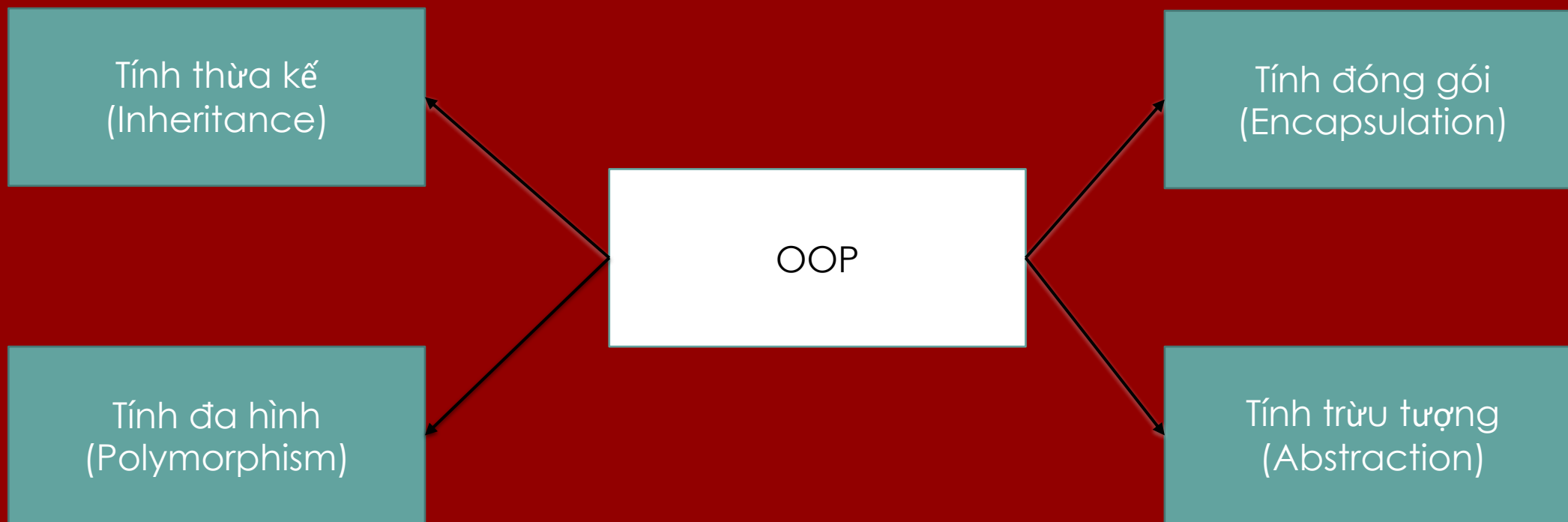


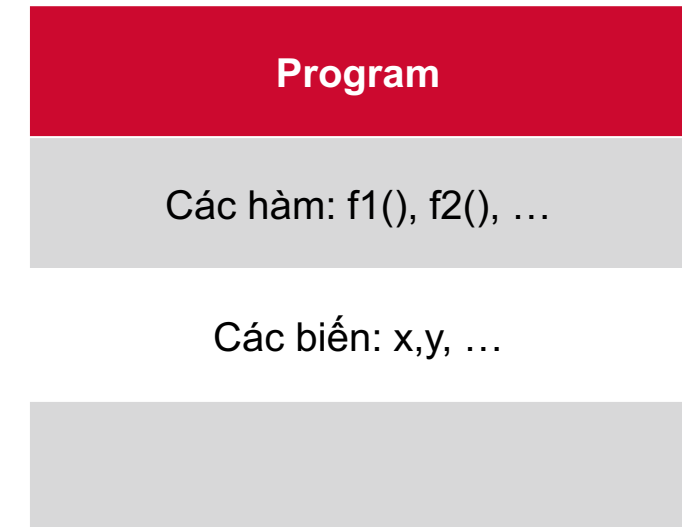
BÀI GIẢNG JAVA

OOP



Trước OOP

- Lập trình thủ tục (Procedural Programming)
- Một chương trình bao gồm:
 - Các hàm
 - Các biến
- Tính chất:
 - Đơn giản, dễ hiểu
 - Khó quản lý code với chương trình lớn



OOP

- Nhóm các hàm, biến có liên quan thành đối tượng (object)
- Một đối tượng trình bao gồm:
 - Các phương thức (methods)
 - Các thuộc tính (properties)
- Tính đóng gói
- Vd: Lớp nhân viên
 - Thuộc tính: tên, mã, lương tháng, thuế, ...
 - Phương thức: tính lương, thêm ngày nghỉ, ...



OOP

- Phương thức tính lương

- Lập trình thủ tục:

```
int getWage(int salary, int tax){  
    return salary - tax;  
}
```

- OOP

```
int getWage(){  
    return salary - tax;  
}
```

- Đặc điểm:

- Không có tham số
 - Không thể tính lương của một nhân viên với dữ liệu của nhân viên khác.

Tính trừu tượng

- Chỉ đưa ra các phương thức, thuộc tính cần thiết
- Dễ sử dụng
- Giảm tác động của việc thay đổi code

Lớp X

Các phương thức: f1(), f2(), ...

Các thuộc tính: x,y, ...

Các phương thức, biến ẩn: f3(), z

Tính kế thừa

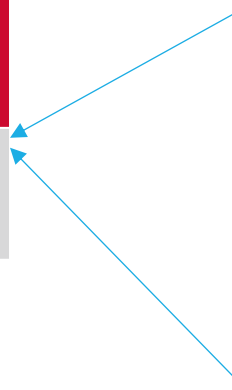
Lớp xe máy
Tham số: hãng, số máy, số khung
Phương thức: khởi động, di chuyển, dừng

Lớp ô tô
Tham số: hãng, số máy, số khung
Phương thức: khởi động, di chuyển, dừng

Lớp xe
Tham số: hãng, số máy, số khung
Phương thức: khởi động, di chuyển, dừng

Lớp xe máy
Các tham số, phương thức riêng

Lớp ô tô
Các tham số, phương thức riêng



Tính đa hình

- switch type{
 case "car":
 startCar();
 case "motorbike":
 startMotorBike();
}
- vehicle.start();
- Việc phương thức start nào được gọi phụ thuộc vào kiểu của đối tượng vehicle.

LỚP VÀ ĐỐI TƯỢNG

Khái niệm

- Lớp: Là mô hình mô tả cho một nhóm đối tượng
- Một lớp trong java có thể chứa:
 - Thuộc tính
 - Hàm tạo
 - Phương thức
- Đối tượng
 - Là một thể hiện cụ thể của lớp



Khai báo lớp

[public | private | protected]
hoặc không có

```
[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...){...}  
}
```

Có thể có nhiều hàm
tạo

Khai báo lớp

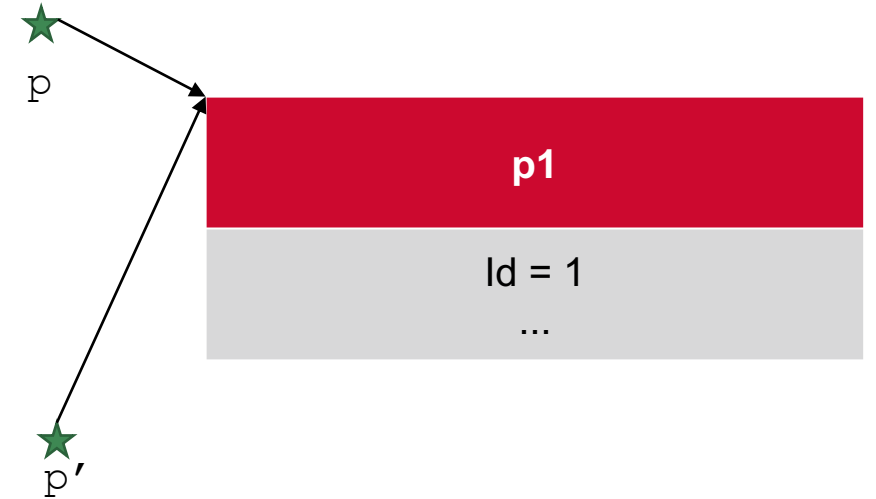
- Ví dụ:

```
public class Employee {  
    //properties  
    public int id;  
    public String name;  
    public int baseSalary;  
  
    //constructor  
    public Employee() {  
  
    }  
  
    //methods  
    public void print() {  
        System.out.println("Id: " + id + ";Name: " + name + "; Base Salary: "  
+ baseSalary);  
    }  
}
```

Tham số của phương thức

- Tất cả các tham số trong Java đều truyền theo tham trị (sao chép giá trị)
- Với biến không thuộc kiểu cơ bản trong Java, tham chiếu sẽ được sao chép

```
public class Entry {  
    public static void main(String[] args) {  
        Person p1 = new Person();  
        p1.id = 1;  
        int i = 2;  
        change(p1);  
        System.out.println(p1.id); //10  
        changei(i);  
        System.out.println(i); //2  
    }  
  
    public static void change(Person p) {  
        p.id = 10;  
    }  
  
    public static void changei(int i) {  
        i = 10;  
    }  
}
```



i = 2

i' = 10

Bài tập

- Khai báo lớp **Student** với họ tên, điểm toán, điểm lý điểm hóa (double)
- Viết chương trình cho phép người dùng nhập vào **số sinh viên**, sau đó nhập họ tên và điểm của mỗi sinh viên
- In ra danh sách những sinh viên đã nhập vào và danh sách sinh viên có điểm trung bình > 5

Nạp chồng phương thức (overload)

- Là trường hợp các phương thức trùng tên nhưng khác tham số
- Phương thức nào được gọi phụ thuộc vào danh sách tham số

```
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);  
    }  
}
```

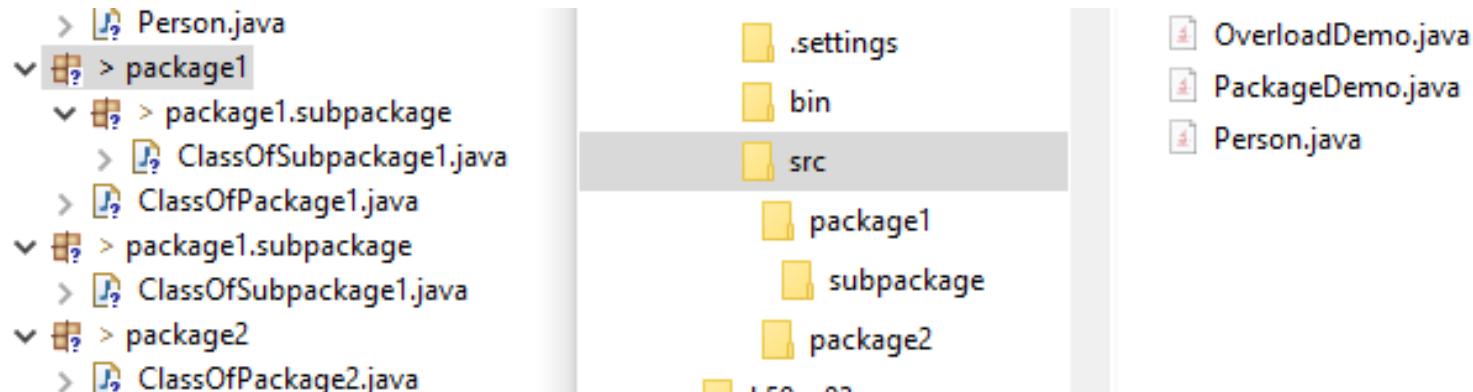
Method signature



GÓI (PACKAGE)

Khái niệm

- Là một nhóm các class, interface, các gói khác
- Tổ chức của 1 package là 1 thư mục có tên là tên của package
 - Sub-package là 1 gói con (thư mục con) của 1 package mức cao hơn (giống cấu trúc thư mục).
- Gói là công cụ tạo khả năng tái sử dụng mã (reusable code).



Khai báo

- Khi khai báo lớp trong gói dùng từ khóa package
- Tên gói trùng với tên folder

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

Sử dụng

- Dùng từ khóa import
- Import tất cả các lớp trong gói: .*

```
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();  
    }  
}
```

ĐẶC TÍNH TRUY SUẤT (access modifier)

Khai báo

```
public class Person {  
    //properties  
    public int id;  
    private String name;  
    protected String address;  
    String zipCode;  
  
    //constructor  
    public Person() {  
  
    }  
  
    //methods  
    public void print() {  
        ...  
    }  
}
```

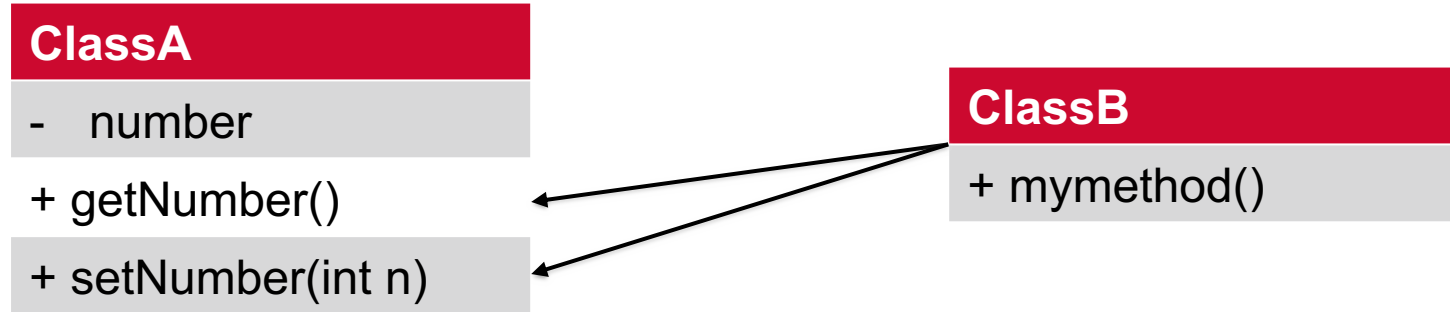
[public | private | protected]
hoặc không có (=friendly)

Đứng trước khai báo lớp, thuộc
tính, phương thức, hàm tạo

Đặc tính truy xuất

Modifier	private	[friendly]	protected	public
Cùng class	YES	YES	YES	YES
Cùng gói, khác class	NO	YES	YES	YES
Lớp con trong cùng gói với lớp cha	NO	YES	YES	YES
Khác gói, khác lớp	NO	NO	NO	YES
Lớp con khác gói với lớp cha	NO	NO	YES	YES

Getter/setter



Đặc tính gói ghém dữ liệu giúp bao bọc lấy các thuộc tính của một lớp. Nó làm cho các thuộc tính của lớp bị ẩn đi so với các lớp khác.

TỪ KHÓA static

Đặc điểm

- 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);  
    }  
}
```

Đặc điểm

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

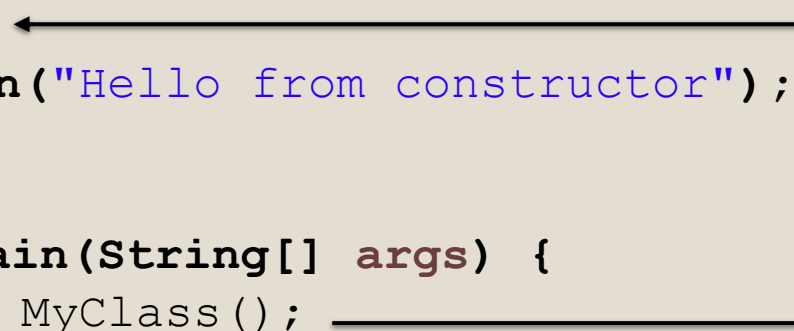
```
public class StaticDemo {  
    public static String name;  
  
    public static void printName() {  
        System.out.println(name);  
    }  
  
    public static void main(String[] args) {  
        StaticDemo.name = "Nguyen Van An";  
        StaticDemo.printName();  
    }  
}
```

XÂY DỰNG VÀ KHỞI TẠO ĐỐI TƯỢNG

Hàm tạo

- Là một hàm đặc biệt được sử dụng để khởi tạo đối tượng
- Hàm tạo có tên trùng tên lớp và không có kiểu dữ liệu trả về

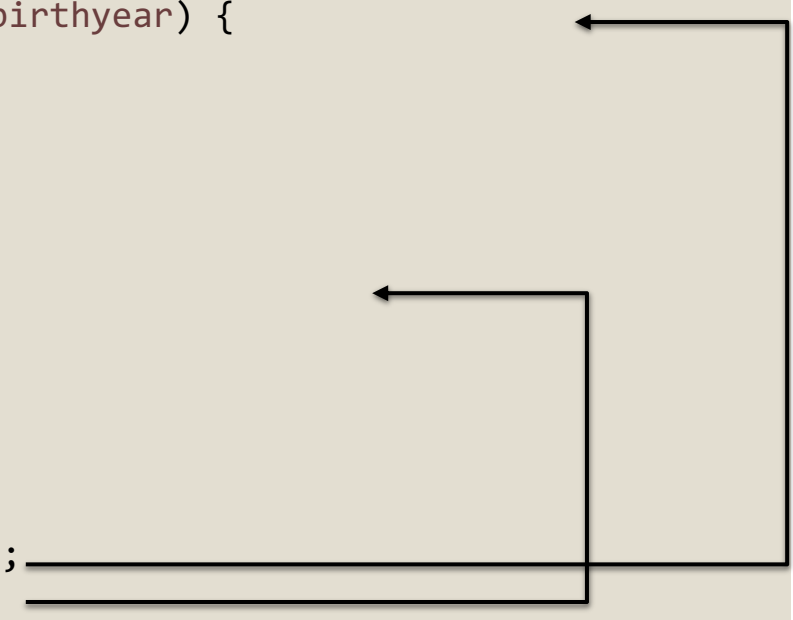
```
public class MyClass {  
  
    public MyClass() {  
        System.out.println("Hello from constructor");  
    }  
  
    public static void main(String[] args) {  
        MyClass obj = new MyClass();  
    }  
}
```



Hàm tạo

- 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");  
    }  
}
```

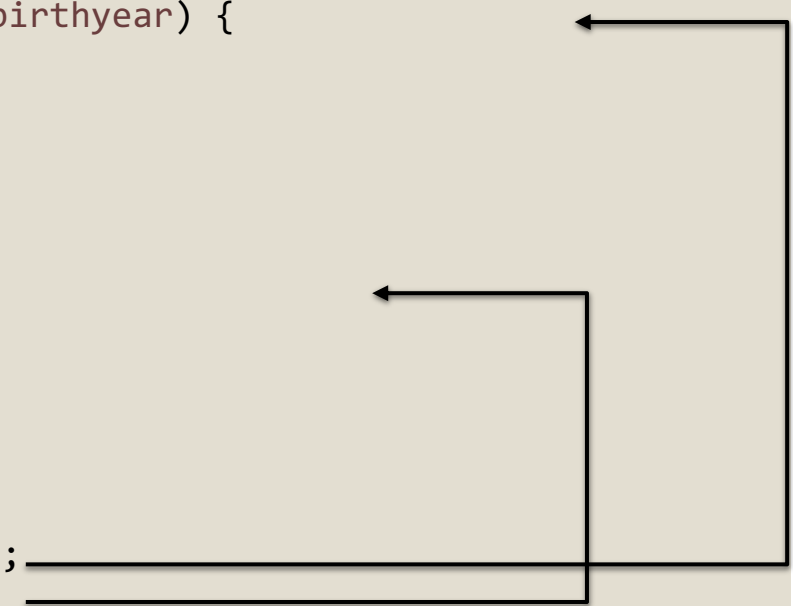


The diagram illustrates the flow of parameters from the `main` method to the constructors. Two horizontal lines originate from the right side of the `main` method, representing the arguments passed to the constructors. One line goes up and left to the `Person(String name, String address, int birthyear)` constructor. The other line goes down and left to the `Person(String name, String address)` constructor.

Hàm tạo

- 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");  
        Person p3 = new Person();  
    }  
}
```

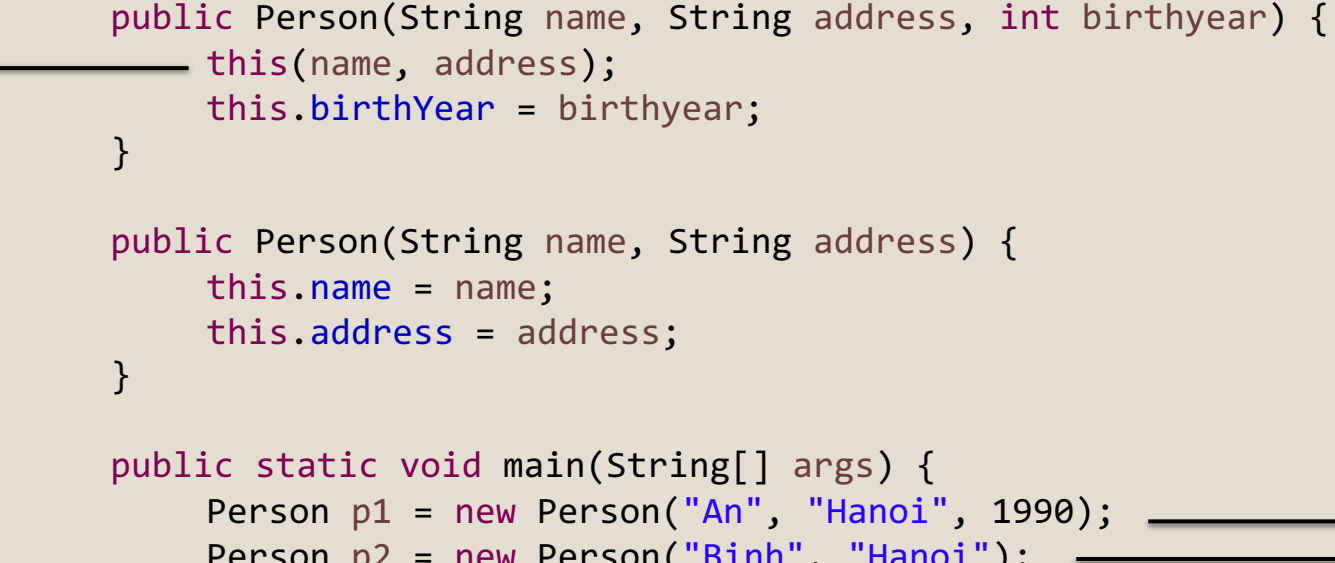


The diagram illustrates the flow of parameters from the `main` method to the constructors. It features three horizontal lines on the right side of the code block. The top line connects to the `Person(String name, String address, int birthyear)` constructor. The middle line connects to the `Person(String name, String address)` constructor. The bottom line connects to the `Person()` constructor. Each connection is made via a horizontal line that then turns vertically to the left, ending in an arrow pointing to the corresponding constructor's opening curly brace.

Hàm tạo

- Để gọi đến hàm tạo khác trong cùng lớp có thể dùng từ **this(<ds tham số>)** (xem ví dụ bên dưới)

```
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");  
    }  
}
```



The diagram illustrates the execution flow of the code. It shows three call paths: 1) From the `main` method to the `Person(String name, String address, int birthyear)` constructor. 2) From the `Person(String name, String address, int birthyear)` constructor to the `Person(String name, String address)` constructor. 3) From the `Person(String name, String address)` constructor to the `main` method.

Hàm tạo

- Nếu trong 1 lớp không khai báo hàm tạo, hàm tạo mặc định (không tham số) sẽ tự động được thêm vào
- Nếu trong 1 lớp có khai báo hàm tạo, Java sẽ không tự thêm hàm tạo mặc định vào

```
public class Person {  
    public String name;  
    public String address;  
    public int birthYear;  
  
    //hàm tạo mặc định sẽ tự động được Java thêm vào  
  
    public static void main(String[] args) {  
        Person p = new Person();  
    }  
}
```


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 mp 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.

THỪA KẾ

Thừa kế

- Thừa kế: Kỹ thuật cho phép tái sử dụng thông tin (properties + methods)
 - Lớp con = Lớp cha + mở rộng
 - Lớp con không thể truy xuất thành phần private của lớp cha

```
public class Son extends Father {  
    ...  
}
```

Thừa kế

- Ví dụ

```
public class Father {  
    public String p1;  
    public String p2;  
  
    public void m1() {  
        system.out.print("Father.m1");  
    }  
}  
  
public class Son extends Father {  
    public String p3;  
  
    public void m2() {  
        system.out.print("Son.m2");  
    }  
  
    public static void main(String[] args) {  
        Son s1 = new Son();  
        s1.m1();  
        s1.m2();  
    }  
}
```

Tính đa hình (runtime)

- Tính đa hình (Polymorphism) trong Java được hiểu là trong từng trường hợp, hoàn cảnh khác nhau thì đối tượng có hình thái khác nhau tùy thuộc vào từng ngữ cảnh.
- Đa hình chỉ có trong 1 phân cấp thừa kế và các class của phân cấp có cùng method.
- Kỹ thuật đa hình cho phép 1 lớp con override 1 method ở lớp cha (cùng 1 method nhưng code trong lớp cha và code trong lớp con khác nhau)

Tính đa hình

```
public class Animal {
    public void sound() {
    }
}

public class Dog extends Animal{
    public void sound() {
        system.out.print("Dog");
    }
    public void eat() {}
}

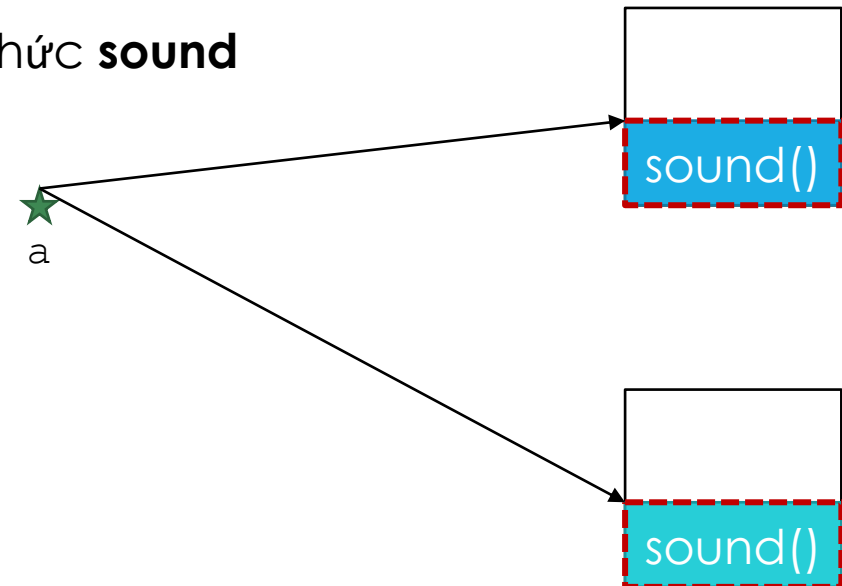
public class Cat extends Animal{
    public void sound() {
        system.out.print("Cat");
    }
    public void climb() {}
}

public class Demo {

    public static void main(String[] args) {
        Animal a;
        a = new Dog(); //upcasting
        a.sound();      //sound from dog
        a = new Cat(); //upcasting
        a.sound();      //sound from cat
    }
}
```

- **a** là tham chiếu của lớp cha **Animal**.
- **a** có thể tham chiếu tới các đối tượng của lớp con (upcasting)
- Thông qua **a**, chỉ có thể truy suất tới

phương thức **sound**



Bài tập

- Hãy cải tiến chương trình về hình cho phép người dùng nhập các hình từ bàn phím.
- **B1**: Chương trình hỏi người dùng muốn nhập bao nhiêu hình?
- **B2**: Chương trình hỏi người dùng muốn nhập vào hình gì? (1. Tam Giác, 2. Chu Nhat, 3. Hình Tron)
- **B3**: Tùy vào hình đã chọn chương trình hỏi người dùng:
Tam Giác: Nhập vào tọa độ 3 điểm
Chu Nhat: Nhập vào độ dài 2 cạnh
Hình Tron: Nhập vào tâm O và bán kính
- Sau khi nhập xong, chương trình in ra danh sách bao gồm: Loại hình, chu vi, diện tích

Bài tập

- Hãy khai báo lớp **Hinh** với 2 phương thức tính chu vi và diện tích
- Thay đổi các lớp **TamGiac**, **HinhTron**, **ChuNhat** để kế thừa lớp **Hinh** và ghi đè (override) phương thức tính chu vi và diện tích
- Thay đổi chương trình chính, lưu tất cả các hình vào 1 mảng **Hinh**.

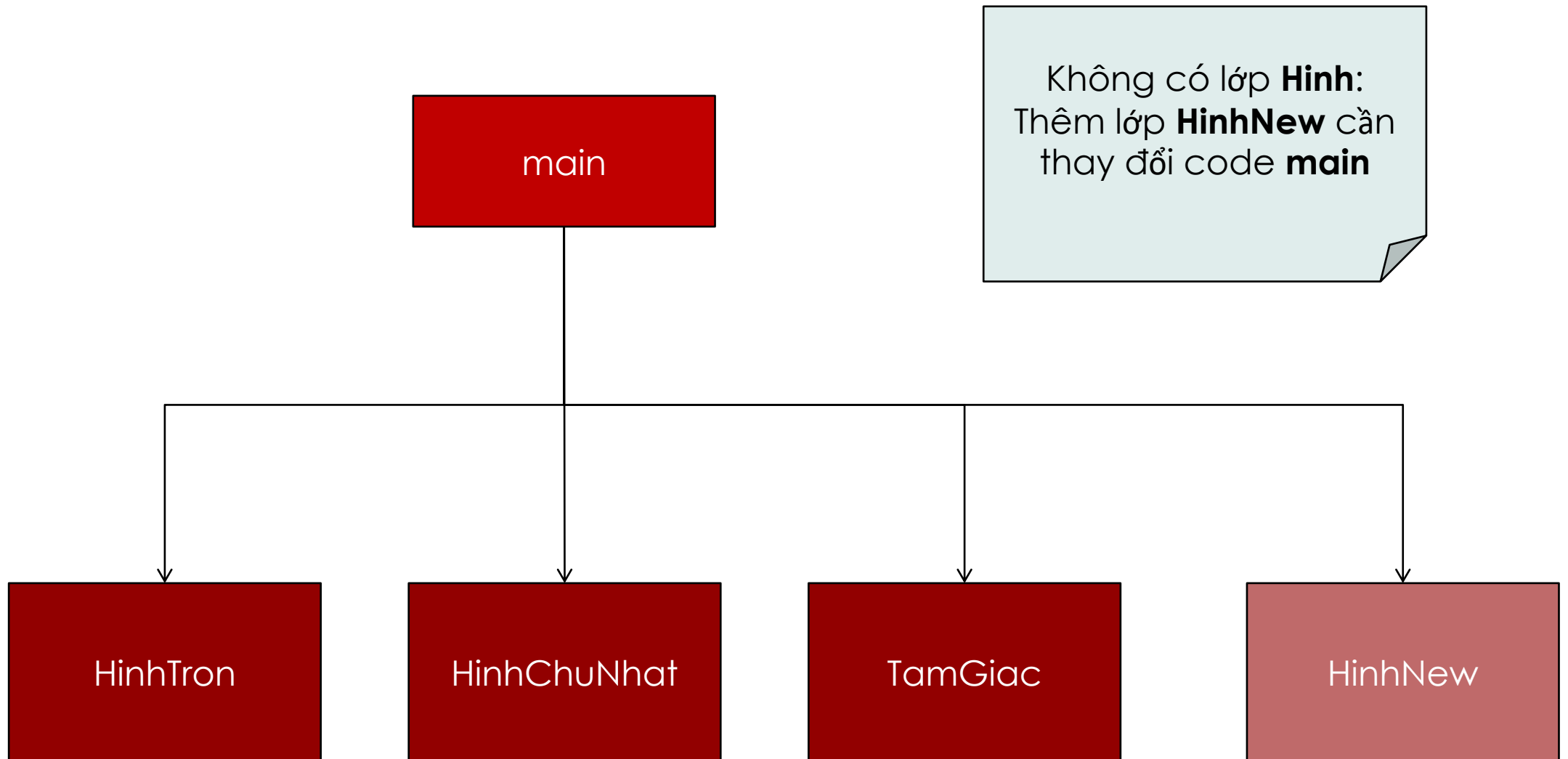
Như vậy mặc dù lớp **Hinh** không có code nhưng bằng việc các lớp thừa kế từ lớp Hinh giúp **thống nhất xử lý** và làm chương trình trở nên gọn gàng hơn

LỚP TRỪU TƯỢNG VÀ GIAO DIỆN

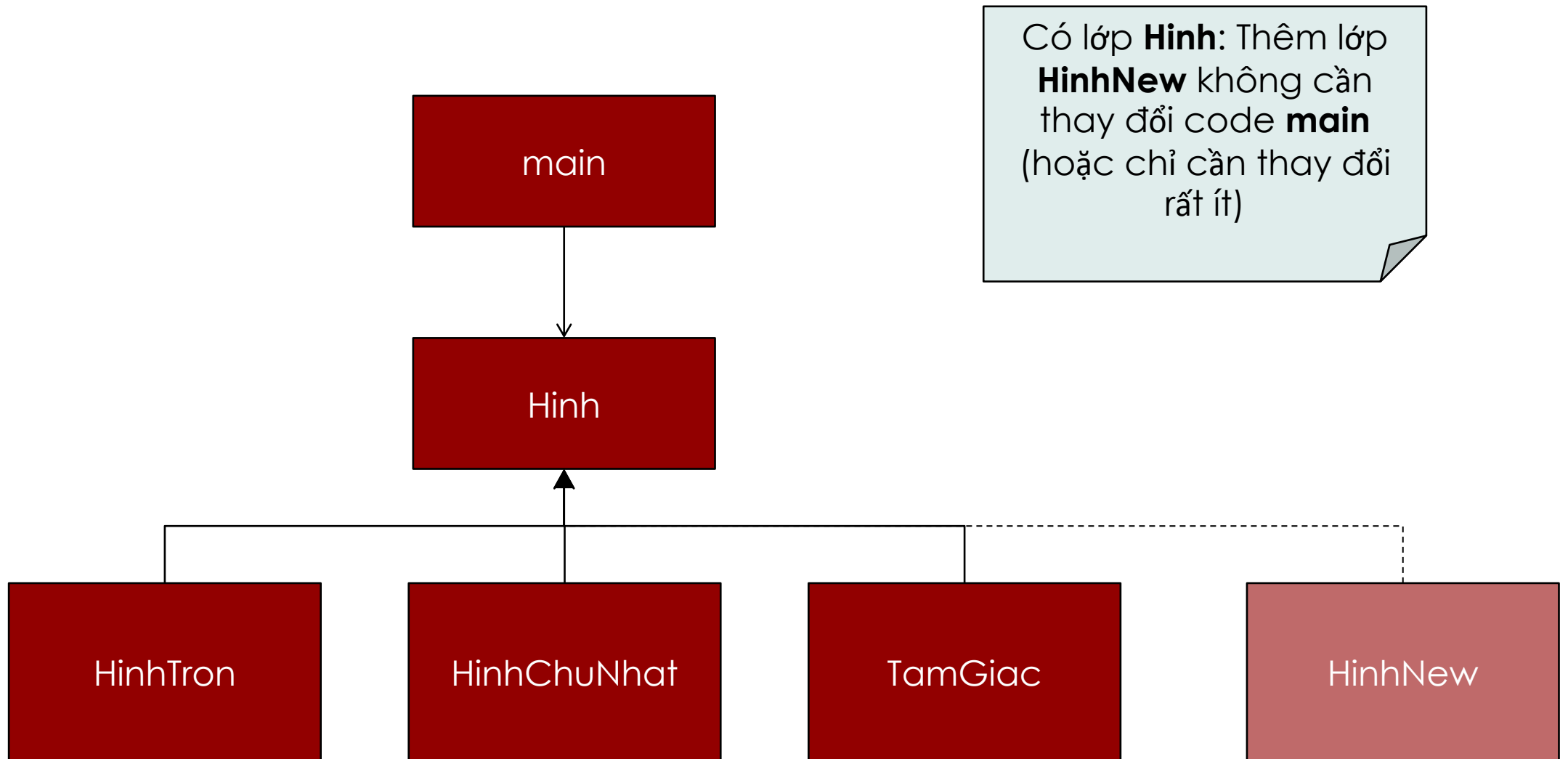
Tính trừu tượng

- Lớp **Hình** trong bài tập trước mặc dù không có code tính ChuVi và DienTich nhưng giúp thống nhất xử lý (duyệt danh sách, in) trên các loại hình khác nhau.
- Các xử lý như duyệt mảng, in ra danh sách các hình chỉ cần quan tâm tới:
 - Đã là **Hình** thì có phương thức tính chu vi
 - Đã là **Hình** thì có phương thức tính diện tích
 - Không cần quan tâm tới công thức tính chu vi và diện tích cụ thể
- Tính trừu tượng cung cấp khả năng mở rộng dễ dàng và giúp người lập trình thiết lập biên (boundary) giữa các thành phần trong hệ thống

Tính trừu tượng



Tính trừu tượng



Lớp trừu tượng

- Khai báo lớp trừu tượng thông qua từ khóa **abstract**
- Lớp trừu tượng là lớp có chứa phương thức trừu tượng
- Không thể khởi tạo đối tượng thuộc lớp trừu tượng

```
public abstract class Hình{  
    ...  
}
```

Phương thức trừu tượng

- Khai báo phương thức trừu tượng thông qua từ khóa **abstract**. Phương thức trừu tượng chỉ có phần khai báo
- Phương thức trừu tượng phải thuộc lớp trừu tượng
- Trong lớp trừu tượng có thể có cả phương thức trừu tượng và phương thức bình thường

```
public abstract class Hình {  
    public abstract double chuVi();  
  
    public abstract double dienTich();  
  
    public void print() {  
        System.out.println("Chu vi: " + chuVi() + "; Dien tich: " + dienTich());  
    }  
}
```

Interface

- Interface chỉ chứa các phương thức trừu tượng (không cần từ khóa `abstract`)
- Có thể hiểu interface là 1 lớp trừu tượng hoàn toàn

```
public interface Hình {  
    public double chuVi();  
  
    public double dienTich();  
}  
...  
public class HìnhTron implements Hình {  
    ...  
}
```


Interface

- Một lớp có thể thực hiện nhiều interface

```
interface Interface1 {  
    public void method1();  
}  
interface Interface2 {  
    public void method2();  
}  
  
class DemoClass implements Interface1, Interface2 {  
    public void method1() {  
        System.out.println("Some text..");  
    }  
    public void method2() {  
        System.out.println("Some other text...");  
    }  
}  
  
class MyMainClass {  
    public static void main(String[] args) {  
        DemoClass myObj = new DemoClass();  
        myObj.method1();  
        myObj.method2();  
    }  
}
```

HÀM TẠO TRONG THỪA KẾ

Hàm tạo

- Hàm tạo không được thừa kế
- Hàm tạo của lớp con phải gọi tới hàm tạo của lớp cha thông qua **super(<ds tham số>)**

- Lời gọi tới hàm tạo khác phải là câu lệnh đầu tiên trong hàm tạo

```
public class Student extends Person{  
    public String studentId;  
  
    public Student(String name, String address, String studentId) {  
        super(name, address);  
        this.studentId = studentId;  
    }  
}
```

- Trong trường hợp lớp cha có hàm tạo mặc định thì không cần gọi một cách tường minh

NÂNG CAO VỀ THỪA KẾ

Object class

- Tất cả các lớp trong java đều là một lớp con, trừ lớp object.
- Khi khai báo một lớp, nếu lớp đó không thừa kế một lớp nào khác thì được hiểu là đang thừa kế lớp object.
- Lớp object nằm trong thư viện java.lang và được khai báo tự động. Vì vậy hai khai báo sau có chung ý nghĩa:

```
public class Student{  
  
}  
  
public class Student extends Object{  
  
}
```

Object class

- Object class có một số phương thức mà các lớp con có thể sử dụng, nạp chồng hoặc ghi đè:

Phương thức	Ý nghĩa
Object clone()	Tạo và return một bản copy của đối tượng.
boolean equals(Object obj)	So sánh xem đối tượng hiện tại và đối tượng đưa vào trong tham số có bằng nhau không.
String toString()	Trả lại một biểu diễn dạng String của đối tượng.

Phương thức toString()

- Lớp object có phương thức toString() được định nghĩa sẵn

```
public class Student{  
  
    public int id;  
    public String name;  
  
    public static void main(String args[]){  
        Student s = new Student();  
        String studentString = s.toString();  
        System.out.println(studentString);  
    }  
}
```

Ghi đè phương thức toString()

- Ghi đè phương thức toString trong các lớp con giúp chúng ta in thông tin của đối tượng ra một cách dễ dàng.

```
public class Student{

    public int id;
    public String name;

    @Override
    public String toString(){
        return this.id + ": " + this.name;
    }

    public static void main(String args[]){
        Student s = new Student();
        String studentString = s.toString();
        System.out.println(studentString);
    }
}
```


Phương thức equals()

- Lớp object có phương thức equals() được định nghĩa sẵn như sau:
 - `public boolean equals(Object obj)`
- Chúng ta có thể sử dụng, nạp chồng hoặc ghi đè phương thức này.

Một vài chú ý

- CHỈ định nghĩa các phương thức nếu chúng là trừu tượng, KHÔNG viết phần thân hàm.
- ĐẶT DẤU ; sau khi định nghĩa phương thức trừu tượng.
- GHI ĐỀ tất cả các phương thức trừu tượng trong lớp con.
- KHÔNG nạp chồng thay vì ghi đề phương thức trừu tượng.
- KHÔNG tạo đối tượng đối với lớp trừu tượng.
- GHI ĐỀ tất cả các phương thức trừu tượng khi sử dụng Interface.

Bài tập

- Viết một lớp BankAccount với một thuộc tính balance (số thực), hàm tạo, getter, setter và ba phương thức: deposit, withdraw, transfer.
- Viết một lớp SavingAccount kế thừa BankAccount và có thêm thuộc tính interestRate (số thực), hàm tạo, hàm tính lãi (lãi được tính bằng $\text{balance} * \text{interestRate} / 100$).
- Viết một lớp CheckingAccount kế thừa BankAccount và có thêm thuộc tính TRANSACTION_FEE có giá trị bằng 2.0 (số thực, biến toàn cục) và FREE_TRANSACTION có giá trị bằng 3 (số nguyên, biến toàn cục). Ngoài ra có thuộc tính transactionCount. Ghi đè hàm deposit, withdraw và transfer, mỗi lần cộng thêm 1 vào transactionCount.

Bài tập

- Viết hàm deductFees để trừ phí cuối tháng đối với CheckingAccount. Phí được tính bằng $\text{TRANSACTION_FEE} * (\text{transactionCount} - \text{FREE_TRANSACTION})$.
- Viết một chương trình chính kiểm tra tính đúng đắn của các lớp vừa viết. Nếu cần có thể ghi đè hàm toString() để hiển thị thông tin tài khoản.

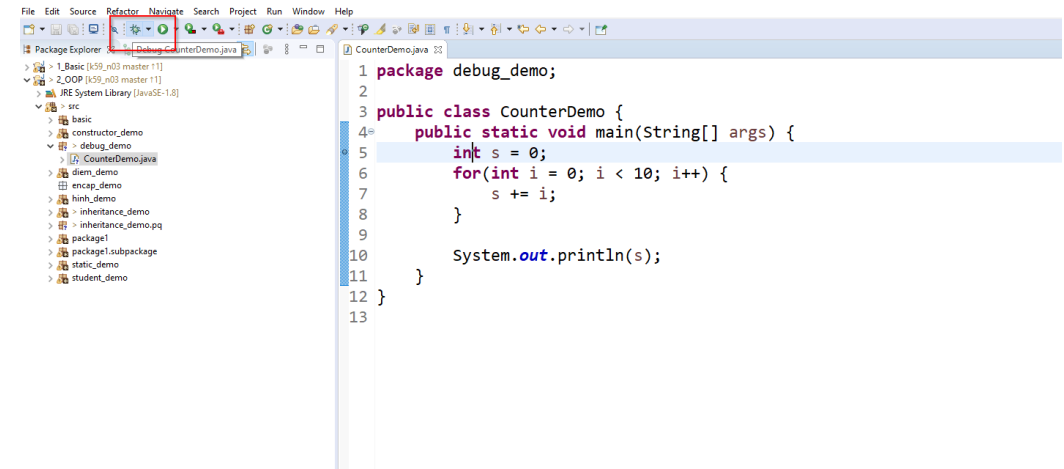
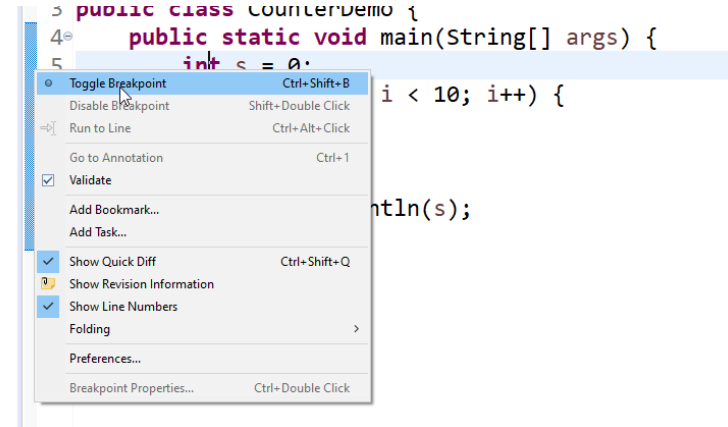
DEBUG VỚI ECLIPSE

Debug là gì

- Debug (gỡ lỗi) giúp kiểm tra và gỡ lỗi chương trình dễ dàng hơn. Bao gồm:
 - Chạy từng dòng lệnh theo ý muốn
 - Theo dõi giá trị các biến
 - Theo dõi danh sách lời gọi hàm
- Debug giúp việc gỡ lỗi thuận tiện và mạnh mẽ hơn so với việc in biến ra dòng lệnh (`system.out.print`)
- Nội dung tiếp theo sẽ hướng dẫn debug với Eclipse, với các IDE khác các bước cũng tương tự

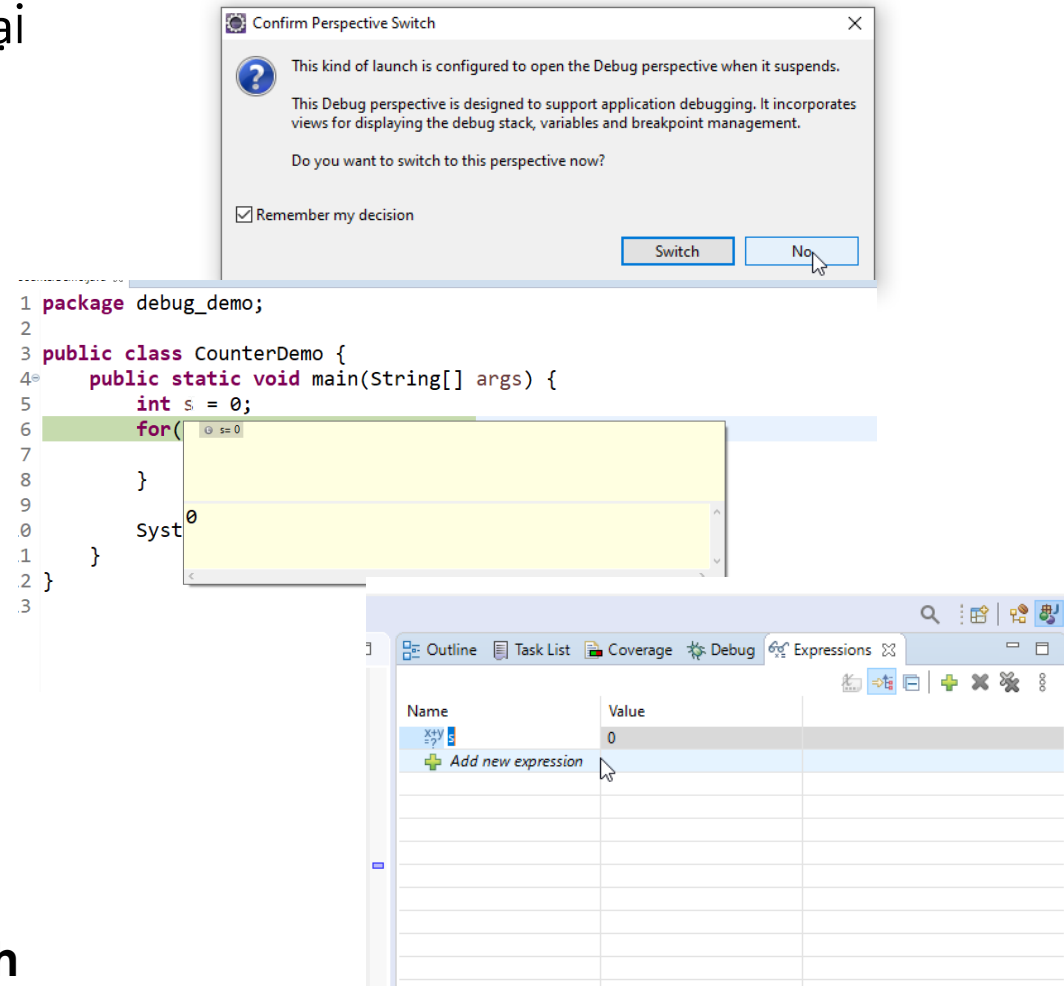
Đặt breakpoint

- Debug theo các bước cơ bản sau:
 - Đặt breakpoint (Ctrl + Shift + B)
- Bắt đầu chương trình ở chế độ debug
 - Chương trình sẽ dừng khi gặp breakpoint



Quan sát giá trị các biến

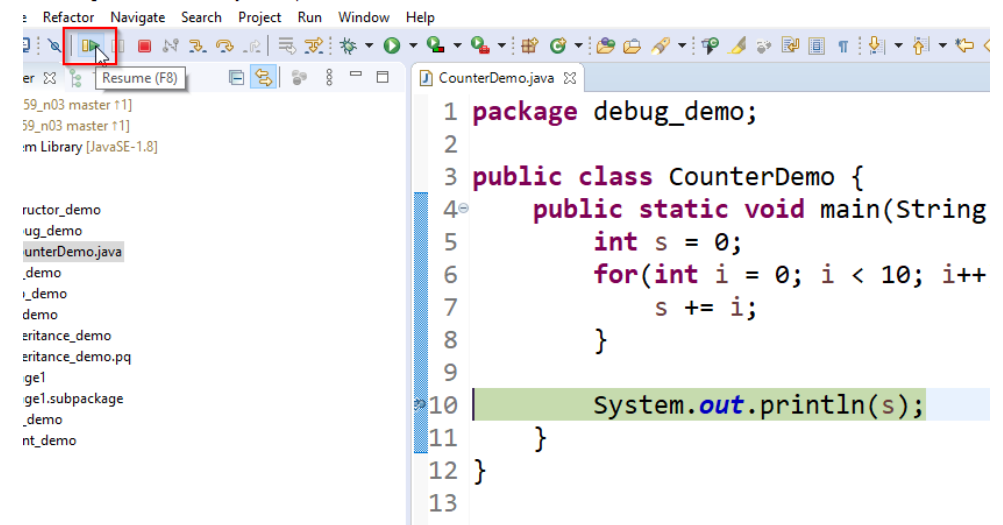
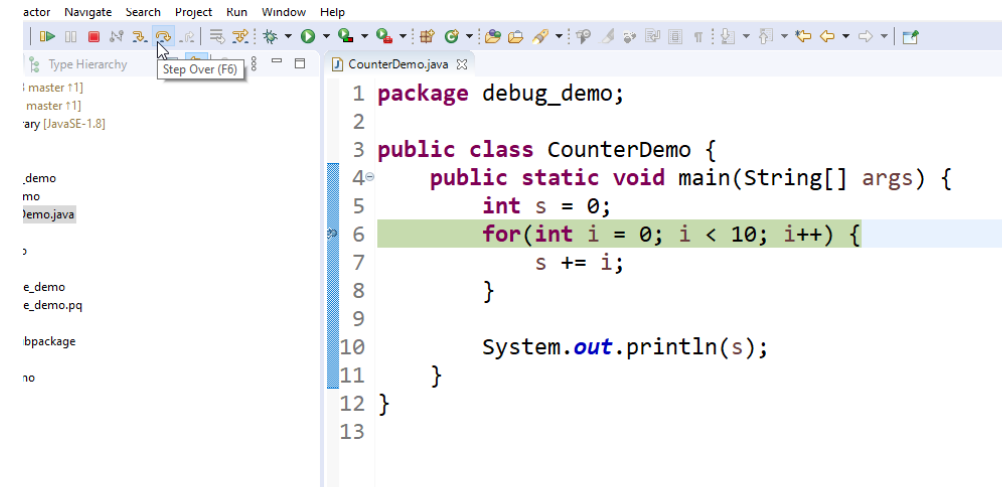
- Eclipse sẽ hỏi để chuyển chế độ khung nhìn sang **Debug**
 - Chọn **No** để giữ vị trí của sổ như hiện tại



- Quan sát giá trị các biến
 - Chương trình sẽ dừng lại ở **breakpoint**
 - Quan sát giá trị biến bằng cách:
 - Di chuột qua
 - Gõ tên biến vào cửa sổ **expression**
 - Bôi đen biểu thức, phải chuột chọn **Watch**

Chạy tiếp chương trình

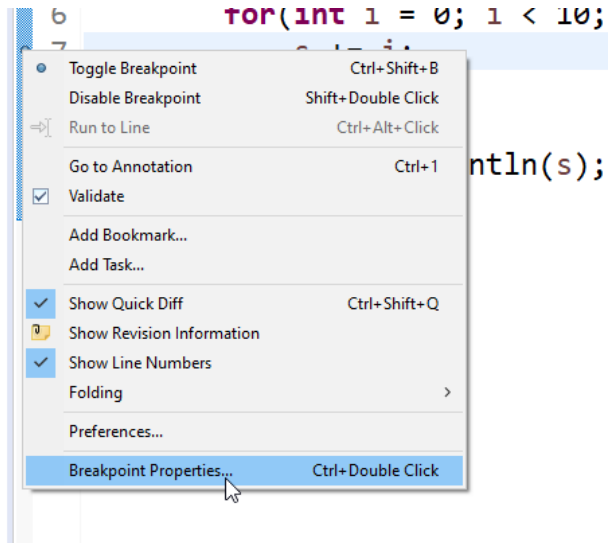
- Chạy tới dòng lệnh kế tiếp:
 - **F6** để bước qua lời gọi phương thức
 - **F5** để vào trong phương thức
- **F8** để tới breakpoint tiếp theo hoặc tới khi kết thúc



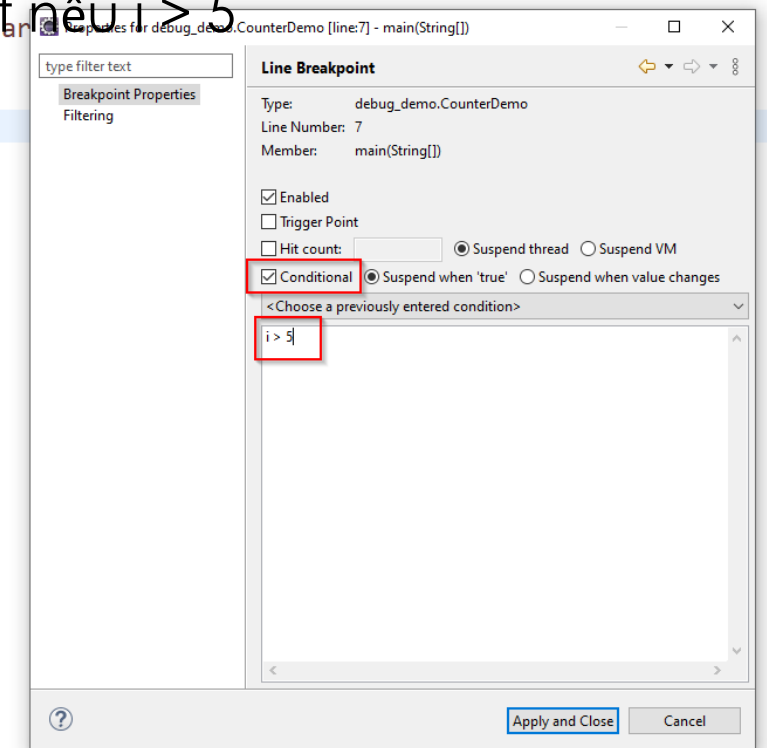
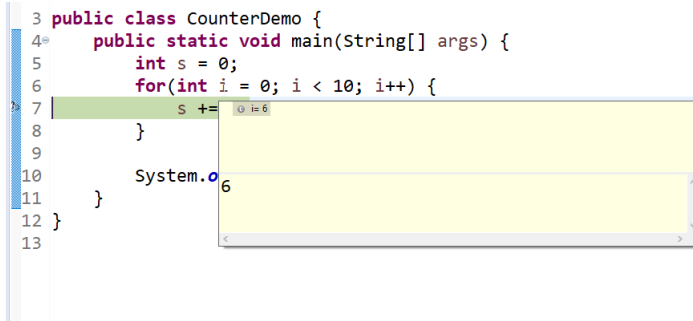
Breakpoint có điều kiện

- Chọn breakpoint properties
- Check Conditional và gõ vào điều kiện

• Ví dụ: Chương trình chỉ dừng lại ở breakpoint nếu $i > 5$



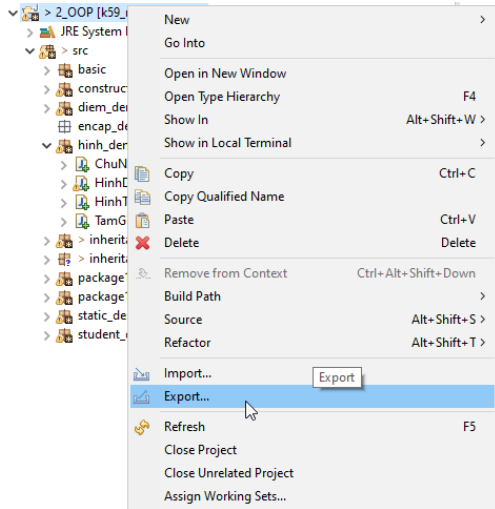
```
public class CounterDemo {  
    public static void main(String[] args) {  
        int s = 0;  
        for(int i = 0; i < 10; i++) {  
            s += i;  
        }  
        System.out.println(s);  
    }  
}
```



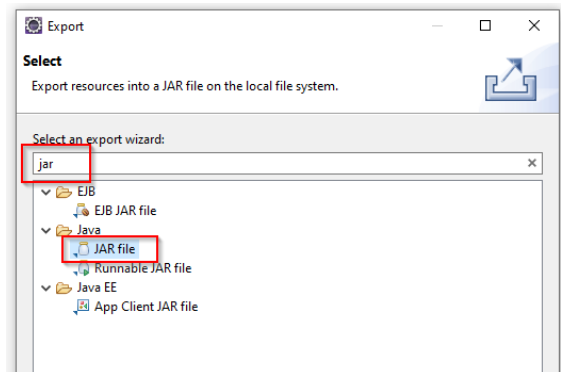
ĐÓNG GÓI FILE .JAR

ĐÓNG GÓI .JAR FILE

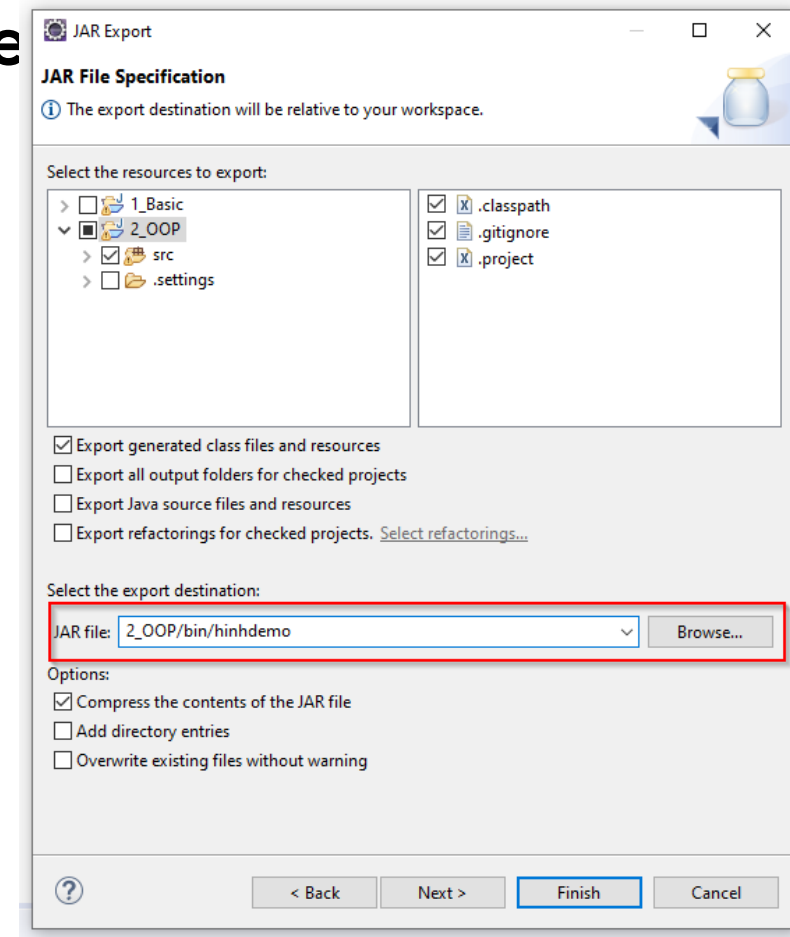
- Chọn project > Export



- Chọn **JAR** file

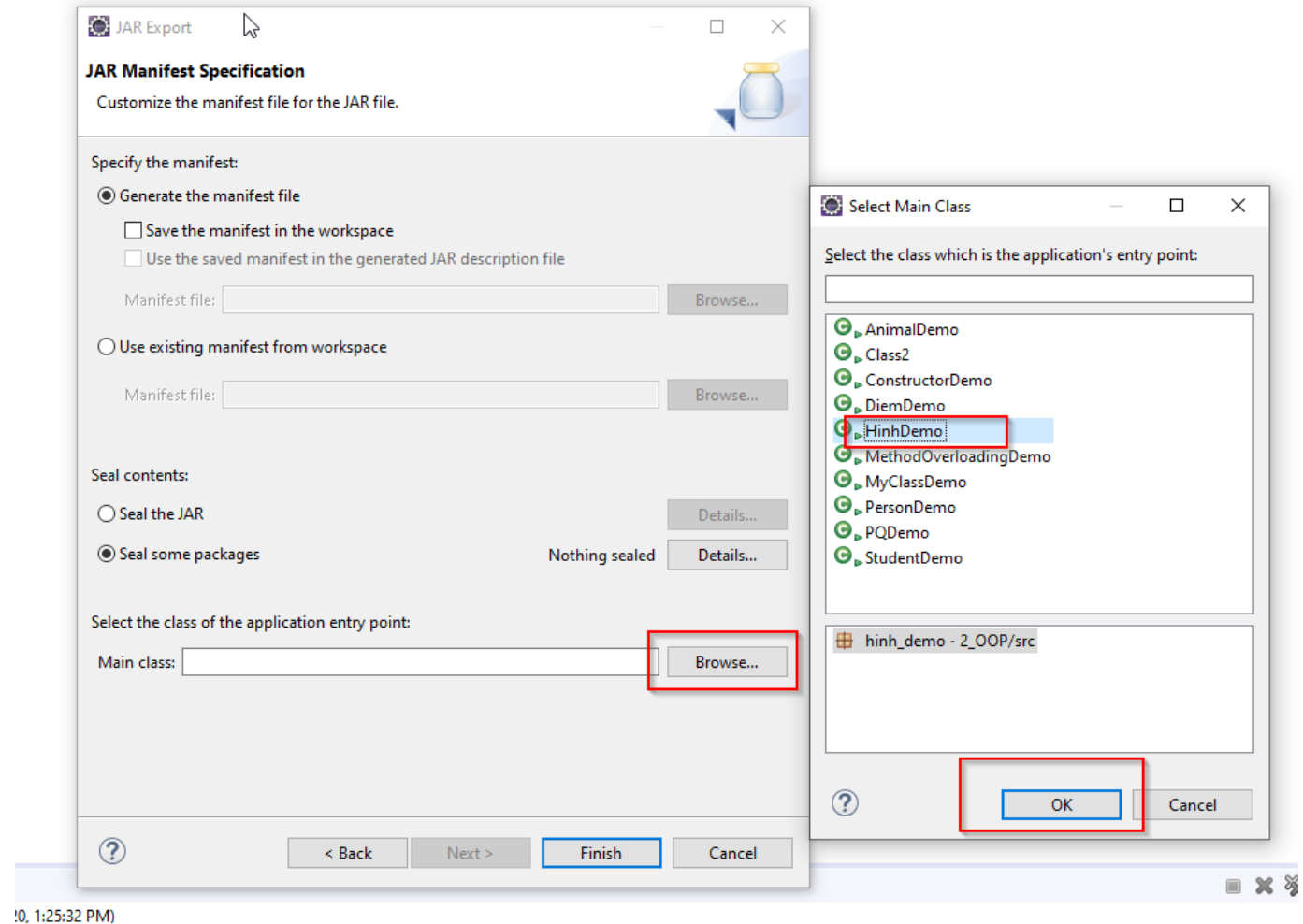
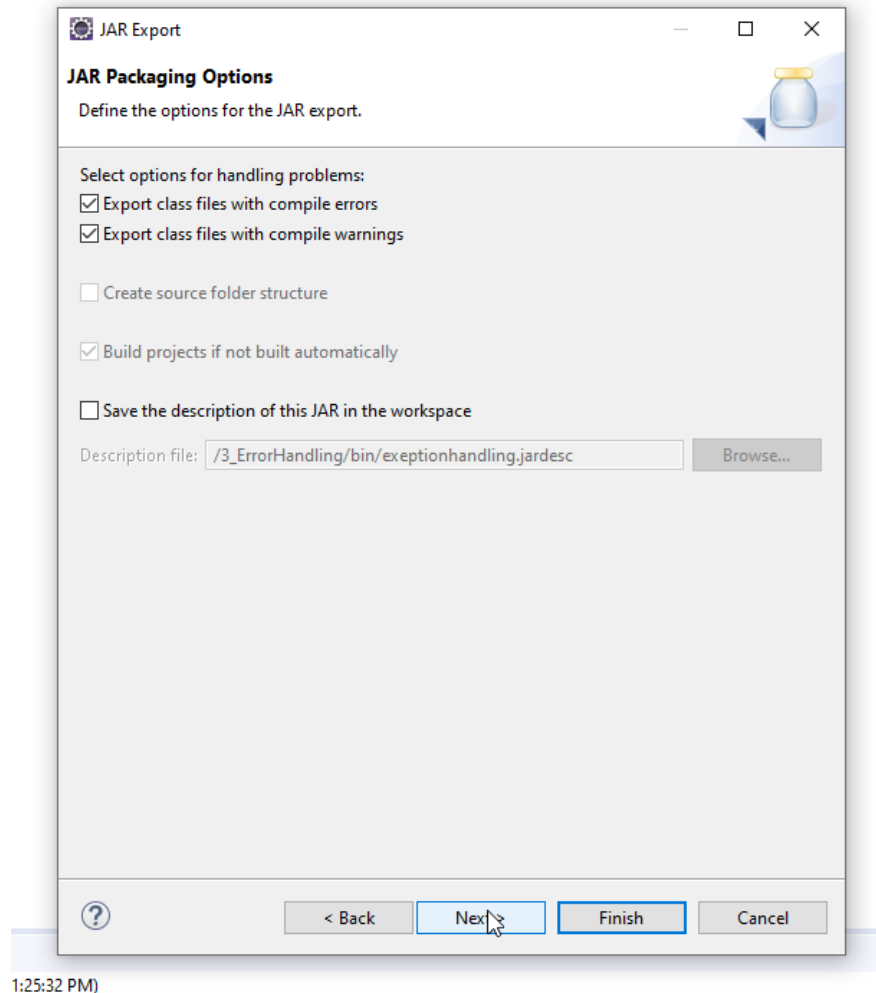


- Chọn project, danh sách các packages, tên file jar cần export và ấn **Ne**



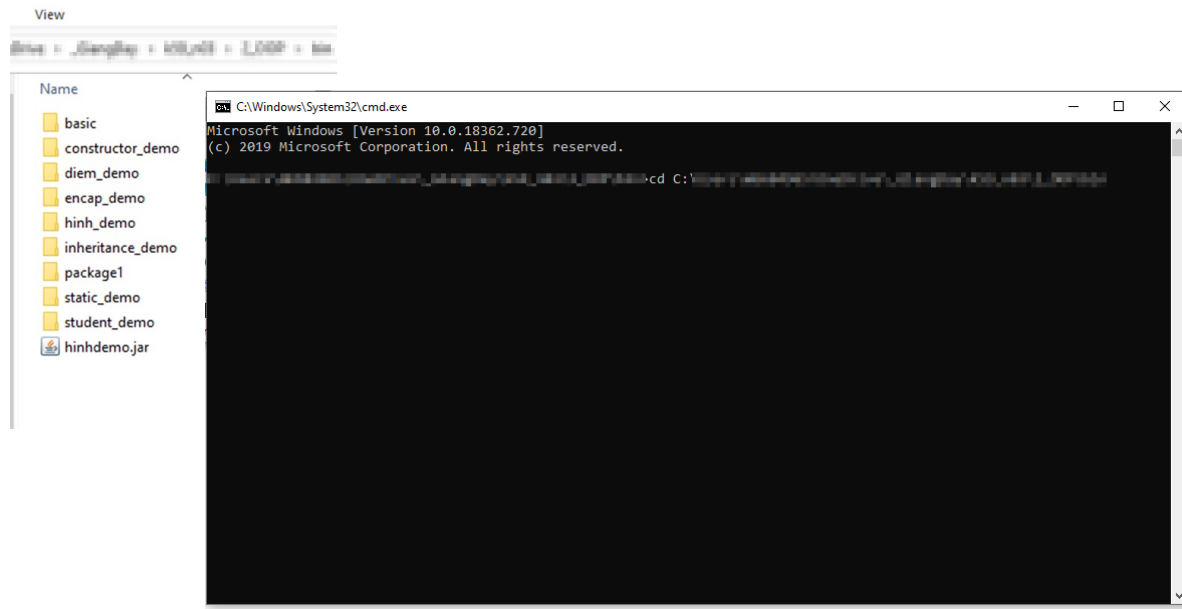
ĐÓNG GÓI .JAR FILE

- Ấn Next > tiếp
- Chọn hàm main của file jar và ấn

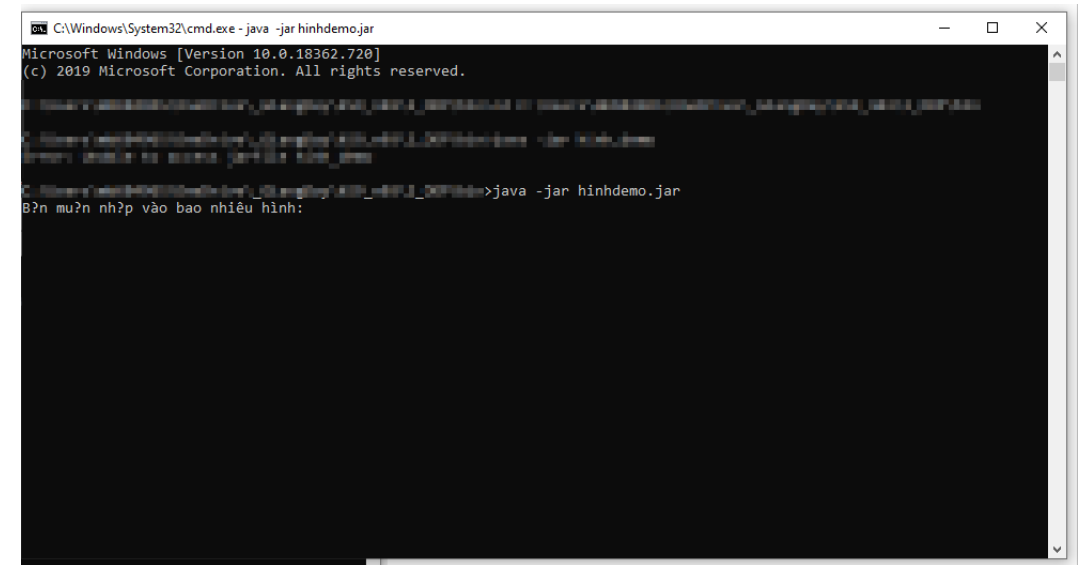


Chạy thử file jar

- **cd** vào thư mục đích



- Chạy lệnh: `java -jar <tên file jar>`



Tài liệu tham khảo

<https://freetuts.net/cac-kieu-du-lieu-trong-java-1037.html>

<https://www.geeksforgeeks.org/interning-of-string/>

<http://tutorials.jenkov.com/java/constructors.html>

https://www.w3schools.com/java/java_interface.asp

Core Java(TM), Volume I—Fundamentals - Cay S. Horstmann