LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

CT276 – Lập trình Java

NỘI DUNG

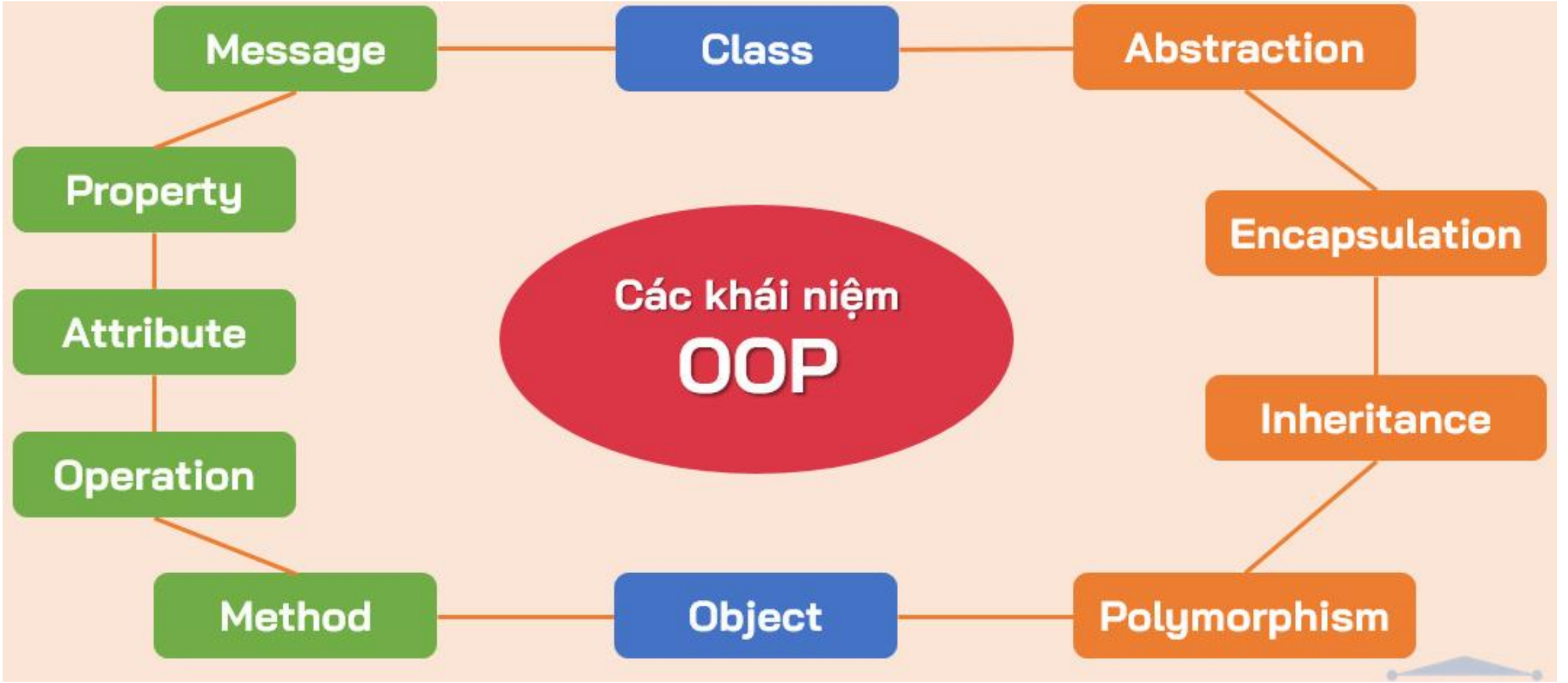
- Khái niệm
- Lớp và các thành phần của lớp
- Đối tượng
- Kế thừa
- Đa hình
- Interface
- Package
- Lớp lồng nhau
- Ngoại lệ

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

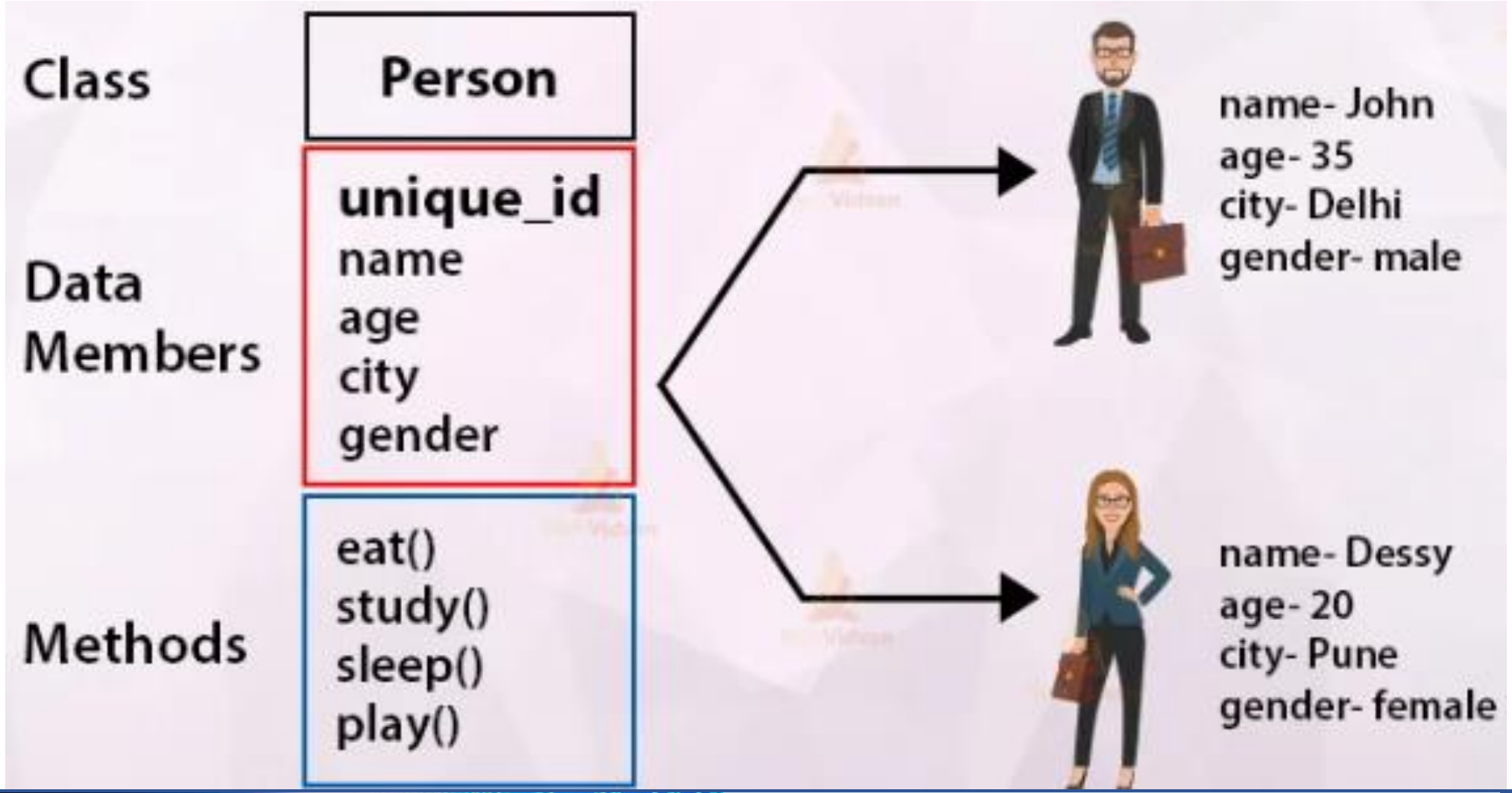
- Tập trung vào dữ liệu hơn là cách thao tác với dữ liệu

Đơn giản hóa các hệ thống phức tạp	Tái sử dụng cao	Dễ sửa lỗi / bảo trì / mở rộng
Bảo mật cao	Tiết kiệm tài nguyên	Hiệu suất cao

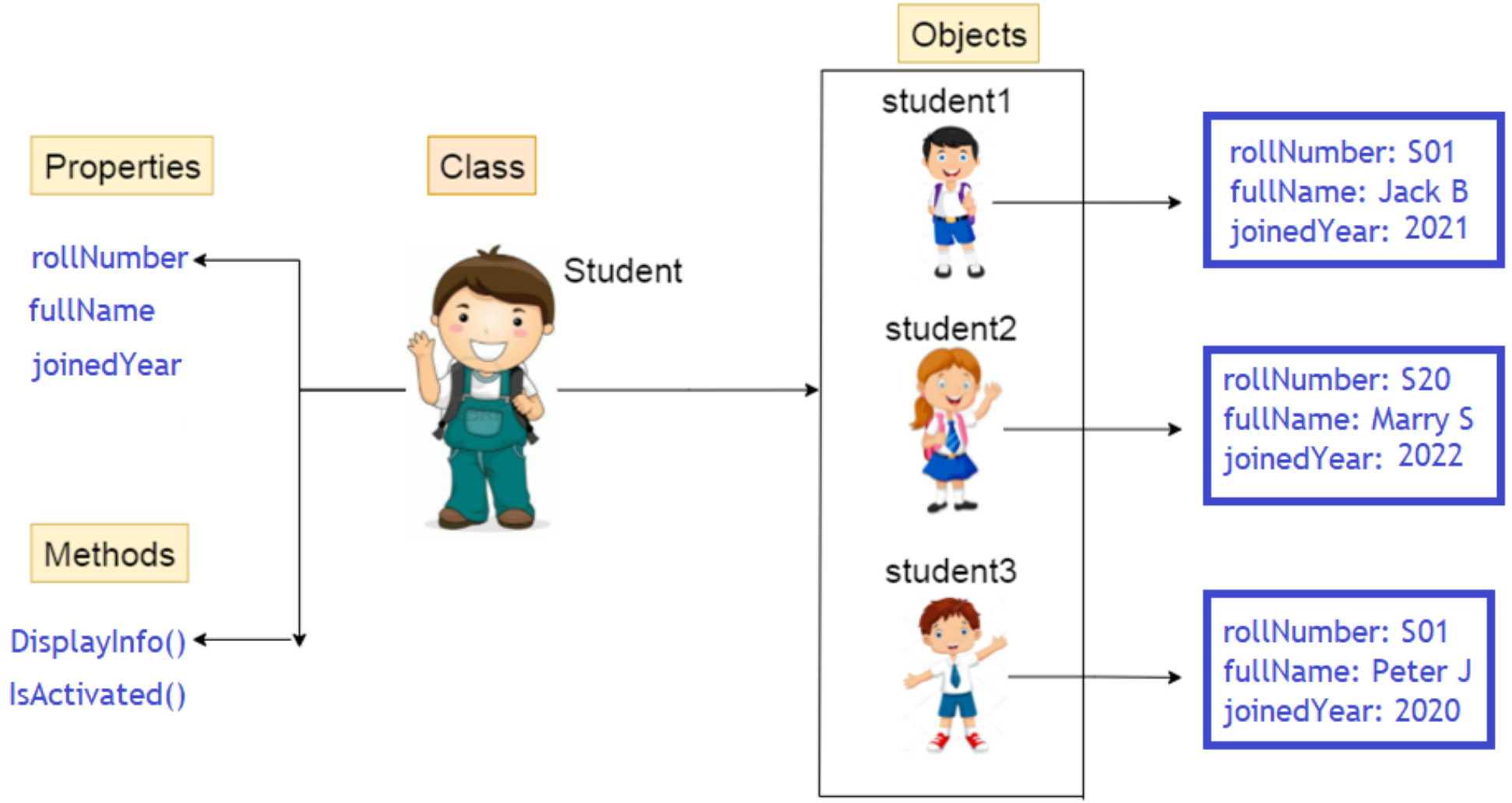
LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG



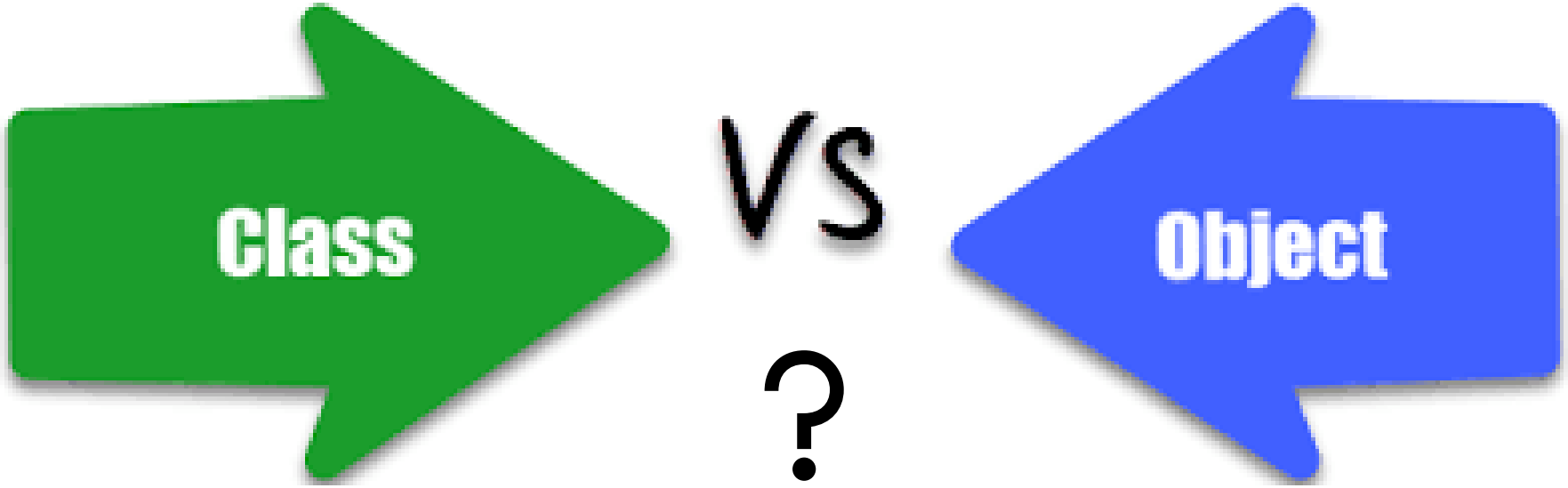
CLASS VS OBJECT



CLASS VS OBJECT



CLASS VS OBJECT ?

- 

LỚP

- Khai báo:

public/(none)

```
access-modifier class <ClassName>
{
        // Properties

        // Methods
        …..
}
```

CHÚ Ý: *1 file java có thể chứa nhiều class, nhưng chỉ có 1 public class*

THUỘC TÍNH

- Thuộc tính:

```
access-modifier DataType PropertyName;
```

- Các giá trị access-modifier: `public / private / protected`

```
class Employee {
    public String empId;
    public String empName;
    protected int salary;
    protected float commission;
}
```

PHƯƠNG THỨC

- Thuộc tính:

```
access-modifier ReturnType MethodName(parameters){}
```

- Các giá trị access-modifier: `public / private / protected`

```
public void calcCommission(float sales){
    if(sales > 10000)
        commission = salary * 20 / 100;
    else
        commission=0;
}
```

HÀM XÂY DỰNG

- Được dùng để tạo các đối tượng của lớp
- Cú pháp:

```
AccessModifier ClassName(parameters){}
```

- Đặc điểm:
  - Có tên trùng với tên lớp
  - Không có giá trị trả về
  - Có nhiều hàm xây dựng trong 1 lớp
  - Có thể overload

HÀM XÂY DỰNG

```
class Employee {
    public String empId;
    public String empName;
    private int salary;
    protected float commission;

    public Employee(){}
    public Employee(String id, String name, int sal) {
        empId=id;
        empName=name;
        salary=sal;
    }
}
```

TỪ KHÓA "THIS"

- "`this`": tham chiếu đến đối tượng hiện tại:
  - Gọi các biến của đối tượng hiện tại: this.name
  - Gọi phương thức của đối tượng hiện tại: this.method()
  - Truyền đối tượng hiện tại khi gọi phương thức: display(this)
  - Gọi hàm xây dựng khác trong cùng lớp: this("E001", 10)
  - Gọi lớp nội bộ (inner class): Outer.this.name

TỪ KHÓA "THIS"

- "`this`": tham chiếu đến đối tượng hiện tại

```
class A{
    int ID = 100;

    void hello(int ID){
        System.out.println("Gia tri ID trong A: " + this.ID);
        System.out.println("Gia tri ID trong ham hello(): " + ID);
    }
}
```

```
Gia tri ID trong A: 100
Gia tri ID trong ham hello(): 500
```

TỪ KHÓA "THIS"

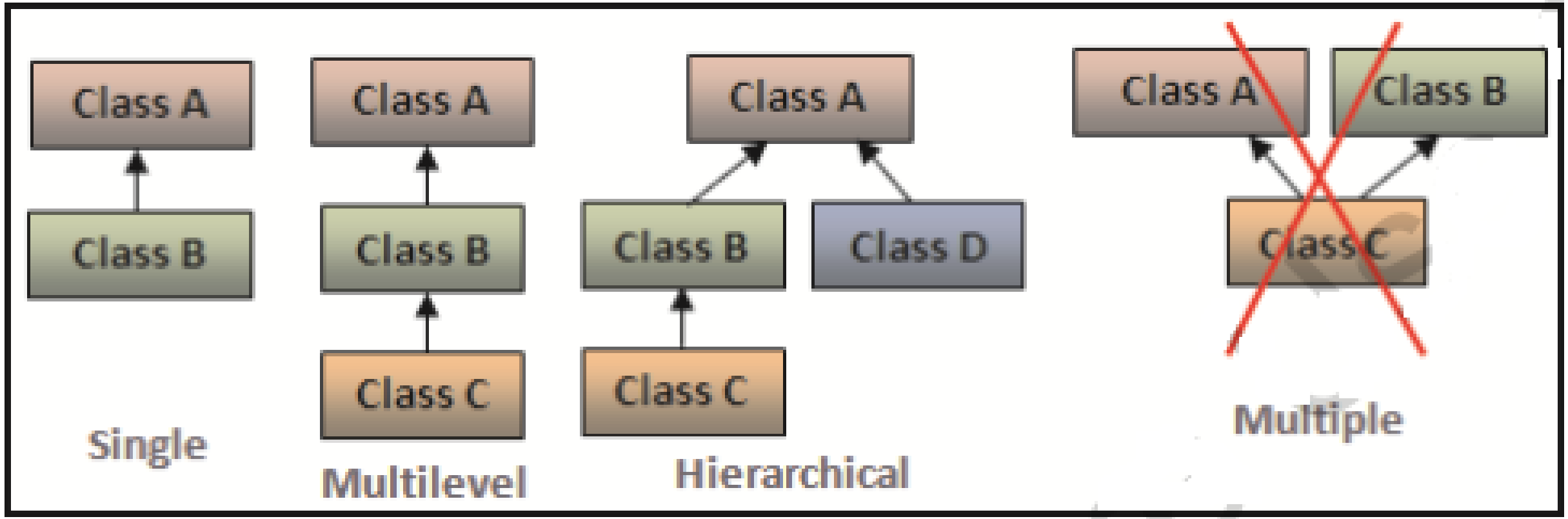
```
class Outer {
    private int x = 10;

    class Inner {
        private int x = 20;

        public void display() {
            System.out.println("Inner x: " + x);              // Biến x của Inner
            System.out.println("Outer x: " + Outer.this.x); // Biến x của Outer
        }
    }
}
```

TẠO ĐỐI TƯỢNG

```
Employee empl =new Employee();
empl.empId = "E001";
empl.empName = "Nguyen Van A";
empl.commission = 2.5f;

empl.displayDetails();

Employee emp2 = new Employee("E002","Maria Nemeth", 40000);

emp2.calcCommission(20000F);
emp2.displayDetails();
```

```
Employee ID:E001
Employee Name:Nguyen Van A
Salary:0
Commission:2.5
Employee ID:E002
Employee Name:Maria Nemeth
Salary:40000
Commission:8000.0
```

KẾ THỪA


Class A
Class B
Single
Class A
Class B
Class C
Multilevel
Class A
Class B
Class D
Class C
Hierarchical
Class A
Class B
Class C
Multiple

KẾ THỪA

- Cú pháp:

```
class Child extends Parent{

}
```

- Được kế thừa các thành phần của lớp cha, ngoài trừ các thành phần `private`
  - Thuộc tính
  - Phương thức
  - Hàm xây dựng

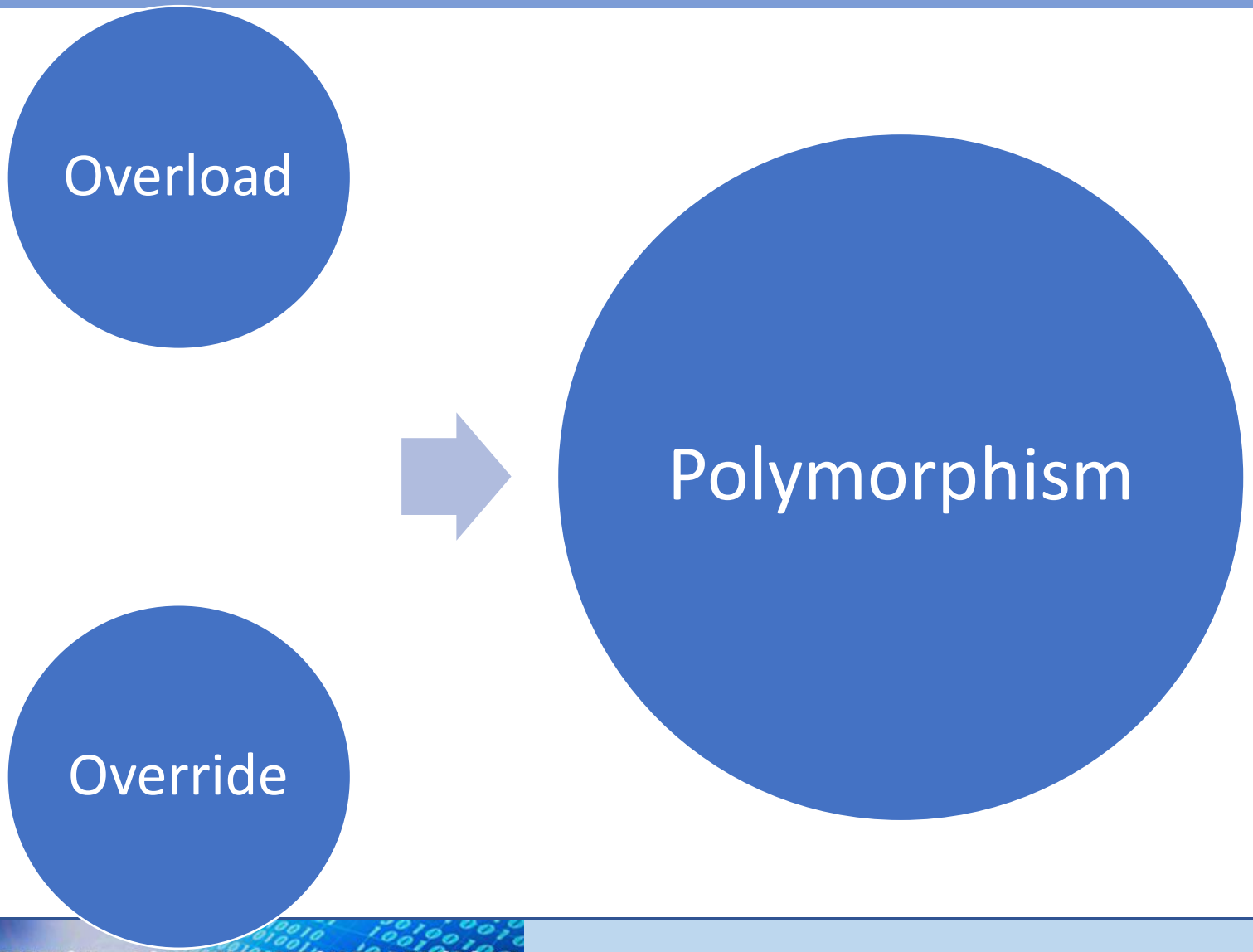
KẾ THỪA

```
class PartTimeEmployee extends Employee{
    String shift;

    public PartTimeEmployee(String id, String name, int sal, String shift){
        // Invoke the super class constructor
        super(id, name, sal);
        this.shift=shift;
    }

    @Override
    public void displayDetails(){
        calcCommission(12); // Invoke the inherited method
        super.displayDetails(); // Invoke the super class display method
        System.out.println("Working shift:"+shift);
    }
}
```

TỪ KHÓA "SUPPER"

- "`supper`": tham chiếu đến lớp cha
- Được sử dụng để gọi các thành phần trong lớp cha:
  - Hàm xây dựng
  - Thuộc tính
  - Phương thức

ĐA HÌNH



OVERLOAD

```
public class Polymorphism {
    public static void main(String[] args) {
        Calculator cal = new Calculator();
        int tong1 = cal.add(5,10);
        int tong2 = cal.add(5, 10, 15);
        double tong3 = cal.add(2.5f, 7.5f);
    }

}
class Calculator {
    int add(int a, int b) {
        return a + b;
    }

    int add(int a, int b, int c) {
        return a + b + c;
    }

    double add(double a, double b) {
        return a + b;
    }
}
```

OVERRIDE

```
public class Polymorphism {
    public static void main(String[] args) {
        Dog d = new Dog();
        d.makeSound();
    }

}
class Animal {
    void makeSound() {
        System.out.println("Animal makes a sound");
    }
}

class Dog extends Animal {
    @Override
    void makeSound() {
        System.out.println("Dog barks");
    }
}
```

```
Dog barks
```

INTERFACE

- Định nghĩa một khung/mẫu/hợp đồng cho lớp.
- Quy định các phương thức mà một lớp phải tuân thủ
- Thành phần interface:
  - Hằng
  - Phương thức trừu tượng (abstract method)
  - Phương thức mặc định (default method)
  - Phương thức tĩnh (static method)

ABSTRACT METHOD

- Chỉ có khai báo, không có phần thân
- Cú pháp: `ReturnType MethodName(parameters);`

```
interface InterfaceName {
    public static final int CONSTANT_NAME = 100;

    public void method1();
    public void method2();
}
```

DEFAULT METHOD

- Có chứa phần thân phương thức
- Được khai báo với từ khóa `default`
- Có thể được sử dụng ở lớp kế thừa
- Có thể được ghi đè (override) ở lớp kế thừa

```
interface InterfaceName {
    default void defaultMethod() {
        System.out.println("Phương thức mặc định");
    }
}
```

STATIC METHOD

- Được gọi trực tiếp thông qua tên lớp mà không cần tạo đối tượng
- Không được `override`
- Được khai báo với từ khóa `static`

```
interface InterfaceName {
    static void staticMethod() {
        System.out.println("Phuong thức static");
    }
}
```

INTERFACE

```
interface InterfaceName {
    public static final int CONSTANT_NAME = 100;

    public void method1();
    public void method2();

    default void defaultMethod() {
        System.out.println("Phương thức mặc định");
    }
    static void staticMethod() {
        System.out.println("Phương thức static");
    }
}
```

TRIỂN KHAI INTERFACE

- Một lớp có thể triển khai (kế thừa) một hoặc nhiều interface
  - Hỗ trợ đa kế thừa
- Một lớp kế thừa từ 1 interface:
  - Có tất cả các thành phần mà interface quy định
  - Định nghĩa phần thân cho tất cả các phương thức trừu tượng
  - Có thể định nghĩa thêm các thành phần khác

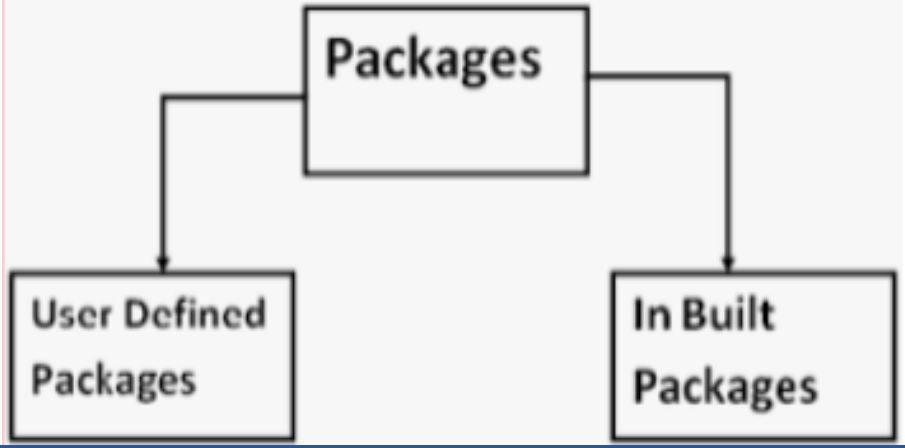
TRIỂN KHAI INTERFACE

```java
interface InterfaceName {
    public static final int CONSTANT_NAME = 100;

    public void method1();
    public void method2();

    default void defaultMethod() {
        System.out.println("Phương thức mặc định");
    }
    public static void staticMethod() {
        System.out.println("Phương thức static");
    }
}
class MyClass implements InterfaceName {
    @Override
    public void method1() {
        System.out.println("Phương thức method1 được triển khai.");
    }

    @Override
    public void method2() {
        System.out.println("Phương thức method2 được triển khai.");
    }
}
```

```java
public static void main(String[] args) {
    MyClass obj = new MyClass();
    obj.method1();
    obj.method2();
    obj.defaultMethod();
    InterfaceName.staticMethod();
}
```

PACKAGE

- Là một không gian tên (namespace) được sử dụng để gom nhóm các class và interface có liên quan với nhau.
- Một package có thể chứa nhiều package khác.
- Trong một package không có các thành phần trùng tên nhau.



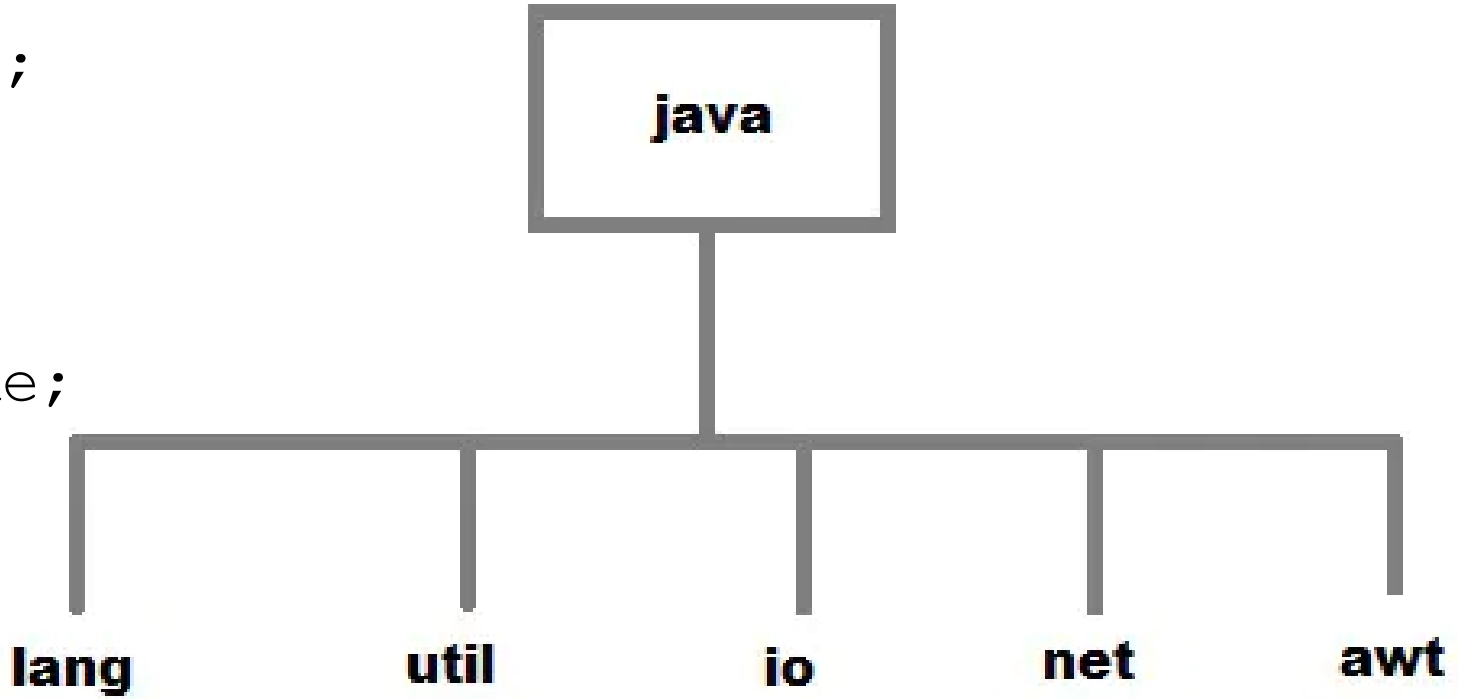
IN-BUILT PACKAGE

- Khai báo sử dụng các class trong package:

```
import packagename.ClassName;

import packagename.*;
```

- Gọi class trong package:

```
packagename.ClassName;
```



```
String str = "Hello";
System.out.println("String length is:" + str.length());
System.out.println("Character at index 2 is:" + str.charAt(2));
System.out.println("Concatenated string is:" + str.concat("World"));
System.out.println("String comparison is:" + str.compareTo("World"));
System.out.println("Index of o is:" + str.indexOf("o"));
System.out.println("Last index of l is:" + str.lastIndexOf("l"));
System.out.println("Replaced string is:" + str.replace('e', 'a'));
System.out.println("Substring is:" + str.substring(2, 5));
String str1 = " Hello ";
System.out.println("Untrimmed string is:" + str1);
System.out.println("Trimmed string is:" + str1.trim());
```

String class

```
String length is:5
Character at index 2 is:l
Concatenated string is:HelloWorld
String comparison is:-15
Index of o is:4
Last index of l is:3
Replaced string is:Hallo
Substring is:llo
Untrimmed string is: Hello
Trimmed string is:Hello
```

STRINGBUILDER CLASS

```
StringBuilder str = new StringBuilder("JAVA ");
System.out.println("Appended String is " + str.append("7"));
System.out.println("Inserted String is " + str.insert(5, "SE "));
System.out.println("Deleted String is " + str.delete(4, 7));
System.out.println("Reverse String is " + str.reverse());
```

StringBuilder class

```
Appended String is JAVA 7
Inserted String is JAVA SE 7
Deleted String is JAVA 7
Reverse String is 7 AVAJ
```

```
int a =2, b=5;
System.out.println("2~5 = " + Math.pow(a, b));
System.out.println("Max of a and b is " + Math.max(a, b));
int n;
n = (int) (Math.random()*10);
System.out.println("A random number (0-10): " + n);
System.out.println("The square root of 81 is: " +Math.sqrt(81));
System.out.println("Ceiling of 3.6 is: " + Math.ceil(3.6));
System.out.println("Floor of 3.6 is: " + Math.floor(3.6));
System.out.println("Round of 3.6 is: " + Math.round(3.6));
```

Math class

```
2~5 = 32.0
Max of a and b is 5
A random number (0-10): 2
The square root of 81 is: 9.0
Ceiling of 3.6 is: 4.0
Floor of 3.6 is: 3.0
Round of 3.6 is: 4
```

```java
LocalDate localDate = LocalDate.now();
System.out.println(localDate);

//Specific date
  LocalDate localDate2 = LocalDate.of(2022, Month.MAY, 20);
  System.out.println(localDate2); // 2022-05-20
System.out.println("Year : " + localDate.getYear());
System.out.println("Month : " + localDate.getMonth().getValue());
System.out.println("Day of Month : " + localDate.getDayOfMonth());
System.out.println("Day of Week : " + localDate.getDayOfWeek());
System.out.println("Day of Year : " + localDate.getDayOfYear());

System.out.println("Addition of days : " + localDate.plusDays(5));
System.out.println("Addition of months : " + localDate.plusMonths(15));
System.out.println("Addition of weeks : " + localDate.plusWeeks(45));
System.out.println("Addition of years : " + localDate.plusYears(5));
// LocalDate's minus methods
System.out.println("Subtraction of days : " + localDate.minusDays(5));
System.out.println("Subtraction of months : " + localDate.minusMonths(15));
System.out.println("Subtraction of weeks : " + localDate.minusWeeks(45));
System.out.println("Subtraction of years : " + localDate.minusYears(5));
```

```
2024-11-12
Year : 2024
Month : 11
Day of Month : 12
Day of Week : TUESDAY
Day of Year : 317
Addition of days : 2024-11-17
Addition of months : 2026-02-12
Addition of weeks : 2025-09-23
Addition of years : 2029-11-12
Subtraction of days : 2024-11-07
Subtraction of months : 2023-08-12
Subtraction of weeks : 2024-01-02
Subtraction of years : 2019-11-12
```

LocalDate class

USER DEFINED PACKAGE

- Khai báo bằng từ khóa package

```
package packagename;
```

```
package Demo1;
```

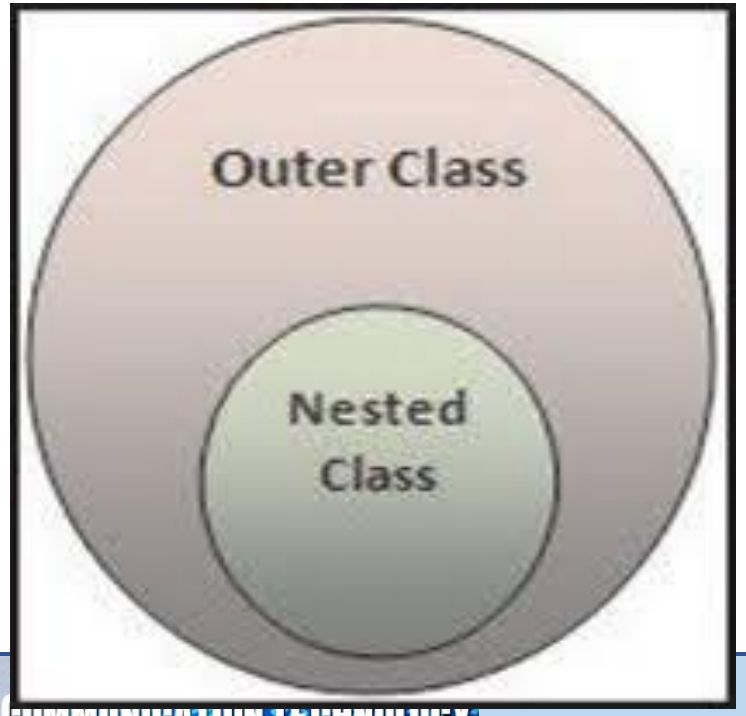
- Chỉ các thành phần trong 1 package có thể truy xuất lẫn nhau.

```
package Demo1;

import Demo2.*;
public class PackageDemo {
    public static void main(String[] args) {
        ClassA objA = new ClassA();

        ClassC objC = new ClassC();
    }

}
class ClassA{}
class ClassB{}
```

```
package Demo2;

public class ClassC {}

class ClassD{}
```

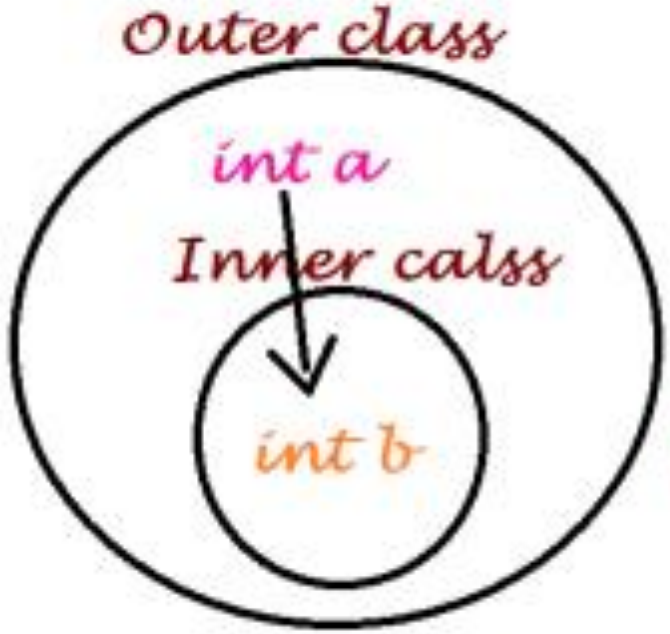
LỚP LỒNG NHAU (NESTED CLASS)

- Nested class là một lớp (inner) được định nghĩa bên trong 1 lớp khác (outer)
- Được sử dụng để giải quyết các vấn đề gắn liền với lớp bên trong, và cung cấp một cách linh hoạt để truy cập các thành phần của lớp bên ngoài

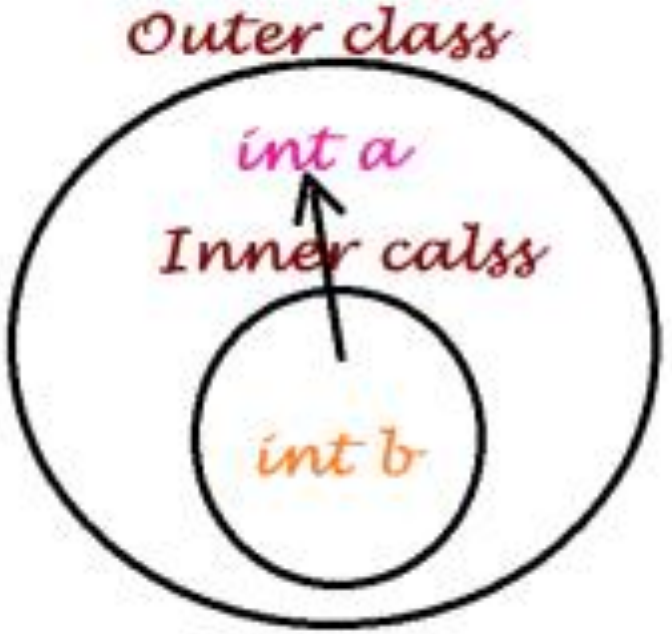


LỚP LỒNG NHAU (NESTED CLASS)

Scope of variables in nested classes.



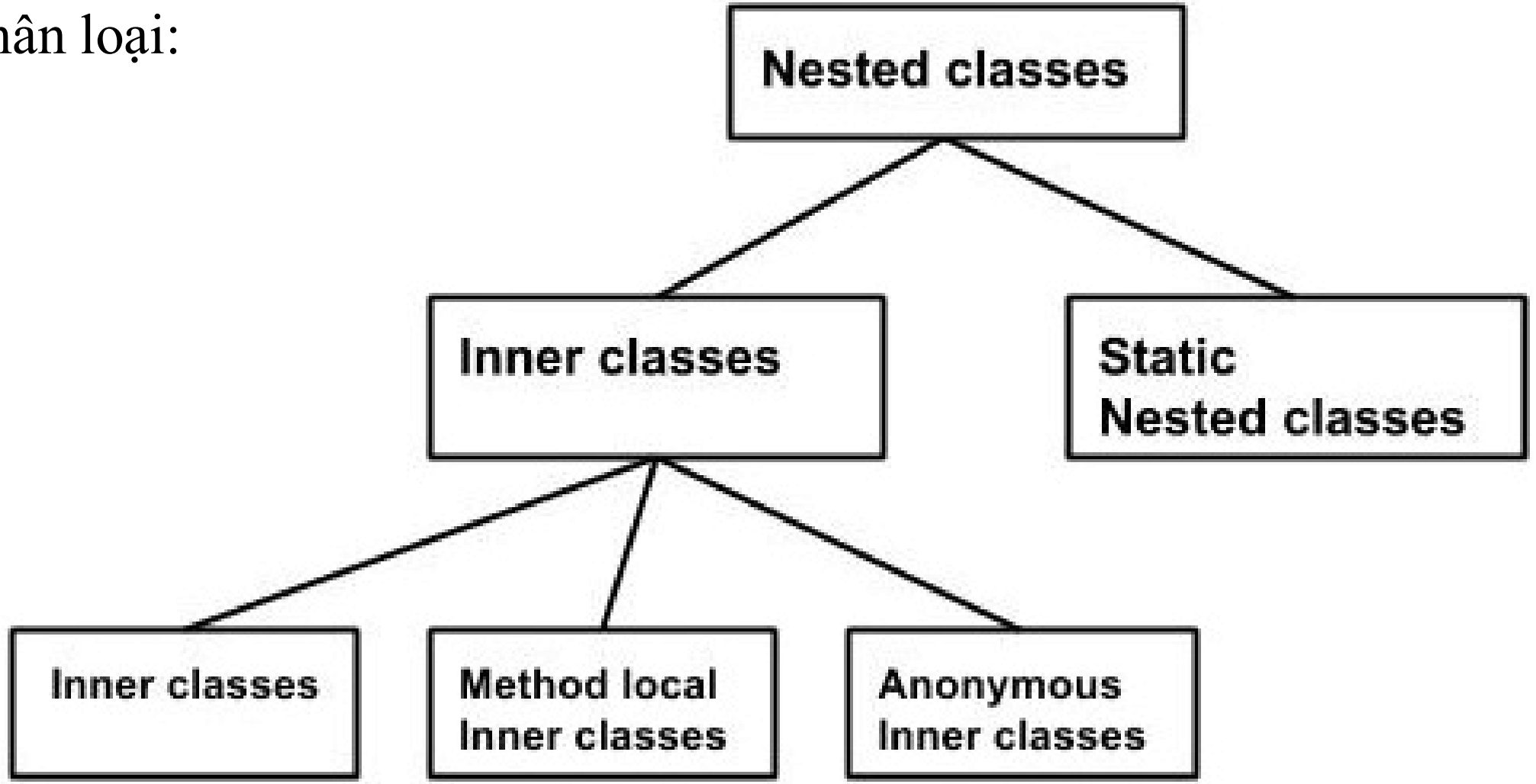
We can access outer class variables in inner class



We cannot access inner class variables in outer class.

LỚP LỒNG NHAU (NESTED CLASS)

- Phân loại:



STATIC NEDTED CLASS

```
public class NestedClass_Static {
    public static void main(String[] args) {
        Outer.StaticNestedClass obj = new Outer.StaticNestedClass();
        obj.display();
    }
}
class Outer {
    static int outerStaticVar = 10;

    static class StaticNestedClass {
        void display() {
            System.out.println("Static nested class: " + outerStaticVar);
        }
    }
}
```

- Không cần tạo đối tượng Outer khi truy xuất
- Chỉ có thể truy xuất các thành viên static của outer class

INNER CLASS

- Inner class là một lớp được định nghĩa bên trong 1 lớp khác

```
class OuterClass {
    ...
     class InnerClass {
        ...
     }
}
```

```
OuterClass outer = new OuterClass();
OuterClass.InnerClass inner = outer.new InnerClass();
```

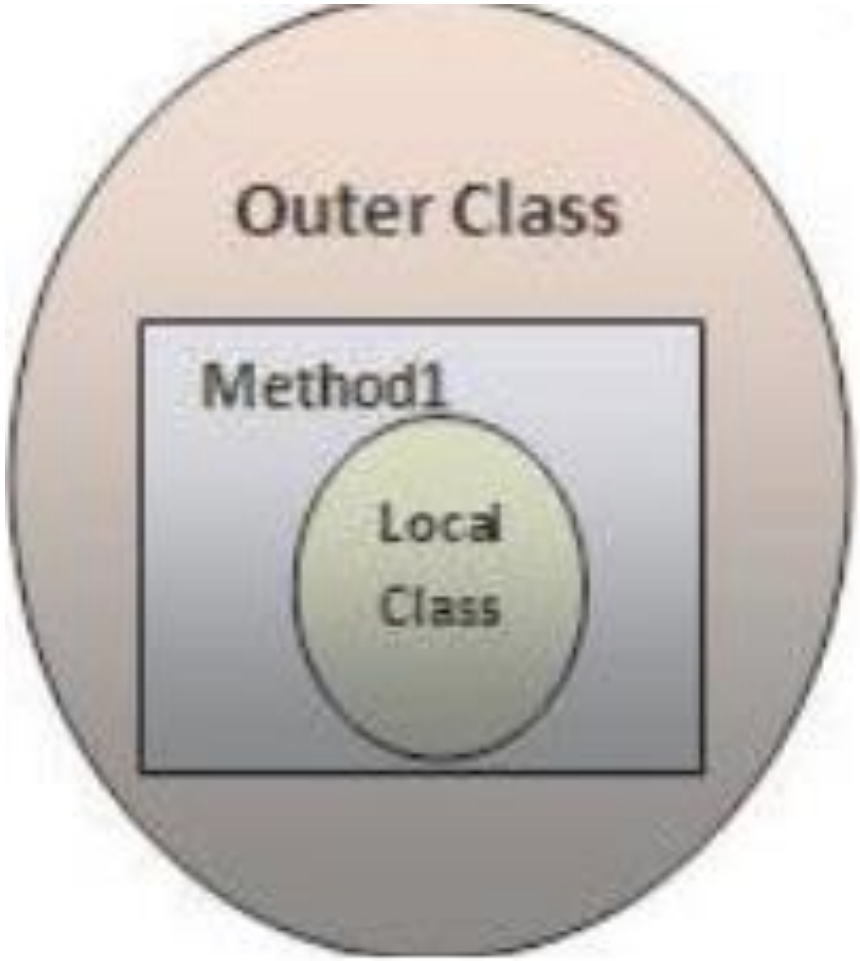
INNER CLASS

```
public class NestedClass_Inner {

    public static void main(String[] args) {
        Outer outerObj = new Outer();
        Outer.Inner innerObj = outerObj.new Inner();
        innerObj.display();
    }

}

class Outer {
    int outerVar = 5;

    class Inner {
        void display() {
            System.out.println("Inner class: " + outerVar);
        }
    }
}
```

- Cần tạo đối tượng Outer để truy cập
- Truy cập tất cả các thành viên của outer class

METHOD LOCAL INNER CLASS

- Được gọi tắt là Local class
- Được định nghĩa bên trong 1 phương thức
  - Chỉ được sử dụng bên trong phương thức
  - Truy cập đến các thành phần của phương thức

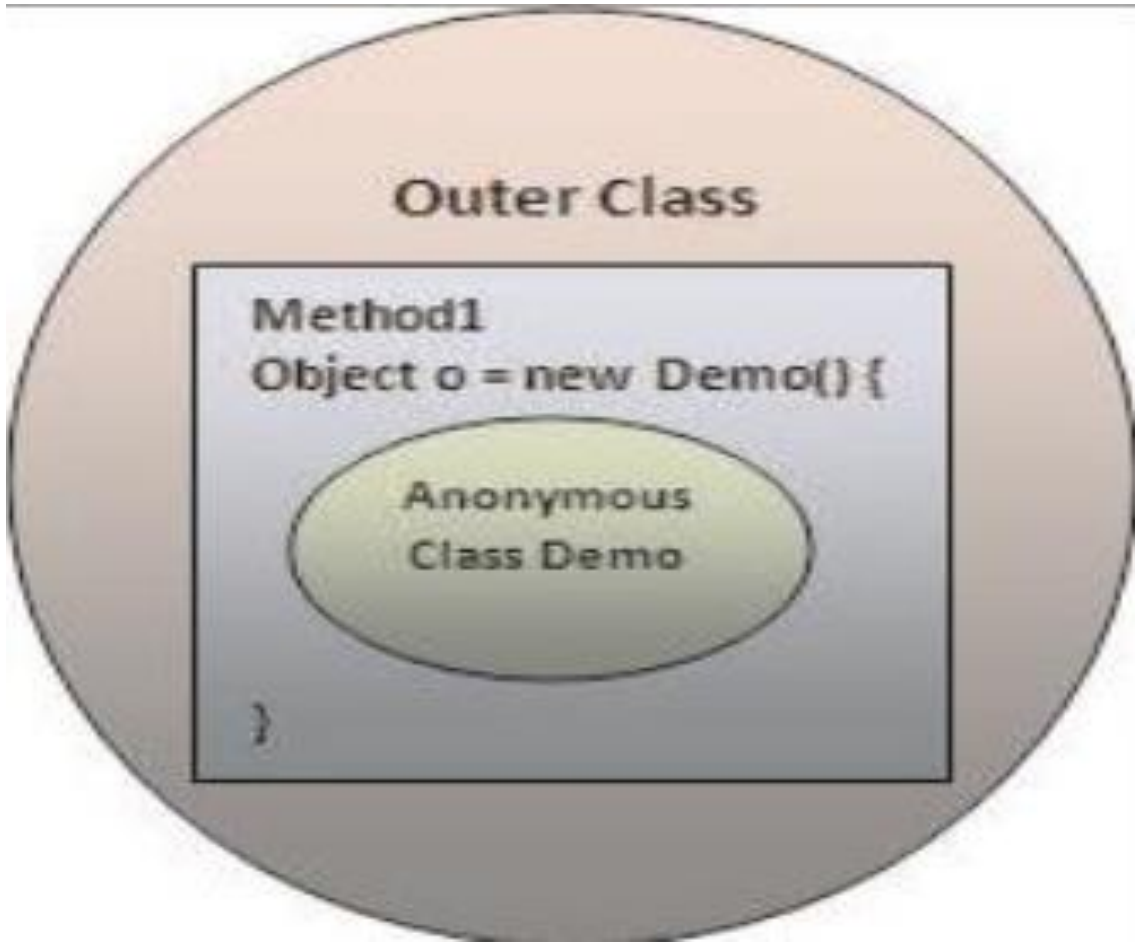


METHOD LOCAL INNER CLASS

```
public class NestedClass_Local {
    public static void main(String[] args) {
        Outer outer = new Outer();
        outer.method();
    }

}
class Outer {
    void method() {
        int a = 100;
        class LocalClass {
            void display() {
                System.out.println("Local class inside method " + a);
            }
        }

        LocalClass local = new LocalClass();
        local.display();
    }
}
```

ANONYMOUS INNER CLASS

- Được gọi tắt là Anonymous Class
- Không có tên
- Được sử dụng ngay sau định nghĩa



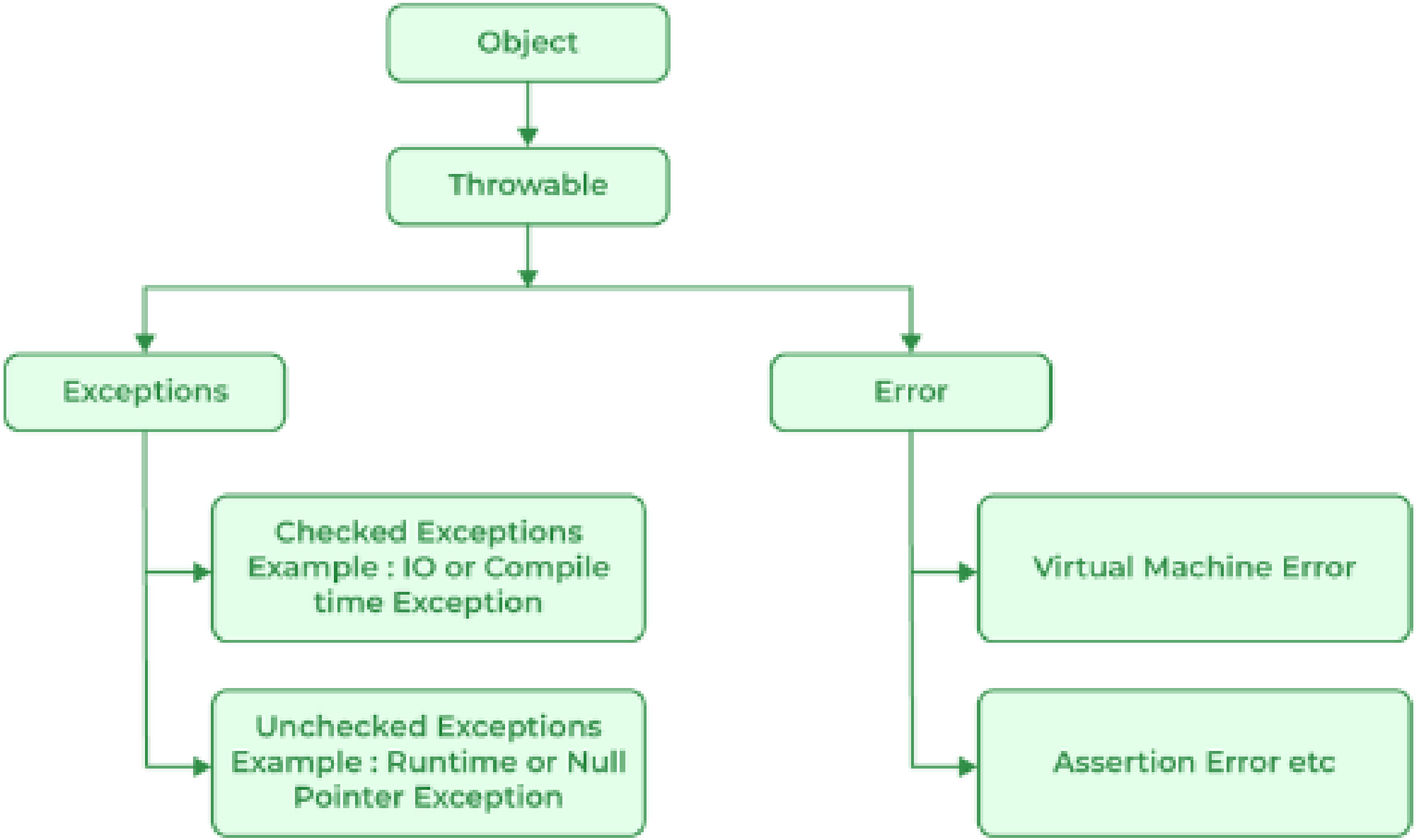
ANONYMOUS INNER CLASS

```
interface Greeting {
    void greet();
}
public class NestedClass_Anonymous {

    public static void main(String[] args) {
        Greeting greeting = new Greeting() {
            public void greet() {
                System.out.println("Hello from anonymous class!");
            }
        };
        greeting.greet();
    }
}
```

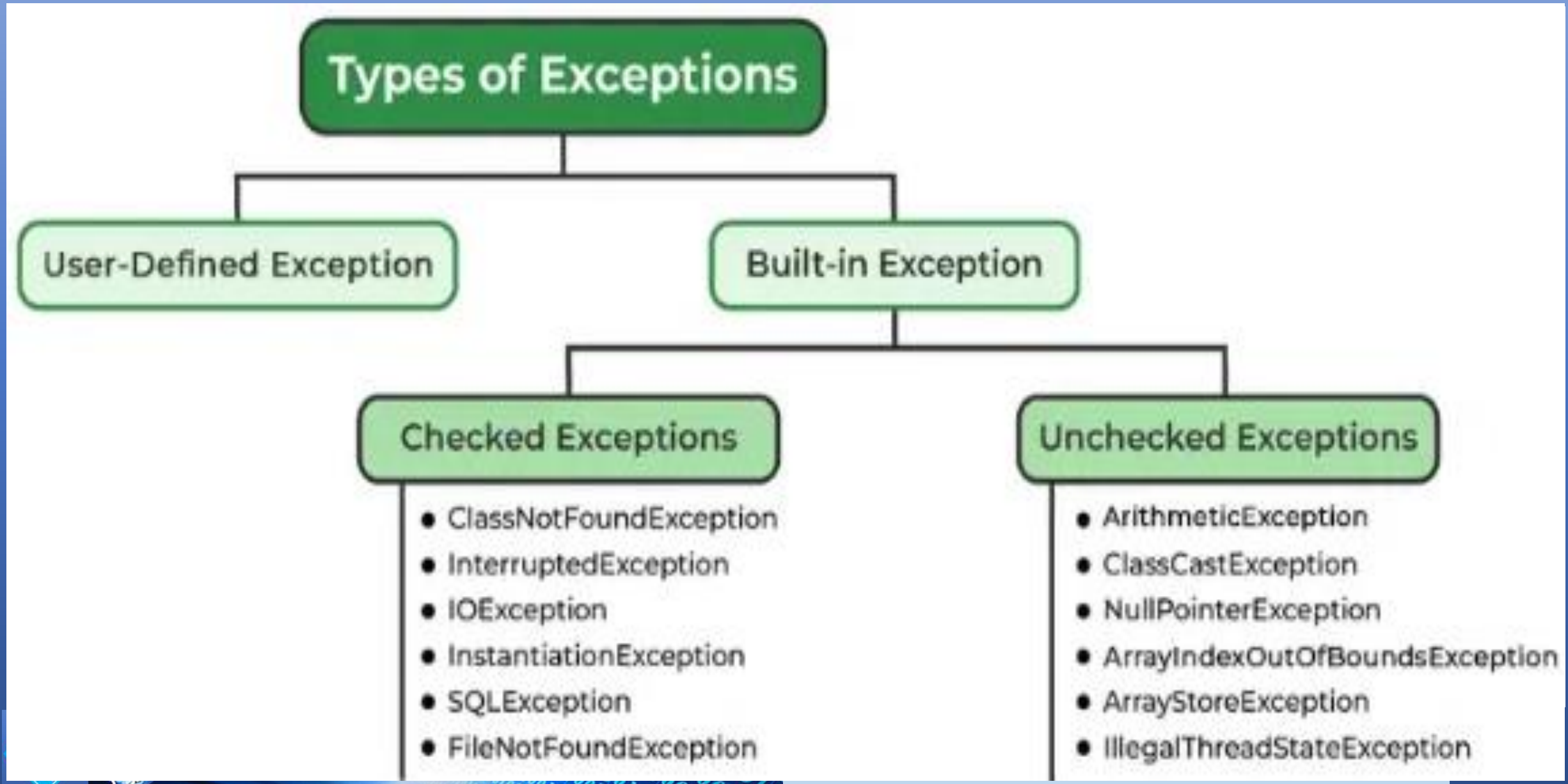
NGOẠI LỆ (EXCEPTION)

- Exception là một lỗi không mong muốn xảy ra trong quá trình thực thi chương trình, làm kết thúc chương trình.
- Nếu ngoại lệ không được xử lý, chương trình sẽ kết thúc.
- Khi một ngoại lệ xảy ra, chương trình sẽ sinh ra một đối tượng của lớp Throwable (hoặc lớp con của nó)

ERROR VS EXCEPTION



CÁC LOẠI EXCEPTION



TRY … CATCH

```
try {
    // Mã có thể gây ra ngoại lệ
} catch (ExceptionType1 e1) {
    // Xử lý ngoại lệ loại ExceptionType1
} catch (ExceptionType2 e2) {
    // Xử lý ngoại lệ loại ExceptionType2
} finally {
    // Mã trong block finally sẽ luôn được thực thi,
       dù có ngoại lệ hay không
}
```

TRY … CATCH

```
try {
    int[] arr = new int[5];
    arr[1] = 25;
    int a = 5, b=0,z;
    z = a/b;
} catch (ArithmeticException e) {
    System.out.println("Ngoai le toan hoc.");
} catch (ArrayIndexOutOfBoundsException e) {
    System.out.println("Chi so mang khong hop le.");
} catch (Exception e) {
    System.out.println("Ngoai le khac.");
}
```

TRY-WITH-RESOURCES

- Tự động đóng các tài nguyên (tập tin, kết nối cơ sở dữ liệu, …) mà không cần phải sử dụng finally để đóng tài nguyên thủ công.
- Điều này yêu cầu tài nguyên đó phải kế thừa interface AutoCloseable hoặc Closeable.

```
try (ResourceType resource = new ResourceType()) {
    // Sử dụng tài nguyên
} catch (ExceptionType e) {
    // Xử lý ngoại lệ nếu có
}
```

TRY-WITH-RESOURCES

```
try (FileReader fileReader = new FileReader("file.txt");
     BufferedReader bufferedReader = new BufferedReader(fileReader)) {

    String line;
    while ((line = bufferedReader.readLine()) != null) {
        System.out.println(line);
    }

} catch (IOException e) {
    System.out.println("Loi khi doc file: " + e.getMessage());
}
```

TRY-WITH-RESOURCES

```
try (FileReader fileReader = new FileReader("file.txt")) {
    // Gây ra ngoại lệ khi đọc file
    int result = 10 / 0;
} catch (ArithmeticException e) {
    System.out.println("Lỗi: " + e.getMessage());
} catch (IOException e) {
    System.out.println("Lỗi khi đọc tệp: " + e.getMessage());
}
```

CÂU HỎI