



Lập trình hướng đối tượng - Java

Chương III

Lập trình hướng đối tượng - Java

Biên soạn: Lý Quỳnh Trân



Nội dung..

- Khái niệm OOP
- Khái niệm đối tượng - Lớp
- 3 đặc điểm lớn của OOP
- Cấu trúc Lớp – thành phần – cách truy xuất
- Trừu tượng hóa dữ liệu
- Tính bao bọc dữ liệu - từ chỉ định truy xuất
- Tính kế thừa - lớp con
- Tính đa hình – override & overload

Hạn chế lập trình truyền thống

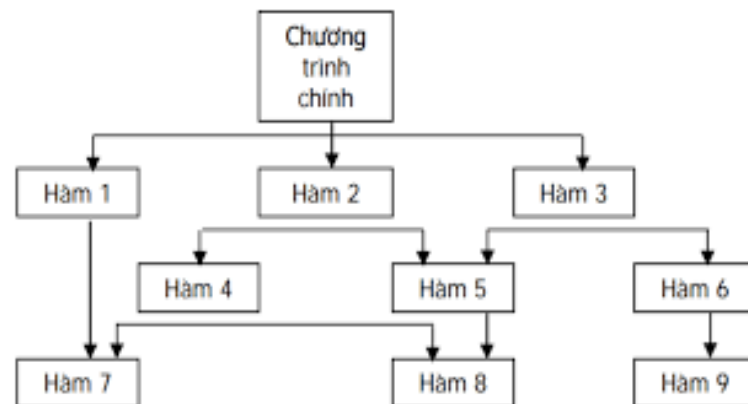
- Lập trình truyền thống:

- Lập trình tuyến tính: chương trình được thực hiện tuần tự từ đầu đến cuối
- Lập trình có cấu trúc: chia bài toán lớn thành các bài toán nhỏ;
 - Chương trình = Cấu trúc dữ liệu + Giải thuật

- Ưu điểm: sáng sủa, dễ hiểu, dễ theo dõi và tư duy giải thuật rõ ràng.

- Nhược điểm: không hỗ trợ việc sử dụng lại mã nguồn

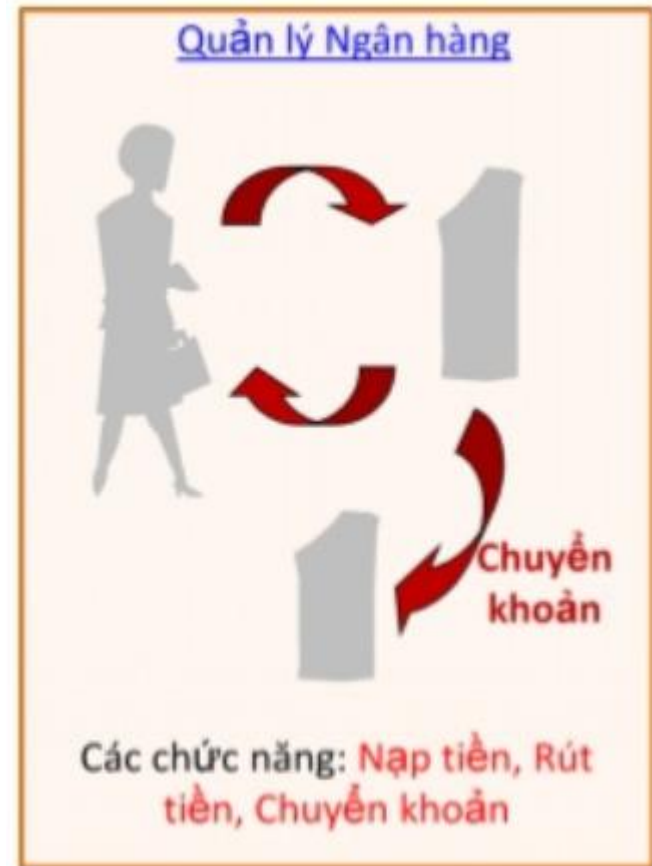
- Thứ nhất, thay đổi cấu trúc dữ liệu, phải thay đổi giải thuật, phải viết lại chương trình.
- Thứ hai, khó có khả năng lấy một đoạn code lớn từ một chương trình cũ lắp vào một dự án mới mà không phải sửa đổi lớn.
- Thứ ba, việc thiết kế linh hoạt mềm dẻo là cái mà các nhà phát triển phần mềm mong muốn.



Lập trình truyền thống

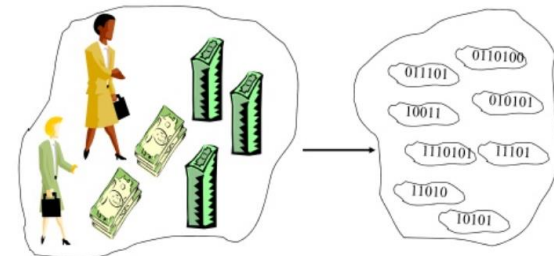
Đặt vấn đề: Bài toán quản lý giao dịch của Khách Hàng với Ngân hàng

- Khách hàng có giao dịch với ngân hàng:
 - Nạp tiền vào tài khoản
 - Rút tiền từ tài khoản
 - Chuyển khoản giữa các tài khoản
- Lập trình thủ tục
 - Chương trình chính chia thành các chức năng nhỏ, hoạt động độc lập với nhau.
 - Sử dụng dữ liệu chung



Hướng đối tượng

- **Lập trình hướng đối tượng khắc phục các ưu điểm của LTTT**
- Thứ nhất, đóng gói dữ liệu bằng cách trừu tượng hóa các đối tượng trong bài toán thực tế thành lớp trong bài toán tin học
 - Mỗi đối tượng chịu trách nhiệm quản lý riêng dữ liệu và các chức năng của nó
 - Các đối tượng tác động và trao đổi thông tin với nhau qua các chức năng thông điệp
- Thứ hai, việc cho phép sử dụng lại mã nguồn được thực hiện thông qua cơ chế kế thừa



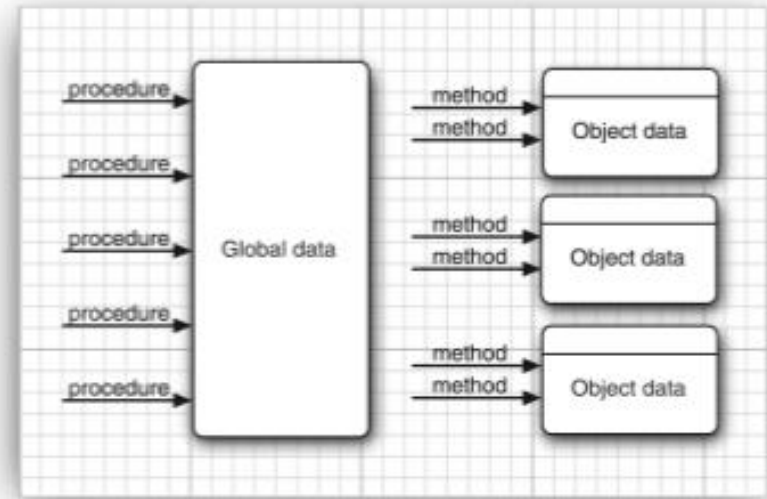
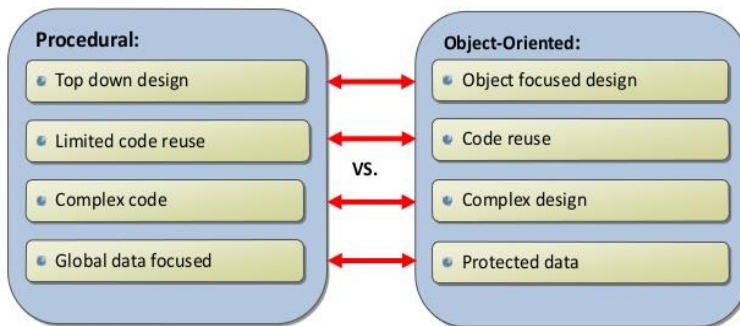


Tiếp cận hướng đối tượng

- Tiếp cận và thiết kế bài toán theo kiểu hướng đối tượng bao gồm bốn bước căn bản.
 - Thứ nhất, đọc kỹ bài toán và **gạch chân** các đối tượng (Object) tham gia trong bài toán (Danh từ)
 - Thứ hai, **gạch chân các thuộc tính** (cái biết về đối tượng) và các **hành động của từng đối tượng** (cái đối tượng làm) trong bài toán và thiết kế class.
 - Thứ ba, thiết kế **phạm vi truy cập** của phương thức và dữ liệu đồng thời thiết kế **mối quan hệ** giữa các lớp trong hệ thống.
 - Thứ 4, **khai báo lớp Main** để khởi tạo và thực thi tất cả cái đối tượng trong bài toán.

Truyền thống vs hướng đối tượng

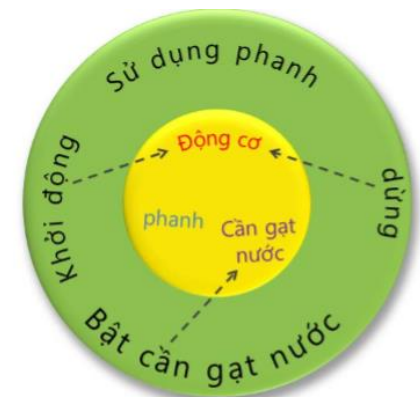
Procedural vs. Object-Oriented Programming



Procedural vs. OO programming

Đối tượng(Object)

- Đối tượng là khái niệm cơ sở của lập trình hướng đối tượng
- Đối tượng là một thực thể hoạt động khi chương trình đang chạy
- Một đối tượng được biểu diễn dựa trên thế giới thực
- Mỗi đối tượng được đặc trưng bởi thuộc tính và hành vi riêng của nó
- Có 2 loại thông tin quan trọng về mỗi đối tượng
 - Những thông tin mà đối tượng đó **biết**
 - Biểu diễn **thuộc tính** của đối tượng
 - Dữ liệu của đối tượng
 - Những thông tin mà đối tượng đó **làm**
 - Biểu diễn **hành vi** của đối tượng
- Ví dụ Ô tô
 - Thuộc tính: Hãng sản xuất, Model, Năm, Màu
 - Hành vi: Khởi động, dừng, Phanh, bật cần gạt



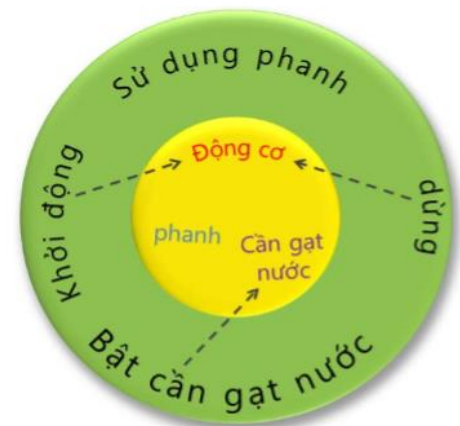
Trừu tượng hóa đối tượng theo thuộc tính

- Mô hình hóa các thuộc tính của lớp dựa trên các thông tin biết về đối tượng.
 - Gạch chân các thuộc tính của đối tượng có trong bài toán
 - Nhóm các đối tượng có các thuộc tính tương tự nhau, loại bỏ các thuộc tính cá biệt, tạo thành một nhóm chung.
 - Mỗi nhóm đối tượng đề xuất một lớp tương ứng.
 - Các thuộc tính chung của đối tượng sẽ cấu thành các thuộc tính tương ứng của lớp được đề xuất.
- Ví dụ trong bài toán quản lý xe ô tô,
- thuộc tính:
 - Các xe đều có năm sản xuất
 - Các xe đều có nhà sản xuất
 - Các xe đều có model
 - Các xe đều có màu sắc



Trừu tượng hóa đối tượng theo hành vi

- Mô hình hóa phương thức của lớp dựa trên các hành động đối tượng làm.
 - Tập hợp tất cả các hành động có thể có của đối tượng.
 - Nhóm các đối tượng có hoạt động tương tự nhau, loại bỏ các hoạt động cá biệt, tạo thành nhóm chung.
 - Mỗi nhóm đối tượng đề xuất một lớp tương ứng.
 - Các hành động chung của đối tượng sẽ cấu thành các phương thức của lớp tương ứng.
- Hành vi
 - Có thể khởi động máy
 - Có thể dừng lại
 - Có thể phanh

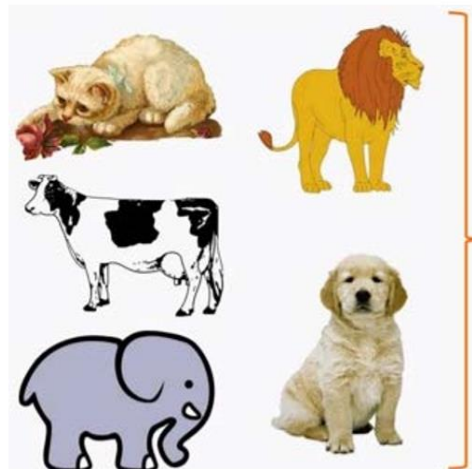


Lớp (Class)

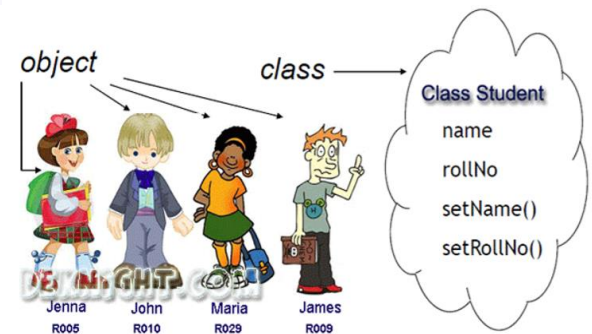
- Một đối tượng được tạo ra từ một lớp
- Lớp là một khuôn mẫu được sử dụng để mô tả các đối tượng cùng loại
- Lớp bao gồm các thuộc tính(trường dữ liệu) và các phương thức thành viên



Nhóm các **Xe ô-tô**



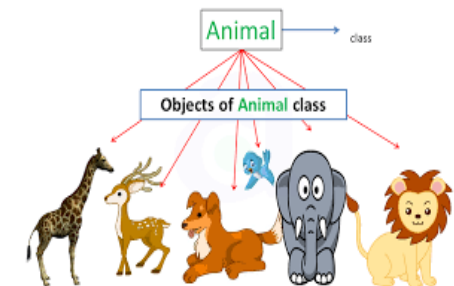
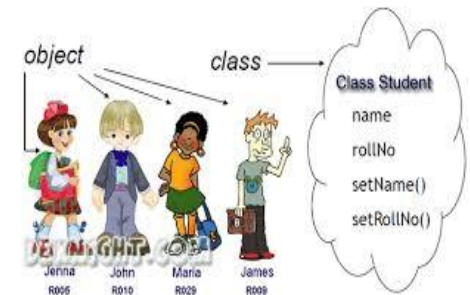
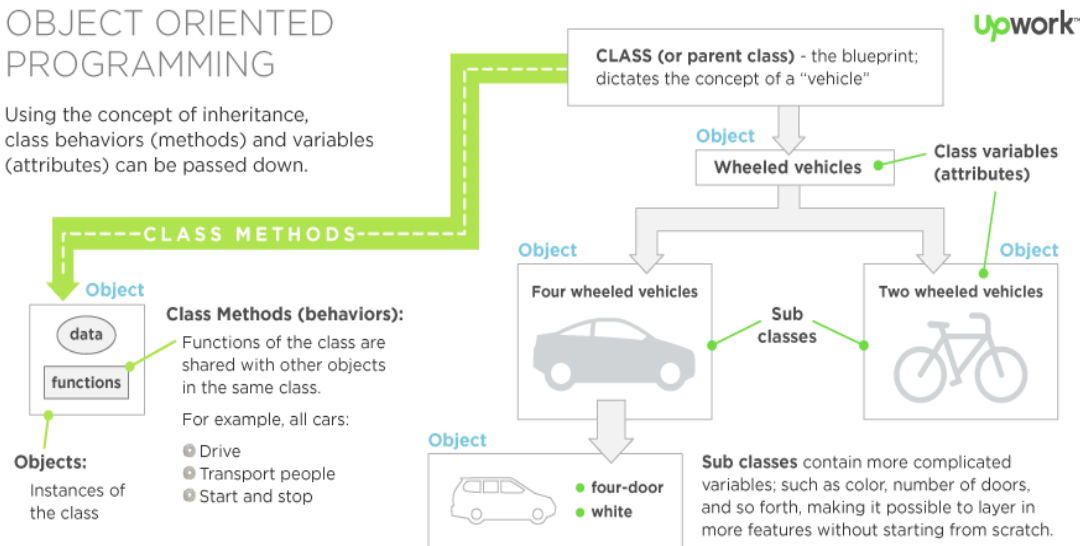
Nhóm các **Động vật**



Lớp (Class)

OBJECT ORIENTED PROGRAMMING

Using the concept of inheritance, class behaviors (methods) and variables (attributes) can be passed down.

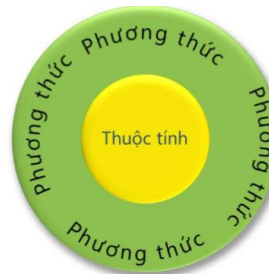


Thuộc tính(Biết) và phương thức(Làm)

- Thuộc tính(Biết)
 - Hãng sản xuất
 - Model
 - Năm
 - Màu
- Phương thức(Làm)
 - Khởi động()
 - Phanh()
 - Bật cần gạt nước()

Danh từ

Động từ



Ô tô

Thuộc tính

- Năm
- Nhà sản xuất
- Model
- Màu

Phương thức

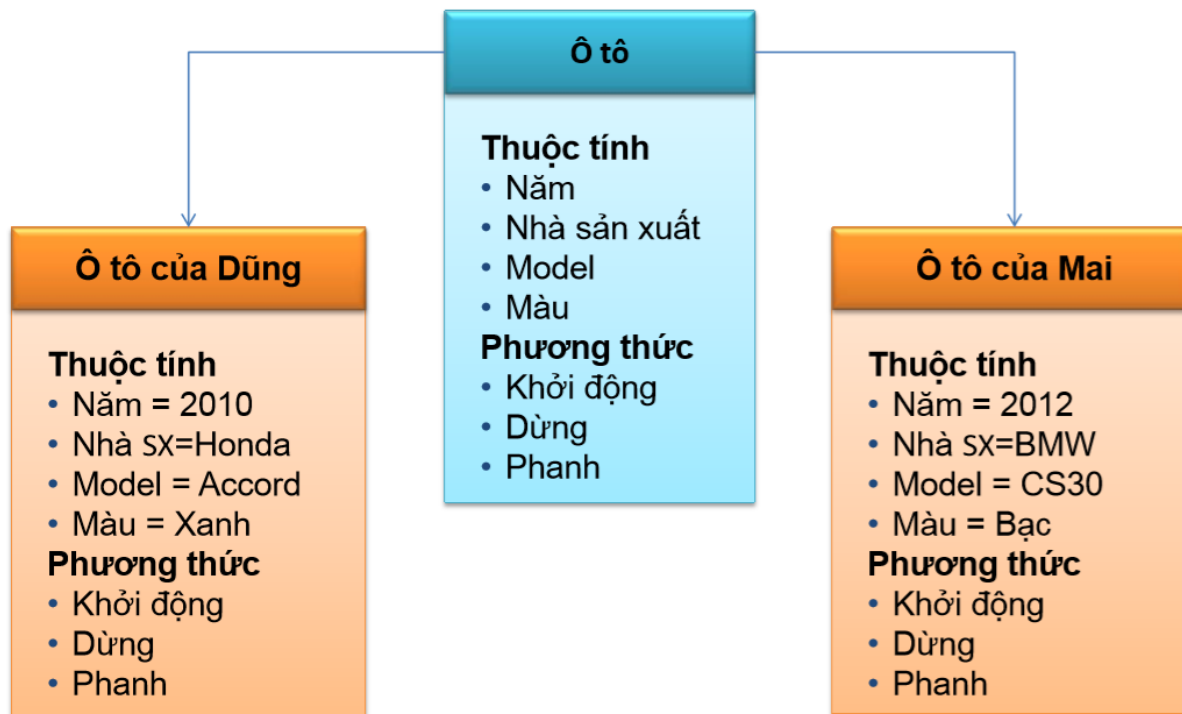
- Khởi động
- Dừng
- Phanh

Circle	SoccerPlayer	Car
radius color	name number xLocation yLocation	plateNumber xLocation yLocation speed
getRadius() getArea()	run() jump() kickBall()	move() park() accelerate()

Examples of classes

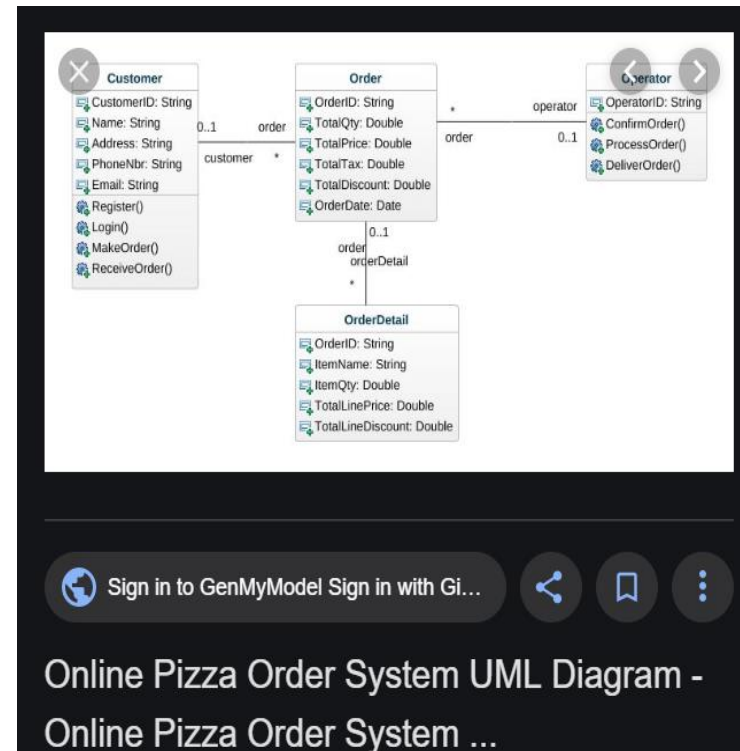
Quan hệ giữa Lớp và Đối tượng

- Tất cả Đối tượng của 1 Lớp có chung các thuộc tính & hành vi.
- Ví dụ: 2 đối tượng Ô tô của Dũng và Ô tô của Mai thuộc **lớp Ô tô**



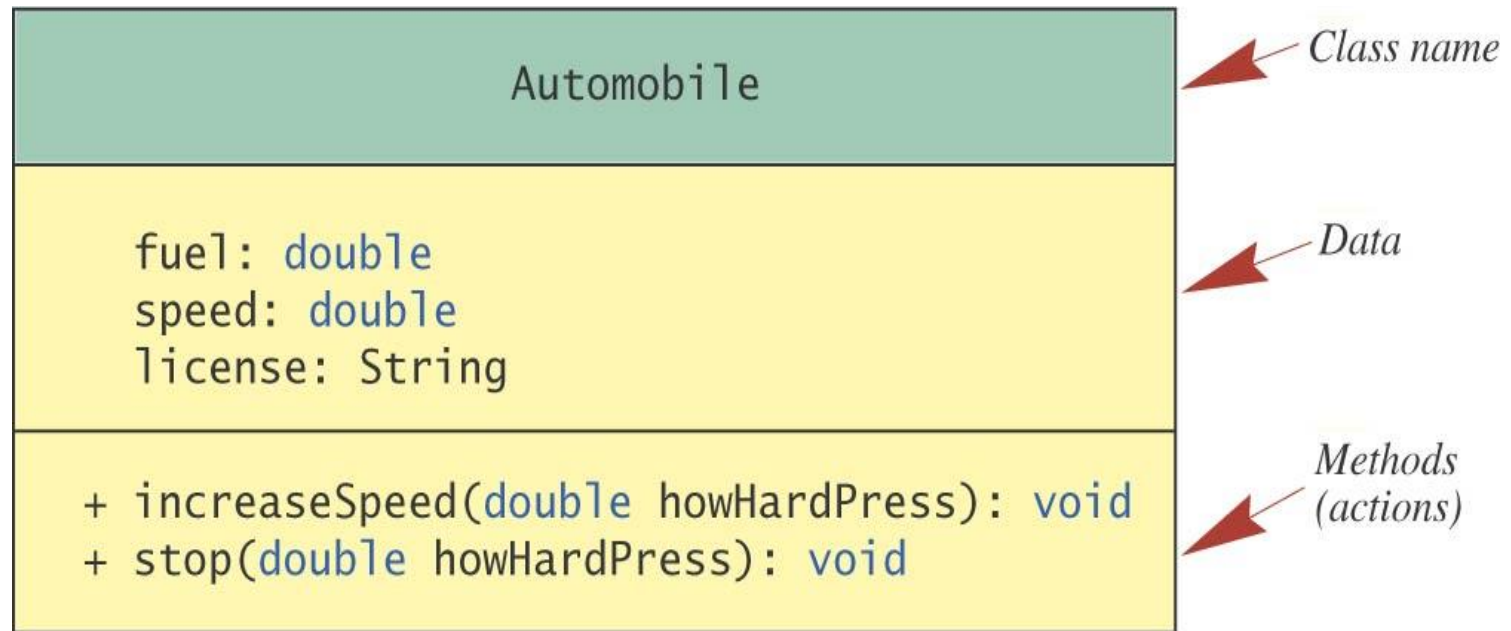
Thiết kế class

- A simple rule of thumb in identifying classes is to look for nouns in the problem analysis.
- Methods, on the other hand, correspond to verbs.
- For example, in an Pizza order-processing system, some of the nouns are
 - Customer • Order • OrderDetails • Operator
 - Next, look for verbs. Customer need to Register(), Login(), MakeOrder() and ReceiveOrder()
 - With each verb, identify the object that has the major responsibility for carrying it out. Only experience can help you decide which nouns and verbs



Thiết kế class

- An UML Class Diagram



Display 4.2

A Class Outline as a UML Class Diagram

Khai báo class

```
class <<ClassName>>
```

```
{
```

```
<<type>> <<field1>>;
```

Khai báo các trường

```
...
```

```
<<type>> <<fieldN>>;
```

Khai báo các phương thức

```
<<type>> <<method1>>([parameters]) {
```

```
    // body of method
```

```
}
```

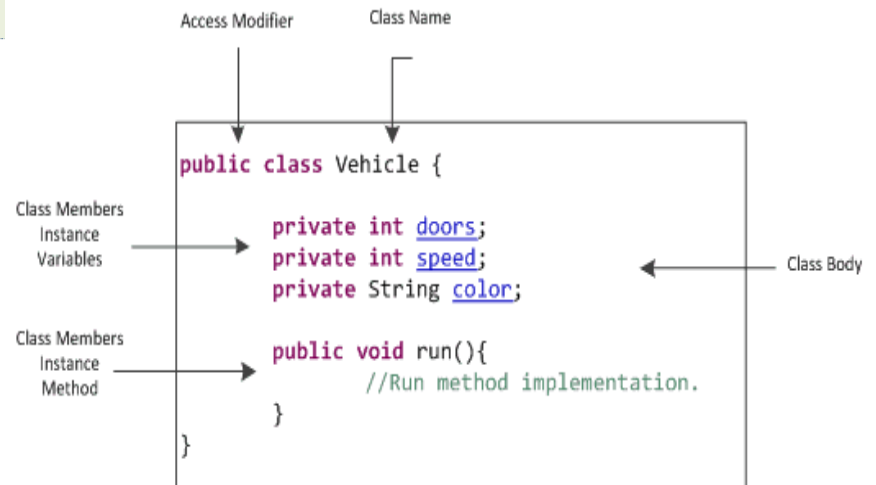
```
...
```

```
<<type>> <<methodN>>([parameters]) {
```

```
    // body of method
```

```
}
```

```
}
```



Ví dụ về class

```
12 public class OtoDemo {
13     String nam;
14     String nhaSX;
15     String model;
16     String mau;
17
18     OtoDemo(String nam, String nhaSX, String model, String mau)
19     {
20         this.nam = nam;
21         this.nhaSX = nhaSX;
22         this.model = model;
23         this.mau = mau;
24     }
25
26     void setModel(String model)
27     {
28         this.model = model;
29     }
30
31     String getModel()
32     {
33         return model;
34     }
35
36     void printAll()
37     {
38         System.out.println("O to " + model + " mau " + mau);
39     }
40 }
```

Trường

Phương thức

Ô tô

Thuộc tính

- Năm
- Nhà sản xuất
- Model
- Màu

Phương thức

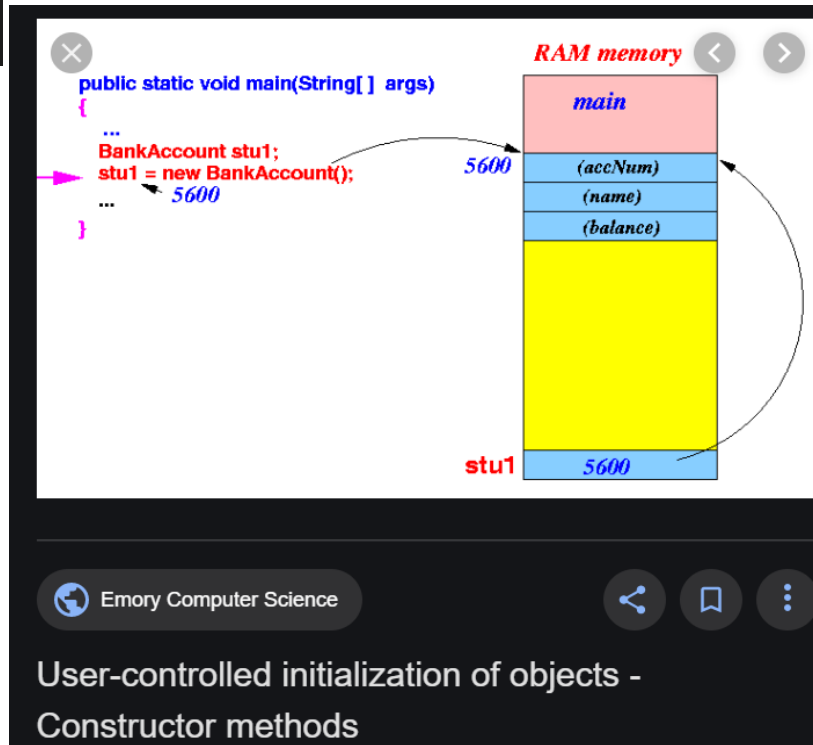
- Khởi động
- Dừng
- Phanh

Hàm dựng - constructor

- Là hàm đặc biệt dùng để khởi tạo đối tượng của 1 lớp.
 - cấp bộ nhớ cho đối tượng được khởi tạo
 - khởi tạo dữ liệu thành phần của đối tượng
 - được gọi khi khởi tạo 1 đối tượng dùng từ khóa new
 - tất cả các lớp đều phải có phương thức constructor
- Cú pháp
 - Tên hàm trùng tên lớp
 - Không có kiểu trả về
 - Có thể có nhiều hàm dựng với đối số khác nhau
- Hàm dựng mặc định
 - Không có đối số
 - được trình dịch tự động tạo ra khi lớp không khai báo hàm dựng

```
12 public class OtoDemo {
13     String nam;
14     String nhaSX;
15     String model;
16     String mau;
17
18     OtoDemo(String nam, String nhaSX, String model, String mau)
19     {
20         this.nam = nam;
21         this.nhaSX = nhaSX;
22         this.model = model;
23         this.mau = mau;
24     }
25
26     void setModel(String model)
27     {
28         this.model = model;
29     }
30
31     String getModel()
32     {
33         return model;
34     }
35
36     void printAll()
37     {
38         System.out.println("O to " + model + " mau " + mau);
39     }
40 }
```

Hàm dựng - constructor



Beginnersbook.com

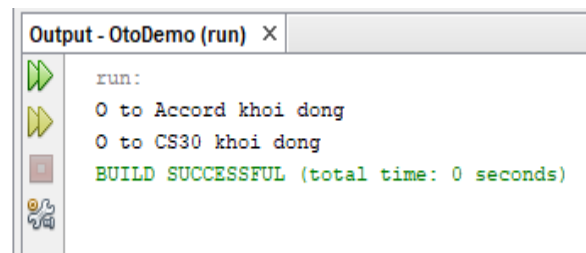
```
public class MyClass{
    // Constructor
    MyClass(){
        System.out.println("BeginnersBook.com");
    }
    public static void main(String args[]){
        MyClass obj = new MyClass();
        ...
    }
}
```

New keyword creates the object of MyClass & invokes the constructor to initialize the created object.

Tạo và sử dụng Đối tượng

- Cần 2 lớp để tạo và sử dụng đối tượng
- **Lớp 1:** dành để khai báo kiểu đối tượng cần tạo (Ô tô hoặc SinhVien, TaiKhoan)
- **Lớp 2:** lớp thử nghiệm (Driver), nơi đặt phương thức main(), tại đó ta tạo và sử dụng các đối tượng vừa xây dựng

```
12 public class OtoDemoDriver {  
13  
14     public static void main(String[] args) {  
15         // TODO code application logic here  
16         OtoDemo oToDung = new OtoDemo("2000", "Honda", "Accord", "Xanh");  
17         OtoDemo oToMai = new OtoDemo("2012", "BMW", "CS30", "Bac");  
18         oToDung.khoiDong();  
19         oToMai.khoiDong();  
20  
21     }  
22  
23 }
```



```
Output - OtoDemo (run) X  
run:  
O to Accord khoi dong  
O to CS30 khoi dong  
BUILD SUCCESSFUL (total time: 0 seconds)
```

Tạo và sử dụng Đối tượng

- Lớp thử nghiệm, nơi đặt phương thức main(), tại đó ta tạo và sử dụng các đối tượng vừa xây dựng
- Toán tử **new** được sử dụng để tạo Đối Tượng
- Sử dụng dấu chấm (.) để truy xuất tới hành vi và phương thức trong lớp

```
class Cow {  
    String name;  
    String breed;  
    int age;  
  
    void moo() {  
        System.out.println("Moo...!");  
    }  
}
```

Cow
name breed age
moo()

```
public class CowTestDrive {  
    public static void main (String[] args) {  
        Cow c = new Cow(); // make a Cow object  
        c.age = 2; // set the age of the Cow  
        c.moo(); // call its moo() method  
    }  
}
```

Định nghĩa phương thức

- Phương thức là một module code thực hiện một công việc cụ thể nào đó.
- Phương thức có thể không có, có một hoặc nhiều tham số
- Phương thức có kiểu trả về hoặc **void** (không trả về gì cả)
- Cú pháp

<<kiểu trả về>> <<tên phương thức>> ([danh sách tham số])

```
{  
    // thân phương thức  
}
```

```
public class Employee{  
    public String fullname;  
    public double salary;
```

```
    public void input(){...}  
    public void output(){...}
```

Kiểu trả về là **void** nên thân phương thức không chứa lệnh **return** giá trị

```
    public void setInfo(String fullname, double salary) {  
        this.fullname = fullname;  
        this.salary = salary;  
    }
```

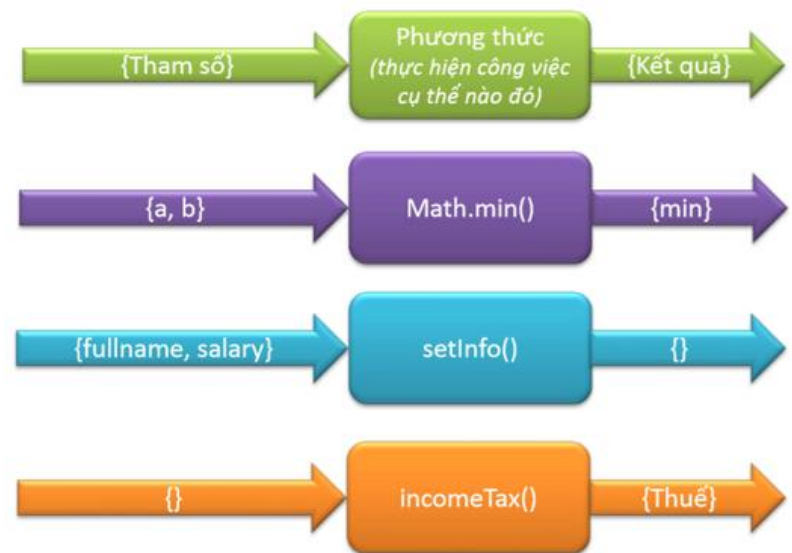
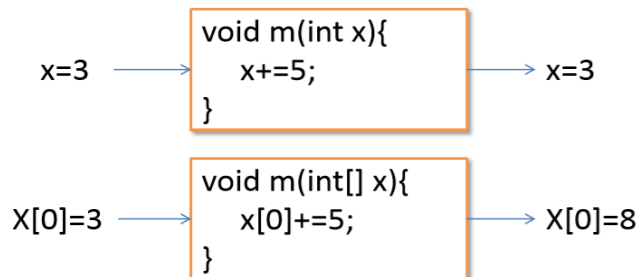
Kiểu trả về là **double** nên thân phương thức phải chứa lệnh **return** số thực

```
    public double incomeTax(){  
        if(this.salary < 5000000){  
            return 0;  
        }  
        double tax = (this.salary - 5000000) * 10/100;  
        return tax;  
    }
```

```
}
```

Truyền tham số cho phương thức

- Tham số được truyền vào cho phương thức bao gồm hai loại
 - Tham biến: Mảng, Class(Object), Interface
 - Tham trị: các kiểu nguyên thủy
- Nếu là tham trị thì giá trị của tham số sẽ không bị thay đổi
- Nếu là tham biến thì giá trị của tham số sẽ thay đổi



`void m(int...x){...}`

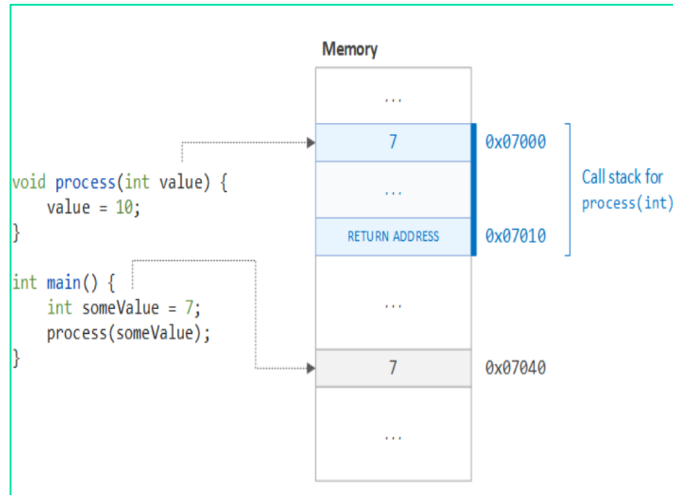
Gọi phương thức

`m(2,6,8)`

`m(2)`

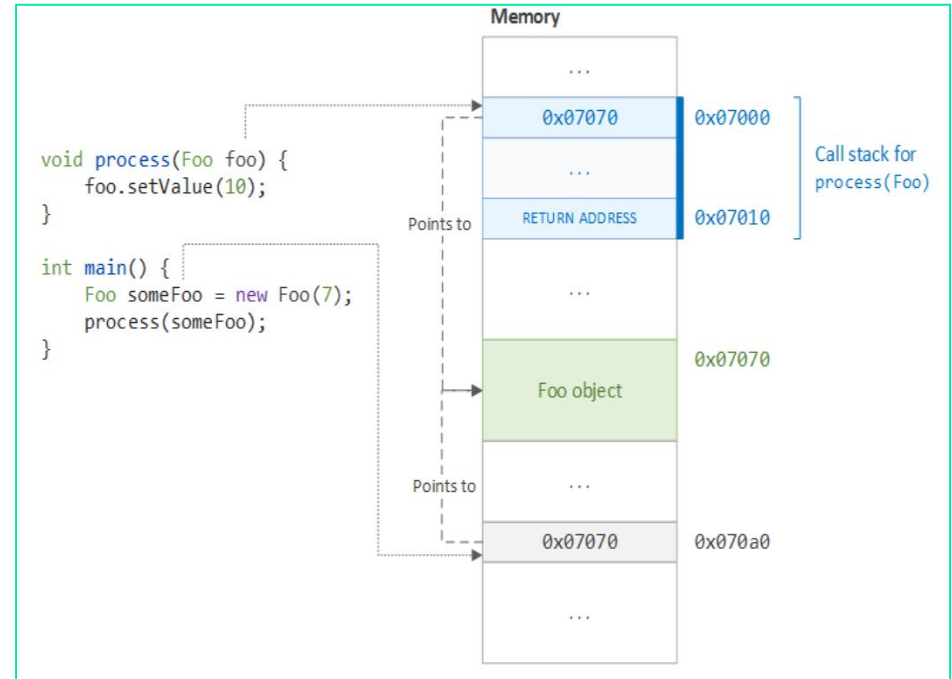
`int[] x = {2,6,8}`
`m(x)`

Truyền tham số cho phương thức



Assigning Values to Variable

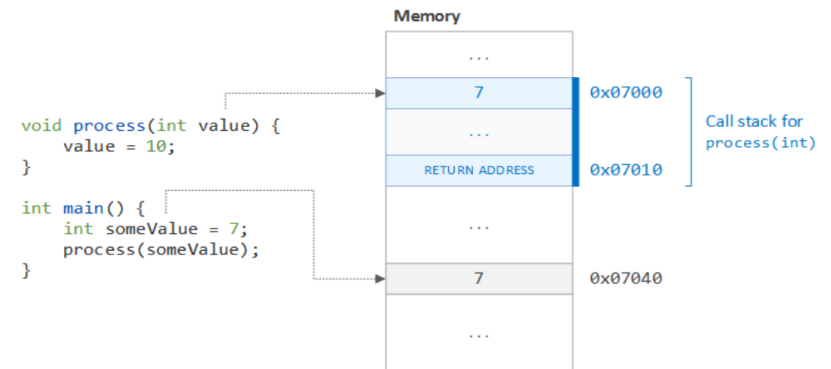
If we assign an existing primitive value, such as `someValue`, to a new variable, `value`, the primitive value is copied to the new variable. Since the value is copied, the two variables are *not* aliases of one another, and therefore, when the original variable, `someValue`, is changed, the change is *not* reflected in `value`.



For example, if we define an expression such as `Foo foo = new Foo()`, the variable `foo` does not hold the `Foo` object created, but rather, a pointer value to the created `Foo` object. The value of this pointer to the object (what the Java specification calls an **object reference**, or simply reference) is copied each time it is passed.

Truyền Primitive data

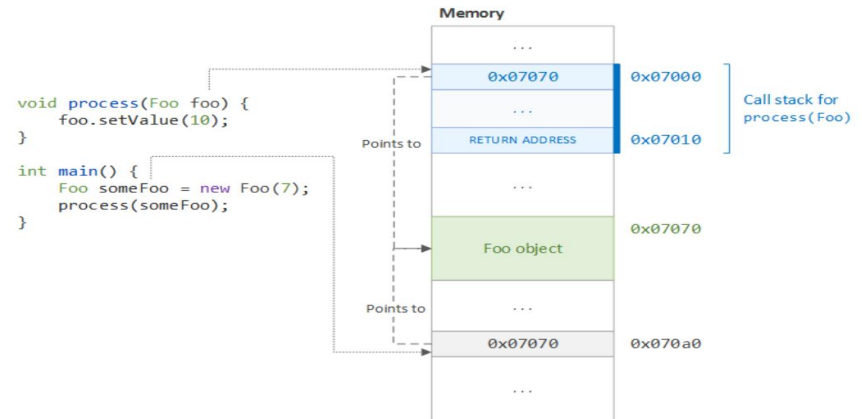
```
12 public class PassPrimiryDataDemo {
13
14     /**
15      * @param args the command line arguments
16      */
17     public void process(int value) {
18         System.out.println("Entered method (value = " + value + ")");
19         value = 50;
20         System.out.println("Changed value within method (value = " + value + ")");
21         System.out.println("Leaving method (value = " + value + ")");
22     }
23
24     public static void main(String[] args) {
25         // TODO code application logic here
26         PassPrimiryDataDemo processor = new PassPrimiryDataDemo();
27         int someValue = 7;
28         System.out.println("Before calling method (value = " + someValue + ")");
29         processor.process(someValue);
30         System.out.println("After calling method (value = " + someValue + ")");
31     }
32 }
33
34 }
35
```



Variables	Output - PassPrimiryDataDemo (run) X
	run:
	Before calling method (value = 7)
	Entered method (value = 7)
	Changed value within method (value = 50)
	Leaving method (value = 50)
	After calling method (value = 7)
	BUILD SUCCESSFUL (total time: 0 seconds)

Truyền Preference data

```
12 class Foo {
13     private int value;
14     public Foo(int value) {
15         this.value = value;
16     }
17     public void setValue(int value) {
18         this.value = value;
19     }
20     public int getValue() {
21         return value;
22     }
23 }
24 public class PassPreferenceDataDemo {
25     public void process(Foo foo) {
26         foo.setValue(10);
27     }
28     public static void main(String[] args) {
29         // TODO code application logic here
30         PassPreferenceDataDemo obj = new PassPreferenceDataDemo();
31         Foo someFoo = new Foo(7);
32         System.out.println("someFoo = "+someFoo);
33         System.out.println("someFoo Value= "+ someFoo.getValue());
34         obj.process(someFoo);
35         System.out.println("someFoo = "+someFoo);
36         System.out.println("someFoo Value= "+ someFoo.getValue());
37     }
38 }
39
```



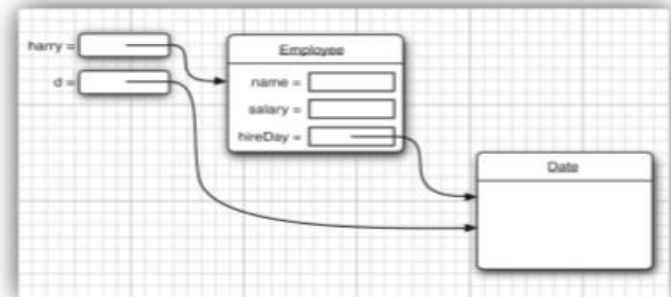
Variables	Output - PassPreferenceDataDemo (run) X
run:	
	someFoo = passpreferencedatademo.Foo@15db9742
	someFoo Value= 7
	someFoo = passpreferencedatademo.Foo@15db9742
	someFoo Value= 10
	BUILD SUCCESSFUL (total time: 0 seconds)

getter/setter

- Sometimes, it happens that you want to get and set the value of an instance field. Then you need to supply three items:
 - A private data field;
 - A public field accessor method; and (getter)
 - public field mutator method. (setter)

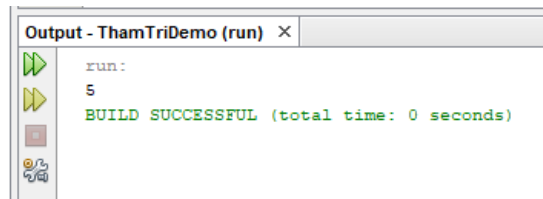
```
8 import java.time.LocalDate;
9
10 class Employee {
11
12     private String name;
13     private double salary;
14     private LocalDate hireDay;
15
16     public Employee(String n, double s, int year, int month, int day) {
17         name = n;
18         salary = s;
19         hireDay = LocalDate.of(year, month, day);
20     }
21
22     public String getName() {
23         return name;
24     }
25
26     public double getSalary() {
27         return salary;
28     }
29
30     public LocalDate getHireDay() {
31         return hireDay;
32     }
33
34     public void raiseSalary(double byPercent) {
35         double raise = salary * byPercent / 100;
36         salary += raise;
37     }
38
39 }
40
```

```
41 public class EmployeeDemo {
42
43     public static void main(String[] args) {
44         // TODO code application logic here
45         Employee[] staff = new Employee[3];
46         staff[0] = new Employee("Carl Cracker", 75000, 1987, 12, 15);
47         staff[1] = new Employee("Harry Hacker", 50000, 1989, 10, 1);
48         staff[2] = new Employee("Tony Tester", 40000, 1990, 3, 15);
49         // raise everyone's salary by 5% 20
50         for (Employee e : staff) {
51             e.raiseSalary(5);
52         }
53         // print out information about all Employee objects
54         for (Employee e : staff) {
55             System.out.println("name=" + e.getName() + ",salary=" + e.getSalary() + ",hireDay="
56                 + e.getHireDay());
57         }
58     }
59 }
60
```

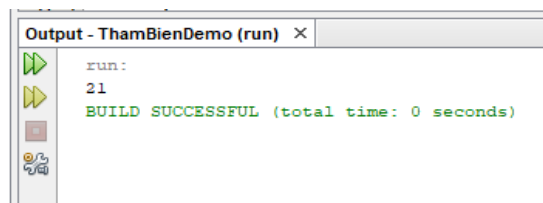


Ví dụ về Truyền tham số cho phương thức

```
12 public class ThamTriDemo {
13
14     static void addOneTo(int num) {
15         num = num + 1;
16     }
17
18     public static void main(String[] args) {
19         // TODO code application logic here
20         int x = 5;
21         addOneTo(x);
22         System.out.println(x);
23     }
24 }
25
```



```
Output - ThamTriDemo (run) x
run:
5
BUILD SUCCESSFUL (total time: 0 seconds)
```



```
Output - ThamBienDemo (run) x
run:
21
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
12 class Person {
13     private String Name;
14     private int Age;
15
16     public Person(String name) {
17         this.Name = name;
18     }
19
20     public void setAge(int age) {
21         this.Age = age;
22     }
23
24     public int getAge() {
25         return this.Age;
26     }
27
28 }
29 public class ThamBienDemo {
30
31     /**
32      * @param args the command line arguments
33      */
34     static void celebrateBirthday(Person p) {
35         p.setAge(p.getAge() + 1);
36     }
37
38     public static void main(String[] args) {
39         // TODO code application logic here
40         Person j;
41         j = new Person("John");
42         j.setAge(20);
43         celebrateBirthday(j);
44         System.out.println(j.getAge());
45     }
46 }
```

Truy xuất các thành phần của lớp: this

- tham chiếu từ bên trong lớp
 - `this.tentruong`
 - `this.tenham(đối số)` - nếu có
 - chỉ cần dùng **this** khi cần phân biệt các biến thành phần của lớp trùng tên với biến cục bộ của 1 hàm
- truy xuất từ bên ngoài lớp
 - `tenbien.tentruong`
 - `tenbien.tenham(đối số)` - nếu có
- truy xuất các thành phần dùng chung static
 - `tenlop.tenThanhPhan`
- các phương thức static chỉ được phép tham chiếu các thành phần static của lớp

```
public class MyClass{  
    int field;  
    void method(int field){  
        this.field = field;  
    }  
}
```

Trường

Tham số

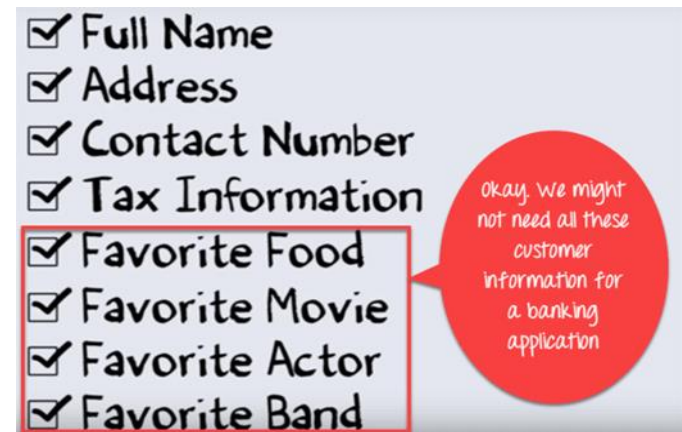


Đặc điểm của Lập trình hướng ĐT

- **Tính trừu tượng (data abstraction)**
 - quá trình chỉ ra các đối tượng với những thuộc tính & hành vi phù hợp với bài toán thực tế đang giải quyết.
- **Tính che dấu dữ liệu (Encapsulation)**
 - Tính bao bọc, che dấu dữ liệu
 - Bảo vệ dữ liệu từ các truy xuất trái phép bên ngoài
 - Từ chỉ định truy xuất : public, private, protected
- **Tính kế thừa (Inheritance)**
 - ý tưởng “đắt giá” nhất của OOP
 - Sử dụng lại mã nguồn
 - Lớp cơ sở (lớp cha) và Lớp dẫn xuất (lớp con)
 - Sử dụng từ khóa extends(class), implements (interface)
- **Tính đa hình (Polymorphism)**
 - Sự đa dạng khi đối tượng thể hiện các hành vi
 - Nạp chồng – override & overload

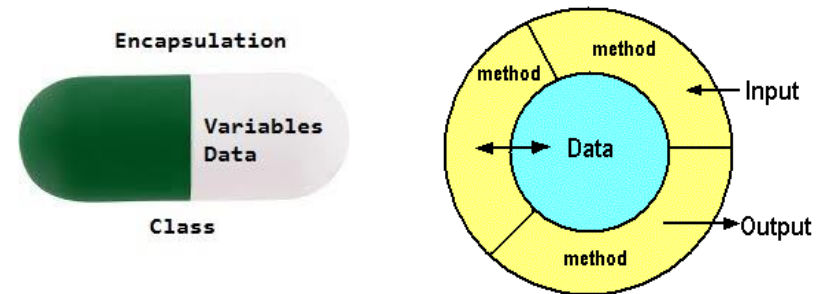
Trừu tượng hóa dữ liệu (Abstraction)

- Abstraction là công việc:
 - lựa chọn các thuộc tính và hành vi của thực thể vừa đủ để mô tả thực thể đó trong bối cảnh cụ thể
 - **không** phải liệt kê tất cả các thuộc tính hành vi của thực thể.
- Ưu điểm của data abstraction:
 - Xác định và chú trọng vào vấn đề đang giải quyết
 - Loại bỏ các chi tiết không quan tâm
- Ví dụ:
 - Mô tả nhân viên công ty ABC có rất nhiều thuộc tính hành vi.
 - Ở đây ta chỉ sử dụng tên, địa chỉ và số dt, thông tin về thuế
 - Không sử dụng món ăn yêu thích, phim, diễn viên hoặc bang nhạc yêu thích



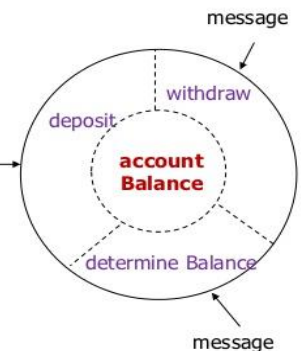
Encapsulation – bao bọc dữ liệu

- Encapsulation là tính che dấu bên trong đối tượng.
 - Nên che dấu bên trong đối tượng
 - Sử dụng phương thức để truy xuất các trường dữ liệu
 - `getThuộcTÍNH();`
 - `setThuộcTÍNH(thuộcTÍNH)`
- Mục đích của che dấu
 - Bảo vệ dữ liệu
 - Tăng cường khả năng mở rộng



Encapsulation - Example

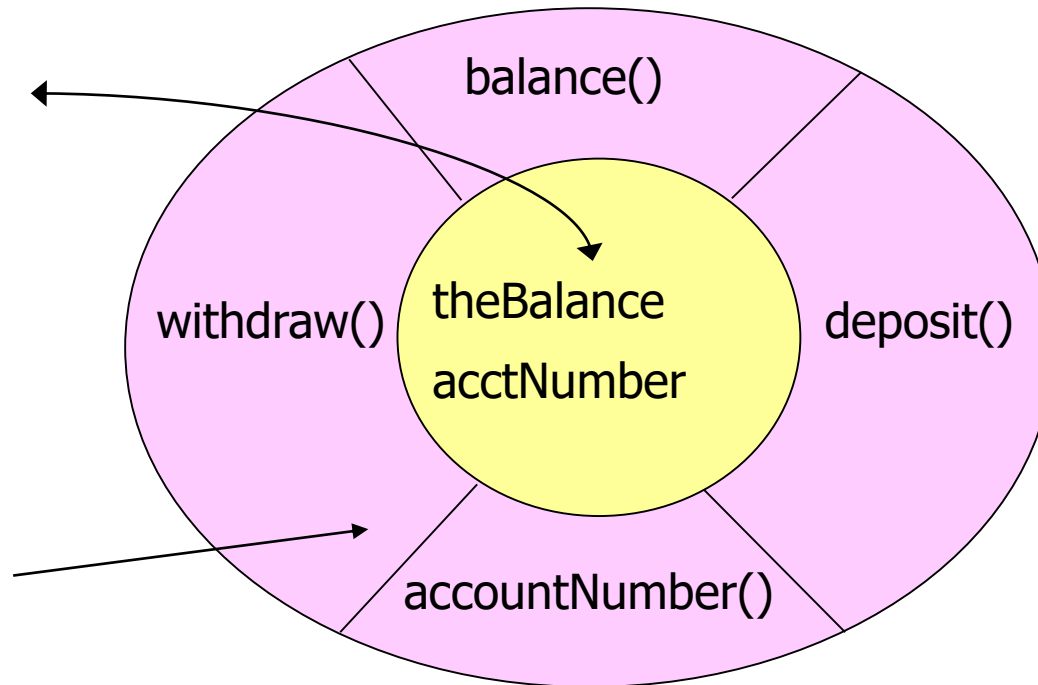
```
class Account {  
    private double accountBalance;  
  
    public withdraw();  
    public deposit();  
    public determineBalance();  
}  
// Class Account
```



Encapsulation

**Instance
variables**

Methods



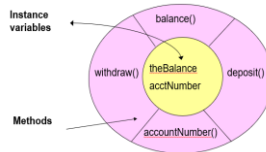
State variables make up the nucleus of the object. Methods surround and hide (encapsulate) the state variables from the rest of the program.

Encapsulation Example

```

2 import java.util.Date;
3
4 class Account {
5     //define variables
6     private int accountNumber;
7     private double balance; // balance for account
8     private double annualInterestRate; //stores the curr
9     private Date dateCreated; //stores the date account
10
11     //no arg constructor
12     Account() {
13         accountNumber = 0;
14         balance = 0.0;
15         annualInterestRate = 0.0;
16     }
17     //constructor with specific accountNumber and initial balance
18     public Account(int accountNumber, double balance) {
19         this.accountNumber = accountNumber;
20         this.balance = balance;
21     }
22     //constructor
23     public Account(int accountNumber, double balance, double annualInterestRate) {
24         this.accountNumber = accountNumber;
25         this.balance = balance;
26         this.annualInterestRate = annualInterestRate;
27     }
28
29     //accessor/mutator methods for accountNumber, balance, and annualInterestRate
30     public int getAccountNumber() {
31         return accountNumber;
32     }
33     public double getBalance() {
34         return balance;
35     }
36     public double getAnnualInterestRate() {
37         return annualInterestRate;
38     }
39     //accessor method
40     public void setAccountNumber(int accountNumber) {
41         this.accountNumber = accountNumber;
42     }
43
44     public void setBalance(double balance) {
45         this.balance = balance;
46     }
47

```



```

48     public void setAnnualInterestRate(double annualInterestRate) {
49         this.annualInterestRate = annualInterestRate;
50     }
51
52     public void setDateCreated(Date dateCreated) {
53         this.dateCreated = dateCreated;
54     }
55     //define method withdraw
56     double withdraw(double amount) {
57         return balance -= amount;
58     }
59     //define method deposit
60     double deposit(double amount) {
61         return balance += amount;
62     }
63 }
64
65 public class BankAccountDemo {
66
67     public static void main(String[] args) {
68         // TODO Auto-generated method stub
69
70         Account account1 = new Account(1122, 20000, .045);
71         account1.withdraw(2500);
72         account1.deposit(3000);
73         Date dateCreated=new java.util.Date();
74         System.out.println("Date Created:" + dateCreated);
75         System.out.println("Account ID:" + account1.getAccountNumber());
76         System.out.println("Balance:" + account1.getBalance());
77         System.out.println("Interest Rate:" + account1.getAnnualInterestRate());
78         System.out.println("Balance after withdraw of 2500:" + account1.getAnnualInterestRate());
79         System.out.println("Balance after deposit of 3000:" + account1.getAnnualInterestRate());
80         System.out.println("Monthly Interest:" + account1.getAnnualInterestRate());
81
82         System.out.println("Process completed.");
83     }
84 }
85

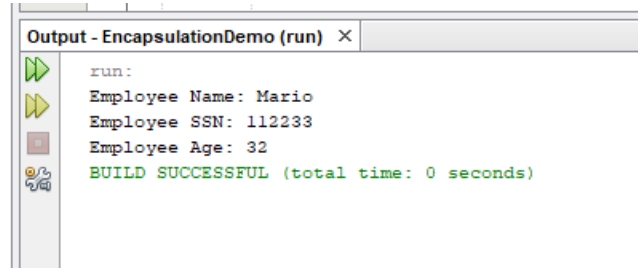
```

Variables	Output - Account (run) x
Run:	
	Date Created:Mon Aug 26 16:55:17 ICT 2019
	Account ID:1122
	Balance:20500.0
	Interest Rate:0.045
	Balance after withdraw of 2500:0.045
	Balance after deposit of 3000:0.045
	Monthly Interest:1122
	Process completed.
	BUILD SUCCESSFUL (total time: 0 seconds)

Ví dụ về Encapsulation

```
8 class Encapsulation {
9     private int ssn;
10    private String empName;
11    private int empAge;
12
13    //Getter and Setter methods
14    public int getEmpSSN() {
15        return ssn;
16    }
17
18    public String getEmpName() {
19        return empName;
20    }
21
22    public int getEmpAge() {
23        return empAge;
24    }
25
26    public void setEmpAge(int newValue) {
27        empAge = newValue;
28    }
29
30    public void setEmpName(String newValue) {
31        empName = newValue;
32    }
33
34    public void setEmpSSN(int newValue) {
35        ssn = newValue;
36    }
37 }
```

```
38 public class EncapsulationDemo {
39
40    /**
41     * @param args the command line arguments
42     */
43    public static void main(String[] args) {
44        // TODO code application logic here
45        Encapsulation obj = new Encapsulation();
46        obj.setEmpName("Mario");
47        obj.setEmpAge(32);
48        obj.setEmpSSN(112233);
49        System.out.println("Employee Name: " + obj.getEmpName());
50        System.out.println("Employee SSN: " + obj.getEmpSSN());
51        System.out.println("Employee Age: " + obj.getEmpAge());
52    }
53
54 }
```



```
Output - EncapsulationDemo (run) x
run:
Employee Name: Mario
Employee SSN: 112233
Employee Age: 32
BUILD SUCCESSFUL (total time: 0 seconds)
```

- Các từ chỉ định truy xuất -Access modifiers – được sử dụng để thực hiện encapsulation

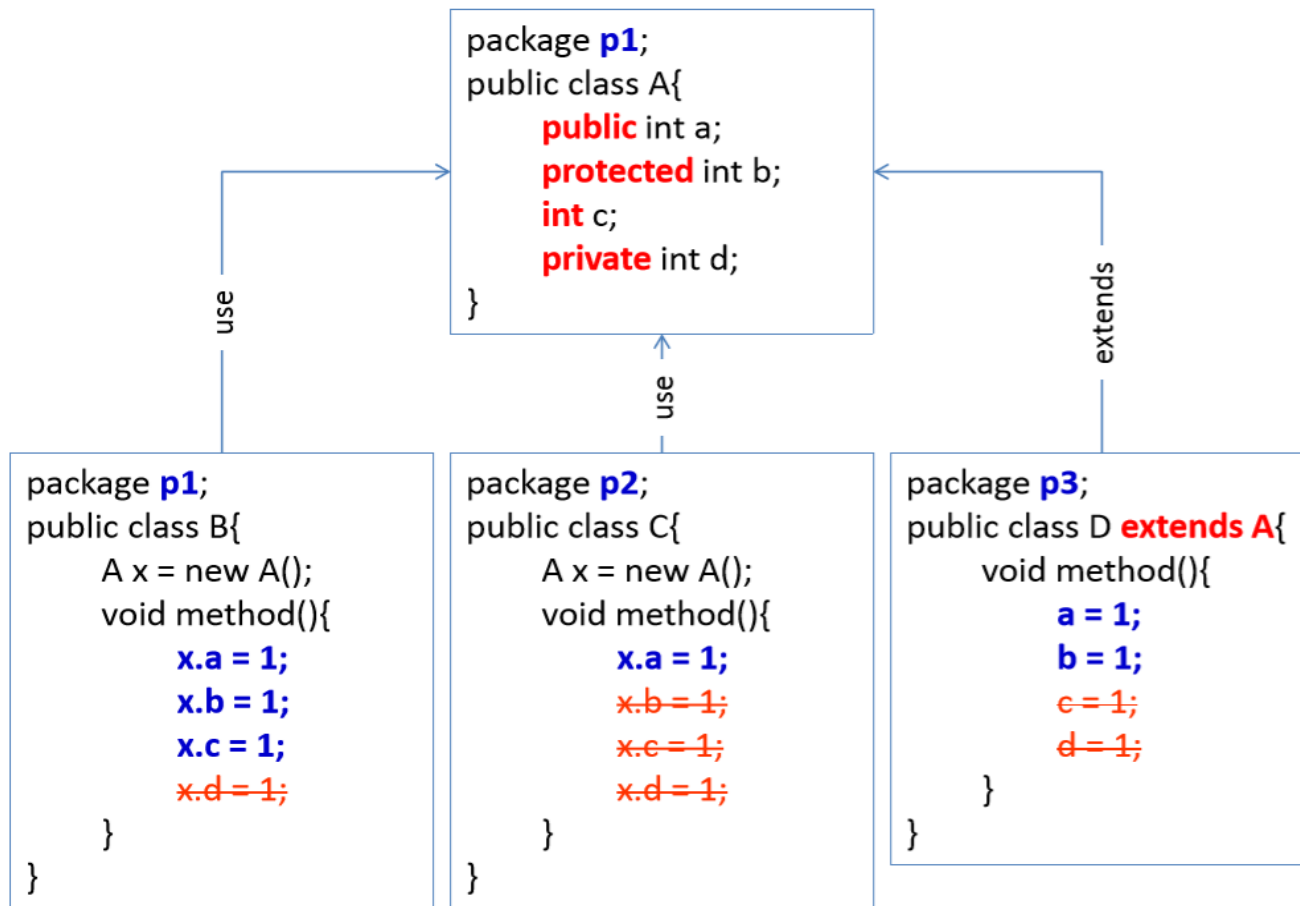
public → protected → {default} → private

Đặc tả truy xuất

- Đặc tả truy xuất được sử dụng để định nghĩa khả năng cho phép truy xuất các thành viên của lớp.
- Trong java có 4 đặc tả khác nhau:
 - **public**: cho phép truy xuất rộng rãi
 - áp dụng cho class, applet, application, thành phần của lớp
 - **private**: chỉ cho phép truy xuất bên trong lớp
 - không áp dụng đối với khai báo class
 - **protected**: cho phép truy xuất từ lớp con cùng thuộc package hoặc lớp kế thừa và lớp con và lớp cha có thể khác gói package
 - **{default}**: không khai báo chỉ định truy xuất
 - là public đối với các lớp khác trong cùng package
 - Là private đối với các lớp truy xuất khác gói
- Mức độ che dấu tăng dần theo chiều mũi tên

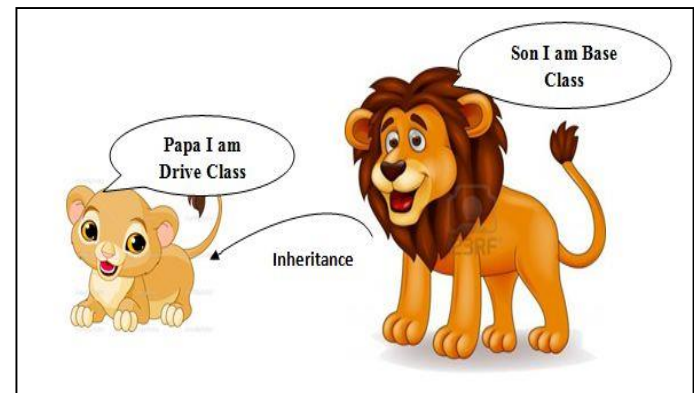
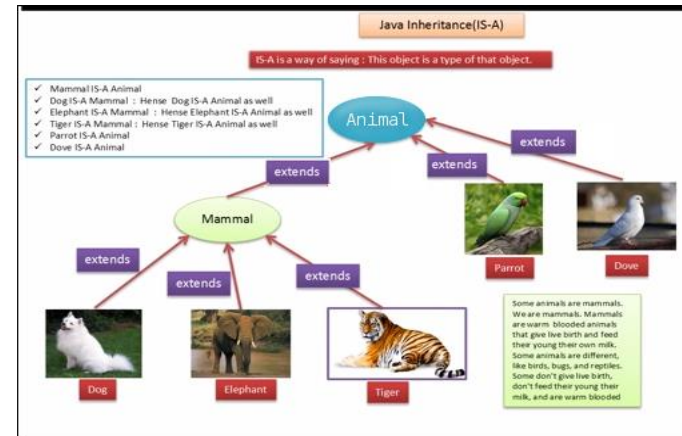
public → **protected** → **{default}** → **private**

Ví dụ đặc tả truy xuất



Inheritance – tính kế thừa

- Kế thừa là sử dụng lại code từ các module của các lập trình viên khác.
- Kế thừa:
- cho phép **1 lớp (lớp cha) chia sẻ** các thành phần (thuộc tính, hành vi) cho các lớp khác (lớp con).
- cho phép định nghĩa **1 lớp mới (lớp con)** từ **1 lớp đã có (lớp cha)**.
- Lớp cho kế thừa được gọi là lớp cha, gọi là super class, Base Class.
- Lớp được kế thừa gọi là lớp con, gọi là sub class, Drive Class

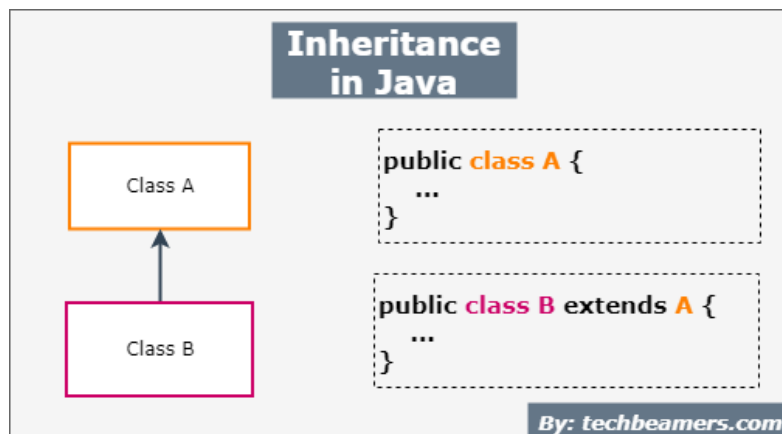


Inheritance – tính kế thừa

Sử dụng **extends** để kế thừa từ một class và **implements** để kế thừa từ interface.

- Cú pháp

```
class ten_lop_con extends ten_lop_cha
{
    // trường và phương thức bên trong;
}
```



```
8 class Animal {
9     public String name;
10    public String food;
11    public Animal(String name, String food)
12    { this.name = name;
13      this.food = food; }
14    void eat() {
15        System.out.println("eating..." + food);
16    } }
17 class Dog extends Animal {
18     String chaseCatName;
19     public Dog(String name, String food, String chaseCatName)
20     { super(name, food);
21       this.chaseCatName = chaseCatName; }
22     public void chaseCat() {
23         System.out.print("chasing cat..."); }
24 }
```

```
8 interface Animal {
9     public void eat();
10    public void travel();
11 }
12 class MammalInt implements Animal {
13
14     public void eat()
15     { System.out.println("Mammal eats"); }
16
17     public void travel()
18     { System.out.println("Mammal travels"); }
19 }
```


Mục đích kế thừa

Sử dụng lại code phù hợp của những chương trình khác, hoặc module được viết từ các lập trình viên khác. Những phương thức chung có thể được viết một lần, và sử dụng nhiều lần ở các lớp con.

- Lớp con khi được thừa kế từ lớp cha, **được phép sở hữu các tài sản (thuộc tính và hành vi)** của lớp cha.
 - Lớp con **sở hữu các tài sản public hoặc protected** của lớp cha.
 - Lớp con sở hữu các tài sản mặc định **{default}** của lớp cha, nếu lớp con và lớp cha được định nghĩa **cùng gói**.
 - Lớp con **không** thể truy cập vào thành viên **private** của lớp cha. Khi muốn cho phương thức thừa kế, ta tránh không khai báo private
 - Lớp con **không** kế thừa các **hàm dựng** của lớp cha, phải gọi **super**
 - Hàm dựng với từ khóa **super** phải đặt dòng đầu tiên trong hàm dựng lớp con

```
8 class Animal {
9     public String name;
10    public String food;
11    public Animal(String name, String food)
12    {
13        this.name = name;
14        this.food = food;
15    }
16    void eat() {
17        System.out.println("eating..." + food);
18    }
19 }
20 class Dog extends Animal {
21     String chaseCatName;
22     public Dog(String name, String food, String chaseCatName)
23     {
24         super(name, food);
25         this.chaseCatName = chaseCatName;
26     }
27     public void chaseCat() {
28         System.out.print("chasing cat...");
29     }
30 }
31 public class KeThuaAnimalDemo2 {
32     public static void main(String[] args) {
33         // TODO code application logic here
34         Dog d = new Dog("Bulldog", "milk", "Cat Tom");
35         System.out.println(d.name);
36         d.eat();
37         d.chaseCat();
38         System.out.println(d.chaseCatName);
39     }
40 }
```

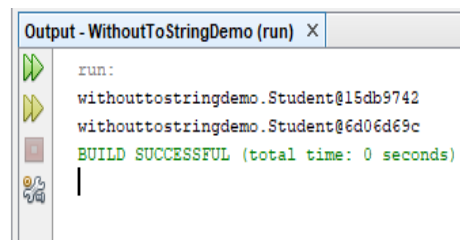
```
Output - KeThuaAnimalDemo2 (run) x
run:
Bulldog
eating...milk
chasing cat...Cat Tom
BUILD SUCCESSFUL (total time: 0 seconds)
```

Mục đích kế thừa: overriding

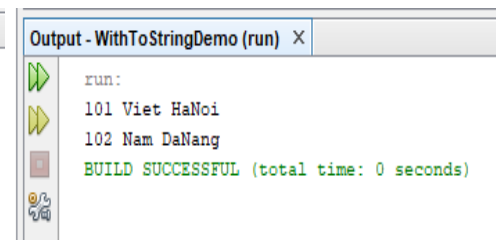
Cho phép lớp con **được ghi đè (overriding) code** của phương thức từ lớp cha, theo mục đích của chính lớp con đó.

- Phương thức ghi đè, cùng tên, nhưng khác nội dung bên trong
- Ghi đè lại phương thức **public String toString() {}** trong class
 - Phương thức toString() được thừa kế từ Object class
 - Mục đích trả về nội dung cần in ra của đối tượng
 - Sử dụng để in đối tượng **System.out.println(s1);**

```
9 class Student {
10     int rollno;
11     String name;
12     String city;
13     Student(int rollno, String name, String city) {
14         this.rollno = rollno;
15         this.name = name;
16         this.city = city;
17     }
18     public String toString() { //Ghi đè phương thức toString()
19         return rollno + " " + name + " " + city;
20     }
21 }
22 public class WithToStringDemo {
23
24     public static void main(String[] args) {
25         // TODO code application logic here
26         Student s1 = new Student(101, "Viet", "HaNoi");
27         Student s2 = new Student(102, "Nam", "DaNang");
28
29         System.out.println(s1); // compiler writes here s1.toString()
30         System.out.println(s2); // compiler writes here s2.toString()
31     }
32 }
33
```



```
Output - WithoutToStringDemo (run) X
run:
withouttostringdemo.Student@15db9742
withouttostringdemo.Student@6d06d69c
BUILD SUCCESSFUL (total time: 0 seconds)
```

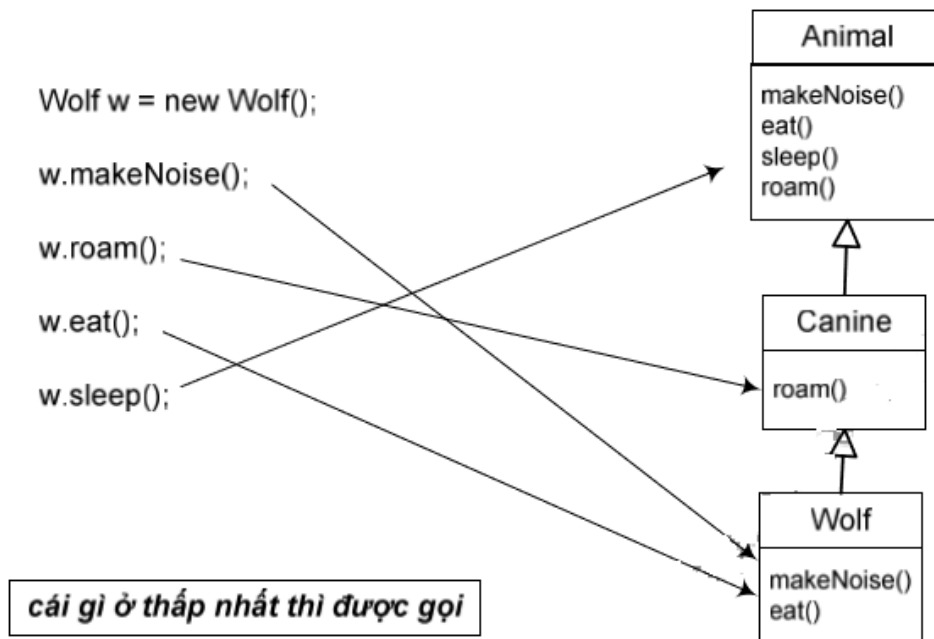


```
Output - WithToStringDemo (run) X
run:
101 Viet HaNoi
102 Nam DaNang
BUILD SUCCESSFUL (total time: 0 seconds)
```

Overriding nào được gọi

- Khi một lớp con có **nhều phương thức được cài đặt**, nghĩa là viết lại code của thân chương trình của một phương thức trong lớp cha.

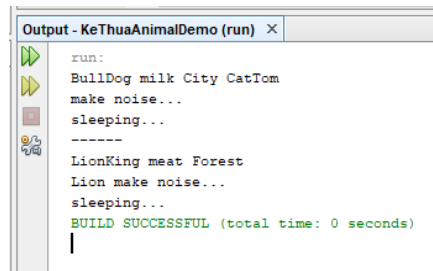
- Khi ta tạo một đối tượng thuộc lớp con và gán một biến tham chiếu tới nó, thì **phiên bản cài đặt của chính phương thức lớp con gần với nó nhất, thấp nhất sẽ được gọi.**



- Ví dụ: biến w để gọi phương thức cho một đối tượng Wolf, thì w.makeNoise() ở Wolf; w.roam() ở Canine; w.eat() và w.sleep() ở Animal.

Ví dụ về overriding

```
8 class Animal {
9     public String picture;
10    public String food;
11    public int hunger;
12    public String boundary;
13    public String location;
14
15    public Animal(String picture, String food, int hunger, String boundary, String location)
16    {
17        this.picture = picture;
18        this.food = food;
19        this.hunger = hunger;
20        this.boundary = boundary;
21        this.location = location;
22    }
23
24    void eat() {
25        System.out.println("eating...");
26    }
27    void makeNoise() {
28        System.out.println("make noise...");
29    }
30    void sleep() {
31        System.out.println("sleeping...");
32    }
33    void roam() {
34        System.out.println("roaming...");
35    }
36 }
37
38 class Dog extends Animal {
39     String chaseCatName;
40     public Dog(String picture, String food, int hunger, String boundary, String location, String chaseCatName)
41     {
42         super(picture, food, hunger, boundary, location);
43         this.chaseCatName = chaseCatName;
44     }
45 }
```



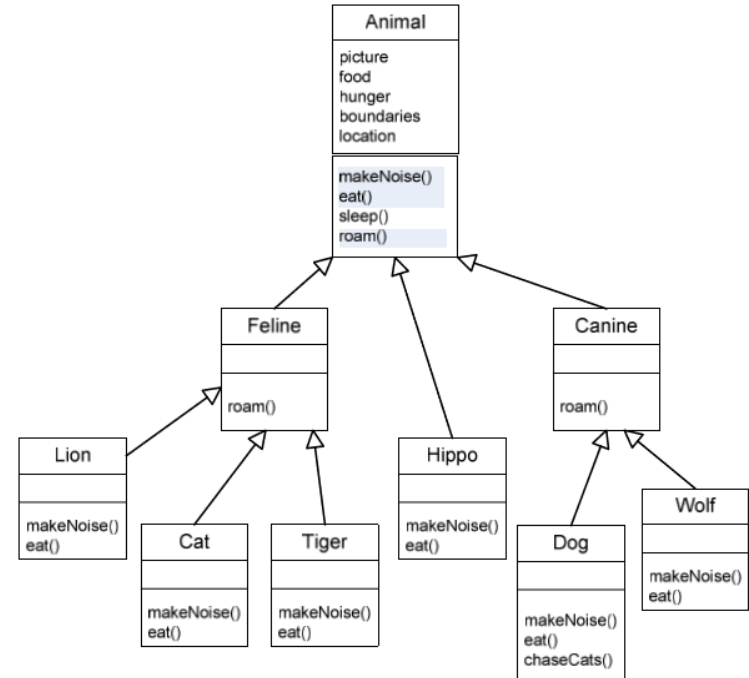
```
Output - KeThuaAnimalDemo (run) X
run:
Bulldog milk City CatTom
make noise...
sleeping...
-----
LionKing meat Forest
Lion make noise...
sleeping...
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
46    public void chaseCat() {
47        System.out.println("chasing cat...");
48    }
49    public String toString()
50    { return picture + " " + food + " " + boundary + " " + chaseCatName;
51    }
52 }
53
54 class Lion extends Animal {
55
56     public Lion(String picture, String food, int hunger,
57         String boundary, String location)
58     {
59         super(picture, food, hunger, boundary, location);
60     }
61     void makeNoise() {
62         System.out.println("Lion make noise...");
63     }
64     public String toString()
65     { return picture + " " + food + " " + boundary;
66     }
67 }
68
69 public class KeThuaAnimalDemo {
70
71     public static void main(String[] args) {
72         // TODO code application logic here
73         Dog d = new Dog("Bulldog", "milk", 1, "City", "Block A", "CatTom");
74         System.out.println(d);
75         d.makeNoise();
76         d.sleep();
77         System.out.println("-----");
78         Lion l = new Lion("LionKing", "meat", 1, "Forest", "Block Mountain");
79         System.out.println(l);
80         l.makeNoise();
81         l.sleep();
82     }
83 }
```

Thiết kế cây kế thừa

Việc thiết kế cây kế thừa trải qua bốn bước căn bản sau:

- **Bước 1:** xác định các **đặc điểm chung** và trừu tượng mà các đối tượng đều có. Ví dụ: Lớp động vật có các đặc điểm chung là picture, food... và các hành vi makeNoise(), eat()....
- **Bước 2:** thiết kế lớp với các hành vi chung trên (Lớp cha)
- **Bước 3:** Xác định các lớp con có cần **thêm** các hành vi **đặc thù cụ thể**
- **Bước 4:** Tiếp tục dùng trừu tượng hóa tìm các lớp con có hành vi giống nhau, phân nhóm mịn hơn nếu cần.
- Ví dụ: Giả sử ta cần thiết kế một chương trình giả lập cho phép người dùng thả một đám các con động vật thuộc các loài khác nhau vào một môi trường sống.



Ví dụ Tính kế thừa

```
12 class Animal {
13     public String picture;
14     public String food;
15     public int hunger;
16     public String boundary;
17     public String location;
18
19     public Animal(String picture, String food, int hunger,
20         String boundary, String location)
21     {
22         this.picture = picture;
23         this.food = food;
24         this.hunger = hunger;
25         this.boundary = boundary;
26         this.location = location;
27     }
28
29     void eat() {
30         System.out.println("eating...");
31     }
32     void makeNoise() {
33         System.out.println("make noise...");
34     }
35     void sleep() {
36         System.out.println("sleeping...");
37     }
38     void roam() {
39         System.out.println("roaming...");
40     }
41 }
42 class Canine extends Animal {
43
44     public Canine(String picture, String food, int hunger,
45         String boundary, String location)
46     {
47         super(picture, food, hunger, boundary, location);
48     }
49     void roam() {
50         System.out.println("Canine roaming...");
```

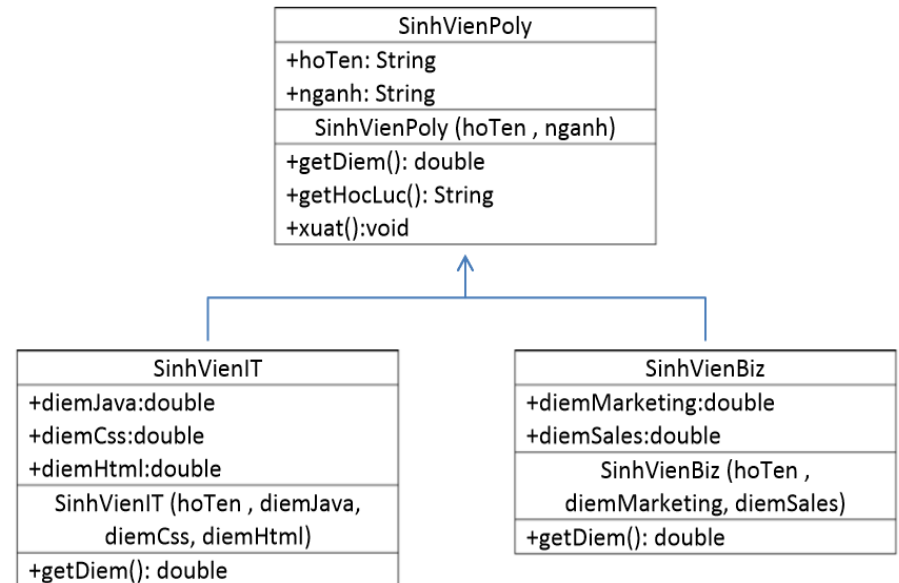
```
51     }
52 }
53 class Dog extends Canine {
54     String chaseCatName;
55     public Dog(String picture, String food, int hunger, String boundary,
56         String location, String chaseCatName)
57     {
58         super(picture, food, hunger, boundary, location);
59         this.chaseCatName = chaseCatName;
60     }
61     void makeNoise() {
62         System.out.println("Dog make noise Gow...");
63     }
64     void eat() {
65         System.out.println("Dog eating milk...");
66     }
67     public void chaseCat() {
68         System.out.println("chasing cat...");
69     }
70     public String toString()
71     { return picture + " " + food + " " + boundary + " " + chaseCatName;
72     }
73 }
74
75 class Wolf extends Canine {
76
77     public Wolf(String picture, String food, int hunger,
78         String boundary, String location)
79     {
80         super(picture, food, hunger, boundary, location);
81     }
82     void makeNoise() {
83         System.out.println("Wolf make noise Yahooo...");
84     }
85     void eat() {
86         System.out.println("Wolf eating meat...");
87     }
88     public String toString()
89     { return picture + " " + food + " " + boundary;
```

```
90     }
91 }
92
93 public class KeThuaAnimalDemo3 {
94
95     public static void main(String[] args) {
96         // TODO code application logic here
97         Wolf w = new Wolf("WolfKing", "meat", 1, "ForestA", "Inland");
98         System.out.println(w);
99         w.eat();
100        w.makeNoise();
101        w.sleep();
102        w.roam();
103    }
104 }
105
106 }
```

```
Output - KeThuaAnimalDemo3 (run) X
run:
WolfKing meat ForestA
Wolf eating meat...
Wolf make noise Yahooo...
sleeping...
Canine roaming...
BUILD SUCCESSFUL (total time: 0 seconds)
```

Bài tập về Tính kế thừa (20 phút)

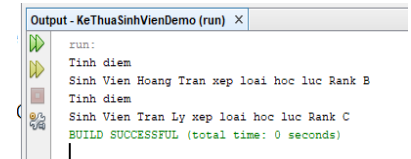
- Viết chương trình tạo lớp SinhVienPoly và hai lớp SinhVienIT và SinhVienBiz kế thừa từ SinhVien theo hình vẽ bên.
- Lớp **SinhVienPoly** gồm hai thuộc tính là hoTen, ngành cùng các phương thức như getDiem(); getHocLuc và xuất(). Trong đó getHocLuc được tính $Fail \leq 4, 4 < D < 6; 6 \leq C < 7; 7 \leq B < 8.5; A \geq 8.5$
- Lớp **SinhVienIT** bao gồm thêm 3 thuộc tính, và overriding phương thức getDiem là trung bình cộng của 3 thuộc tính trên
- Lớp **SinhVienBiz** bao gồm thêm 2 thuộc tính và overriding phương thức tính điểm là trung bình cộng của 2 thuộc tính trên



Gợi ý

```
8 class SinhVienPoly
9 {
10     public String hoTen;
11     public String ngành;
12
13     SinhVienPoly(String hoTen, String ngành)
14     {
15         //Student add more Code here
16     }
17     public double getDiem()
18     {
19         System.out.println("Tinh diem");
20         return 1;
21     }
22     public String tinhHocLuc(double diem)
23     {
24         String result;
25         //Student add more Code here
26         return result;
27     }
28
29     public void xuat(String hl)
30     {
31         System.out.println("Sinh Vien "+ hoTen +" xep loai hoc luc "+hl);
32     }
33 }
34 class SinhVienIT extends SinhVienPoly
35 {
36     //Student add more Code here
37
38     SinhVienIT(String hoTen, String ngành, double diemJava,
39         double diemCss, double diemHtml)
40     {
41         //Student add more Code here
42     }
43 }
```

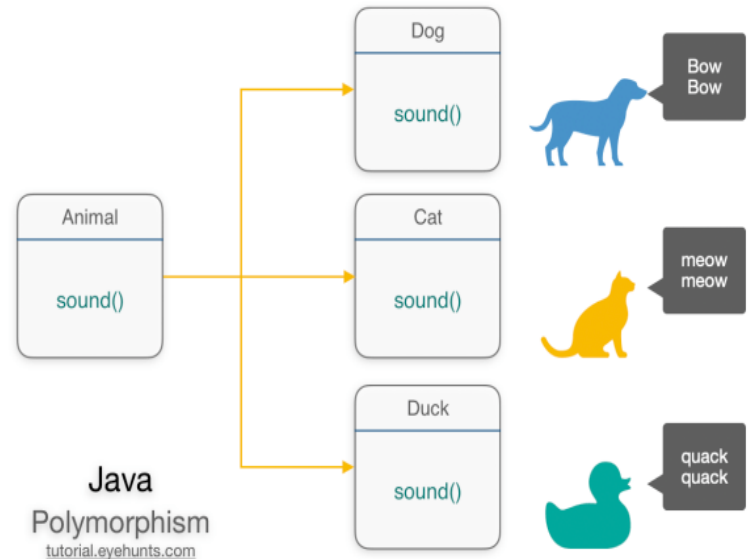
```
44     public double getDiem()
45     {
46         //Student add more Code here
47     }
48 }
49 class SinhVienBiz extends SinhVienPoly
50 {
51     //Student add more Code here
52
53     SinhVienBiz(String hoTen, String ngành, double diemMarketing,
54         double diemSales)
55     {
56         super(hoTen,ngành);
57         //Student add more Code here
58     }
59
60     public double getDiem()
61     {
62         //Student add more Code here
63     }
64
65 }
66 public class KeThuaSinhVienDemo {
67
68     public static void main(String[] args) {
69         // TODO code application logic here
70         SinhVienIT svIT=new SinhVienIT("Hoang Tran","Phan Mem",8.5,9,6.3);
71         double diem=svIT.getDiem();
72         String hocluc=svIT.tinhHocLuc(diem);
73         svIT.xuat(hocluc);
74         SinhVienBiz svBiz=new SinhVienBiz("Tran Ly","Thuong Mai",8,4.2);
75         diem=svBiz.getDiem();
76         hocluc=svBiz.tinhHocLuc(diem);
77         svBiz.xuat(hocluc);
78     }
79
80 }
```



```
run:
Tinh diem
Sinh Vien Hoang Tran xep loai hoc luc Rank B
Tinh diem
Sinh Vien Tran Ly xep loai hoc luc Rank C
BUILD SUCCESSFUL (total time: 0 seconds)
```


Đa hình Polymorphism

- Đa hình là khái niệm mô tả đối tượng có khả năng mang nhiều hình thái khác nhau, đối tượng khác nhau.
 - **phương thức có được viết lại và trùng tên trong cùng 1 lớp hay khác lớp**, tùy vào kiểu dữ liệu đối tượng, phương thức được gọi.
 - Ví dụ ta khai báo một mảng kiểu Animal, nghĩa là một mảng để chứa các đối tượng thuộc loại Animal.
 - Khi i chạy từ 0 tới 4, animals[i] lần lượt chứa Dog, Cat, Wolf, Hippo, Lion
-
- Hai cơ chế thể hiện tính đa hình trong Java:
 - phương thức nạp chồng: overloading method
 - phương thức ghi đè: overriding methods





Overloading vs. Overriding

- Overloading cho phép ghi lại code nhiều phương thức cùng tên trong cùng class, nhưng nội dung khác nhau. Tùy vào kiểu dữ liệu của đối tượng, mà phương thức được gọi thích hợp. Nếu có thì phương thức gọi tiếng kêu của chó..
- Overriding cho phép ghi đè 2 phương thức cùng tên, một ở lớp cha, và một ở lớp con, nội dung khác nhau
- Overloading cho phép định nghĩa thuật toán giống nhau cho các đối tượng khác nhau
- Overriding định nghĩa thuật toán giống nhau theo nhiều các hiểu khác nhau cho các đối tượng khác nhau
- Lớp con ghi đè phương thức của lớp cha thì sẽ che dấu phương thức lớp cha.
- Mục đích của ghi đè là sửa lại phương thức của lớp cha trong lớp con.

Lớp trừu tượng abstract

- Từ khóa **abstract** được sử dụng để định nghĩa lớp và phương thức trừu tượng
- Lớp trừu tượng là lớp mà các hành vi chưa xác định rõ.
- Lớp chứa phương thức trừu tượng thì phải là lớp trừu tượng.
- Phương thức **trừu tượng là phương thức không có phần thân xử lý** và được khai báo bằng từ khóa **abstract**
- Trong lớp trừu tượng có thể định nghĩa các phương thức cụ thể hoặc khai báo các trường dữ liệu.
- **Không sử dụng new** để tạo đối tượng từ lớp trừu tượng

```
abstract public class DongVat{  
    abstract public void speak();  
}
```

```
DongVat cho = new Cho();  
DongVat meo = new Meo();  
DongVat vit = new Vit();  
  
cho.speak();  
meo.speak();  
vit.speak();
```

```
public class Cho extends DongVat{  
    public void speak(){  
        System.out.println("Woof");  
    }  
}
```

```
public class Meo extends DongVat{  
    public void speak(){  
        System.out.println("Meo");  
    }  
}
```

```
public class Vit extends DongVat{  
    public void speak(){  
        System.out.println("Quack");  
    }  
}
```

Ví dụ về tính Đa hình

```
12 public abstract class Animal {
13     public String picture;
14     public String food;
15     public int hunger;
16     public String boundary;
17     public String location;
18     public Animal()
19     {}
20     public Animal(String picture, String food, int hunger,
21         String boundary, String location)
22     {
23         this.picture = picture;
24         this.food = food;
25         this.hunger = hunger;
26         this.boundary = boundary;
27         this.location = location;
28     }
29
30     public void eat()
31     {System.out.println("eating...");}
32     public abstract void makeNoise();
33     public void sleep() {
34         System.out.println("sleeping...");
35     }
36     public void roam() {
37         System.out.println("roaming...");
38     }
39 }
40 public class Cat extends Animal {
41     public Cat()
42     {}
43     public Cat(String picture, String food, int hunger, String boundary,
44         String location)
45     {
46         super(picture, food, hunger, boundary, location);
47     }
48     public void makeNoise() {
49         System.out.println("Cat make noise...");
50     }
```

```
51     public String toString()
52     { return picture + " " + food + " " + boundary;
53     }
54 }
55
56 public class Dog extends Animal {
57     String chaseCatName;
58     public Dog()
59     {}
60     public Dog(String picture, String food, int hunger, String boundary,
61         String location, String chaseCatName)
62     {
63         super(picture, food, hunger, boundary, location);
64         this.chaseCatName = chaseCatName;
65     }
66     public void chaseCat() {
67         System.out.println("chasing cat...");
68     };
69
70     @Override
71     public void makeNoise() {
72         System.out.println("Dog make noise...");
73     }
74     public String toString()
75     { return picture + " " + food + " " + boundary + " " + chaseCatName;
76     }
77 }
78 public class Lion extends Animal {
79     public Lion()
80     {}
81     public Lion(String picture, String food, int hunger,
82         String boundary, String location)
83     {
84         super(picture, food, hunger, boundary, location);
85     }
86     public void makeNoise() {
87         System.out.println("Lion make noise...");
88     }
```

```
89     public String toString()
90     { return picture + " " + food + " " + boundary;
91     }
92 }
93 public class PolymorphismDemo {
94
95     /**
96     * @param args the command line arguments
97     */
98     public static void main(String[] args) {
99         // TODO code application logic here
100        // TODO code application logic here
101        Animal[] kingdom = new Animal[5];
102        kingdom[0] = new Dog();
103        kingdom[1] = new Cat();
104        kingdom[2] = new Cat();
105        kingdom[3] = new Lion();
106        kingdom[4] = new Cat();
107        for(int i=0; i<kingdom.length; i++)
108            kingdom[i].makeNoise();
109
110    }
111
112 }
```

```
Output - PolymorphismDemo (run) x
run:
Dog make noise...
Cat make noise...
Cat make noise...
Lion make noise...
Cat make noise...
BUILD SUCCESSFUL (total time: 0 seconds)
```



Bài tập thực hành
