



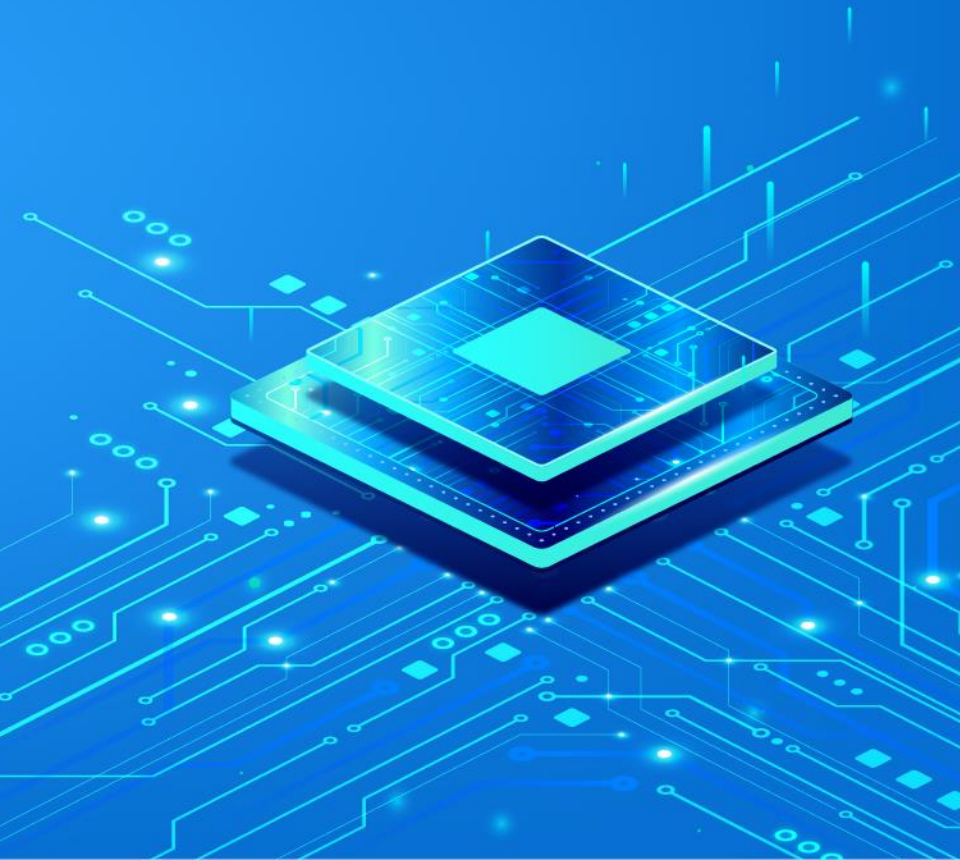
CMC UNIVERSITY



OOP: Inherit principle

PhD. Ngo Hoang Huy

Monday, July 10, 2023



Study concepts: superclass, subclass

Understand common relationships

Functions in inheritance

Using an “instanceof” operator

Casting

.

Derived and Super Classes

Object-oriented languages implement reusability of coding structure through inheritance

It refers to the relationship between classes where one class inherits the entire structure of another class

The root of our design is a relatively abstract entity, and we build upon that entity to produce progressively more concrete entities

the higher-level entities are “parent”, “base” or “super” classes

the lower-level ones built from them are “child”, “derived” or “sub” classes.

common relationships in classes:

“is-a/ a kind of”

“has-a”

Examples:

Student is a person

“A home is a house that has a family and a pet.”

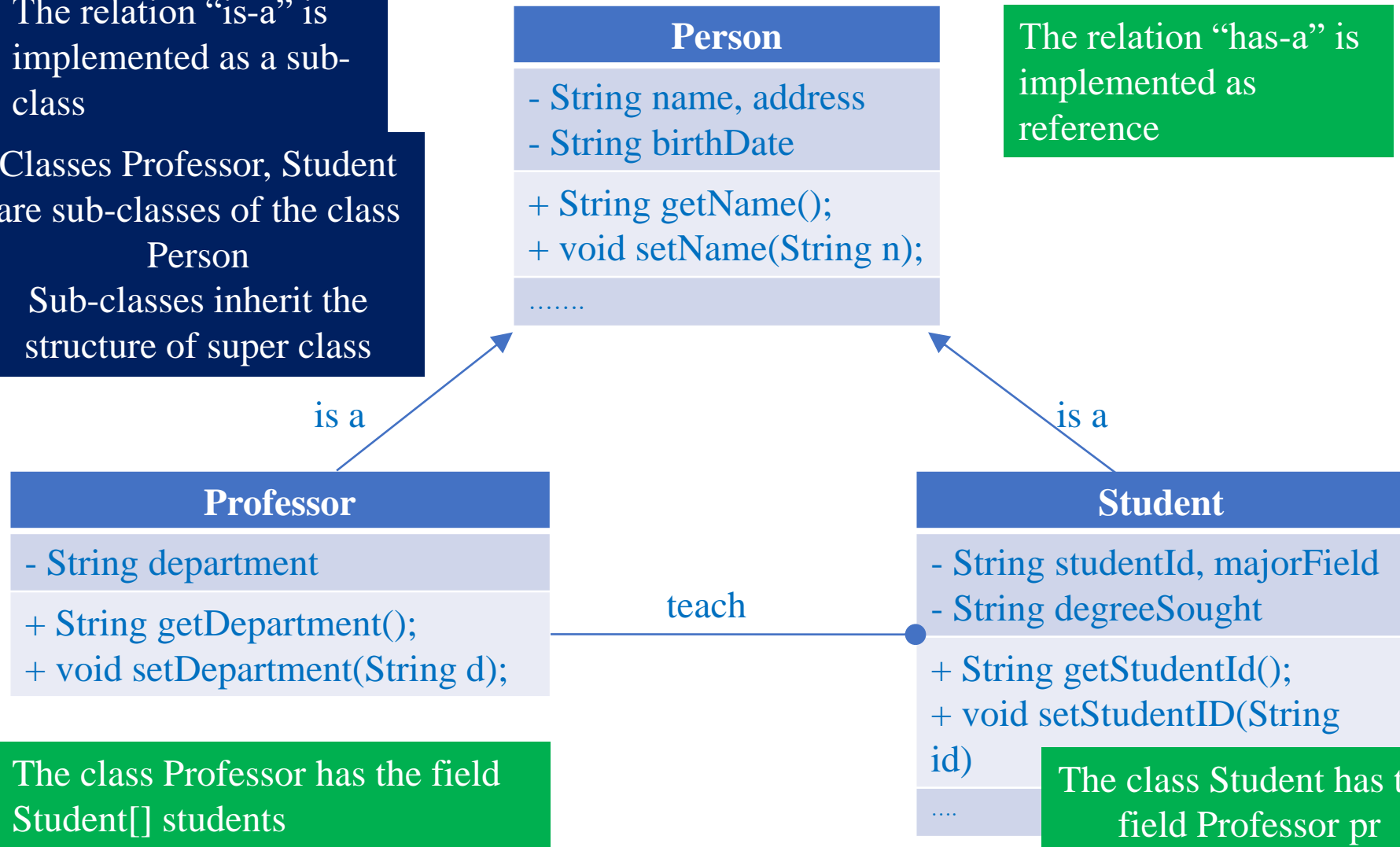
An invoice contains some products and a product can be contained in some invoices

Object-Oriented Relationships

The relation “is-a” is implemented as a sub-class

Classes Professor, Student are sub-classes of the class Person
Sub-classes inherit the structure of super class

The relation “has-a” is implemented as reference



The class Professor has the field Student[] students

The class Student has the field Professor pr

There are some sub-classes from one super class: An inheritance is a relationship where objects share a common structure: the structure of one object is a sub-structure of another object.

The **extends** keyword is used to create **sub-class**.

A class can be directly derived from only one class (Java is a single-inherited OOP language).

If a class **does not have any superclass**, then it is implicitly derived from **Object class**.

Unlike other members, **constructor cannot be inherited** (constructor of super class can not initialize sub-class objects)

How to use extends keyword

```
class Parent {  
    public void print() {System.out.println("This is the parent class.");}  
}  
class Child extends Parent{  
    public void printChild() {System.out.println("This is the child class.");}  
}  
public class Example {  
    public static void main(String[] args) {  
        Child child = new Child();  
        child.print();    // Gọi phương thức của lớp cha từ đối tượng lớp con  
        child.printChild(); // Gọi phương thức của lớp con  
    } //main  
}
```

Object class of Java

Lớp **Object** là lớp cơ sở cho tất cả các lớp trong Java. Mọi lớp trong Java đều được kế thừa từ lớp Object trực tiếp hoặc gián tiếp.

Lớp Object chứa các phương thức cơ bản:

equals(Object obj): So sánh hai đối tượng xem chúng có bằng nhau hay không.

hashCode(): Trả về mã băm (hash code) của đối tượng để tìm kiếm và lưu trữ dữ liệu hiệu quả.

toString(): Trả về một chuỗi biểu diễn của đối tượng.

getClass(): Trả về đối tượng Class biểu diễn lớp của đối tượng hiện tại. Class được sử dụng để lấy thông tin về lớp, như tên lớp, phương thức, thuộc tính, v.v.

clone(): Tạo ra một bản sao (clone) của đối tượng hiện tại. Phương thức này thường được ghi đè lại để cung cấp việc sao chép đối tượng theo cách phù hợp.

Lớp Object cũng chứa các phương thức hỗ trợ khác như **notify**(), **notifyAll**(), **wait**(), v.v., được sử dụng trong quản lý luồng (thread) trong Java.

Object class example

```
public class ObjectExample {  
    public static void main(String[] args) {  
        Object obj = new Object();  
        // Sử dụng phương thức toString() để in ra chuỗi biểu diễn của đối tượng  
        System.out.println(obj.toString());  
        // Sử dụng phương thức equals() để so sánh đối tượng với một đối tượng khác  
        boolean isEqual = obj.equals(new Object());  
        System.out.println("isEqual: " + isEqual);  
        // Sử dụng phương thức hashCode() để lấy mã băm của đối tượng  
        int hashCode = obj.hashCode();  
        System.out.println("hashCode: " + hashCode);  
    }  
}
```

Constructor cannot be inherited

```
class Parent {  
    public Parent() {System.out.println("This is the parent constructor.");}  
}  
class Child extends Parent {  
    public Child() {System.out.println("This is the child constructor.");}  
}  
public class Example {  
    public static void main(String[] args) {  
        Child a_child = new Child();  
    }  
}
```



Subclass inherits superclass constructor

Constructors Are Not Inherited

super(...) for Constructor Reuse

super(arguments); //invoke a superclass constructor

Subclass constructor must invoke super class constructor

The call must be the first statement in the subclass constructor

If a **constructor** does not explicitly invoke a superclass constructor, the Java compiler automatically inserts a call to the no-argument constructor of the superclass.

Implicit Superclass Constructor Invocation

```
class Parent {  
    public Parent(int value) {System.out.println("Parent constructor called with  
value: " + value);}  
}  
class Child extends Parent {  
    public Child() {  
        super(10); // Gọi constructor của lớp cha với giá trị 10  
        System.out.println("Child constructor called");  
    }  
}  
public class Example {  
    public static void main(String[] args) {Child child = new Child();}  
}
```



Superclass Missing No-Arg Constructor

```
class Parent {  
    // No constructor defined  
}  
class Child extends Parent {  
    // No constructor defined  
    public Child(){System.out.println("Hàm tạo của lớp con đây!");}  
}  
  
public class Example {  
    public static void main(String[] args) {  
        Child obj = new Child(); // ok  
    }  
}
```



Inheritance

```
public class Rectangle {  
    private int length = 0;  
    private int width = 0;  
    // Overloading constructors  
    public Rectangle() // Default constructor  
    {  
    }  
    public Rectangle(int l, int w)  
    {  
        length = l>0? l: 0;    width= w>0? w: 0;  
    }  
    // Overriding the toString method of the java.lang.Object class  
    public String toString()  
    {  
        return "[" + getLength() + "," + getWidth() + "]]";  
    }  
    // Getters, Setters  
    public int getLength() { return length; }  
    public void setLength(int length) { this.length = length; }  
    public int getWidth() { return width; }  
    public void setWidth(int width) { this.width = width; }  
    public int area() { return length*width; }  
}
```





```
public class Box extends Rectangle {  
    private int height=0; // additional data  
    public Box() { super(); }  
    public Box (int l, int w, int h)  
    { super(l, w); // Try swapping these statements  
      height = h>0? h: 0;  
    }  
    // Additional Getter, Setter  
    public int getHeight() { return height; }  
    public void setHeight(int height)  
    { this.height = height; }  
    // Overriding methods  
    public String toString()  
    { return "[" + getLength() + "," +  
      getWidth() + "," + getHeight() + "];"  
    }  
    public int area(){  
        int l = this.getLength();  
        int w = this.getWidth();  
        int h = this.getHeight();  
        return 2*(l*w + w*h + h*l);  
    }  
    // additional method  
    public int volumn(){  
        return this.getLength()*this.getWidth()*height;  
    }  
}
```

```
1 public class Demo_1 {  
2     public static void main (String[] args)  
3     { Rectangle r= new Rectangle(2,5);  
4         System.out.println("Rectangle: " + r.toString());  
5         System.out.println("    Area: " + r.area());  
6         Box b= new Box(2,2,2);  
7         System.out.println("Box " + b.toString());  
8         System.out.println("    Area: " + b.area());  
9         System.out.println("    Volumn: " + b.volumn());  
10    }  
11 }
```

Output - Chapter06 (run)

```
run:  
Rectangle: [2,5]  
    Area: 10  
Box [2,2,2]  
    Area: 24  
    Volumn: 8  
BUILD SUCCESSFUL (total time: 0 seconds)
```



Functions in inheritance

A derived class inherits from superclass is limited to the normal member functions of the superclass.

We use the Java keyword `super` as the qualifier for calling a superclass 's method: `super.methodName(arguments);`

To invoke the version of method `methodName` that was defined by our superclass.

Hiding a method: Re-implementing a static method implemented in super class.

Sử dụng từ khóa **super** để gọi phương thức có cùng tên với phương thức hiện tại được định nghĩa trong superclass. Điều này cho phép chúng ta truy cập và sử dụng phiên bản của phương thức trong superclass trong trường hợp muốn thực hiện một số xử lý bổ sung hoặc ghi đè phương thức trong subclass.



Inherited Class Access Limitations

```
class Superclass {  
    public void publicMethod() {System.out.println("This is a public method of the  
superclass");}  
    private void privateMethod() {System.out.println("This is a private method of the  
superclass");}  
}  
  
public class Main {  
    public static void main(String[] args) {  
        Superclass obj = new Superclass();  
        obj.publicMethod(); // OK, accessible from superclass instance  
        // obj.privateMethod(); // Error: privateMethod() has private access in Superclass  
    }  
}
```



Using "super" to call superclass methods

```
class Superclass {  
    public void display() {System.out.println("This is a superclass method");}  
}  
class Subclass extends Superclass {  
    public void display() {  
        super.display(); // calling superclass method  
        System.out.println("This is a subclass method");  
    }  
}  
public class Main {  
    public static void main(String[] args) {  
        Subclass obj = new Subclass();  
        obj.display();  
    }  
}
```



Invoke superclass method in Java

```
class Superclass {  
    public void display() {System.out.println("This is a superclass method");}  
}  
class Subclass extends Superclass {  
    public void display() {  
        super.display(); // calling superclass method  
        System.out.println("This is a subclass method");  
    }  
}  
public class Main {  
    public static void main(String[] args) {  
        Subclass obj = new Subclass();  
        obj.display();  
    }  
}
```



Hiding static methods in inheritance.

```
class Superclass {  
    public static void display() { System.out.println("This is a superclass method"); }  
}  
class Subclass extends Superclass {  
    public static void display() { System.out.println("This is a subclass method"); }  
}  
  
public class Main {  
    public static void main(String[] args) {  
        Superclass.display(); // invoking superclass method  
        Subclass.display(); // invoking subclass method  
    }  
}
```

Functions in inheritance: Hiding Method

```

class Father1 {
    public static void m(){
        System.out.println("I am a father");
    }
}

class Son1 extends Father1{
    public static void m(){
        System.out.println("I am a son");
    }
}

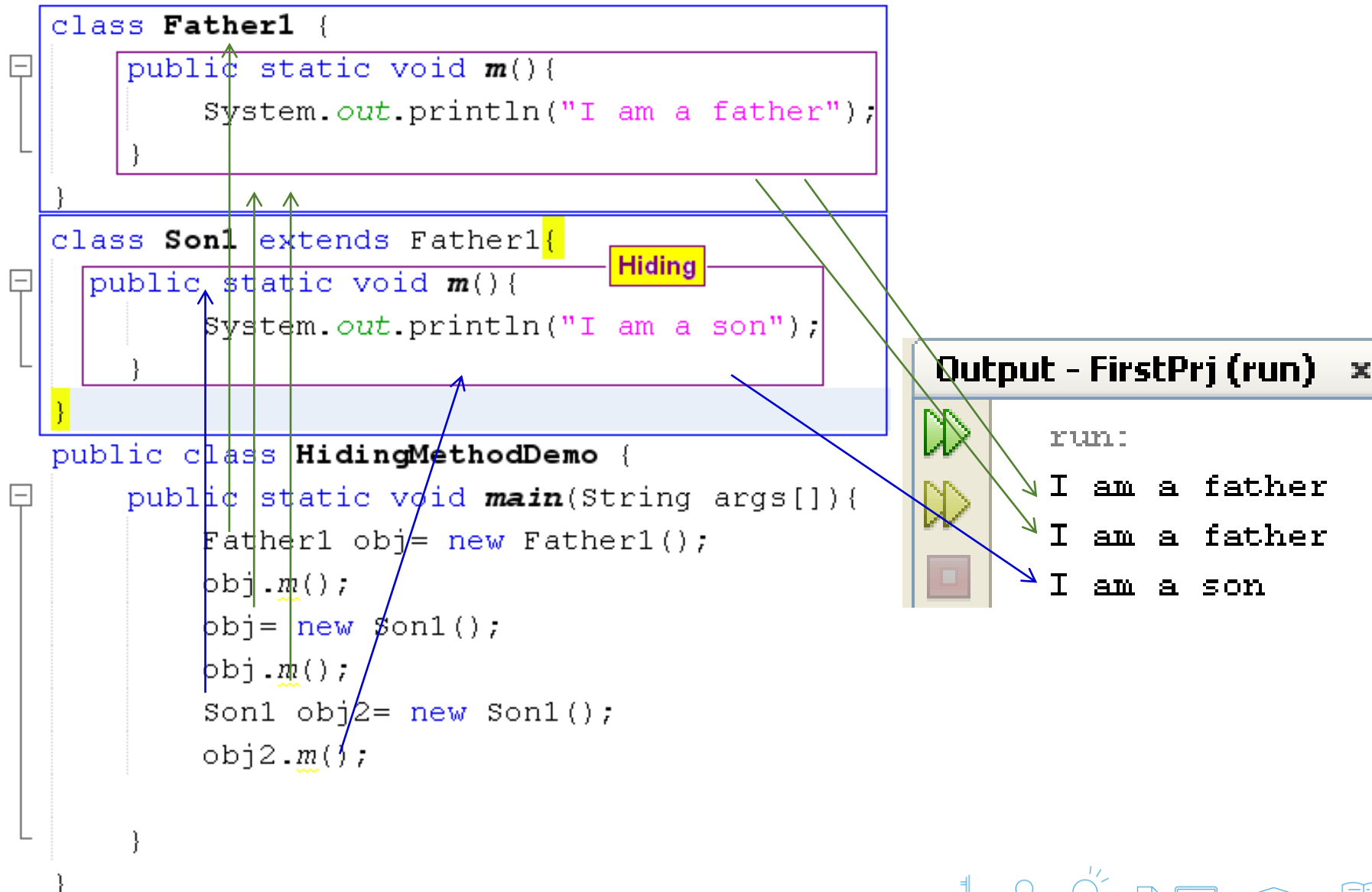
public class HidingMethodDemo {
    public static void main(String args[]){
        Father1 obj= new Father1();
        obj.m();
        obj= new Son1();
        obj.m();
        Son1 obj2= new Son1();
        obj2.m();
    }
}
    
```

Hiding

Output - FirstPrj (run)

```

run:
I am a father
I am a father
I am a son
    
```





Dynamic and Static type

dynamic type: A reference variable that has the type of the superclass can store the address of the object of sub class. It is called to be dynamic type, the type that is has at runtime.

```
Rectangle obj1 = new Box();
```

Static type: The type that it has when first declared. Static type checking is enforced by the compiler.

```
Box obj2 = new Box();
```

“**InstanceOf**” operator: It checks whether the reference of an object belongs to the provided type or not, the instanceof operator will return true or false.

```
If ( obj1 instanceof Box)
```

```
System.out.println(“ obj1 is pointing to the Box object”);
```



Dynamic and Static Type

```
class Superclass {  
    public void dynamicMethod() {System.out.println("This is a dynamic method in Superclass");}  
    public static void staticMethod() {System.out.println("This is a static method in Superclass");}  
}  
  
class Subclass extends Superclass {  
    public void dynamicMethod() {System.out.println("This is a dynamic method in Subclass");}  
}  
  
public class Main {  
    public static void main(String[] args) {  
        Superclass obj = new Subclass();  
        obj.dynamicMethod(); // Gọi phương thức dynamicMethod() của Subclass  
        obj.staticMethod(); // Gọi phương thức staticMethod() của Superclass  
    }  
}
```



Casting

A variable that has the type of the superclass only calls methods of the superclass. To call methods of the subclass we must cast explicitly

Example:

```
class Parent {  
    public void display() {System.out.println("This is a parent object");}  
}  
class Child extends Parent {  
    public void display() {System.out.println("This is a child object");}  
}  
public class Main {  
    public static void main(String[] args) {  
        Parent parent = new Child(); // Ép kiểu Child thành Parent  
        parent.display(); // Gọi phương thức display của Child  
    }  
}
```




Summary

Object-oriented languages implement **reusability** of coding structure through inheritance

A derived class does not by default inherit the **constructor** of a **super** class

Constructors in an inheritance hierarchy execute in order from the **super** class to the derived class

Using the **instanceof** keyword if we need to check the type of the reference variable.

Check the type of the reference variable before **casting** it explicitly.



CMC UNIVERSITY



THANK YOU