối tượng (object).
- Trong lớp bao gồm dữ liệu của đối tượng (fields hay properties) và các phương thức (methods) tác động lên thành phần dữ liệu đó gọi là các phương thức của lớp.
- Các đối tượng được xây dựng bởi các lớp nên được gọi là các thể hiện của lớp (class instance).
- *Các lớp được gom nhóm lại thành package.*

## 2.7.2. Khai báo định nghĩa lớp

```
class <ClassName>
{
    <kiểu dữ liệu> <field_1>; // thuộc tính của lớp
    <kiểu dữ liệu> <field_2>;
    constructor // hàm khởi tạo
    method_1 // phương thức của lớp
    method_2
}
```

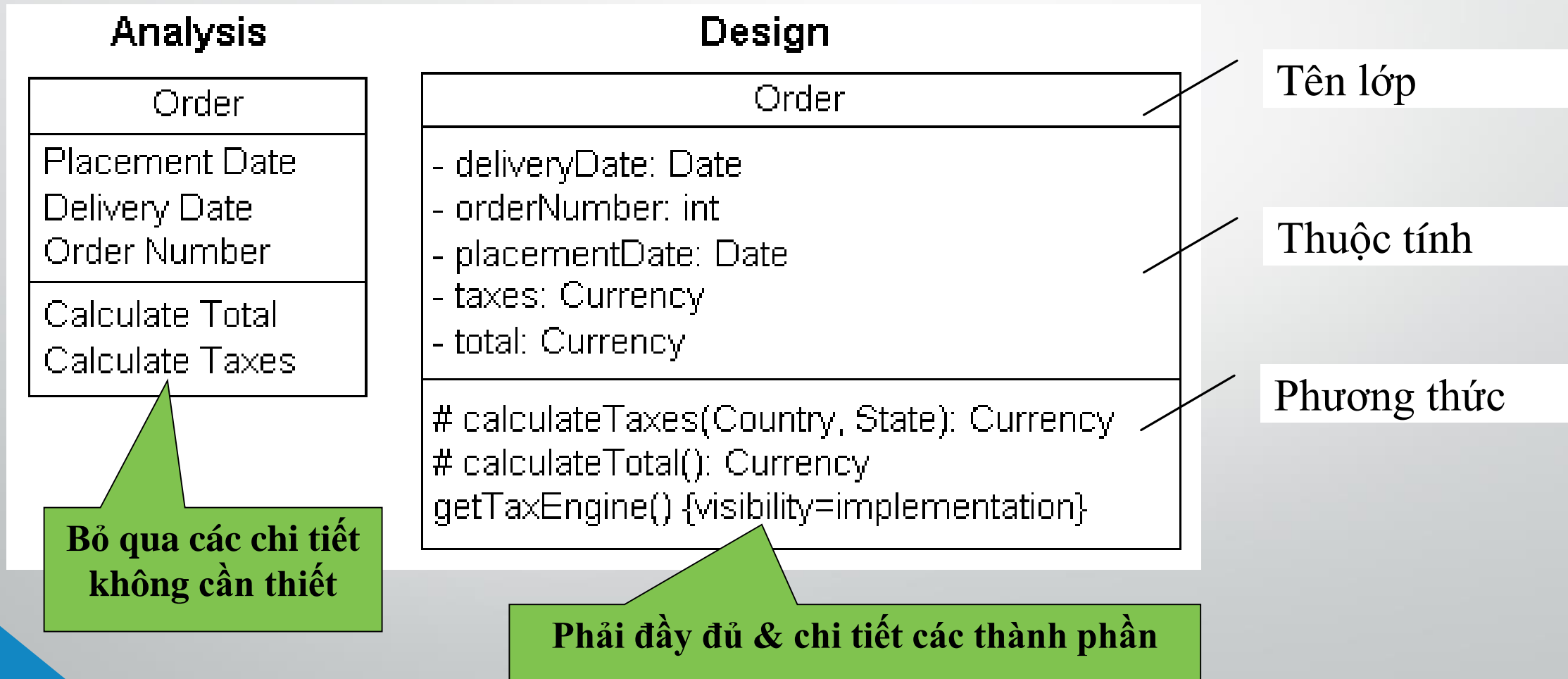
- **class**: là từ khóa của Java
- **ClassName**: là tên của lớp
- **field\_1, field\_2**: các thuộc tính, các biến, hay các thành phần dữ liệu của lớp.
- **constructor**: là hàm xây dựng, khởi tạo đối tượng lớp.
- **method\_1, method\_2**: là các phương thức/hàm thể hiện các thao tác xử lý, tác động lên các thành phần dữ liệu của lớp.

## 2.7.2. Khai báo định nghĩa lớp (tt)

- UML (Unified Model Language) là một ngôn ngữ dùng cho phân tích thiết kế hướng đối tượng (OOAD – Object Oriented Analysis and Design)
- UML thể hiện phương pháp phân tích hướng đối tượng nên không lệ thuộc ngôn ngữ LT.
- Dùng UML để biểu diễn 1 lớp trong Java
  - Biểu diễn ở mức phân tích (analysis)
  - Biểu diễn ở mức thiết kế chi tiết (design)

## 2.7.2. Khai báo định nghĩa lớp (tt)

- Ví dụ UML để biểu diễn 1 lớp trong Java



## 2.7.3. Thuộc tính của lớp

- Thuộc tính của lớp được khai báo bên trong lớp

```
class <ClassName>  
{ // khai báo những thuộc tính của lớp  
    //<quyền truy xuất> <kiểu dữ liệu> field1;  
    // ...  
}
```

- Quyền truy xuất của các đối tượng khác đối với thuộc tính của lớp:
  - **public:** có thể truy xuất từ tất cả các đối tượng khác.
  - **private:** một lớp không thể truy xuất vùng private của 1 lớp khác.
  - **protected:** vùng protected của 1 lớp chỉ cho phép bản thân lớp đó và những lớp thừa kế từ lớp đó truy cập đến.



## 2.7.3. Thuộc tính của lớp (tt)

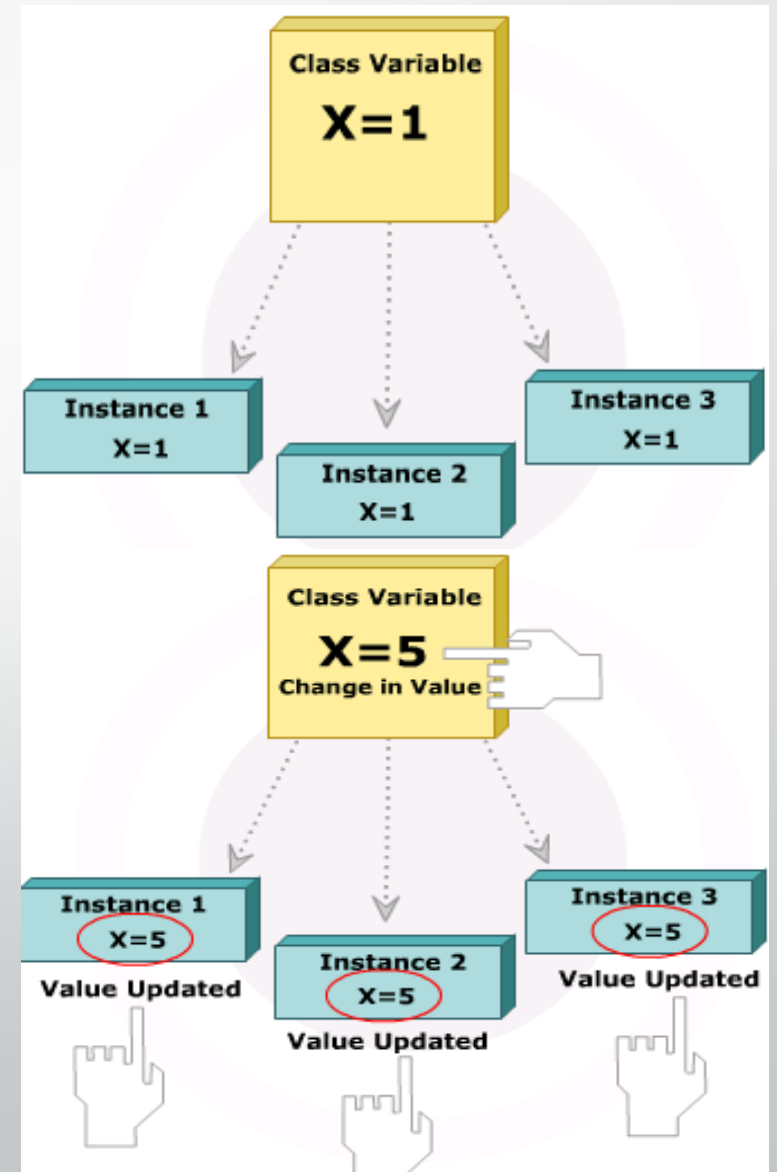
- Ví dụ: Lớp sinh viên

```
class SinhVien
{
    public String hoTen;
    private int    namSinh;
    protected String lopHoc;
    public static String tenTruong = "DHCN";
    // ...
}
```

## 2.7.3. Thuộc tính của lớp (tt)

Biến lớp (Class Variables) - (Biến tĩnh - Static Variables)

- Là biến được truy xuất mà không có sử dụng đối tượng của lớp đó.
- Khai báo dùng thêm từ khóa **static** keyword.
- Chỉ có 1 bản copy biến này được chia sẻ cho tất cả các đối tượng của lớp
  - Sự thay đổi giá trị của biến này sẽ ảnh hưởng tới tất cả các đối tượng của lớp.



## 2.7.3. Thuộc tính của lớp (tt)

### Ví dụ: Biến của lớp

```
public class Circle
```

```
{
```

```
    public float radius = 0;
```

```
    public static float PI = 3.14F; {
```

```
    public Circle(float r)
```

```
    {
```

```
        this.radius = r;
```

```
    }
```

```
}
```

```
public static void main(String[] args)
```

```
    Circle c1 = new Circle(10.3F);
```

```
    Circle c2 = new Circle(5.7F);
```

```
    System.out.println("C1 radius = " + c1.radius);
```

```
    System.out.println("C1 PI = " + c1.PI);
```

```
    System.out.println("\nC2 radius = " + c2.radius);
```

```
    System.out.println("C2 PI = " + c2.PI);
```

```
    System.out.println("\nPI = " + Circle.PI);
```

```
    c1.radius = 16.2F;
```

```
    c1.PI = 7.8F;
```

```
    System.out.println("C1 radius = " + c1.radius);
```

```
    System.out.println("C1 PI = " + c1.PI);
```

```
    System.out.println("\nC2 radius = " + c2.radius);
```

```
    System.out.println("C2 PI = " + c2.PI);
```

```
}
```

```
Console X
<terminated> Circle [Java Applicatio
C1 radius = 10.3
C1 PI = 3.14

C2 radius = 5.7
C2 PI = 3.14

PI = 3.14
C1 radius = 16.2
C1 PI = 7.8

C2 radius = 5.7
C2 PI = 7.8
```

## 2.7.4. Phương thức của lớp

- Có hai loại phương thức trong ngôn ngữ Java:
  - Hàm khởi tạo (Constructor)
- Các phương thức/hàm khác
  - Phương thức thể hiện (Instance Method)
  - Gọi phương thức và truyền tham số kiểu trị (Passing Arguments by Value).
  - Gọi phương thức và truyền tham số kiểu tham chiếu (Passing Arguments by Reference).
  - Phương thức tĩnh (Static Methods)
  - Phương thức tham số biến (Variable Argument Methods)

## 2.7.4. Phương thức của lớp (tt)

### Hàm khởi tạo (Constructor)

- Constructor là phương thức đặc biệt được gọi khi tạo object
- *Mục đích*: Khởi động trị cho biến instance của class.
- A constructor phải thỏa 2 điều kiện:
  - Cùng tên class
  - Không giá trị trả về
- Một lớp có thể có nhiều Constructors
- Nếu không viết Constructor, trình biên dịch tạo default constructor
  - Default constructor không thông số và không làm gì cả.

## 2.7.4. Phương thức của lớp (tt)

### Phương thức thể hiện (Instance Method)

- Là hàm định nghĩa trong lớp
- Định nghĩa hành vi của đối tượng
  - Ta có thể làm được gì với đối tượng này?
  - Những phương thức có thể áp dụng?
- Cung cấp cách thức truy xuất tới các dữ liệu **riêng** của đối tượng
- Truy xuất thông qua tên đối tượng

```
public class Rectangle
{
    private float length;
    private float width;

    public Rectangle()
    {
        length = 0;
        width = 0;
    }

    public Rectangle(float l, float w)
    {
        length = l;
        width = w;
    }

    public float area()
    {
        return length * width;
    }
}
```

## 2.7.4. Phương thức của lớp (tt)

Gọi phương thức và truyền tham số kiểu trị

- Các giá trị từ phương thức gọi (calling method) sẽ được truyền như đối số tới phương thức được gọi (called method).
- Bất kỳ sự thay đổi của đối số trong phương thức được gọi đều không ảnh hưởng đến các giá trị được truyền từ phương thức gọi.
- Các biến có giá trị kiểu nguyên thủy (primitive types int, float ...) sẽ được truyền theo kiểu này.

## 2.7.4. Phương thức của lớp (tt)

```
public class PassByValue
{
    public PassByValue()
    {
    }

    public void doubleValue(int a)
    {
        a = a * 2;
    }
}

public static void main (String [] args )
{
    int x = 10;
    PassByValue obj = new PassByValue();

    System.out.println("Before pass to method: " + x);

    obj.doubleValue(x);

    System.out.println("After pass to method: " + x);
}
```

Console X

<terminated> PassByValue [Java Application] C:\Program

Before pass to method: 10  
After pass to method: 10



## 2.7.4. Phương thức của lớp (tt)

Gọi phương thức và truyền tham số kiểu tham biến

- Sự thay đổi giá trị trong phương thức được gọi sẽ ảnh hưởng tới giá trị truyền từ phương thức gọi.
- Khi các tham chiếu được truyền như đối số tới phương thức được gọi, các giá trị của đối số có thể thay đổi nhưng tham chiếu sẽ không thay đổi.

## 2.7.4. Phương thức của lớp (tt)

```
public class MyClass
{
    private int a = 0;

    public int getA()
    {
        return a;
    }

    public void setA(int a)
    {
        this.a = a;
    }
}
```

```
public class PassByRef
{
    public void doSomething(MyClass o)
    {
        o.setA(100);
    }
}

public static void main(String[] args)
{
    PassByRef passByRef = new PassByRef();

    MyClass aObj = new MyClass();

    aObj.setA(2);
    System.out.println("Before: " + aObj.getA());

    passByRef.doSomething(aObj);

    System.out.println("After: " + aObj.getA());
}
```

Console X

<terminated> PassByRef [Java Applicati

**Before: 2**

**After: 100**

## 2.7.4. Phương thức của lớp (tt)

### Phương thức tĩnh (Static Methods)

- Là những phương thức được gọi thông qua tên Lớp (không cần đối tượng).
- Khai báo phương thức thêm từ khóa **static** .
- Chỉ có thể truy xuất 1 cách trực tiếp tới các biến tĩnh(static) và các phương thức tĩnh khác của lớp.
- Không thể truy xuất đến các phương thức và biến không tĩnh (non-static).

## 2.7.4. Phương thức của lớp (tt)

Việc sử dụng phương thức tĩnh

- Khi phương thức không truy xuất tới các trạng thái của đối tượng.
- Khi phương thức chỉ quan tâm đến các biến tĩnh.

```
public class Calculator
{
    public static int add(int a, int b)
    {
        return (a + b);
    }

    public int sub(int a, int b)
    {
        return (a - b);
    }
}
```

```
public static void main(String[] args)
{
    Calculator cal = new Calculator();
    int s = cal.sub(3, 5);

    System.out.println("Subtraction result: " + s);

    //calling static method
    int a = Calculator.add(3,5);

    System.out.println("Addition result: " + a);
}
```

```
Console
<terminated> Calculator [Java Application] C:\Program Fil
Subtraction result: -2
Addition result: 8
```

## 2.7.4. Phương thức của lớp (tt)

- Phương thức tham số biến. Phương thức này cho phép gọi phương thức với số tham số thay đổi.

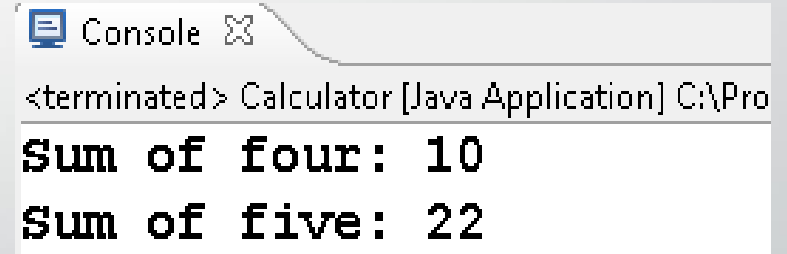
```
public class Calculator
{
    public int add(int... a)
    {
        int result = 0;

        for (int i = 0; i < a.length; i++)
        {
            result += a[i];
        }

        return result;
    }
}
```

```
public static void main(String[] args)
{
    Calculator cal = new Calculator();

    System.out.println("Sum of four: " + cal.add(1,2,3,4));
    System.out.println("Sum of five: " + cal.add(5,2,5,3,7));
}
```



Console

<terminated> Calculator [Java Application] C:\Pro

Sum of four: 10

Sum of five: 22

## 2.7.5. Tạo đối tượng của lớp

- Tạo đối tượng (object) dùng toán tử ***new***
- Cú pháp

*// gọi tới constructor mặc định*

*ClassName objectName = new ClassName();*

*// gọi tới constructor có tham số*

*ClassName objectName1 = new ClassName(ts1, ts2, ...);*

Truy xuất các thuộc tính và phương thức

- Khi tạo object bằng toán tử ***new***, vùng nhớ được cấp phát cho mỗi thuộc tính và phương thức của class.
- Để truy cập các thuộc tính và phương thức của đối tượng dùng toán tử dấu chấm (***dot operator***).

## 2.7.5. Tạo đối tượng của lớp (tt)

- Nếu một class không có constructor, trình biên dịch tạo ra constructor mặc định không có tham số.
- Nếu class có một hoặc nhiều constructor, bất kể tham số kiểu gì, trình biên dịch sẽ không thêm mặc định constructor nữa.
- Ví dụ: lớp không có constructor mặc định

# Ví dụ lớp Hình chữ nhật

```
class HìnhChuNhat
{
    double cdai, crong;
    public HìnhChuNhat (double cd, double cr)
    {
        cdai = cd;
        crong = cr;
    }
    public double DienTich()
    {
        return cdai * crong;
    }
    public double ChuVi()
    {
        return (cdai + crong)*2;
    }
}
```

```
HìnhChuNhat n; // khai báo đối tượng
// khởi tạo, cấp vùng nhớ bằng toán tử new
n = new HìnhChuNhat(3,6);
```



# Ví dụ lớp Hình chữ nhật (No default Constructor)

```
public class HCN_NoDefauConstructor
{
    double cdai, crong;
    public HCN_NoDefauConstructor (double cd, double cr)
    {
        cdai = cd;
        crong = cr;
    }
    public double DienTich()
    {
        return cdai * crong;
    }
    public double ChuVi()
    {
        return (cdai + crong)*2;
    }
}
```

```
// khai báo đối tượng
HCN_NoDefaultConstructor n, n2;
// khởi tạo, cấp vùng nhớ bằng toán tử new
// dùng Constructor 2 tham số
n = new HCN_NoDefaultConstructor(3,6);
// khởi tạo dùng constructor default
n2 = new HCN_NoDefaultConstructor();
```

# Ví dụ lớp Hình tròn

```
class HìnhTron
{
    double bkinh;
    public HìnhTron (double bk)
    {
        bkinh = bk;
    }
    public HìnhTron ()
    {
        bkinh = 0.0;
    }
    public double DienTich()
    {
        return Math.PI * bkinh * bkinh;
    }
    public double ChuVi()
    {
        return 2 * Math.PI * bkinh;
    }
}
```

```
HìnhTron n1, n2; // khai báo đối tượng
// khởi tạo, cấp vùng nhớ bằng toán tử new
// khởi tạo dùng constructor 1 tham số
n1 = new HìnhTron(3);
// khởi tạo dùng constructor không tham số
n2 = new HìnhTron();
```

# Ví dụ lớp Sinh Viên

```
import java.util.Date;
class SinhVien
{
    String MaSV, HoTen, NoiSinh;
    Date NgaySinh;
    String Lop, MonHoc;
    double diemlt, diemth, diemtb;
    public SinhVien (String ma, String ht, Date ngs,
                     String ns, String l, String mh, double lt, double th )
    {
        MaSV = ma; HoTen = ht;
        NoiSinh = ns; NgaySinh = ngs;
        Lop = l; MonHoc = mh;
        diemlt = lt;
        diemth = th;
    }
    public double DiemTB()
    {
        return (diemlt + diemth)/2;
    }
    public void InDiem ()
    {
        System.out.println(MaSV+ "\t" + HoTen+ "\t" + NgaySinh + "\t"
                           + NoiSinh+ "\t" + Lop + "\t" + MonHoc+ "\t" + diemtb);
    }
}
```

## 2.7.6. this

- Từ khóa this được dùng như biến đại diện cho object hiện tại.
- Để tránh lặp lại code, có thể có constructor gọi một constructor khác trong cùng class. Khi đó sử dụng từ khóa this để gọi constructor khác trong cùng class.
- Nếu một constructor gọi constructor khác bằng từ khóa this, thì từ khóa this phải là dòng lệnh đầu tiên trong constructor đó. (Nếu không, sẽ bị lỗi biên dịch).

## 2.7.6. this (tt)

- Dùng this trong hàm khởi tạo constructor

```
class HìnhChuNhatThis
{
    double cdai, crong;
    public HìnhChuNhatThis (double cd, double cr)
    {
        cdai = cd;
        crong = cr;
    }
    public HìnhChuNhatThis()
    {
        this(0,0);
    }
    public double DienTich()
    {
        return cdai * crong;
    }
    public double ChuVi()
    {
        return (cdai + crong)*2;
    }
}
```

## 2.7.6. this (tt)

- Dùng `this` đại diện cho đối tượng
- Đại diện cho đối tượng, dùng để truy xuất một thành phần của đối tượng *this.tênThànhPhần*.
- Khi tham số trùng với tên thuộc tính thì nhờ từ khóa *this* để phân biệt rõ thuộc tính với tham số.

## 2.7.6. this (tt)

vongtron.java \* SuDungVongTron.java |

```
1 class VONGTRON
2 { protected int x,y,r;
3   int getX() { return x; }
4   public void setX (int xx) { x=xx;}
5   public int getY() { return y; }
6   public void setY (int yy) { y=yy;}
7   public int getR() { return r; }
8   public void setR (int rr) { if (rr>=0) r=rr;}
9   public double getArea () { return Math.PI * r *r; }
10  private double getPerimeter () { return 2*Math.PI * r; }
11  public void OutData()
12  { System.out.println(" " + x + ", " + y + ", " + r);
13  }
14  public void setData (int x, int y, int r)
15  { this.
16  }
17 }
```

◆ setData (int, int, int)	void
◆ setR (int)	void
◆ setX (int)	void
◆ setY (int)	void
◆ toString ()	String
◆ wait (long, int)	void
◆ wait (long)	void
◆ wait ()	void
◆ x	int
◆ y	int

## 2.7.6. this (tt)

vongtron.java

SuDungVongTron.java

```
1 class VONGTRON
2 { protected int x,y,r;
3   int getX() { return x; }
4   public void setX (int xx) { x=xx;}
5   public int getY() { return y; }
6   public void setY (int yy) { y=yy;}
7   public int getR() { return r; }
8   public void setR (int rr) { if (rr>=0) r=rr;}
9   public double getArea () { return Math.PI * r *r; }
10  private double getPerimeter () { return 2*Math.PI * r; }
11  public void OutData()
12  { System.out.println(" " + x + ", " + y + ", " + r);
13  }
14  public void setData (int x, int y, int r)
15  { this.x=x; this.y=y; this.r=r;
16  }
17 }
```

Truy cập thành phần qua  
từ khóa this

Truy cập thành phần  
không qua từ khóa this



## 2.7.7. Phương thức chồng overloading

- Phương thức Overloading:
  - Các phương thức trong cùng class có cùng tên
  - Danh sách tham số phải khác nhau (includes the number, type, and order of the parameters)
- Trình biên dịch so sánh danh sách thông số thực để quyết định gọi phương thức nào.
- Kiểu giá trị trả về của phương thức không được tính vào dấu hiệu của overloading method
- Các constructors có thể được overloaded. Một overloaded constructor cho ta nhiều cách khác nhau để tạo ra một đối tượng mới.

## 2.7.7. Phương thức chồng overloading(tt)

Version 1

```
float tryMe (int x)
{
    return x + .375;
}
```

Version 2

```
float tryMe (int x, float y)
{
    return x*y;
}
```

Invocation

```
result = tryMe (25, 4.32)
```

Version 1

```
float tryMe (int x)
{
    return x + .375;
}
```

Version 2

```
int tryMe (int k)
{
    return k*k;
}
```

Not Overloading method

## 2.7.7. Phương thức chồng overloading(tt)

Ví dụ:

- The `println` method is overloaded:

```
println (String s)
```

```
println (int i)
```

```
println (double d) ...
```

- Những dòng lệnh sau sẽ gọi các versions khác nhau của phương thức `println`:

```
System.out.println ("The total is:");
```

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

## 2.7.8. Encapsulation

- Encapsulation là kỹ thuật làm cho các field trong một class thành private và cung cấp truy cập đến field đó thông qua public method.
- Encapsulation còn được gọi là che dấu dữ liệu.
- Encapsulation là một trong bốn khái niệm cơ bản của OOP.
- In Java, hiện thực encapsulation bằng cách sử dụng phù hợp các bộ từ truy xuất (`visibility modifiers`)
- Java có 3 bộ từ cho thành phần (`visibility modifiers`): `public`, `private`, `protected`

## 2.7.8. Encapsulation (tt)

### ★ Visibility Modifiers

- Thành phần của class được khai báo có bổ từ `public` visibility được truy cập ở bất cứ đâu.
- Thành phần được khai báo `private` visibility chỉ được truy cập bên trong class, bên ngoài lớp không truy xuất được.
- Thành phần được khai báo mặc định (không có visibility modifier) được truy cập bởi bất cứ class nào bên trong cùng một gói (package).

## 2.7.8. Encapsulation (tt)

- Dữ liệu nên được định nghĩa với bổ từ `private`
  - Dữ liệu `private` chỉ có thể được truy cập bởi các phương thức của class.
- Các method có thể được định nghĩa `public` hoặc `private`
  - `public` method: cung cấp dịch vụ cho lớp khác dùng class này, phương thức này có thể gọi là: *service methods*
  - `private` method: không thể được gọi từ bên ngoài class. Mục đích duy nhất của `private` method là để giúp cho những phương thức khác trong cùng một class để làm công việc của nó, các phương thức này còn gọi là phương thức hỗ trợ (*support methods*).

