



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

Chương 1. Tổng quan về cách tiếp cận hướng đối tượng

Nội dung

- 1.1. Phương pháp tiếp cận của lập trình truyền thống
- 1.2. Phương pháp tiếp cận hướng đối tượng
- 1.3. So sánh sự khác biệt giữa hai cách tiếp cận
- 1.4. Xu hướng phát triển của lập trình hướng đối tượng

1.1. Phương pháp tiếp cận của LT truyền thống

- Lập trình tuyến tính
 - Đơn giản: tuần tự từ
 - Đơn luồng: chỉ một luồng xử lý

$S =$

$X = Z$
 $Y = Z$
 $Z = X + Y$

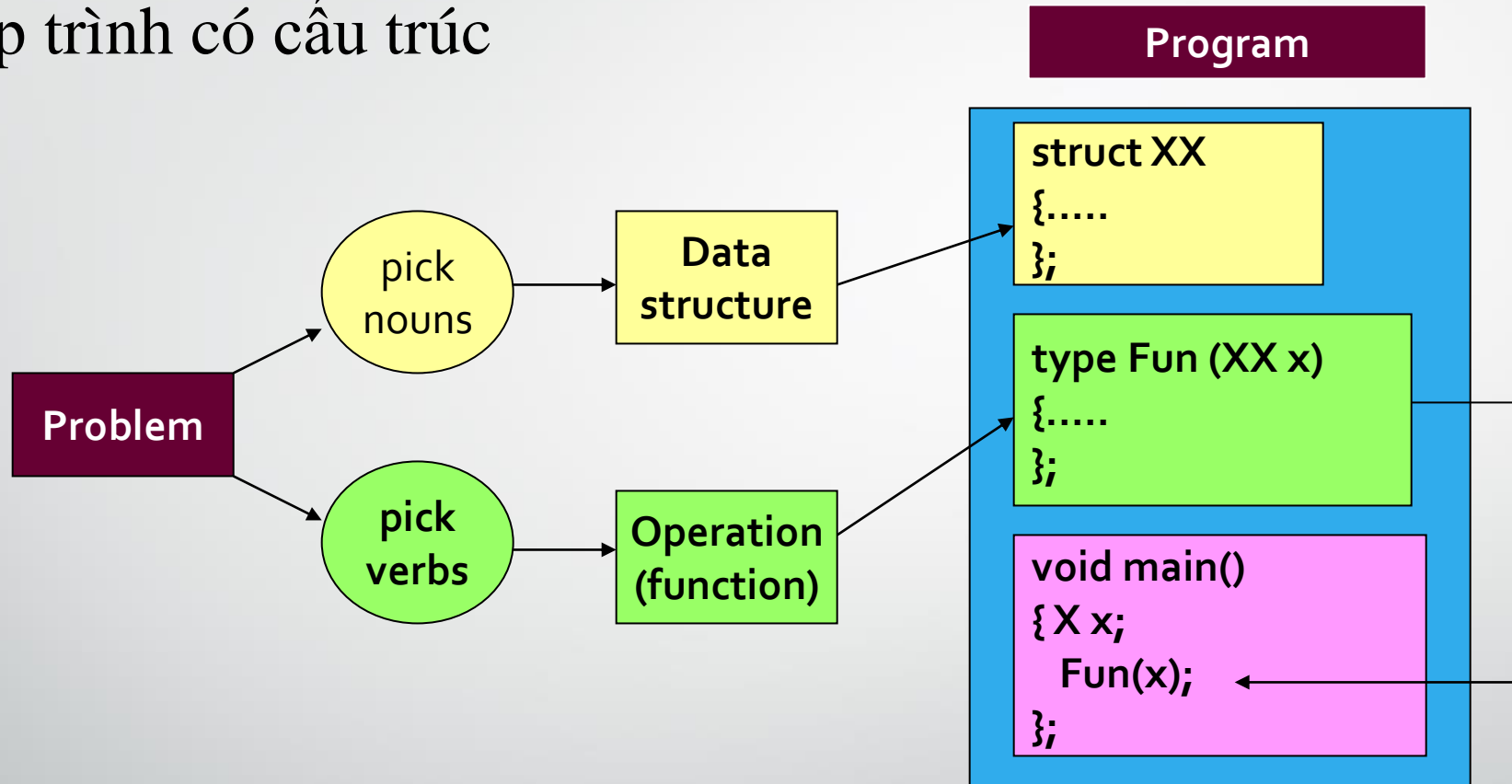
1.1. PP tiếp cận của LT truyền thống (tt)

Lập trình có cấu trúc

- Chia nhỏ thành chương trình con
 - Chương trình sẽ gọi chương trình con theo kịch bản định trước
 - Mỗi chương trình con sẽ được triệu gọi nhiều lần
 - Chương trình con được triệu gọi bất kỳ, không theo thứ tự khai báo
- **Chương trình = cấu trúc dữ liệu + giải thuật**
- Chương trình dễ đọc dễ hiểu
- Tư duy giải thuật rõ ràng
- Khi thay đổi cấu trúc thì giải thuật cũng thay đổi theo
- Phù hợp với phạm vi trong mỗi module, không phù hợp với chương trình có nhiều module, gọi module sẽ khó quản lý

1.1. PP tiếp cận của LT truyền thống (tt)

Lập trình có cấu trúc



1.1. PP tiếp cận của LT truyền thống (tt)

Hạn chế của lập trình truyền thống

- Lập trình hướng cấu trúc đã rất phổ biến trong những năm 80 và đầu những năm 90, nhưng do những hạn chế và những nhược điểm rõ ràng khi lập trình hệ thống lớn, lập trình hướng cấu trúc đã dần bị thay thế cho phương pháp lập trình hướng đối tượng.
- Hiện nay, những ngôn ngữ lập trình hướng cấu trúc chỉ còn được sử dụng để dạy học và lập trình những chương trình nhỏ mang tính chất cá nhân.
- Trong thương mại, phương pháp lập trình truyền thống đã không còn được dùng đến nhiều.

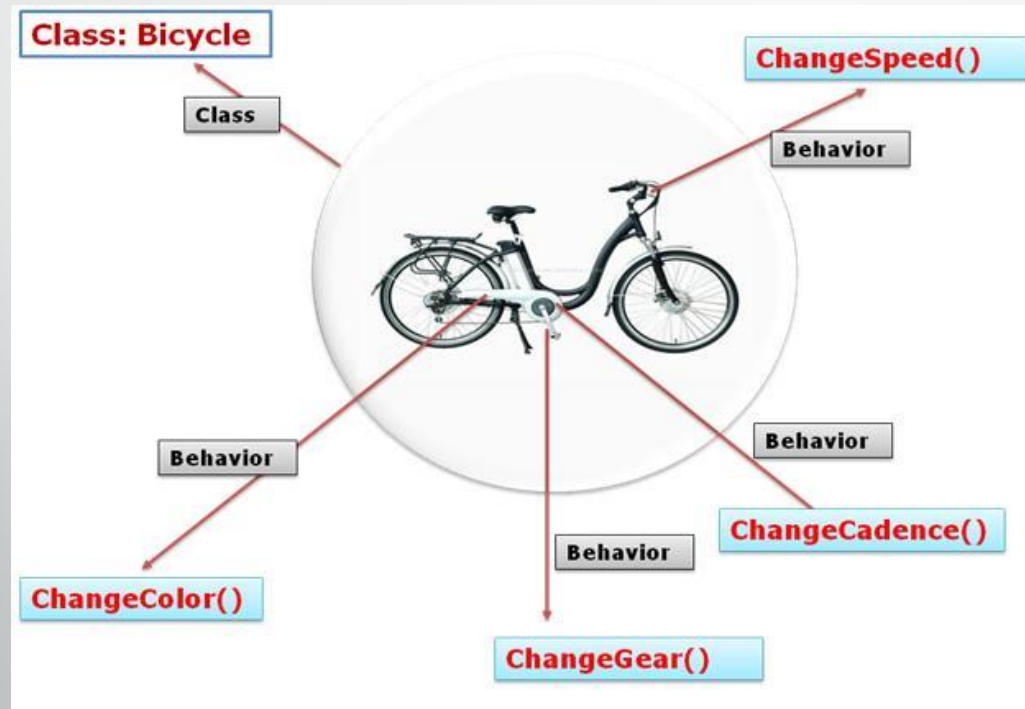
1.2. Phương pháp tiếp cận hướng đối tượng

- OOP – Object Oriented Programming.
- Chương trình là sự hoạt động của các đối tượng → Giống tự nhiên.
- Đối tượng thực thi một hoạt động tức là đối tượng thực hiện một *hành vi* mà đối tượng này có khả năng.
- Một chương trình là một trật tự các lời yêu cầu đối tượng thực hiện hành vi của mình.
- Đóng gói dữ liệu nên hạn chế việc truy cập tự do (private trong hướng đối tượng, chỉ các phương thức thuộc lớp mới truy cập được) làm không kiểm soát được việc cập nhật dữ liệu
- Sử dụng lại mã nguồn, hạn chế việc viết lại mã nguồn

1.2. Phương pháp tiếp cận hướng đối tượng (tt)

Sơ lược về OOP

- Đối tượng (object): dữ liệu + hành vi.
- Đối tượng phải thuộc một lớp (class).
- Một nhóm đối tượng được biểu diễn bởi Lớp(Class)
- **Lớp = data (biến, thuộc tính) + methods (code).**



1.2. Phương pháp tiếp cận hướng đối tượng (tt)

Đặc trưng (tính chất)

- Trừu tượng (Abstraction)
- Đóng gói/Che dấu thông tin (Encapsulation - Information hiding)
- Thừa kế (Inheritance)
- Đa hình (Polymorphism)

Ưu điểm

- Khi thay đổi cấu trúc dữ liệu thì không cần thay đổi mã nguồn của đối tượng khác
- Có thể sử dụng lại mã nguồn, tiết kiệm tài nguyên
- PP tiếp cận HĐT phù hợp với các dự án phần mềm lớn, phức tạp

1.2. Phương pháp tiếp cận hướng đối tượng (tt)

Phương pháp phân tích và thiết kế theo hướng đối tượng

- Phân tích: ngôn ngữ đặc tả mô hình UML(Unified Modeling Language)
- Thiết kế: dựa trên các mô hình phân tích, cài đặt ứng dụng/chương trình theo một ngôn ngữ lập trình hướng đối tượng.
- Các bước phân tích thiết kế hướng đối tượng
 - Mô tả bài toán
 - Đặc tả yêu cầu
 - Trích chọn đối tượng
 - Mô hình hóa lớp đối tượng
 - Thiết kế tổng quan
 - Thiết kế chi tiết

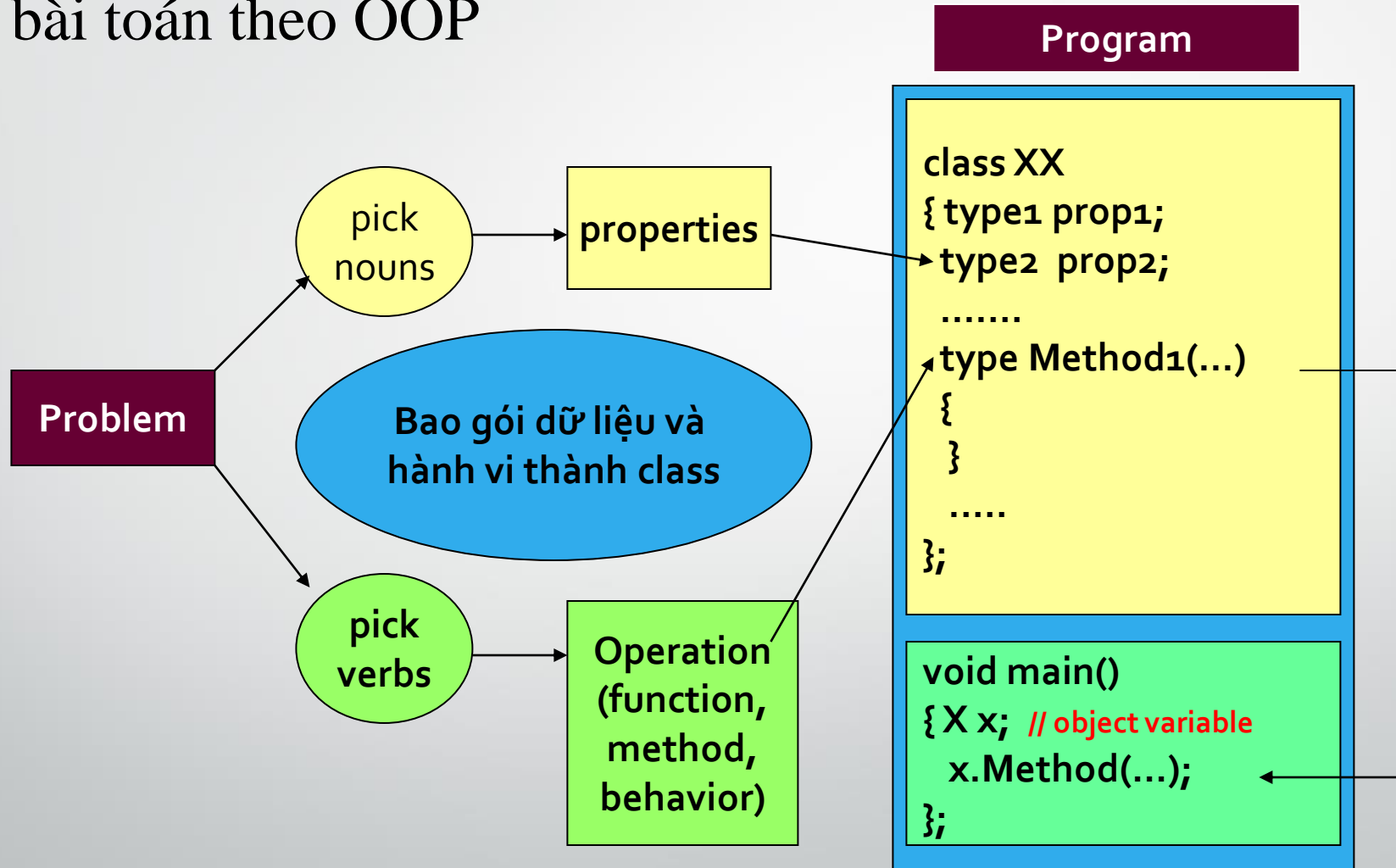
1.2. Phương pháp tiếp cận hướng đối tượng (tt)

Ngôn ngữ lập trình OOP

- C++ (Borland C++, Visual C++)
- Java
- C# (C sharp)
- Visual Basic.
-
- C++, MS VC++: hỗ trợ cả POP (lập trình cấu trúc Procedure Oriented Programming) lẫn OOP → Lai OOP. Hỗ trợ đa thừa kế. Đối tượng là biến của chương trình. Hàm main() là POP.
- Java (Sun), C# (Microsoft): chỉ hỗ trợ OOP, hàm main phải nằm trong một lớp. Chỉ hỗ trợ đơn thừa kế.

1.2. Phương pháp tiếp cận hướng đối tượng (tt)

Giải bài toán theo OOP



1.3. So sánh sự khác biệt giữa 2 cách tiếp cận

Phương pháp

- Phương pháp hướng đối tượng đi từ chi tiết đến trừu tượng hóa ở mức cao
- Phương pháp cấu trúc đi từ tổng quan rồi chia nhỏ thành các bài toán con, cụ thể hơn.

Về hạn chế truy xuất dữ liệu (đóng gói)

- Phương pháp hướng đối tượng cho phép ẩn dữ liệu và hạn chế truy cập dữ liệu. Cho phép sử dụng lại mã nguồn để tiết kiệm tài nguyên.
- Phương pháp cấu trúc có ràng buộc giữa cấu trúc dữ liệu và các thủ tục hoặc hàm đi kèm.

1.3. So sánh sự khác biệt ... (tt)

Ưu nhược điểm

- Phương pháp hướng đối tượng:
 - Hạn chế truy cập từ bên ngoài
 - Tiết kiệm tài nguyên
 - Khó theo dõi luồng dữ liệu
 - Không thiên hướng về giải thuật
- Phương pháp cấu trúc:
 - Thiên hướng về giải thuật
 - Dễ theo dõi luồng giải thuật
 - Khi thay đổi cấu trúc thường phải viết lại giải thuật
 - Chương trình đơn giản dễ hiểu

1.4. Xu hướng phát triển của lập trình HĐT

- Hướng thành phần
- Hướng Agent
- Hướng Aspect

1.4. Xu hướng phát triển của LT HĐT (tt)

Hướng thành phần

- Xuất phát từ lập trình hướng đối tượng
- Giải quyết bài toán từ các thành phần có tính độc lập với nhau, mỗi thành phần đảm nhiệm một công việc nhất định
- Các thành phần được lắp ghép với nhau để thỏa các yêu cầu phần mềm
- Ưu điểm của lập trình theo hướng thành phần
 - Các lập trình viên chia sẻ các thành phần
 - Tiết kiệm công sức lập trình dựa trên các thành phần có sẵn

1.4. Xu hướng phát triển của LT HĐT (tt)

Hướng Agent

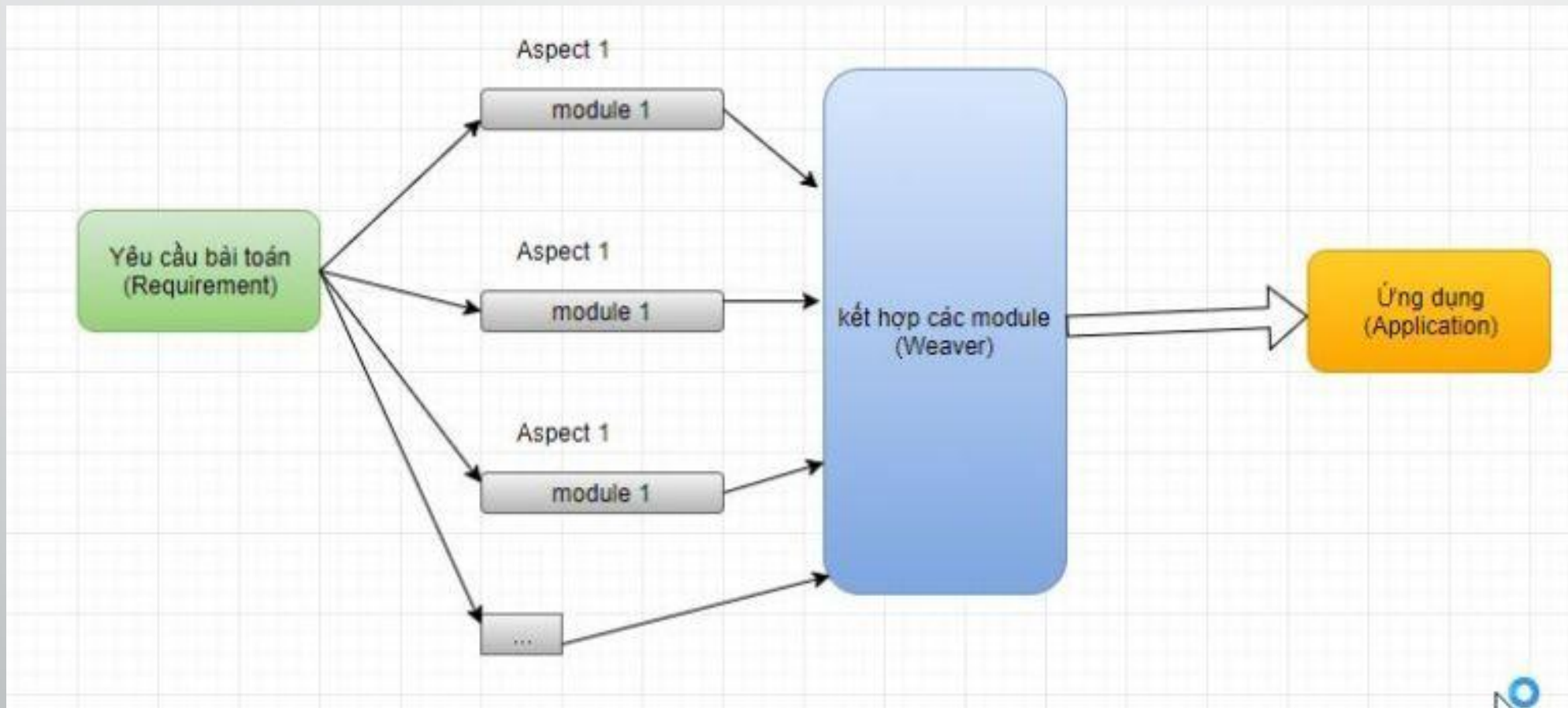
- Agent là một hệ tính toán hoàn chỉnh hay chương trình được đặt trong một môi trường nhất định, có khả năng hoạt động một cách tự chủ và mềm dẻo trong môi trường đó nhằm đạt được mục đích đã thiết kế.
- Lập trình hướng agent giống như hướng thành phần nhưng có mức trừu tượng cao hơn.
- Các agent là các thành phần có khả năng hoạt động độc lập
- Các agent có thể chủ động liên lạc với các agent khác khi cần

1.4. Xu hướng phát triển của LT HĐT (tt)

Hướng Aspect

- Aspect Oriented Programming (AOP) – lập trình hướng khía cạnh: là một kỹ thuật lập trình (kiểu như lập trình hướng đối tượng) nhằm phân tách chương trình thành các module riêng rẽ, phân biệt, không phụ thuộc nhau.
- Khi hoạt động, chương trình sẽ kết hợp các module lại để thực hiện các chức năng nhưng khi sửa đổi 1 chức năng thì chỉ cần sửa 1 module.

1.4. Xu hướng phát triển của LT HĐT (tt)



1.4. Xu hướng phát triển của LT HĐT (tt)

- **Ưu điểm:**

- Thiết kế đơn giản: “You aren’t gonna need it (YAGNI)” – chúng ta chỉ cài đặt những thứ chúng ta thực sự cần mà không bao giờ cài đặt trước.
- Cài đặt chương trình một cách trong sáng: mỗi một module chỉ làm cái mà nó cần phải làm.
- Tái sử dụng dễ dàng.

- **Nhược điểm:**

- Khái niệm khá trừu tượng, độ trừu tượng của chương trình cao
- Luồng chương trình phức tạp



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

Analysis

Order
Placement Date Delivery Date Order Number
Calculate Total Calculate Taxes

**Bỏ qua các chi tiết
không cần thiết**

Design

Order
- deliveryDate: Date - orderNumber: int - placementDate: Date - taxes: Currency - total: Currency
calculateTaxes(Country, State): Currency # calculateTotal(): Currency getTaxEngine() {visibility=implementation}

Phải đầy đủ & chi tiết các thành phần

Tên lớp

Thuộc tính

Phương thức

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)

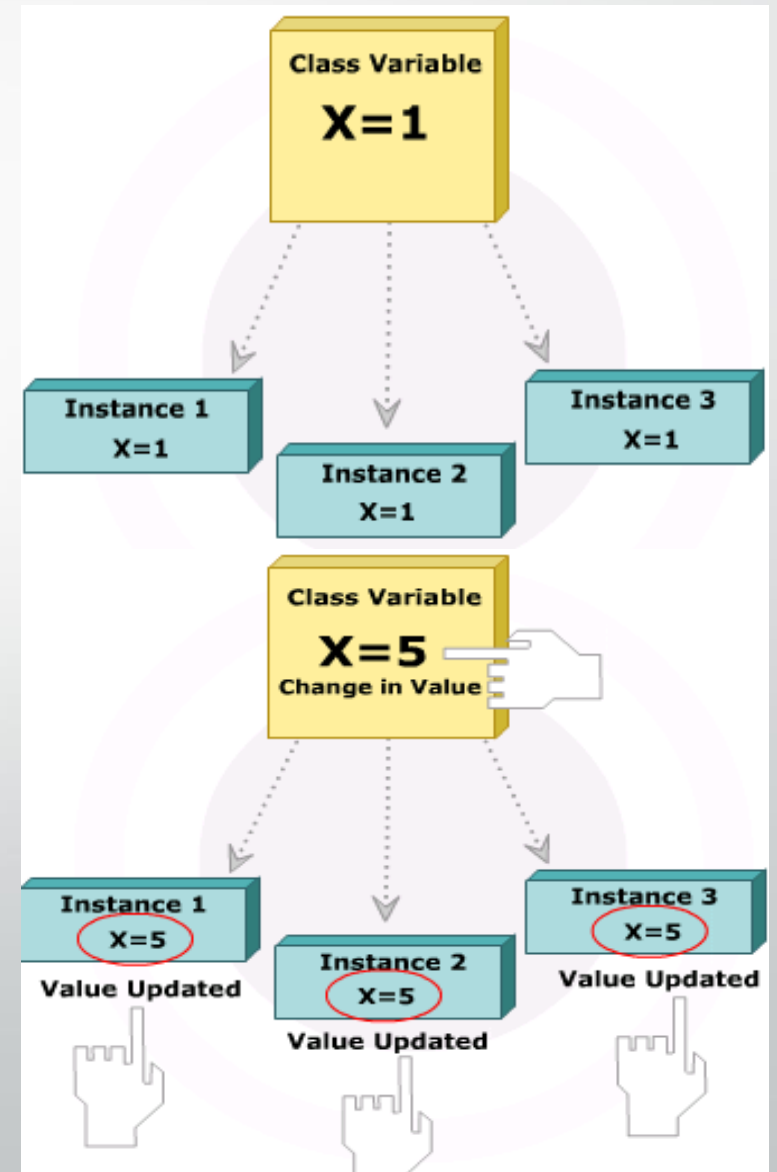
- Ví dụ: Lớp sinh viên

```
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);

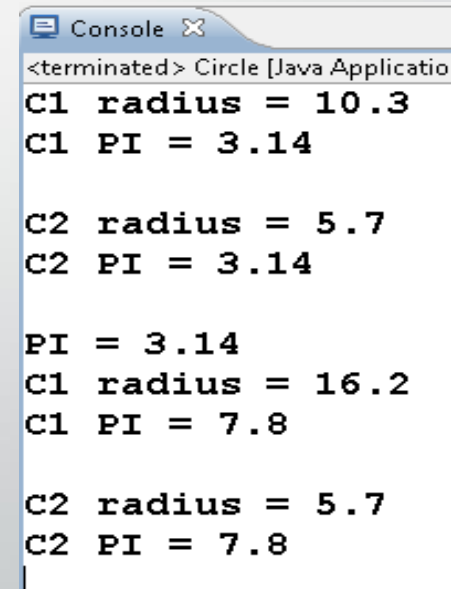
    System.out.println("\nC2 radius = " + c2.radius);
    System.out.println("C2 PI = " + c2.PI);

    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 Application]

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 sẽ 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*).



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

Chương 3. Giới thiệu Java

Nội dung

- 3.1. Lịch sử phát triển của Java
- 3.2. Đặc trưng của Java
- 3.3. Tổng quan lập trình Java
 - 3.3.1. Kiểu dữ liệu cơ bản
 - 3.3.2. Hằng, biến
 - 3.3.3. Toán tử, biểu thức
 - 3.3.4. Các cấu trúc lệnh trên Java (cấu trúc điều khiển, lặp)
- 3.4. Kiến trúc chương trình xây dựng trên Java
- 3.4. Case Study I (simple programs & language)

3.1. Lịch sử phát triển của Java

- 1991: được Sun Microsystems phát triển nhằm mục đích viết phần mềm điều khiển (phần mềm nhúng) cho các sản phẩm gia dụng
 - lúc đầu được đặt tên là Oak
- 1995: được phổ cập với sự phát triển mạnh mẽ của Internet thị trường phần mềm nhúng không phát triển mạnh
 - WWW bùng nổ (1993~)
- Hiện nay, được chấp nhận rộng rãi với tư cách là một ngôn ngữ (công nghệ) đa dụng
 - khả chuyển, an toàn
 - Hướng đối tượng, hướng thành phần

3.1. Lịch sử phát triển của Java (tt)

- Java là một công nghệ
- Java bao gồm
 - Ngôn ngữ lập trình
 - Môi trường phát triển
 - Môi trường thực thi và triển khai

3.1. Lịch sử phát triển của Java (tt)

Mục tiêu của Java

- Ngôn ngữ dễ dùng
 - Khắc phục nhiều nhược điểm của các ngôn ngữ trước đó
 - Hướng đối tượng
 - Rõ ràng
- Môi trường thông dịch
 - Tăng tính khả chuyển
 - An toàn
- Cho phép chạy nhiều tiến trình (threads)
- Nạp các lớp (classes) động vào thời điểm cần thiết từ nhiều nguồn khác nhau
 - Cho phép thay đổi động phần mềm trong khi hoạt động
- Tăng độ an toàn

3.1. Lịch sử phát triển của Java (tt)

Biên dịch và thông dịch

- Chương trình nguồn được biên dịch sang mã đích (bytecode)
- Mã đích (bytecode) được thực thi trong môi trường thông dịch (máy ảo)

Các dạng ứng dụng của Java

- Desktop applications - J2SE
 - Java Applications: ứng dụng Java thông thường trên desktop
 - Java Applets: ứng dụng nhúng hoạt động trong trình duyệt web
- Server applications - J2EE
 - JSP và Servlets
- Mobile (embedded) applications – J2ME

3.2. Đặc trưng của Java

- JVM (Java Virtual Machine) – máy ảo Java
- Cơ chế giải phóng bộ nhớ tự động
- Bảo mật chương trình

3.2. Đặc trưng của Java (tt)

JVM (Java Virtual Machine) – máy ảo Java

- Máy ảo phụ thuộc vào platform (phần cứng, OS)
- Cung cấp môi trường thực thi cho chương trình Java (độc lập với platform)
- Máy ảo đảm bảo an toàn cho hệ thống
- Máy ảo thông thường được cung cấp dưới dạng phần mềm
 - JRE - Java Runtime Environment
- Java platform: JVM + APIs

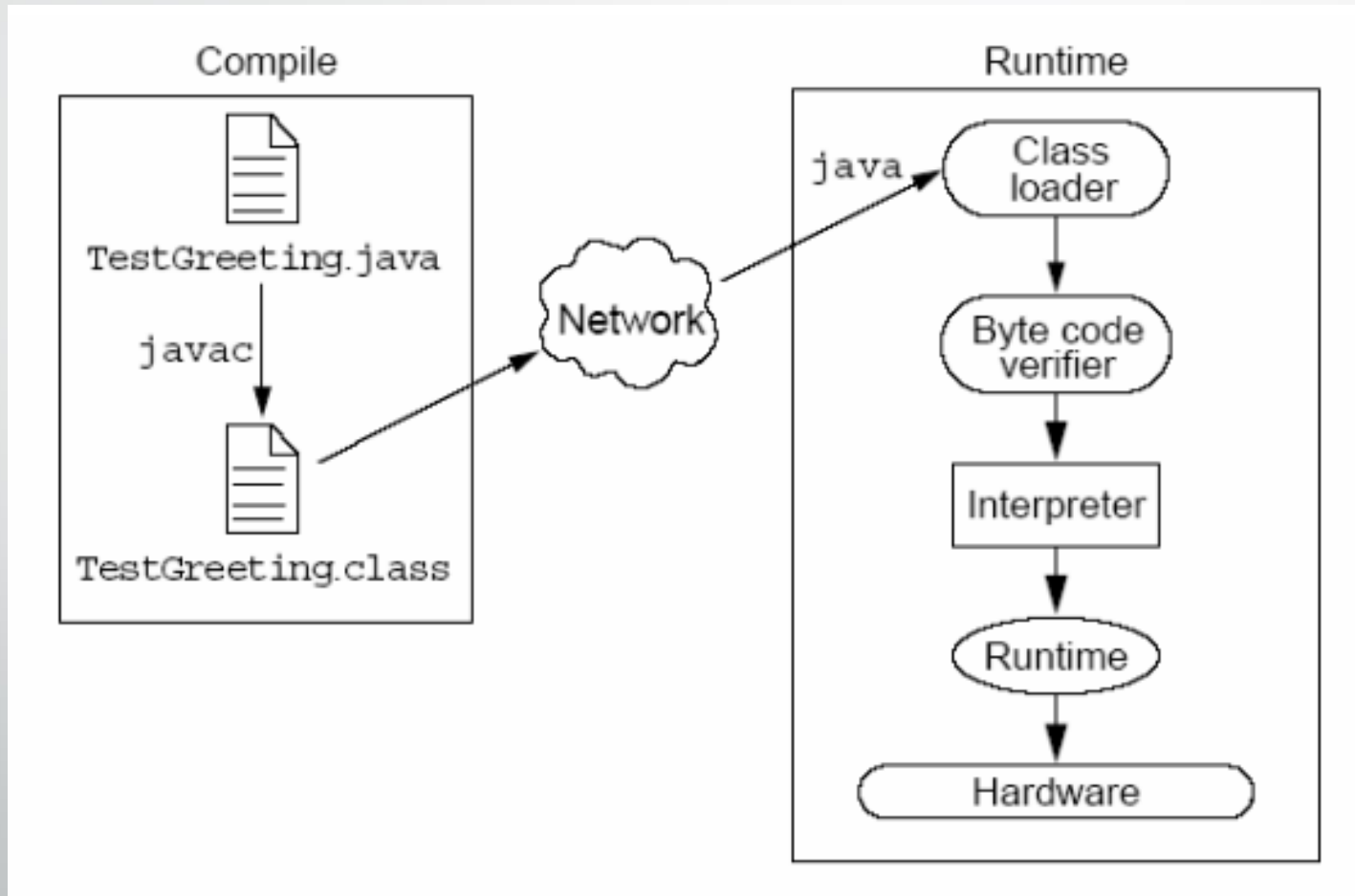
3.2. Đặc trưng của Java (tt)

Cơ chế giải phóng bộ nhớ tự động

- Java cung cấp một tiến trình mức hệ thống để theo dõi việc cấp phát bộ nhớ
- Garbage Collection
 - Đánh dấu và giải phóng các vùng nhớ không còn được sử dụng
 - Được tiến hành tự động
 - Cơ chế hoạt động phụ thuộc vào các phiên bản máy ảo

3.2. Đặc trưng của Java (tt)

Bảo mật chương trình – chống sao chép



3.2. Đặc trưng của Java (tt)

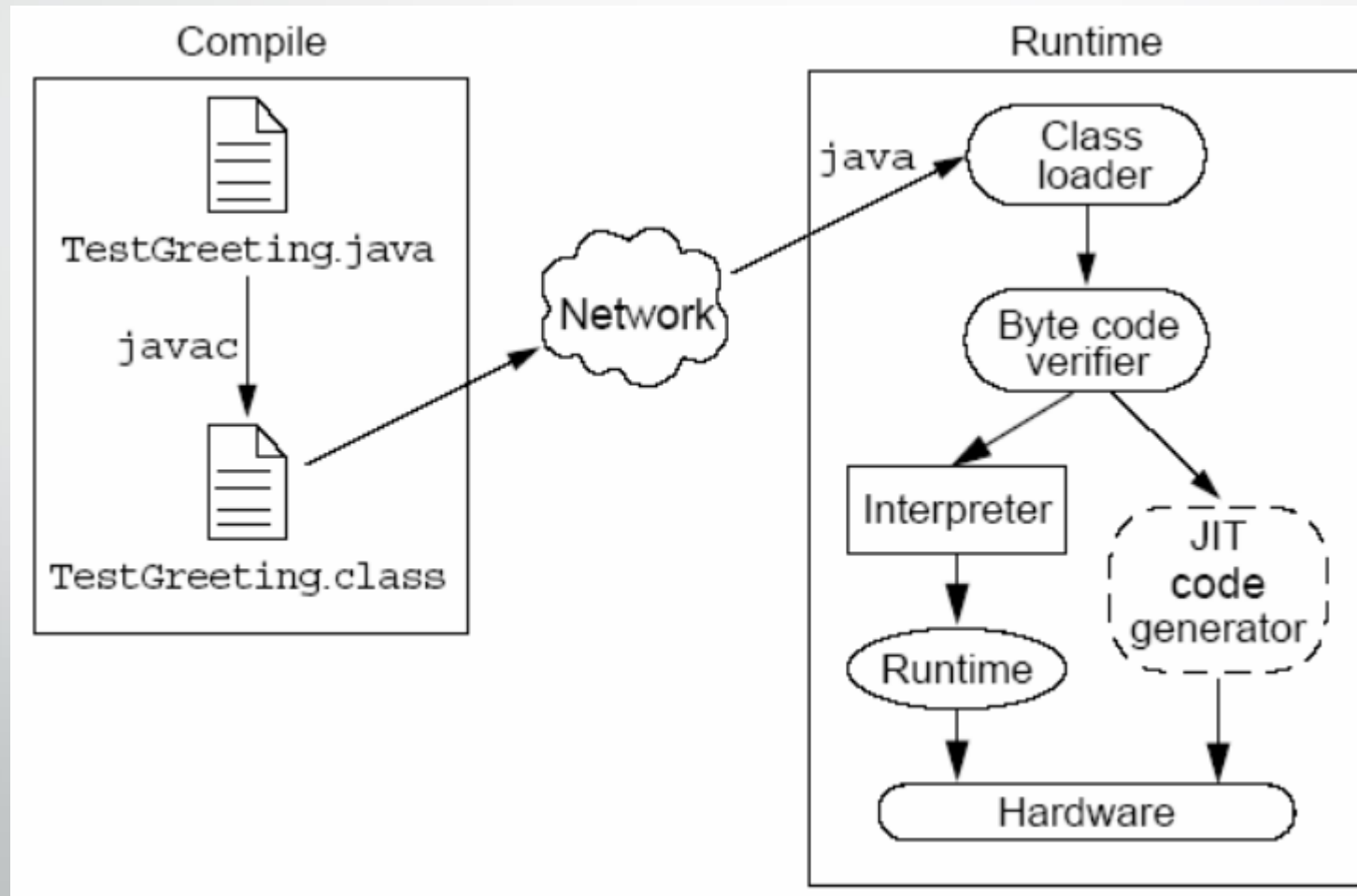
JDK Java Development Kit

- Môi trường phát triển và thực thi do Sun Microsystems cung cấp (<http://oracle.com/java>)
- Phiên bản hiện tại J2SDK 8.0 (1.8)
- Bao gồm
 - javac Chương trình dịch chuyển mã nguồn sang bytecode
 - java Bộ thông dịch: Thực thi java application
 - appletviewer Bộ thông dịch: Thực thi java applet mà không cần sử dụng trình duyệt như Netscape, hay IE, v.v.
 - javadoc Bộ tạo tài liệu dạng HTML từ mã nguồn và chú thích
 - jdb Bộ gỡ lỗi (java debugger)
 - javap Trình dịch ngược bytecode

3.2. Đặc trưng của Java (tt)

Công nghệ JIT

Just-In-Time Code Generator



3.2. Đặc trưng của Java (tt)

- Java Applications
 - Chương trình ứng dụng hoàn chỉnh
 - Giao diện dòng lệnh hoặc đồ họa
 - Được bắt đầu bởi phương thức (hàm) `main()` là phương thức `public static`
- Java Applets
 - Được nhúng trong một ứng dụng khác (web browser)
 - Có giao diện hạn chế (đồ họa)
 - Không truy cập được tài nguyên của client
 - Code nhúng vào trang Web: `Welcome.html`:

```
<html>
<applet code = "Welcome.class" width = "300"
height = "45"></applet>
</html>
```

3.3. Tổng quan lập trình Java

3.3.1. Kiểu dữ liệu cơ bản

3.3.2. Hằng, biến

3.3.3. Toán tử, Biểu thức

3.3.4. Các cấu trúc lệnh trên Java (cấu trúc điều khiển, lặp)

3.3.1. Kiểu dữ liệu cơ bản

Có 8 kiểu dữ liệu cơ bản trong Java.

- 4 kiểu biểu diễn số nguyên:
 - `byte`, `short`, `int`, `long`
- 2 kiểu biểu diễn số thực:
 - `float`, `double`
- 1 biểu diễn các ký tự:
 - `char`
- Và 1 biểu diễn cho giá trị luận lý (*true*, *false*):
 - `boolean`

3.3.2. Hằng, biến

Từ khóa

- Từ khóa cho các kiểu dữ liệu cơ bản : **byte, short, int, long, float, double, char, boolean**
- Từ khóa cho phát biểu lặp: **do, while, for, break, continue**
- Từ khóa cho phát biểu rẽ nhánh: **if, else, switch, case, default, break**
- Từ khóa đặc tả đặc tính một method: **private, public, protected, final, static, abstract, synchronized, volatile..**
- Literal value: **true, false, null**
- Từ khóa liên quan đến method: **return, void**
- Từ khóa liên quan đến package: **package, import**
- Từ khóa cho việc quản lý lỗi: **try, catch, finally, throw, throws**
- Từ khóa liên quan đến đối tượng: **new, extends, implements, class, instanceof, this, super**

3.3.2. Hằng, biến (tt)

Cách đặt tên

- Bắt đầu bằng ký tự, ký tự gạch dưới (underscore ‘_’) hay ký tự ‘\$’
- Sau đó là các ký tự ký số hay ‘_’, ‘\$’, không dùng các ký tự khác như: khoảng trống, ký hiệu phép toán
- Tên có tính chất phân biệt chữ thường chữ hoa (case-sensitive)

3.3.2. Hằng, biến (tt)

Khai báo và sử dụng các biến

- Biến là một giá trị có thể thay đổi khi chương trình thực thi.
- Khi biến được tạo sẽ xuất hiện một vùng nhớ để lưu trữ giá trị của biến.
- 3 đặc điểm của biến: *tên biến*, *giá trị khởi tạo*, *tầm vực (scope)*
- Scope của biến: khối chương trình mà biến có ý nghĩa (tham khảo được)
- Một biến phải được khai báo trước khi sử dụng (tên biến và kiểu dữ liệu). Một biến có thể được khởi tạo giá trị khi khai báo biến.

Kiểu dữ liệu

Tên biến

`int total;`

`int count, temp, result;`

`int sum = 0;`

`int base = 32, max = 149;`

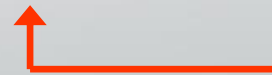
3.3.2. Hằng, biến (tt)

- Hằng số (constant)
 - Một hằng tương tự như biến như giá trị của nó luôn luôn không đổi.
 - Trình biên dịch sẽ phát sinh lỗi nếu ta cố tình thay đổi giá trị của hằng.
 - Trong Java, ta dùng `final` để khai báo hằng.

```
final int MIN_HEIGHT = 69;
```

- Phép gán làm thay đổi giá trị của một biến. Toán tử gán (=).

```
total = 55;
```



- Toán tử gán mở rộng `+=`, `-=`, `*=`, `/=`, `%=`

3.3.3. Toán tử, biểu thức (tt)

- *Các toán tử số học:*

Cộng +

Trừ -

Nhân *

Chia /

- *Toán tử tăng / giảm*

Chia lấy số dư %

- Toán tử tăng (++)

- Toán tử giảm (--)

- Câu lệnh **count++**; tương đương với **count = count + 1;**

3.3.3. Toán tử, biểu thức (tt)

- Toán tử so sánh (quan hệ)

== bằng

!= không bằng

< nhỏ hơn

> lớn hơn

<= nhỏ hơn hoặc bằng

>= lớn hơn hoặc bằng

- *Toán tử luận lý*

! : not

& & : and

| | : or

3.3.3. Toán tử, biểu thức (tt)

- Toán tử điều kiện

Cú pháp: *điều_kiện? biểu_thức1:biểu_thức2*

- Nếu *điều_kiện* là true, *biểu_thức1* được thực thi; Nếu là false, *biểu_thức2* được thực thi.
- Các biểu thức
 - Một biểu thức là một sự kết hợp giữa các toán tử và các toán hạng.
 - Nếu trong biểu thức có chứa số thực thì kết quả trả về số thực.
 - Lưu ý độ ưu tiên của các toán tử trong 1 biểu thức

3.3.3. Toán tử, biểu thức (tt)

- instanceof : toán tử kiểm tra 1 đối tượng có thuộc 1 lớp nào đó → true | false

```
class InstanceOfDemo
{
    public static void main (String args[])
    {
        InstanceOfDemo t = new InstanceofDemo();
        if ( t instanceof InstanceOfDemo)
            System.out.println(" Yes");
        else
            System.out.println(" NO");
    }
}
```

3.3.4. Các cấu trúc lệnh trên Java

Cấu trúc điều khiển – Rẽ nhánh

Cấu trúc if

```
if (Condition)
{
    Statements;
}
else
{
    Statement;
}
```

Cấu trúc switch

```
switch (Expression)
{
    case Cons1: Statements;
        break;
    case Cons2: Statements;
        break;
    . . .
    default : Statements;
}
```

3.3.4. Các cấu trúc lệnh trên Java

Cấu trúc điều khiển – Rẽ nhánh (tt)

- Thường một lệnh *break* được dùng ở cuối danh sách lệnh của mỗi case. Một cấu trúc switch có thể có một case *default*
- Kết quả của *biểu_thức* trong switch phải là kiểu số nguyên (byte, short, int, long) hoặc char.
- Không thể là boolean hoặc số thực (float, double)

3.3.4. Các cấu trúc lệnh trên Java (tt)

- Cấu trúc lặp

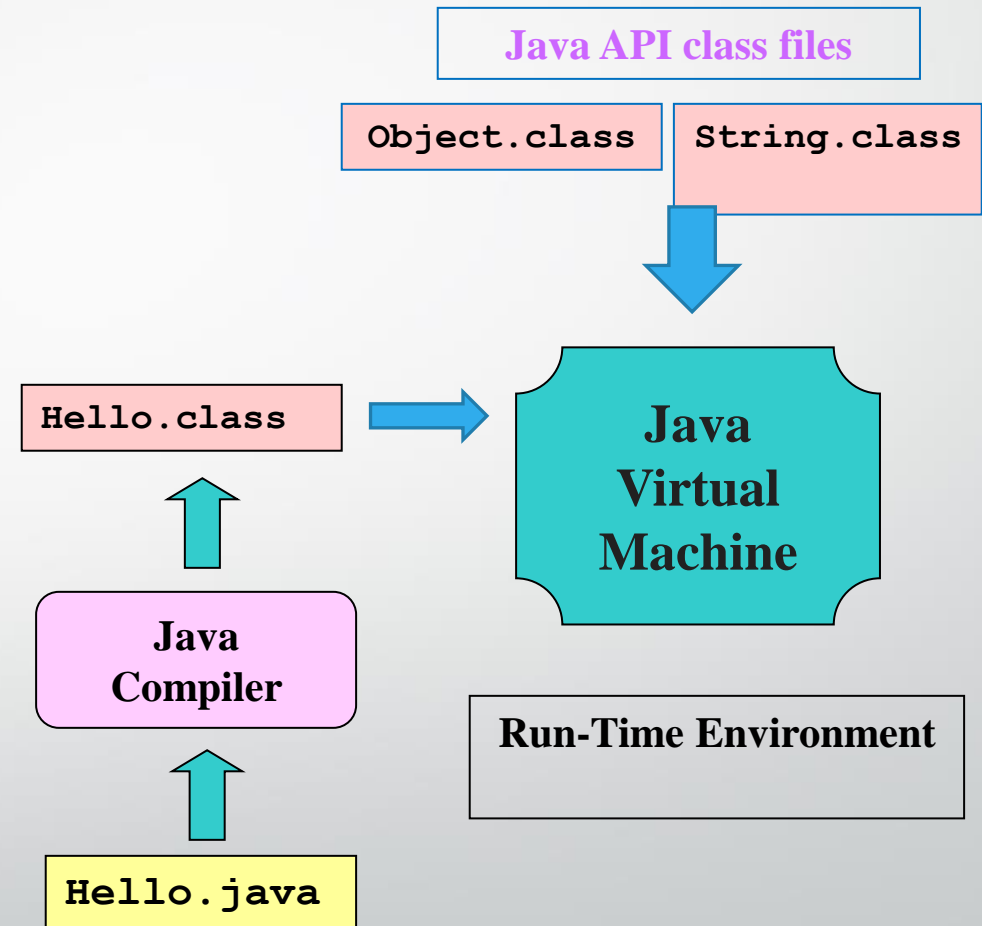
```
while (condition)
{
    Statements;
}
```

```
do
{
    Statements;
} while (condition);
```

```
for ( varInit ; Condition ; Statements)
{
    Statements1;
}
```

3.4. Kiến trúc chương trình xây dựng trên Java

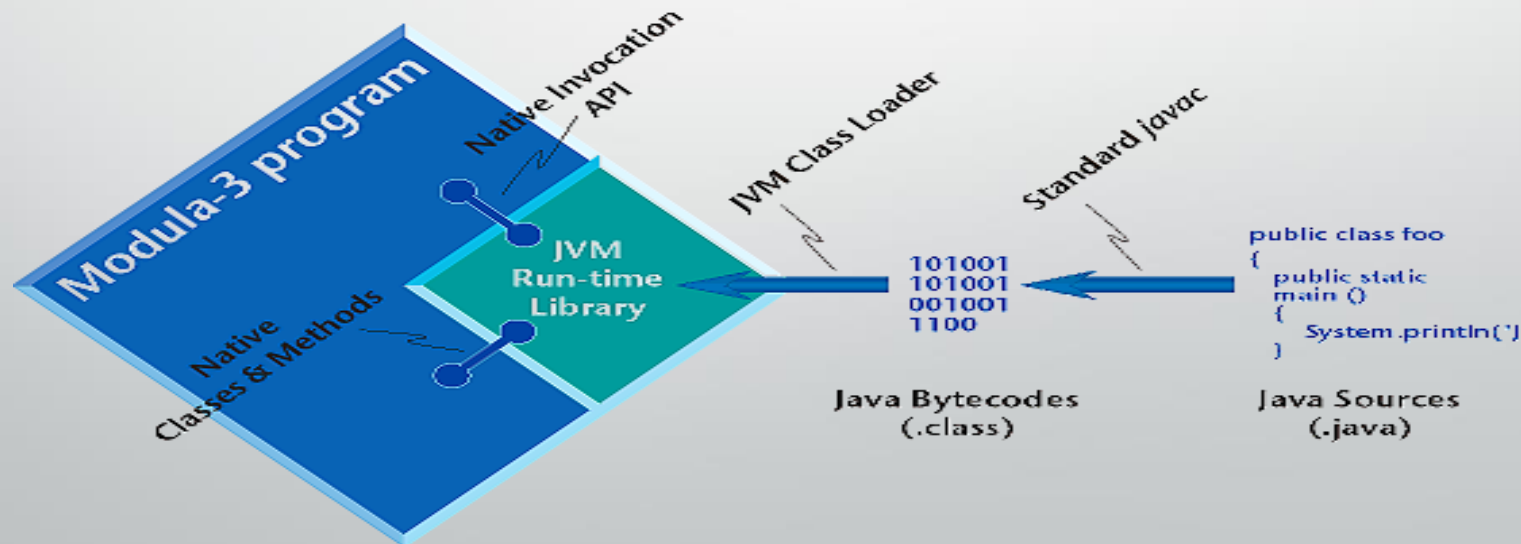
1. Chương trình nguồn (source code) được viết bằng ngôn ngữ Java
2. Các chương trình được biên dịch thành các file dạng lớp (*.class)
3. Các file .class được nạp vào bộ nhớ và thực thi bởi máy ảo Java (JVM)



3.4. Kiến trúc chương trình ... (tt)

JVM và Java “bytecode”

- Chương trình Java không biên dịch mã nguồn thành ngôn ngữ máy đích mà biên dịch thành file dạng “bytecode” – file *.class
- Mỗi HĐH sẽ có thể hiện riêng của máy ảo Java – JVM
- Mã bytecode làm việc với JVM và JVM làm việc với HĐH



3.4. Kiến trúc chương trình ... (tt)

- Các thành phần chính của một tập tin mã nguồn java
 - Gói – Package, Thư viện có sẵn – Library, Lớp – Class

Package

- Các class được lưu trong một khối thống nhất gọi là package
- Cú pháp: package <tên gói>;
- Các gói có thể được lồng vào nhau như cấu trúc thư mục
- Ưu điểm:
 - Các lớp được tổ chức riêng trong các gói riêng
 - Trong một gói không các lớp không trùng tên, trong một dự án lớn việc trùng tên các gói khác nhau.
 - Dễ quản lý trong các dự án lớn
 - Tên lớp có thể dùng định danh đối tượng

3.4. Kiến trúc chương trình ... (tt)

Library

- Dùng thư viện có sẵn
- Cú pháp: `import <ten gọi>;`
- Ví dụ: `import java.util.*; import java.io.File, ...`

Class

- Cú pháp

```
class ClassName  
{  
    //fields, constructors  
    // method declaration  
}
```


3.4. Kiến trúc chương trình ... (tt)

Tên của lớp

1. Sử dụng quy tắc đặt tên
2. Luôn viết hoa chữ cái đầu tiên
3. Dùng danh từ để đặt tên

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

Dữ liệu thành phần

- Là những dữ liệu cần phải có

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

Khởi tạo

- Định nghĩa cách thức thể hiện 1 đối tượng
- Có tên giống tên lớp
- Giống như hàm trong C nhưng không có kiểu dữ liệu trả về

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

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

Các phương thức (method)

- Những hành vi có thể thực hiện
1. Như hàm trong C
 2. Sử dụng động từ để đặt tên
 3. Luôn viết thường chữ cái đầu tiên

3.4. Kiến trúc chương trình ... (tt)

- Để thực thi chương trình, trình ứng dụng Java (Java application) bắt buộc phải có 1 lớp mà trong đó định nghĩa phương thức **main**.
- Phương thức **main()** trong lớp public được triệu hồi bởi JVM để bắt đầu thực thi ứng dụng.

```
public class RectangleDemo
{
    public static void main(String[] args)
    {
        Rectangle rec = new Rectangle(3, 4);

        float area = rec.area();

        System.out.println("Area: " + area);
    }
}
```

3.4. Kiến trúc chương trình ... (tt)

- Đối tượng phải được tạo trước khi được sử dụng trong chương trình.
 1. Khai báo 1 biến để lưu giữ tham chiếu đến đối tượng (đối tượng chỉ có thể được thao tác thông qua tham chiếu)
 2. Tạo đối tượng: bằng cách sử dụng toán tử **new** (ngầm định gọi đến hàm khởi tạo – hàm dựng)

```
public class RectangleDemo
{
    public static void main(String[] args)
    {
        Rectangle rec = new Rectangle(3, 4);

        float area = rec.area();

        System.out.println("Area: " + area);
    }
}
```

3.4. Kiến trúc chương trình ... (tt)

Gọi phương thức

- Sử dụng toán tử dấu chấm (the '.' operator)
- Cú pháp (Syntax):
 - <tên biến đối tượng tham chiếu> '.' <tên phương thức được gọi>

```
public class RectangleDemo
{
    public static void main(String[] args)
    {
        Rectangle rec = new Rectangle(3, 4);

        float area = rec.area();

        System.out.println("Area: " + area);
    }
}
```