



TRƯỜNG ĐẠI HỌC
GIAO THÔNG VẬN TẢI
University of Transport and Communications

HỌC PHẦN: CÔNG NGHỆ JAVA

CHƯƠNG 3. OOP – LỚP VÀ ĐỐI TƯỢNG TRONG JAVA

Giảng viên: Đào Thị Lệ Thủy
Khoa Công nghệ Thông tin



Nội dung

- ✓ Các khái niệm về OOP
 - ✓ Khái niệm về OOP
 - ✓ Các đặc trưng của OOP: lớp, đối tượng, thuộc tính, phương thức, kế thừa, đóng gói, đa hình và trừu tượng.
- ✓ OOP trong Java
 - ✓ Lớp và đối tượng
 - ✓ Tính đóng gói
 - ✓ Tính kế thừa
 - ✓ ...
- ✓ Mở rộng
 - ✓ Inner/Outer class.
 - ✓ Debug



CHƯƠNG 3. OOP – LỚP VÀ ĐỐI TƯỢNG TRONG JAVA

CÁC KHÁI NIỆM VỀ OOP

Khái niệm về OOP

- ❖ Khái niệm về đối tượng – object, thực thể – instance ra đời từ những năm 1960 tại MIT trong dự án phát triển phần mềm sử dụng ngôn ngữ Simular.
- ❖ Khái niệm OOP được đưa ra bởi Xerox PARC trong dự án phát triển sử dụng ngôn ngữ Smalltalk – coi các đối tượng là nền tảng của mọi thao tác, tính toán.
- ❖ Simular với tư tưởng của mình chính là nền tảng cho việc xây dựng và phát triển các ngôn ngữ cũng như tư tưởng về OOP. Các ngôn ngữ khác trên tư tưởng đó cũng được hình thành như C++, Java, ...
- ❖ 1980s, OOP trở thành tư tưởng để phát triển mà đại diện tiêu biểu chính là C++: chiếm 73% thị phần 1990.
- ❖ Ngày nay: hầu hết các ngôn ngữ đều hỗ trợ OOP.

Khái niệm về OOP

Lập trình hướng đối tượng - OOP
(Object Oriented Programming)

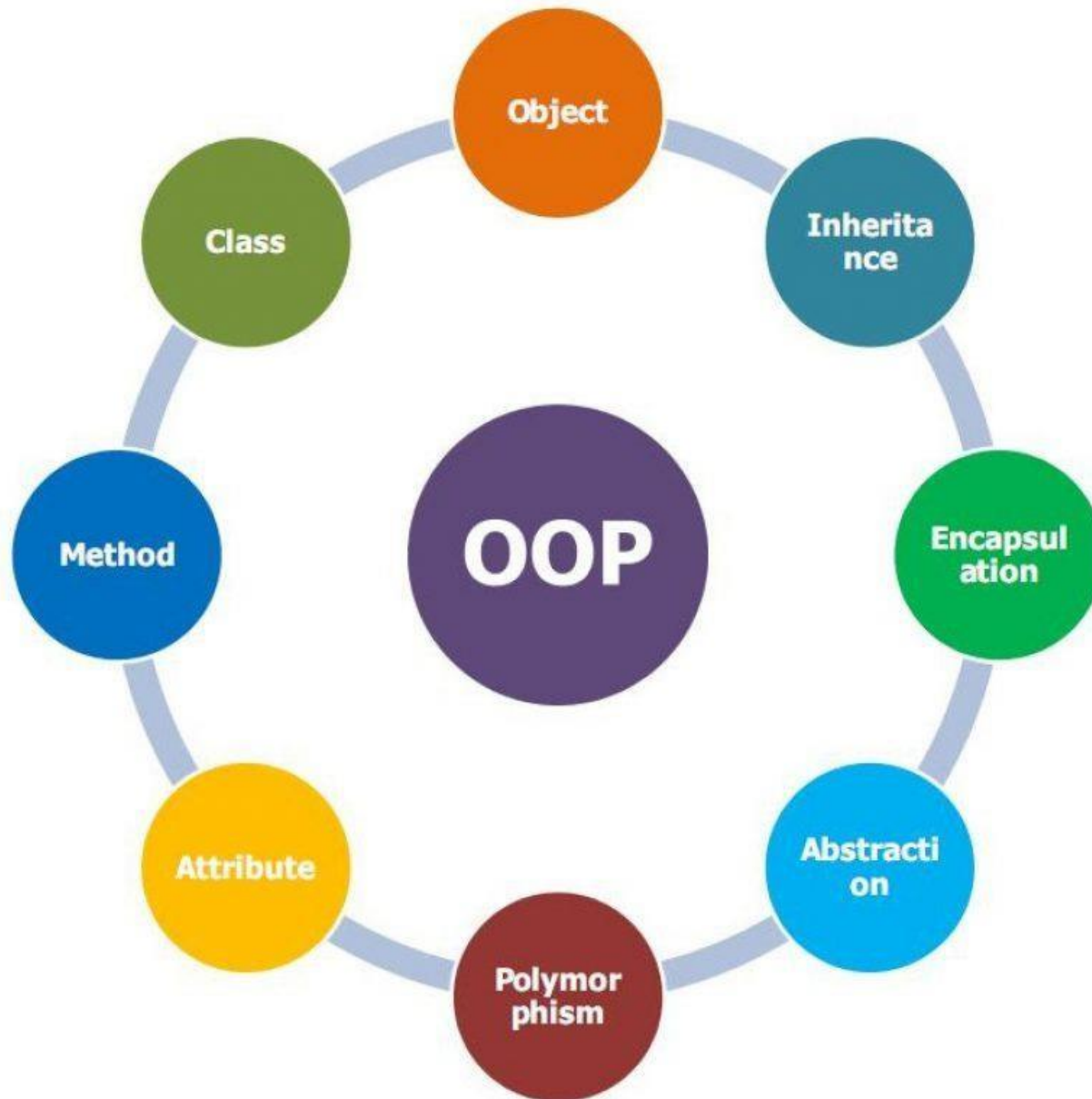
Được xem là:

- Cách tiếp cận mới, hiệu quả, bảo mật hơn
- Cung cấp cấu trúc các module chương trình một cách rõ ràng
- Dễ dàng bảo trì, sửa đổi, nâng cấp
- Giảm bớt thao tác viết trình (tái sử dụng lại code)
- Mô tả chân thực thế giới thực

Tính chất cơ bản:

- Đóng gói
- Kế thừa
- Đa hình
- Trừu tượng

Khái niệm về OOP



OOP – Class vs Object

❑ Đối tượng (Object):

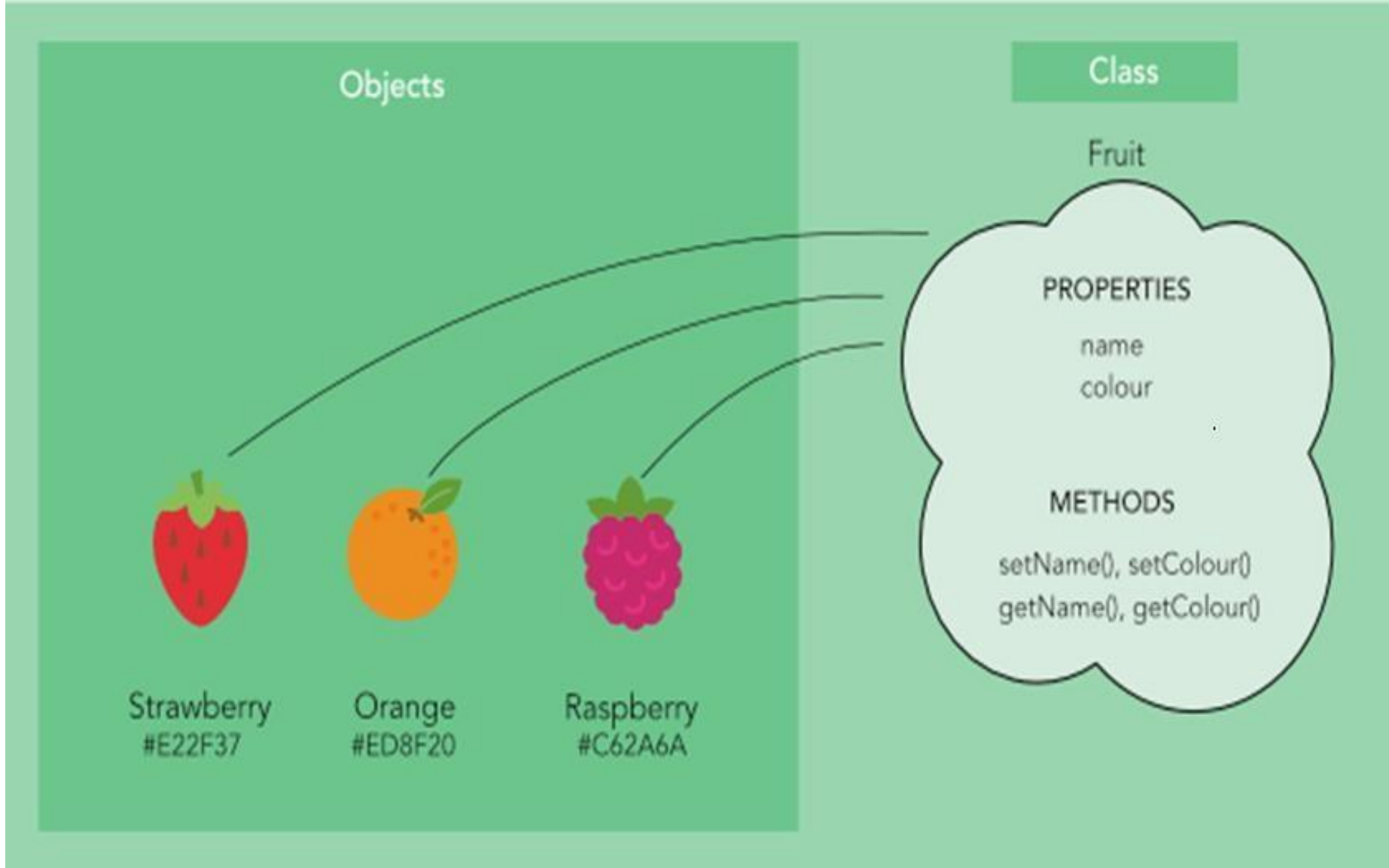
- Là những thực thể trong thế giới thực
- Mỗi đối tượng được đặc trưng bởi các thuộc tính và các hành vi riêng của nó



$$3x + 15 = 0$$

$$A = \frac{20}{10}$$

OOP – Class vs Object



OOP – Class vs Object

- ❑ Lớp (Class) là một khuôn mẫu được sử dụng để mô tả các đối tượng cùng loại.



Breed: Bulldog
Size: large
Colour: light gray
Age: 5 years



Breed: Beagle
Size: large
Colour: orange
Age: 6 years



Breed: German Shepherd
Size: large
Colour: white & orange
Age: 6 years

Dog_Object

Dog_Object

Dog_Object

Dog

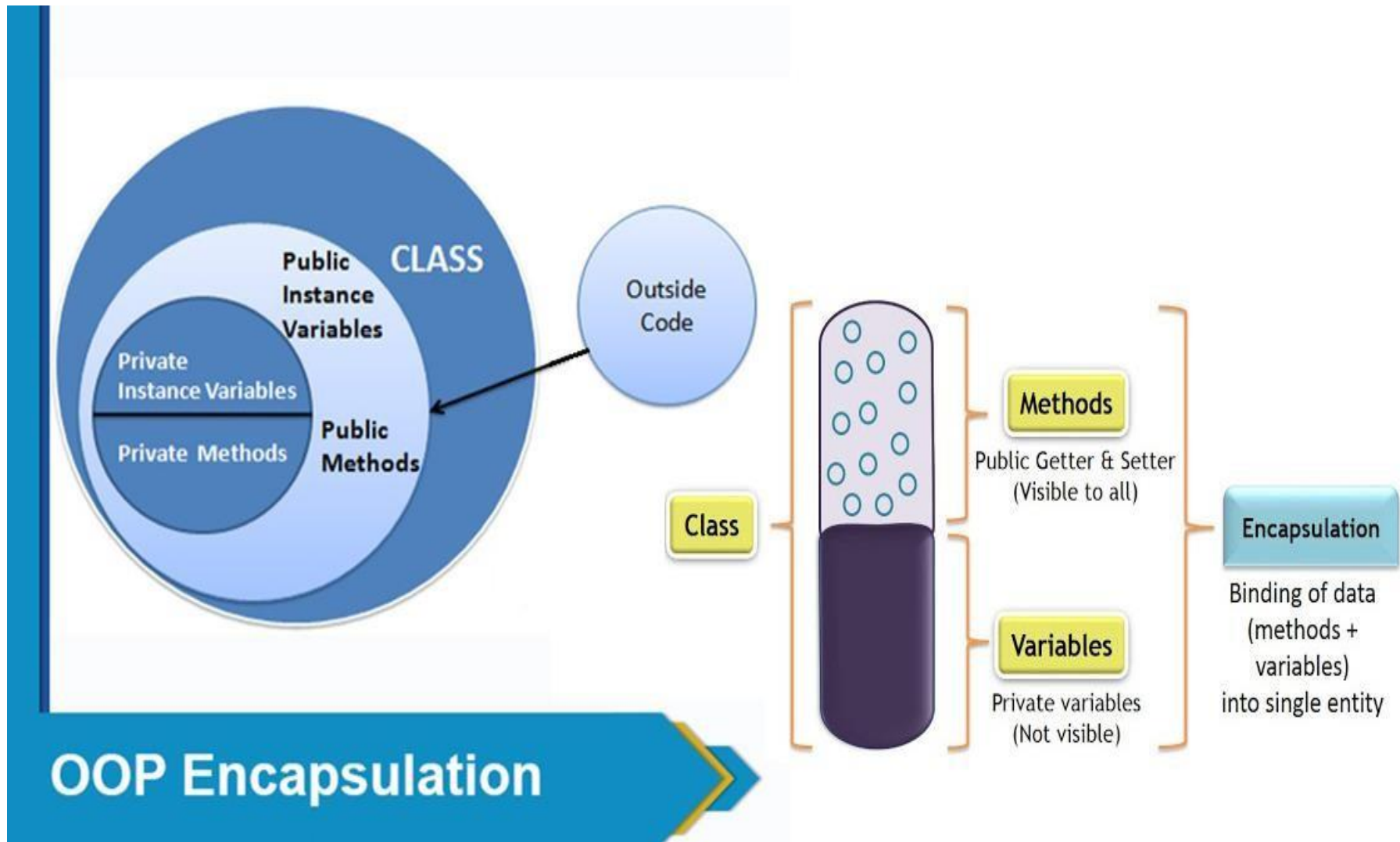
Fields

Breed
Size
Colour
Age

Methods

Eat()
Run()
Sleep()
Name()

OOP – Encapsulation



OOP – Encapsulation

- ❖ **Data hiding:** che giấu việc triển khai bên trong của lớp. Các thuộc tính cũng như một số phương thức được bảo vệ, các lập trình viên chỉ có thể truy cập thông qua các phương thức cho phép.
- ❖ **Increased flexibility:** các thuộc tính của lớp có thể giới hạn ở mức chỉ đọc hoặc chỉ ghi (read, write only) thông qua các getters và setters.
- ❖ **Reusability:** tính đóng gói cho phép chúng ta có thể tái sử dụng và dễ dàng thay đổi khi có yêu cầu mà không cần thay đổi đến các đoạn mã của người khác.
- ❖ **Easy testing:** việc đóng gói dễ dàng cho phép thực hiện unit test – tách dữ liệu và phương thức độc lập.

OOP – Encapsulation

Access Modifiers in Java



01

default

02

public

03

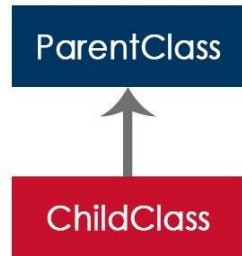
private

04

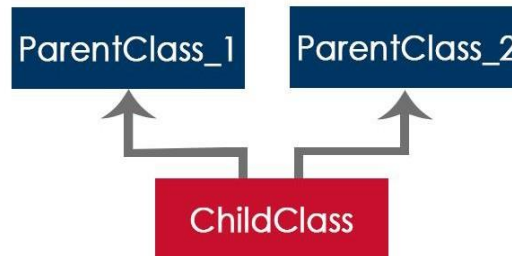
protected

OOP – Inheritance

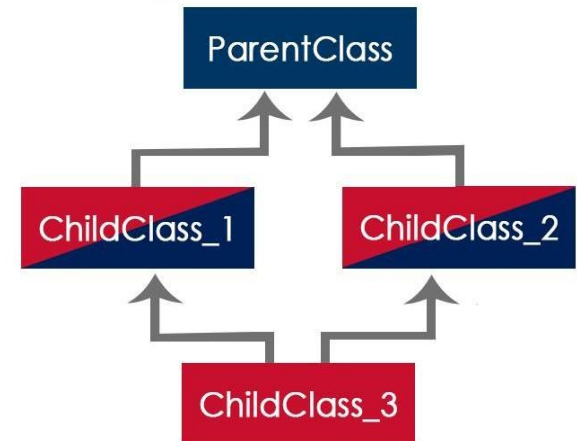
Simple Inheritance



Multiple Inheritance



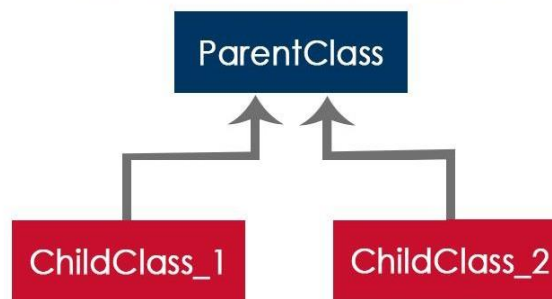
Hybrid Inheritance



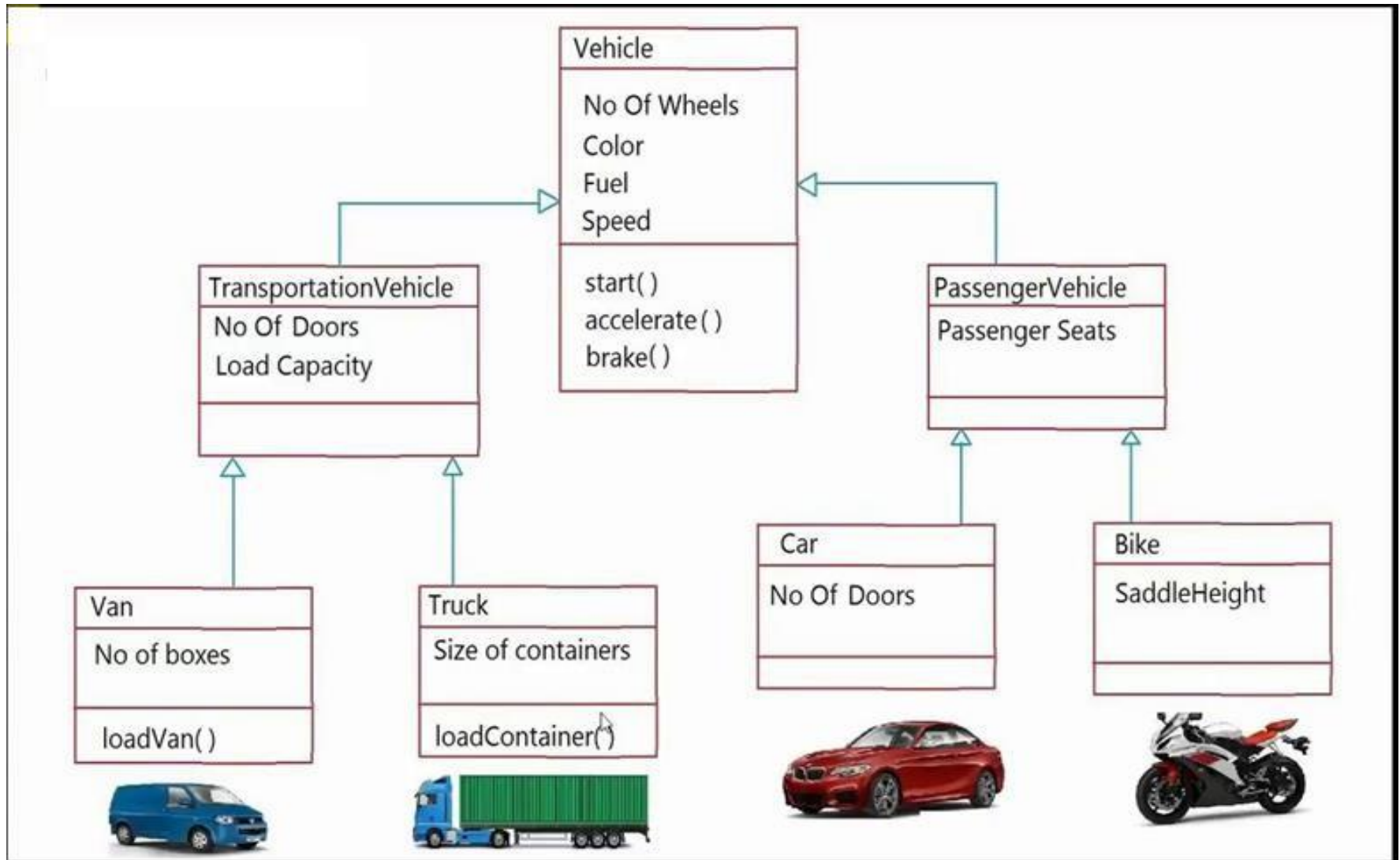
Multi Level Inheritance



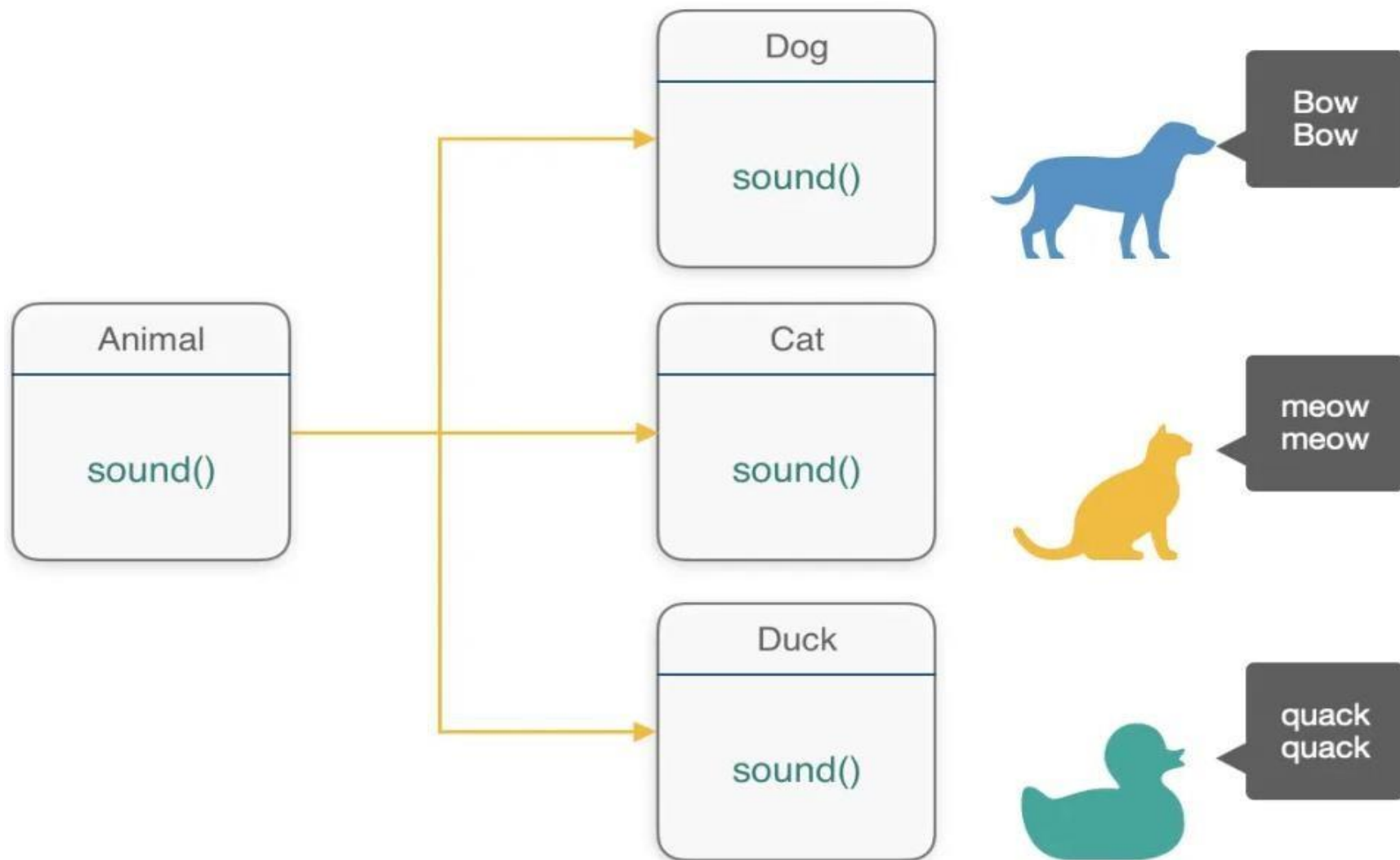
Hierarchical Inheritance



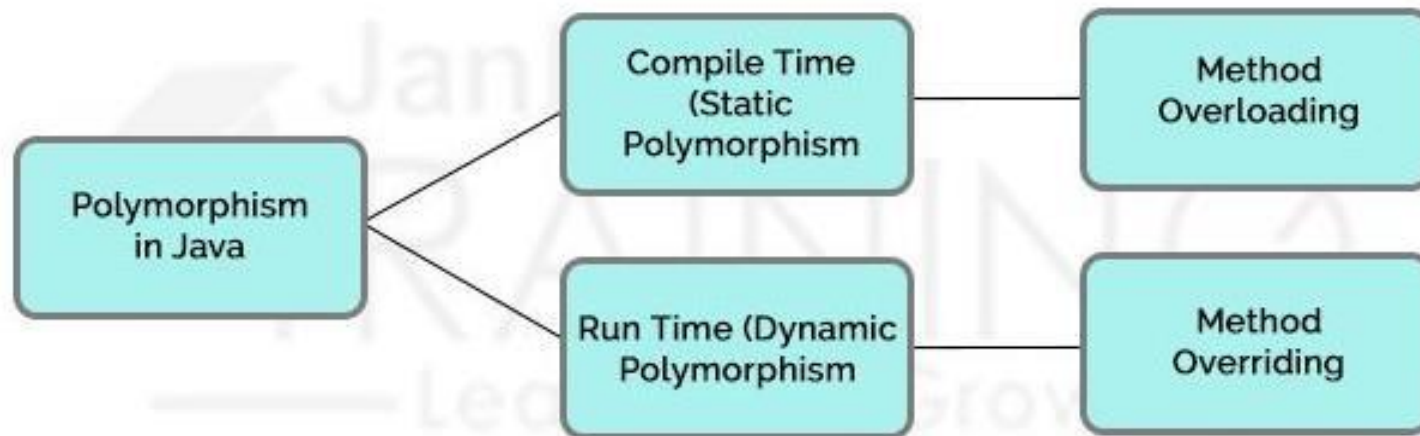
OOP – Inheritance



OOP – Polymorphism



OOP – Polymorphism



Overriding

```
class Dog{
    public void bark(){
        System.out.println("woof ");
    }
}
class Hound extends Dog{
    public void sniff(){
        System.out.println("sniff ");
    }
    public void bark(){
        System.out.println("bowl");
    }
}
```

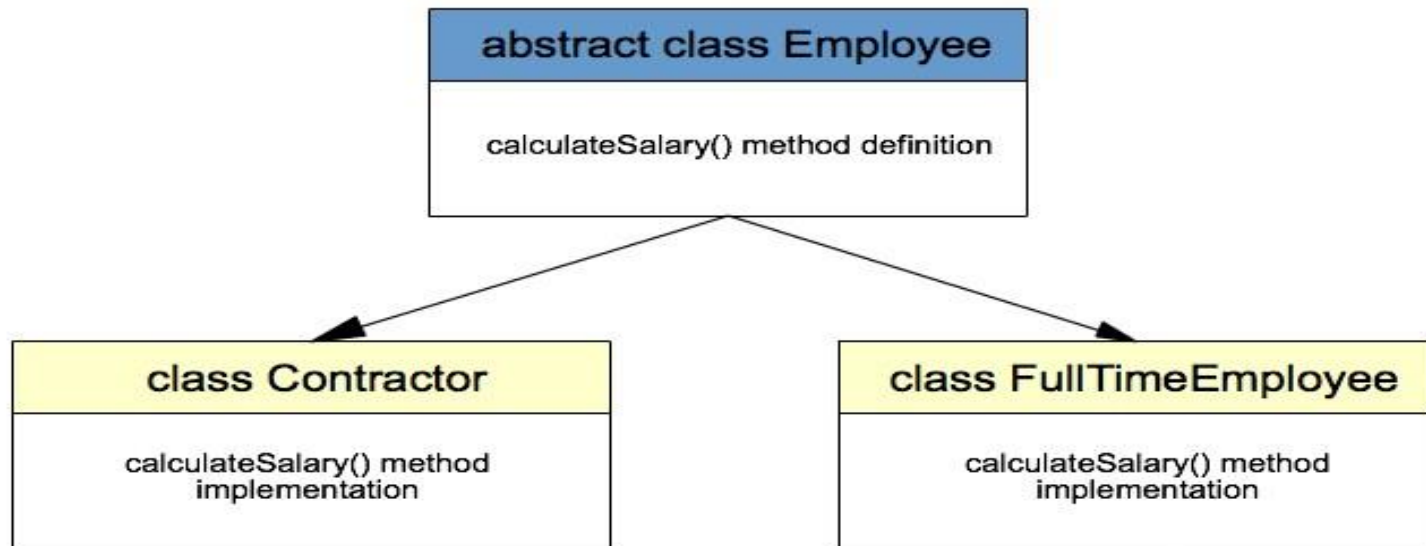
Same Method Name,
Same parameter

Overloading

```
class Dog{
    public void bark(){
        System.out.println("woof ");
    }
    //overloading method
    public void bark(int num){
        for(int i=0; i<num; i++)
            System.out.println("woof ");
    }
}
```

Same Method Name,
Different Parameter

OOP – Abstraction



CHƯƠNG 3. OOP – LỚP VÀ ĐỐI TƯỢNG TRONG JAVA

XÂY DỰNG LỚP ĐỐI TƯỢNG

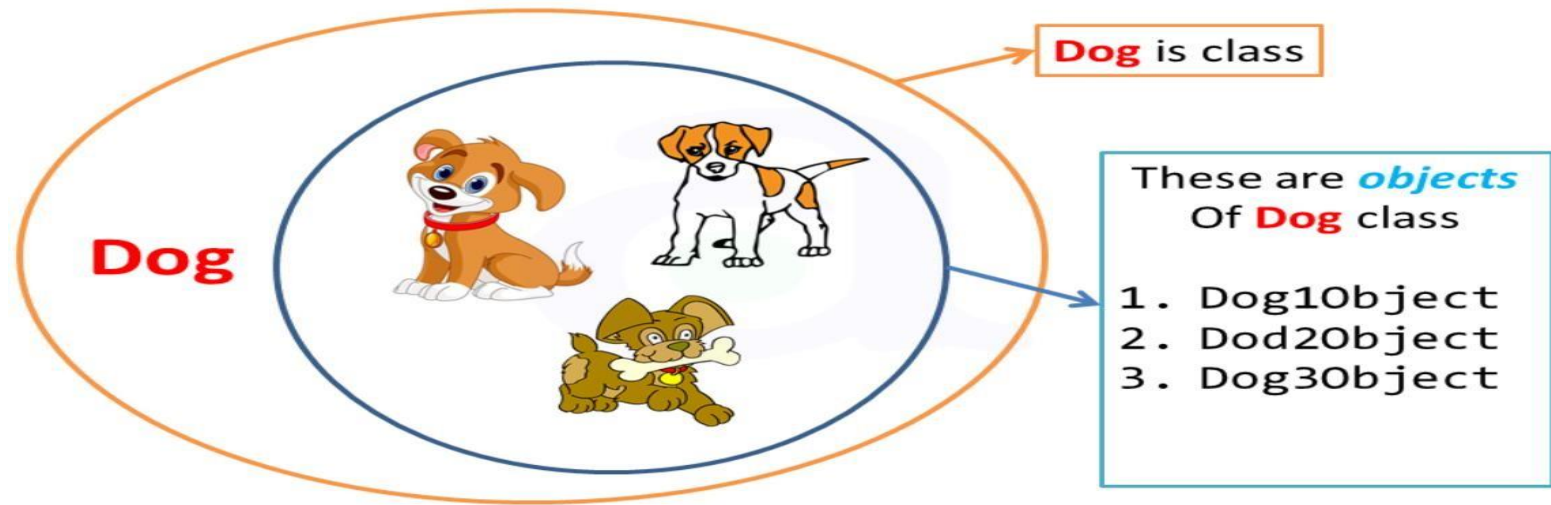
Khái niệm

- ❑ Một lớp trong Java có thể gồm:
 - Thuộc tính (properties)
 - Hàm tạo (constructors)
 - Phương thức (methods)

- ❑ Đối tượng là một thể hiện cụ thể của lớp

Lớp và đối tượng

- ✓ Đối tượng – object: là một thực thể có thuộc tính và hành vi nhằm xác định cụ thể.
- ✓ Đối tượng: thuộc tính + phương thức.
- ✓ Lớp – class: khuôn mẫu chung của một họ các đối tượng cho phép định nghĩa các thuộc tính và các phương thức của họ đối tượng đó cũng như cho phép tạo ra các đối tượng thực thể – instance.

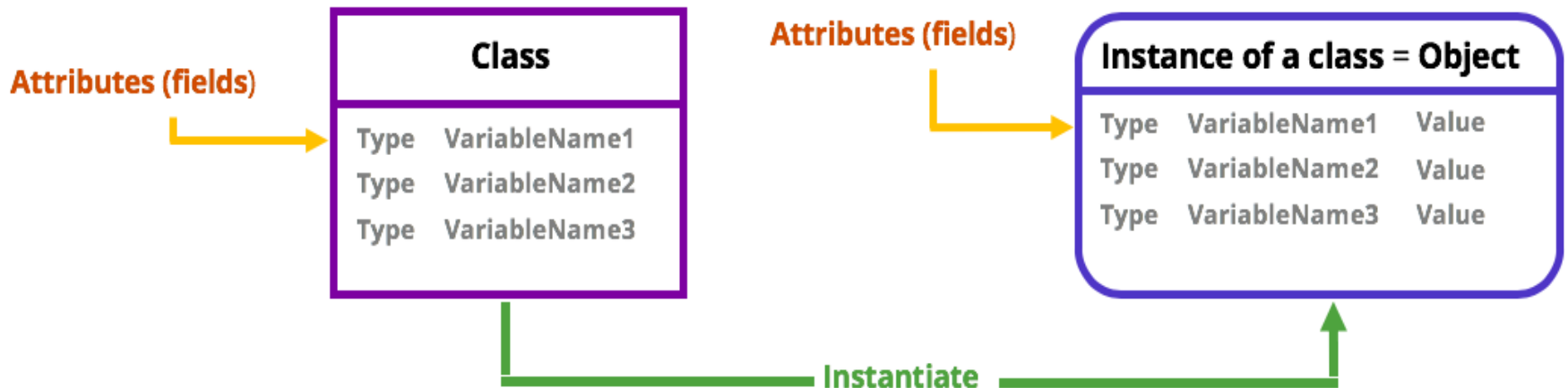


Lớp và đối tượng

No	Đối tượng	Lớp
1.	Đối tượng là thể hiện của một lớp.	Lớp là một khuôn mẫu hay thiết kế để tạo ra các đối tượng trong cùng một nhóm.
2.	Đối tượng là một thực thể trong thực tiễn.	Lớp là một nhóm các đối tượng tương tự nhau.
3.	Đối tượng là một thực thể vật lý	Lớp là một thực thể logic
4.	Đối tượng được tạo ra chủ yếu từ từ khóa new. Student s=new Student();	Lớp được khai báo bằng việc sử dụng từ khóa class. class Student{}
5.	Đối tượng có thể được tạo nhiều lần.	Lớp được khai báo một lần duy nhất.
6.	Đối tượng được cấp bộ nhớ khi nó được tạo ra.	Lớp không được cấp bộ nhớ khi nó được tạo ra.

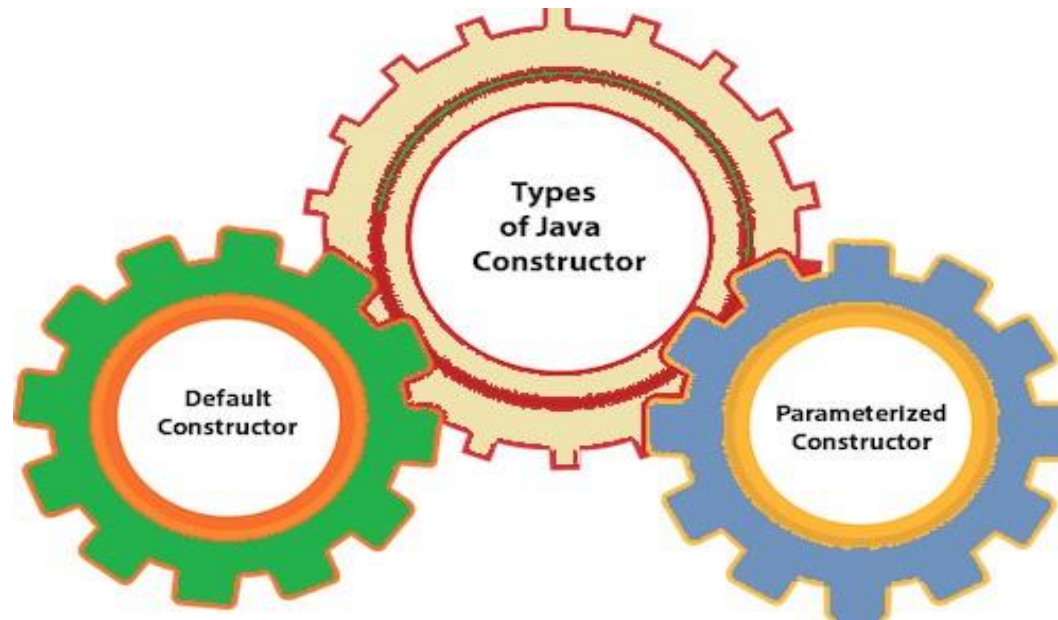
Lớp và đối tượng – Thuộc tính

- ✓ Cho phép định danh các đặc tính của một lớp và sẽ nhận giá trị ứng với mỗi đối tượng thực thể được tạo ra từ lớp.
- ✓ Kiểu dữ liệu + tên thuộc tính + giá trị mặc định.
- ✓ Thường là **private** và được xác định thông thường qua hàm tạo, hàm setters.



Lớp và đối tượng – hàm tạo

- ✓ Là phương thức đặc biệt cho phép khởi tạo một đối tượng từ lớp tương ứng.
- ✓ Có tên trùng với tên lớp và không có kiểu dữ liệu trả về.
- ✓ Hàm tạo: mặc định, không đối, có đối.
- ✓ Từ hàm tạo, để tạo đối tượng ta có thể dùng toán tử ***new***.



Khai báo lớp

```
[access modifier] class <class_name> {  
    //properties  
    [access modifier] <data_type> <property_name> [=value]  
  
    //constructors  
    [access modifier] <class_name>() {...}  
  
    //methods  
    [access modifier] <data_type> <method_name>([<data_type>arg...]){...}  
}
```

access modifier: [public|private|protected] hoặc không có (default)

constructors: để tạo đối tượng, có thể có nhiều hàm tạo, nếu không xây dựng hàm tạo thì java sẽ tạo hàm khởi tạo mặc định

Khai báo lớp – Ví dụ

```
public class Person {  
    //properties  
    public int id;  
    public String name;  
    public String address;  
  
    //constructor  
    public Person() {  
  
    }  
  
    //methods  
    public void print() {  
        System.out.println("Id: " + id + ";Name: " +  
name + "; Adress: " + address);  
    }  
}
```

TẠO ĐỐI TƯỢNG

□ Tạo đối tượng:

<Tên_lớp> <Biến_đối_tượng> = **new** <Tên_lớp> ([Danh_sách_tham_số])

□ Ví dụ:

```
public static void main(String[] args){  
    Person ps = new Person();  
    ps.name = "David";  
    ps.address = "LosAngle";  
    ps.hienthi();  
}
```

- ❖ Toán tử **new** được sử dụng để tạo đối tượng
- ❖ Biến **ps** chứa tham chiếu tới đối tượng
- ❖ Sử dụng dấu chấm (.) để truy xuất các thành viên của lớp

Phạm vi truy xuất

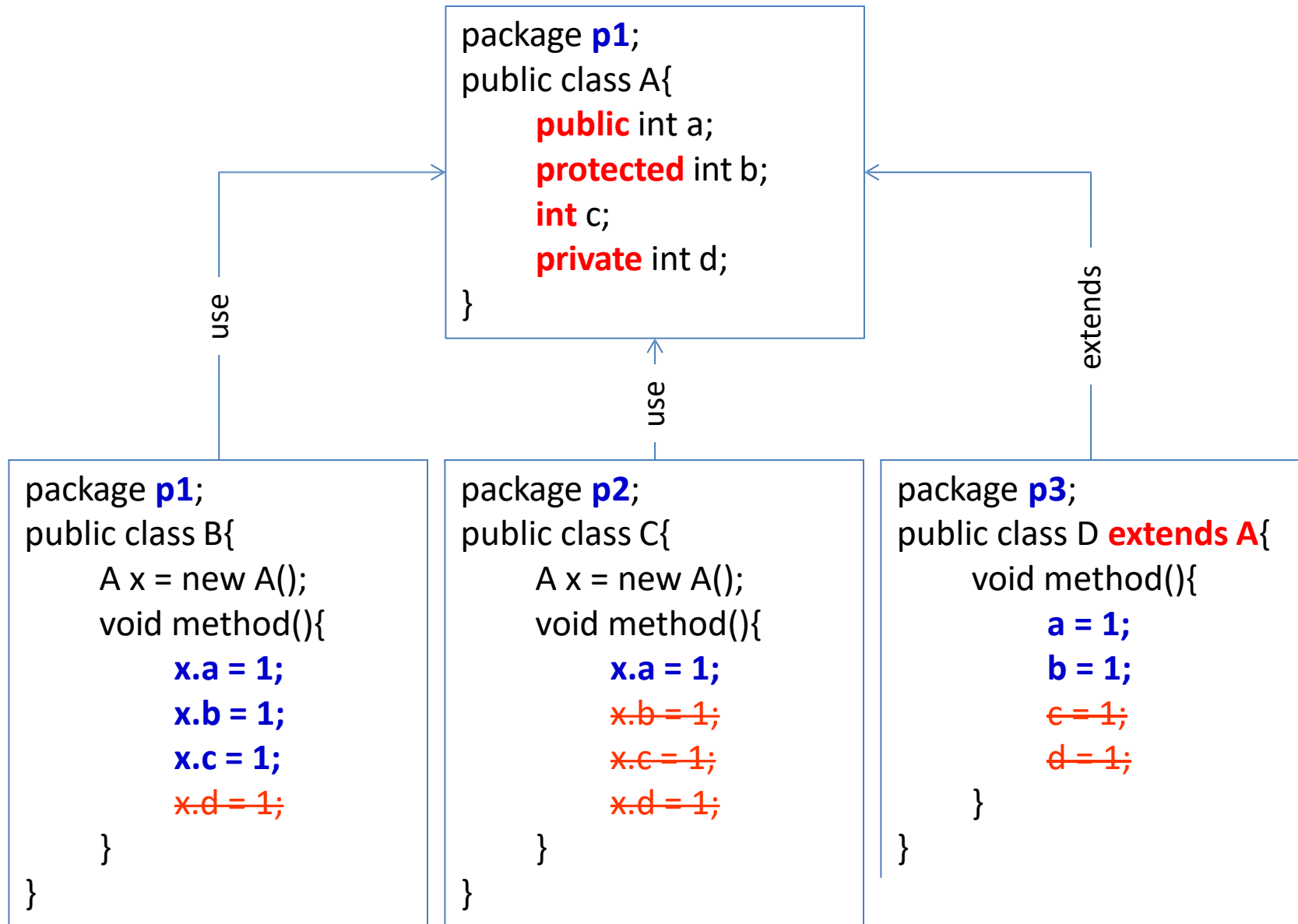
- ❑ Đặc tả truy xuất được sử dụng để định nghĩa khả năng cho phép truy xuất đến các thành viên của lớp. Trong java có 4 đặc tả khác nhau:
 - ❖ **private**: chỉ được phép sử dụng nội bộ trong class
 - ❖ **public**: công khai hoàn toàn
 - ❖ **{default}**:
 - Là public đối với các lớp truy xuất cùng gói
 - Là private với các lớp truy xuất khác gói.
 - ❖ **protected**: tương tự {default} nhưng cho phép kế thừa dù lớp con và cha khác gói.
- ❑ Mức độ che dấu tăng dần theo chiều mũi tên

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

Access Modifier

Access Modifier	Bên trong lớp	Bên trong package	Bên ngoài package chỉ bởi lớp con	Bên ngoài package
private	C	K	K	K
default	C	C	K	K
protected	C	C	C	K
public	C	C	C	C

Ví dụ truy xuất



Access Modifier - private

```
1 package org.o7planning.tutorial.am.privatedemo;
```

```
2  
3 public class Person {
```

```
4     public String name;
```

```
5     private String secret;
```

```
6  
7     public Person(String name) {
```

```
8         this.name= name;
```

```
9     }
```

```
10  
11     public void showSecret() {
```

```
12         System.out.println("Secret: " + this.secret);
```

```
13     }
```

```
14  
15     class Diary {
```

```
16  
17         public void Logging() {
```

```
18             System.out.println("Secret: " + secret);
```

```
19         }
```

```
20     }
```

```
21  
22 }
```

```
23  
24 }
```

Đây là một trường private

Trường private này có thể truy cập bất kỳ nơi đâu trong nội bộ class. Bao gồm cả trong Inner class.

Access Modifier - private

```
1 package org.o7planning.tutorial.am.privateconstructor;
2
3 public class Animal {
4
5     private String name;
6
7     private Animal(String name) {
8         this.name = name;
9     }
10
11     public String getName() {
12         return this.name;
13     }
14
15 }
```

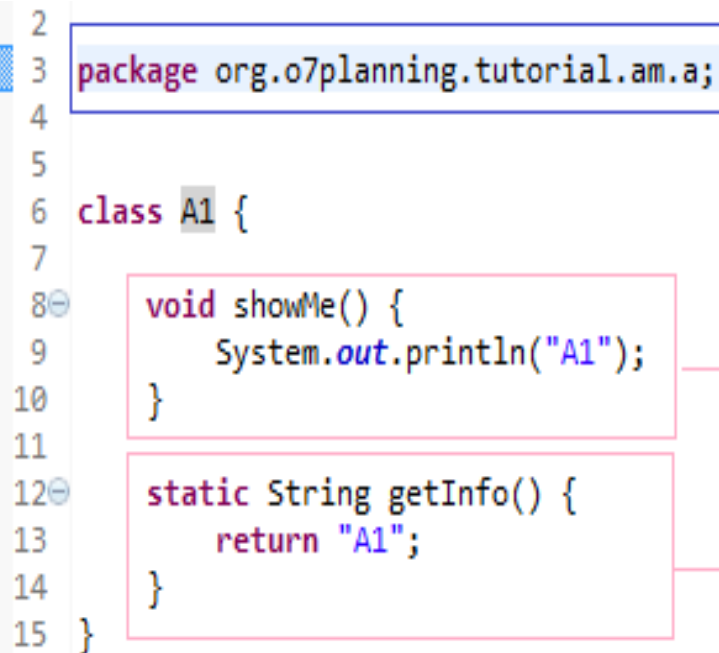
AnimalTest.java

```
1 package org.o7planning.tutorial.am.privateconstructor;
2
3 public class AnimalTest {
4
5
6     public static void main(String[] args) {
7
8         Animal a = new Animal("Tiger");
9     }
10 }
11
```

The constructor Animal(String) is not visible

Access Modifier - default

```
2  
3 package org.o7planning.tutorial.am.a;  
4  
5  
6 class A1 {  
7  
8     void showMe() {  
9         System.out.println("A1");  
10    }  
11  
12     static String getInfo() {  
13         return "A1";  
14    }  
15 }
```



ATest.java

```
1  
2 package org.o7planning.tutorial.am.a;  
3  
4  
5 public class ATest {  
6  
7     public static void main(String[] args) {  
8  
9         A1 a = new A1();  
10  
11         a.showMe();  
12  
13         A1.getInfo();  
14     }  
15 }
```


Access Modifier - default

```
1 // Một class với access modifier mặc định.  
2 // (Không chỉ định public).  
3 class MyClass {  
4  
5     // Một trường private access modifier.  
6     private int myField;  
7  
8     // Một trường access modifier mặc định.  
9     // (Không chỉ định public, protected, private).  
10    String myField2;  
11  
12    // Một method với access modifier mặc định.  
13    // (Không chỉ định public, protected, private).  
14    void showMe() {  
15  
16    }  
17 }
```

Access Modifier - default

The diagram illustrates the default access modifiers in Java across three files: A1.java, A2.java, and ChildOfA2.java. Red lines and numbered annotations (1) through (4) point to specific code elements.

A1.java

```
1
2
3 package org.o7planning.tutorial.am.a;
4
5
6 class A1 {
7
8     void showMe() {
9         System.out.println("A1");
10    }
11
12    static String getInfo() {
13        return "A1";
14    }
15 }
```

A2.java

```
1
2
3 package org.o7planning.tutorial.am.a;
4
5 public class A2 {
6
7     void showMe() {
8         System.out.println("A2");
9     }
10
11    static String getInfo() {
12        return "A2";
13    }
14 }
```

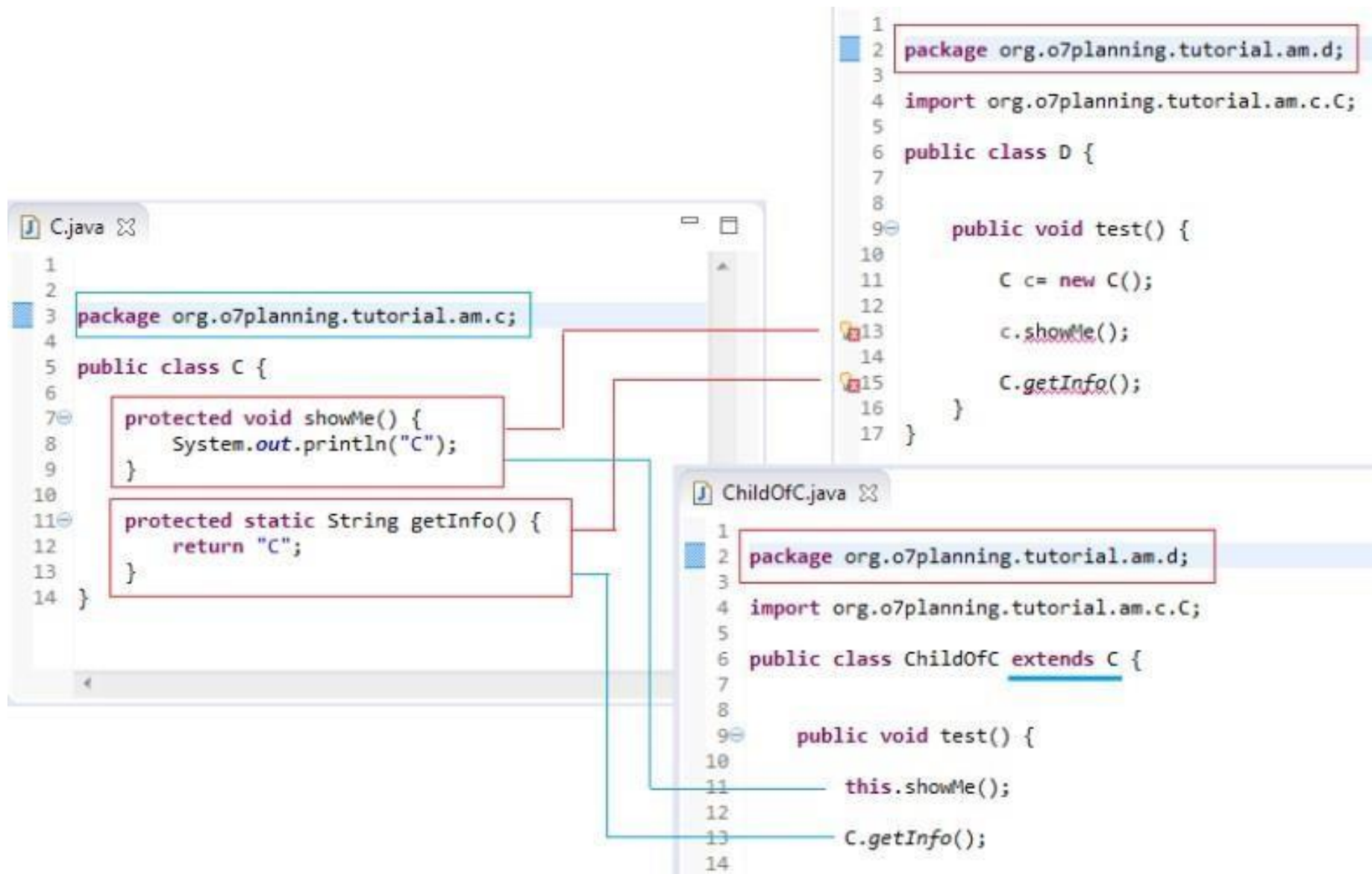
ChildOfA2.java

```
1
2 package org.o7planning.tutorial.am.b;
3
4 import org.o7planning.tutorial.am.a.A1;
5
6 import org.o7planning.tutorial.am.a.A2;
7
8 public class ChildOfA2 extends A2 {
9
10    public void test() {
11
12        this.showMe();
13
14        A2 a = new A2();
15
16        a.showMe();
17
18        A2.getInfo();
19
20    }
21 }
```

Annotations:

- (1) Points to the `package` statement in A1.java.
- (2) Points to the `void` keyword in A1.java.
- (3) Points to the `void` keyword in A2.java.
- (4) Points to the `static` keyword in A2.java.

Access Modifier - protected



Access Modifier - public

```
E.java
1
2
3 package org.o7planning.tutorial.am.e;
4
5 public class E {
6
7     public static int MY_CONSTANT = 100;
8
9     public int myField;
10
11     public E() {
12
13     }
14
15     public void myMethod() {
16
17     }
18
19 }
```

```
F.java
1 package org.o7planning.tutorial.am.f;
2
3 import org.o7planning.tutorial.am.e.E;
4
5 public class F {
6
7
8     public void test() {
9
10         System.out.println("Constant = " + E.MY_CONSTANT);
11
12         E e = new E() ;
13
14         e.myField = 2000;
15
16         e.myMethod();
17
18     }
19 }
```

TỪ KHÓA STATIC

- **static property**: Dữ liệu chung cho mọi đối tượng cùng lớp
- Nằm ngoài vùng nhớ của đối tượng (mang ý nghĩa của 1 biến toàn cục)

```
public class Student {  
    public int count1;  
    public static int count2;  
    public Student() {  
        count1++;  
        count2++;  
    }  
    public static void main(String[] args) {  
        Student s1 = new Student();  
        Student s2 = new Student();  
        Student s3 = new Student();  
        System.out.println(s1.count1);  
        System.out.println(s1.count2);  
        System.out.println(s2.count1);  
        System.out.println(s2.count2);  
        System.out.println(s3.count1);  
        System.out.println(s3.count2);  
        System.out.println(Student.count2);  
    }  
}
```

TỪ KHÓA STATIC

- **static method:** Phương thức cho phép sử dụng mà không cần khai báo đối tượng thuộc lớp.

```
public class StaticDemo {  
    public static String name;  
  
    public static void printName() {  
        System.out.println(name);  
    }  
  
    public static void main(String[] args) {  
        StaticDemo.name = "Nguyen Van An";  
        StaticDemo.printName();  
    }  
}
```

CON TRỎ THIS

- ❑ **this** được sử dụng để đại diện cho đối tượng hiện tại.
- ❑ **this** được sử dụng trong lớp để tham chiếu tới các thành viên của lớp (properties và method)
- ❑ Sử dụng **this.properties** để phân biệt **thuộc tính** với các biến cục bộ hoặc tham số của phương thức

```
public class MyClass{  
    private int a;  
    MyClass (int a){  
        this.a = a;  
    }  
}
```

Thuộc
tính

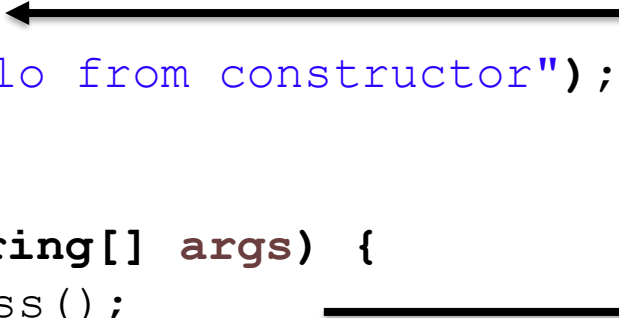
Tham
số

```
public class MyClass{  
    private int a;  
    MyClass(int b){  
        a = b;  
    }  
}
```

HÀM TẠO (CONSTRUCTOR)

- ❑ Hàm tạo là một phương thức đặc biệt được sử dụng để tạo đối tượng. Nếu không khai báo hàm tạo, Java tự động cung cấp hàm tạo mặc định (không tham số)
- ❑ Đặc điểm của hàm tạo
 - ❖ Có tên trùng với tên lớp
 - ❖ Không có giá trị trả về
- ❑ Ví dụ

```
public class MyClass {  
  
    public MyClass() {  
        System.out.println("Hello from constructor");  
    }  
  
    public static void main(String[] args) {  
        MyClass obj = new MyClass();  
    }  
}
```



HÀM TẠO (CONSTRUCTOR)

- Trong một lớp có thể có nhiều hàm tạo
- Hàm tạo nào được sử dụng sẽ phụ thuộc vào danh sách tham số

```
public class Person {  
    public String name;  
    public String address;  
    public int birthYear;  
  
    public Person(String name, String address, int birthyear) {  
        this.name = name;  
        this.address = address;  
        this.birthYear = birthyear;  
    }  
  
    public Person(String name, String address) {  
        this.name = name;  
        this.address = address;  
    }  
  
    public static void main(String[] args) {  
        Person p1 = new Person("An", "Hanoi", 1990);  
        Person p2 = new Person("Binh", "Hanoi");  
    }  
}
```

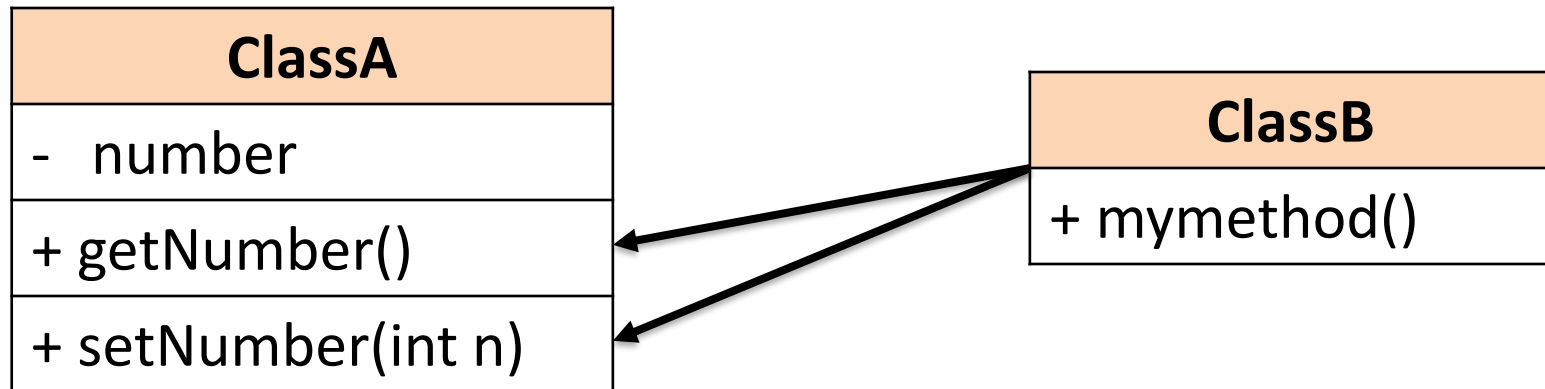
HÀM TẠO (CONSTRUCTOR)

- Để gọi đến hàm tạo khác trong cùng lớp có thể dùng từ: **this(<ds tham số>)**

```
public class Person {  
    public String name;  
    public String address;  
    public int birthYear;  
  
    public Person(String name, String address, int birthyear) {  
        this(name, address);  
        this.birthYear = birthyear;  
    }  
  
    public Person(String name, String address) {  
        this.name = name;  
        this.address = address;  
    }  
  
    public static void main(String[] args) {  
        Person p1 = new Person("An", "Hanoi", 1990);  
        Person p2 = new Person("Binh", "Hanoi");  
    }  
}
```

getters & setters

- ❖ Trong tiếp cận hướng đối tượng, các thuộc tính được bảo vệ (Encapsulation- các thuộc tính của lớp bị ẩn đi so với các lớp khác.).
- ❖ Để truy xuất các thuộc tính private từ bên ngoài, sử dụng các phương thức getters và setters.
- ❖ Thông thường, các getters và setters được định nghĩa cho mỗi thuộc tính và thường là **public**.



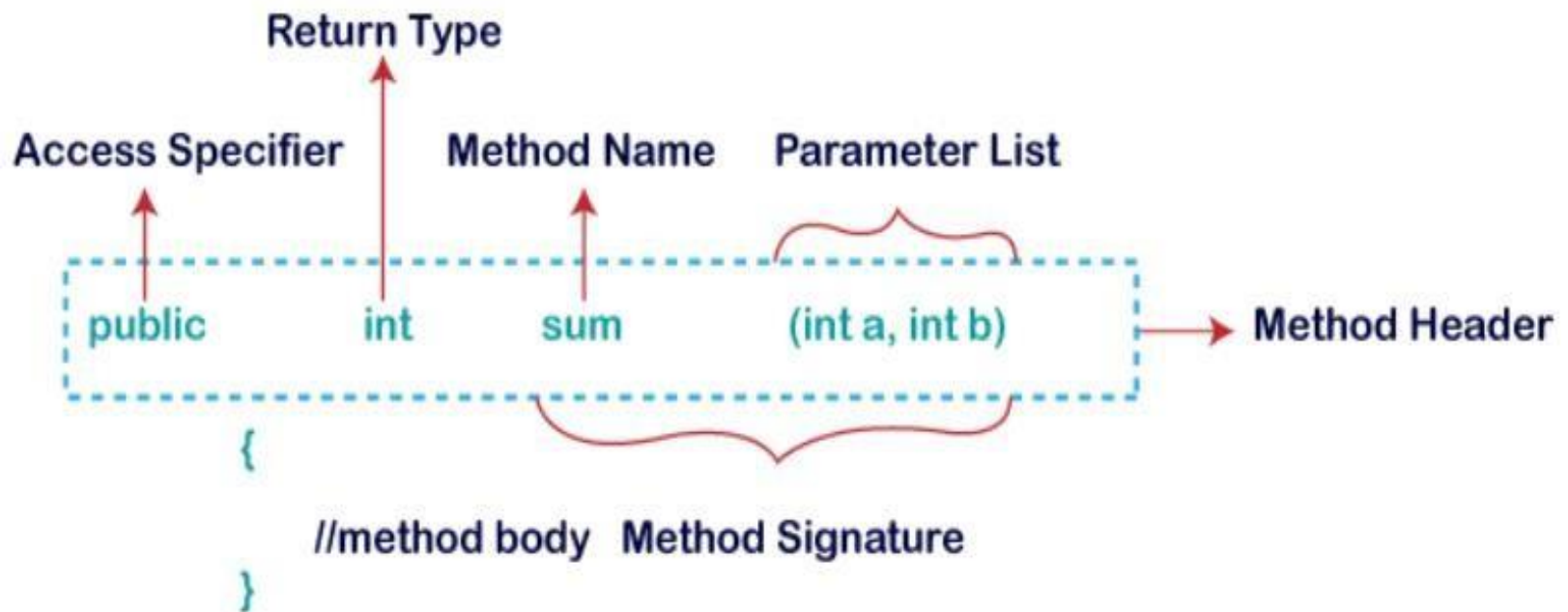
getters & setters

```
public class SinhVien{
    private String hoTen;
    private double diem;
    public void setHoTen(String hoTen){
        this.hoTen = hoTen;
    }
    public String getHoTen(){
        return this.hoTen;
    }
    public void setDiem(double diem){
        if(diem < 0 || > 10){
            System.out.println("Điểm không hợp lệ");
        }
        else{
            this.diem = diem;
        }
    }
    public double getDiem(){
        return this.diem;
    }
}
```

```
public class MyClass{
    public static void main(String[] args){
        SinhVien sv = new SinhVien();
        sv.setHoTen("Nguyễn Văn Tý");
        sv.setDiem(20);
    }
```

methods

- ✓ Phương thức cho phép định nghĩa hành vi của đối tượng
 - và được thực thi khi có lời gọi từ đối tượng.
- ✓ Mỗi đối tượng có thể định nghĩa các phương thức để từ đó có thể đáp ứng yêu cầu bài toán.
- ✓ **Chú ý:** phạm vi truy xuất thông thường của phương thức
 - là **public** hoặc **protected**.



Ví dụ

Xây dựng lớp Circle gồm:

- Thuộc tính tọa độ tâm $O(x,y)$, bán kính r
- Hàm tạo đủ 3 tham số
- Viết các getter/setter
- Phương thức in thông tin ra màn hình
- Phương thức main để thực thi

Lớp và đối tượng – ví dụ

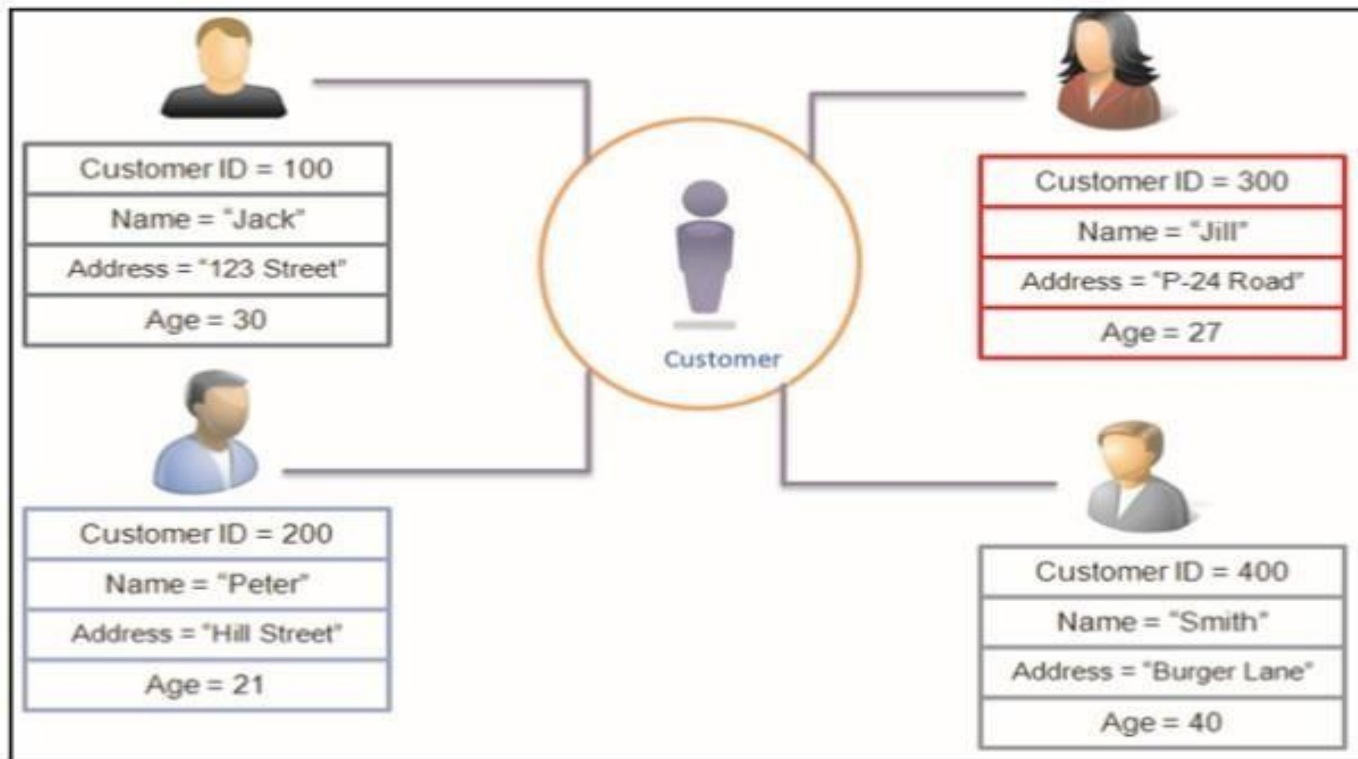
```
public class Circle {  
    //properties  
    private x,y,r;  
  
    //constructor  
    public Circle(int x, int y, int r) {  
        this.x=x;  
        this.y=y;  
        this.r=r;  
    }  
    //getter/setter  
    public int getX() {  
        return x;  
    }  
    public void setX(int x) {  
        this.x = x;  
    }  
}
```

Lớp và đối tượng – ví dụ

```
public int getY() {  
    return y;  
}  
public void setY(int y) {  
    this.y = y;  
}  
//methods  
public void show() {  
    System.out.println("x = " + x + ";y = " + y + "; r = "  
+ r);  
}  
public static void main(String [] args) {  
    Circle c1 = new Circle(3,5,6);  
    c1.show();  
    Circle c2 = new Circle(4,5,8);  
    c2.show();  
}  
}
```


VÍ DỤ

Xây dựng lớp khách hàng



VÍ DỤ

Trường hợp 1:

```
package demo;  
public class Customer  
{  
    public int customerID;  
    public String customerName;  
    public String customerAddress;  
    public int customerAge;  
    public Customer()  
    {  
    }  
}
```

VÍ DỤ

Trường hợp 1 (TT):

```
package demo;
public class TestCustomer {
    public static void main(String[] args) {
        Customer c1 = new Customer();
        c1.customerID = 100;
        c1.customerName = "Jack";
        c1.customerAddress = "123 Street";
        c1.customerAge = 30;
        System.out.println("Mã KH:" + c1.customerID);
        System.out.println("Tên KH:" + c1.customerName);
        System.out.println("Địa chỉ:" + c1.customerAddress);
        System.out.println("Tuổi:" + c1.customerAge);
    }
}
```

VÍ DỤ

Trường hợp 2:

```
package demo;
public class Customer
{
    private int customerId;
    public String customerName;
    public String customerAddress;
    public int customerAge;
    public Customer()
    {
    }
    public void setCustomerID(int cID) {
        customerId = cID;
    }
    public int getCustomerID() {
        return (customerId);
    }
}
```

VÍ DỤ

Trường hợp 2 (TT):

```
package demo;
public class TestCustomer {
    public static void main(String[] args) {
        Customer c1 = new Customer();
        c1.setCustomerID(100);
        c1.customerName = "Jack";
        c1.customerAddress = "123
Street"; c1.customerAge = 30;
        System.out.println("Mã KH:" + c1.getCustomerID(););
        System.out.println("Tên KH:" + c1.customerName);
        System.out.println("Địa chỉ:" + c1.customerAddress);
        System.out.println("Tuổi:" + c1.customerAge);
    }
}
```

VÍ DỤ

Trường hợp 3 (TT):

```
package demo;  
public class Nguoi  
{   public String customerName;  
    public String customerAddress;  
    public int customerAge;  
}
```

```
public class Customer extends Nguoi  
{  
    private int customerID;  
    public Customer()  
    {  
    }  
    public void setCustomerID(int cID)  
    {customerID = cID;  
    }  
    public int getCustomerID() {  
    return (customerID);  
    }  
}
```

VÍ DỤ

Trường hợp 3 (TT):

```
package demo;
public class TestCustomer {
    public static void main(String[] args) {
        Customer c1 = new Customer();
        c1.setCustomerID(100);
        c1.customerName = "Jack";
        c1.customerAddress = "123
Street"; c1.customerAge = 30;
        System.out.println("Mã KH:" + c1.getCustomerID(););
        System.out.println("Tên KH:" + c1.customerName);
        System.out.println("Địa chỉ:" + c1.customerAddress);
        System.out.println("Tuổi:" + c1.customerAge);
    }
}
```

BÀI TẬP

DEMO



SinhVien

+ hoTen, diaChi, gioiTinh: String

+ diem: double

(phương thức set và get cho mỗi thuộc tính)

+ **nhap()**: void

+ **xepLoai()**: String

+ **xuat()**: void

+ SinhVien()

+ SinhVien(hoTen, diaChi, gioiTinh, diem)

Xây dựng lớp mô tả sinh viên như mô hình trên.

Trong đó **nhap()** cho phép nhập thông tin từ bàn phím;

xuat() cho phép xuất thông tin ra màn hình;

xepLoai() dựa vào điểm để xếp loại học lực

Sử dụng 2 hàm tạo để tạo 2 đối tượng sinh viên.

MẢNG ĐỐI TƯỢNG

- ❑ Mảng đối tượng trong Java chứa các đối tượng.
- ❑ Các phần tử mảng lưu trữ các vị trí biến tham chiếu của các đối tượng.
- ❑ Cú pháp khai báo mảng đối tượng

<Tên_lớp> <Biến_mảng> = new <Tên_lớp> [Kích_thước_mảng];

```
class Test{
    int a;
    int b;
    public void setData(int a1, int b1){
        a = a1;
        b = b1;
    }
    // Hàm in thông tin đối tượng
    public void showData(){
        System.out.println("Giá trị của a = "+a);
        System.out.println("Giá trị của b = "+b);
    }
}
```

```
class ObjectArray{
    public static void main(String args[]){
        Test obj[] = new Test[2] ;
        // Gán dữ liệu
        obj[0].setData(1, 2);
        obj[1].setData(3, 4);
        // In đối tượng ra màn hình
        System.out.println("Phần tử mảng thứ 0");
        obj[0].showData();
        System.out.println("Phần tử mảng thứ 1");
        obj[1].showData();
    }
}
```

BÀI TẬP

DEMO



SinhVien

+ hoTen, diaChi, gioiTinh: String

+ diem: double

(phương thức set và get cho mỗi thuộc tính)

+ **nhap()**: void

+ **xepLoai()**: String

+ **xuat()**: void

+ SinhVien()

+ SinhVien(hoTen, diaChi, gioiTinh, diem)

Tạo thêm các hàm:

- nhập vào danh sách n sinh viên
- In danh sách sinh viên ra màn hình
- Hàm sắp xếp danh sách sinh viên tăng dần theo điểm
- Thực thi các phương thức trong main()

NẠP CHỒNG PHƯƠNG THỨC (OVERLOAD)

- ❑ Là trường hợp các phương thức trùng tên nhưng khác tham số (kiểu, số lượng và thứ tự)
- ❑ Phương thức nào được gọi phụ thuộc vào danh sách tham số

```
public class MyClass{  
    public void method(){...}  
    public void method(double x){...}  
    public void method(int x, double y){...}  
}
```

Trong lớp MyClass có 3 phương thức cùng tên là **method** nhưng khác nhau về tham số.

- ❑ Với lời gọi truyền vào giá trị x=5: **method(5)** sẽ chạy phương thức nào?

NẠP CHỒNG PHƯƠNG THỨC (OVERLOAD)

❑ Ví dụ

```
public class OverloadDemo {  
    public void method1() {  
        System.out.println("method1");  
    }  
    public void method1(String s) {  
        System.out.println("method1 with string arg: " + s);  
    }  
    public void method1(int i) {  
        System.out.println("method1 with int arg: " + i);  
    }  
    public static void main(String[] args) {  
        OverloadDemo d1 = new OverloadDemo();  
        d1.method1();  
        d1.method1("String");  
        d1.method1(10);  
    }  
}
```

GÓI (PACKAGE)

- ❑ Là một nhóm các class, interface, các gói khác
- ❑ Package được sử dụng để chia các class và interface thành từng gói khác nhau.
 - ❖ Việc làm này tương tự quản lý file trên ổ đĩa trong đó class (như file) và package (như folder)
- ❑ Gói là công cụ tạo khả năng tái sử dụng mã (reusable code).
 - ❖ Trong một package không có 2 class hoặc interface trùng tên

```
package package1.subpackage;  
public class ClassOfSubpackage1 {  
}
```

IMPORT PACKAGE

- ❑ Lệnh import được sử dụng để chỉ ra lớp hoặc interface đã được định nghĩa trong một package.
- ❑ Các lớp trong các gói sau sẽ tự động import
 - ❖ Các lớp cùng gói với lớp sử dụng chúng
 - ❖ java.lang sẽ được import
- ❑ Cách thức import
 - ❖ **import com.utc.*** *//import tất cả các class và interface có trong gói com.utc*

```
//import từng class
import package1.ClassOfPackage1;
import package1.subpackage.ClassOfSubpackage1;
import package2.ClassOfPackage2;

public class PackageDemo {

    public static void main(String[] args) {
        ClassOfPackage1 c1 = new ClassOfPackage1();
        ClassOfPackage2 c2 = new ClassOfPackage2();
        ClassOfSubpackage1 c3 = new ClassOfSubpackage1();
    }
}
```

IMPORT PACKAGE

- ❑ Trong Java có rất nhiều gói được sử dụng để phân các class và interface theo chức năng
 - ❖ java.util: chứa các lớp tiện ích
 - ❖ java.io: chứa các lớp vào/ra dữ liệu
 - ❖ java.lang: chứa các lớp thường dùng...

Bài tập 1

- Khai báo lớp **Diem** với 2 thuộc tính double x, y
 - Khai báo hàm tạo nhận vào x, y
 - Thực hiện getter/setter với x, y
- Khai báo lớp **DuongThang** với 2 thuộc tính A và B có kiểu **Diem**
 - Khai báo hàm tạo nhận vào A, B
 - Viết getter cho A và B
 - Viết phương thức tính độ dài đường thẳng

Bài tập 2

- Khai báo lớp **Diem** với 2 thuộc tính x và y với kiểu dữ liệu `double` nhằm biểu diễn 1 điểm trên mặt phẳng tọa độ.
- Khai báo lớp **TamGiac** với hàm tạo nhận vào 3 điểm. Viết phương thức tính chu vi và diện tích hình **TamGiac**.
- Khai báo lớp **ChuNhat** với hàm tạo nhận độ dài 2 cạnh. Viết phương thức tính chu vi và diện tích hình **ChuNhat**.
- Khai báo lớp **HinhTron** với hàm tạo nhận vào điểm O và bán kính. Viết phương thức tính chu vi và diện tích hình **HinhTron**.
- Viết chương trình chính khởi tạo 2 **TamGiac**, 2 **ChuNhat**, 2 **HinhTron** bất kỳ, in ra chu vi và diện tích các hình.