ĐẠI NAM
UNIVERSITY

ĐA HÌNH

Thời gian: 4 tiết

NỘI DUNG

- ✓ Khái niệm
- ✓ Upcasting và Downcasting
- ✓ Toán tử instanceof
- ✓ Liên kết tĩnh, liên kết động
- ✓ Đa hình

Đa hình

- Tính đa hình (polymorphism) là một trong bốn tính chất cơ bản của lập trình hướng đối tượng trong Java.
- Tính đa hình là khả năng một đối tượng có thể thực hiện một tác vụ theo nhiều cách khác nhau.
- Tính đa hình được thể hiện rõ nhất qua cách thức truy xuất hay gọi phương thức của đối tượng.
- Các phương thức hoàn toàn có thể giống nhau, nhưng việc xử lý luồng có thể khác nhau.

Đa hình

- Trong Java, sử dụng nạp chồng phương thức (method overloading) và ghi đè phương thức (method overriding) để có tính đa hình.
- Nạp chồng (overloading) là một lớp có nhiều phương thức cùng tên nhưng có các tham số khác nhau.
- Ghi đè (overriding) là hai phương thức cùng tên, cùng tham số, cùng kiểu trả về nhưng phương thức ở lớp con viết lại và dùng theo cách riêng của lớp con.

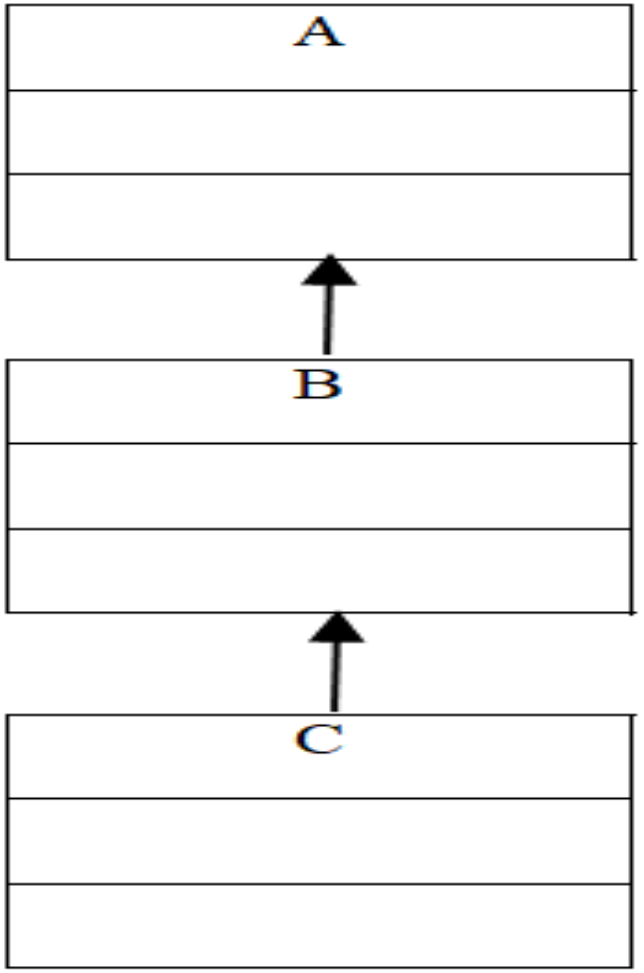
Đa hình

Có các ép kiểu đối tượng theo cây phân cấp kế thừa như sau:

A var1 = **new** B();

A var1 = **new** A();

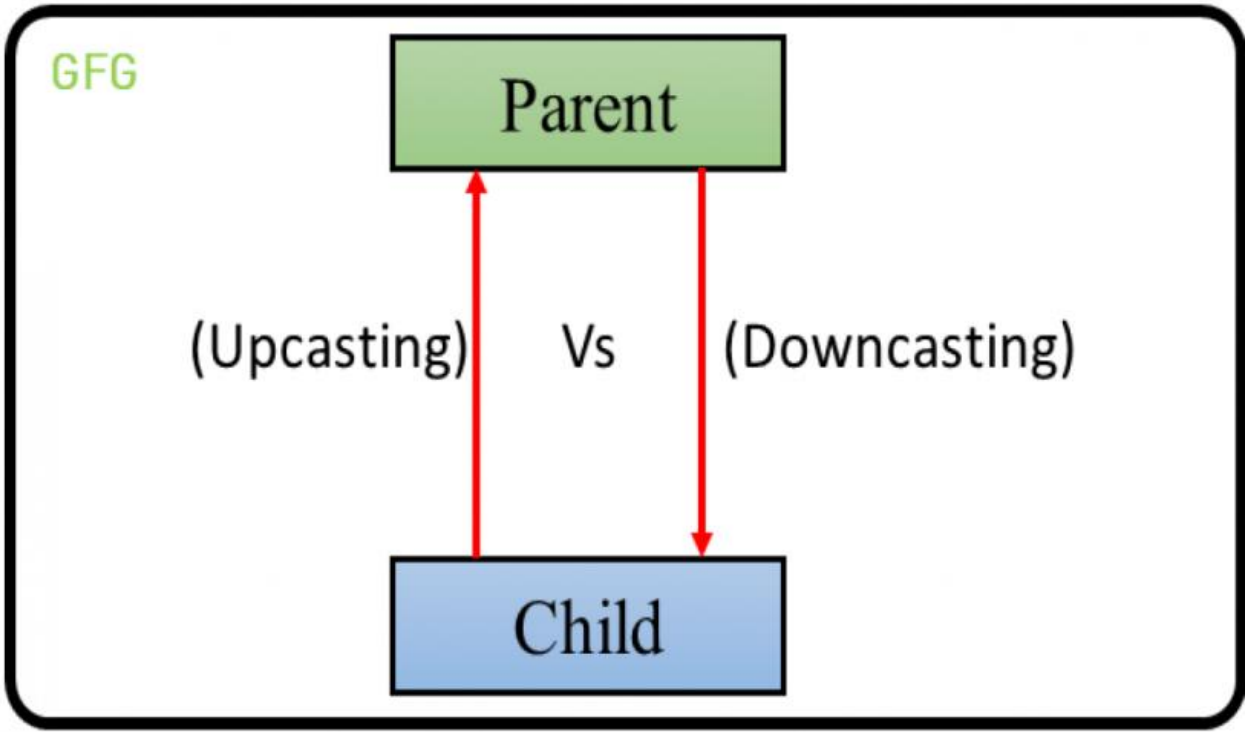
C var2 = (C) var1;



Upcasting và Downcasting

- Trong Java, cơ chế upcasting và downcasting là cơ chế được sử dụng để chuyển kiểu đối với kiểu dữ liệu tham chiếu
- Mục đích chuyển kiểu để tối ưu hóa bộ nhớ, do ép kiểu giúp quá trình sử dụng bộ nhớ hiệu quả hơn bằng cách chuyển đổi kiểu dữ liệu sang một kiểu dữ liệu với kích thước nhỏ hơn
- Ép kiểu trong Java giữa các đối tượng và lớp có thể được thực hiện thông qua hai phương pháp upcasting và downcasting

Upcasting và Downcasting



Upcasting

- Upcasting là quá trình chuyển kiểu (casting) một đối tượng của lớp con (subclass) sang kiểu của lớp cha (superclass).
- Đặc điểm của Upcasting:
  - Ngầm định.
  - Tính đa hình: Một tham chiếu tới lớp cha tham chiếu đến mọi đối tượng của các lớp con
  - Giới hạn phương thức: Khi upcast một đối tượng, chỉ có thể gọi các phương thức trong lớp cha.

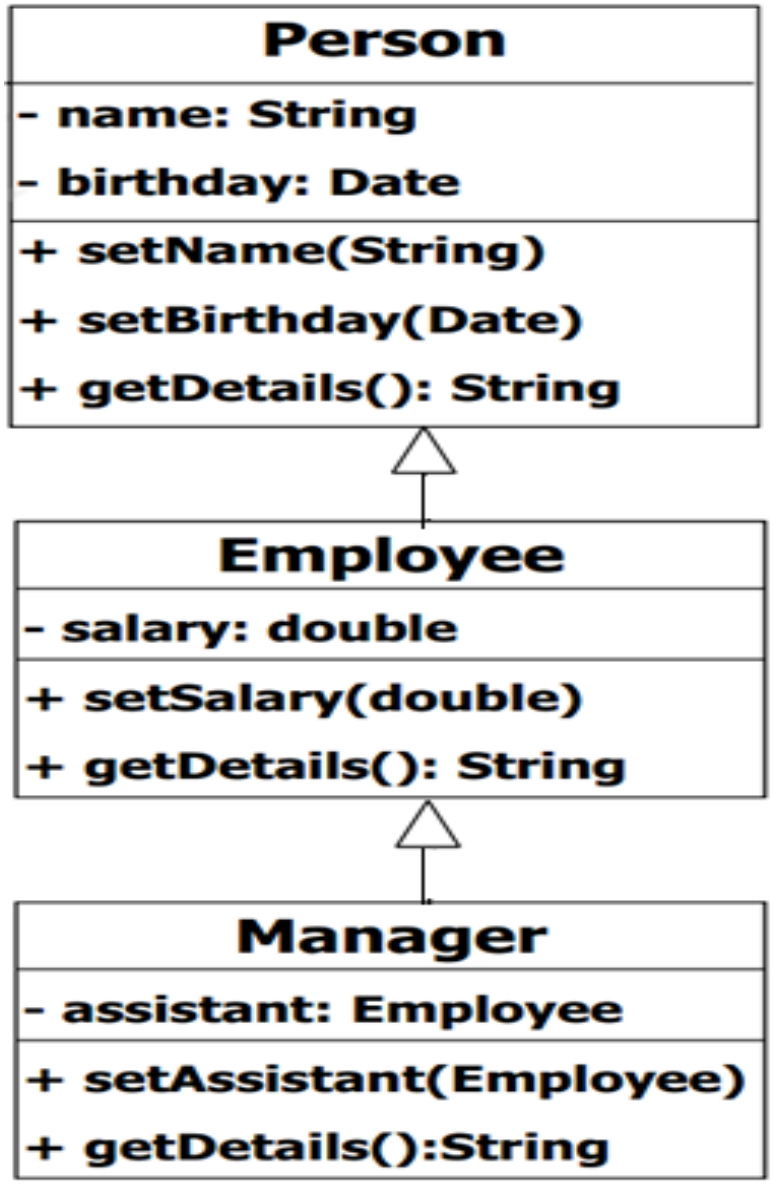
Downcasting

- Upcasting là quá trình chuyển kiểu (casting) từ lớp cha (superclass) sang lớp con (subclass), theo hướng đi xuống cây phân cấp thừa kế.
- Đặc điểm:
  - Không tự động chuyển đổi kiểu
  - Cần phải thực hiện ép kiểu.

Cú pháp của Downcasting:

Subclass obj = (Subclass) superclassObj;

Ví dụ


Person
- name: String
- birthday: Date
+ setName(String)
+ setBirthday(Date)
+ getDetails(): String
Employee
- salary: double
+ setSalary(double)
+ getDetails(): String
Manager
- assistant: Employee
+ setAssistant(Employee)
+ getDetails():String

Ví dụ

```
public class Test1 {
    public static void main(String arg[]) {
        Employee e = new Employee();
        Person p;
        p = e;
        p.setName("Hoa");
        p.setSalary(350000);
        // compile error
    }
}
```

Ví dụ

```
class Manager extends Employee {
      Employee assistant;
      // ...
      public void setAssistant(Employee e) {
            assistant = e;
      }
      // ...
      }
      public class Test2 {
            public static void main(String arg[]) {
                  Manager junior, senior;
                  // ...
                  senior.setAssistant(junior);
            }
      }
```

Ví dụ

```
public class Test3 {
    String static teamInfo(Person p1, Person p2) {
        return "Leader: " + p1.getName() + ", member: "
        + p2.getName();
    }
    public static void main(String arg[]) {
        Employee e1, e2;
        Manager m1, m2;
        // ...
        System.out.println(teamInfo(e1, e2));
        System.out.println(teamInfo(m1, m2));
        System.out.println(teamInfo(m1, e2));
    }
}
```

Ví dụ

```
public class Test2 {
    public static void main(String arg[]) {
        Employee e = new Employee();
        // upcasting - tự động chuyển kiểu
        Person p = e;
        // downcasting - cần có ép kiểu từ Person sang
        // đối tượng Employee
        Employee e1 = (Employee) p;
        Person p2 = new Manager();
        Employee e2 = (Employee) p2;
    }
}
```

Toán tử instanceof

- Toán tử instanceof là một toán tử nhị phân dùng để kiểm tra xem một đối tượng có phải là một thể hiện (instance) của một lớp cụ thể hoặc một lớp con của nó hay không

```
Fruit fruit = new Orange();
if (fruit instanceof Apple) {
  Apple apple = (Apple) fruit;

} else {
  System.out.println("Can't cast");
}
```

Liên kết tĩnh (static binding)

- Liên kết tĩnh trong Java quyết định việc gọi phương thức nào sẽ được thực hiện tại thời điểm biên dịch

```
class Student {
    private void getInfo() {
        System.out.println("Dang lay thong tin
        Student...");
    }
    public static void main(String args[]){
        Student p = new Student();
        p.getInfo();
    }
}
```

Liên kết động (dynamic binding)

- Liên kết động trong Java là quá trình quyết định phương thức nào sẽ được gọi dựa trên loại thực sự của đối tượng tại thời điểm chạy.
- Điều này giúp hỗ trợ tính đa hình, làm cho mã nguồn linh hoạt hơn
- Thích ứng tốt với các tình huống runtime đa dạng.

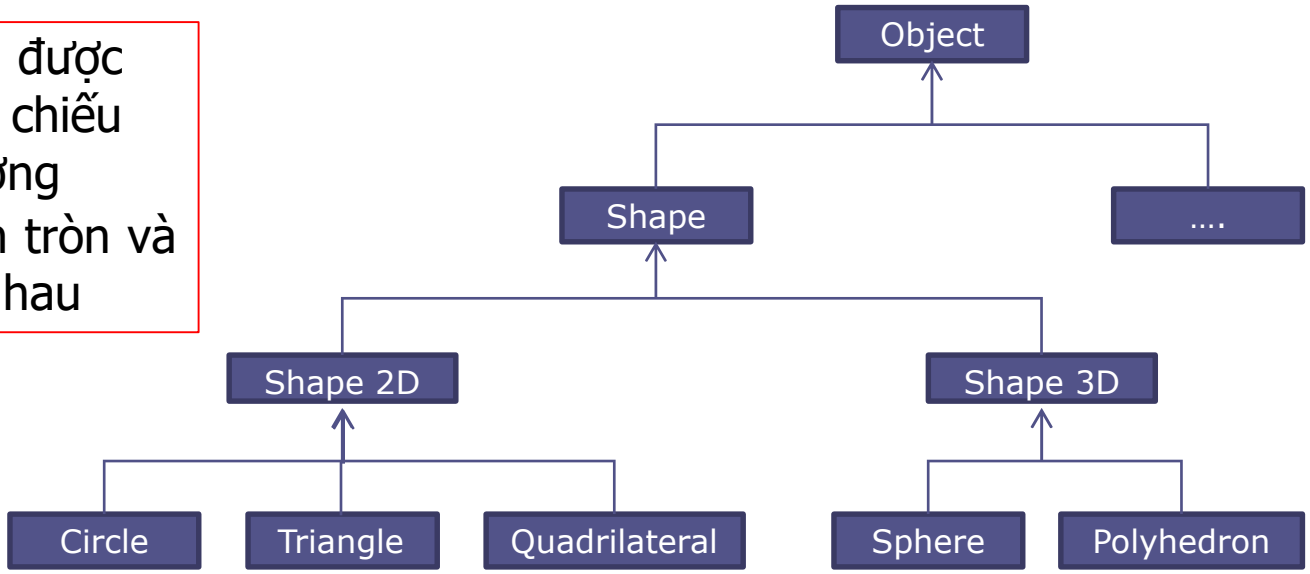
Liên kết động (dynamic binding)

```
class Person {
      private void getInfo() {
            System.out.println("Dang lay thong tin
            Person...");
      }
      class Student extends Person {
            void getInfo(){
                  System.out.println("Dang lay thong tin
                  Student");
            }
      public static void main(String args[]){
            Person p = new Student();
            p.getInfo ();
      }
}
```

Đa hình

- Ví dụ: Một hoạt động có thể được thực hiện trên một đối tượng Shape2D cũng có thể được thực hiện trên một đối tượng thuộc một trong ba lớp Tam giác, Hình tròn, Tứ giác.
  - Lớp cha Shape định nghĩa giao diện chung
  - Các lớp con Tam giác, Vòng tròn, Tứ giác phải theo giao diện này (kế thừa), nhưng cũng được phép cung cấp các triển khai riêng của chúng (ghi đè)

→ Khi một phương thức được yêu cầu thông qua tham chiếu lớp Shape2D, các đối tượng Shape2D, Tam giác, Hình tròn và Tứ giác phản ứng khác nhau



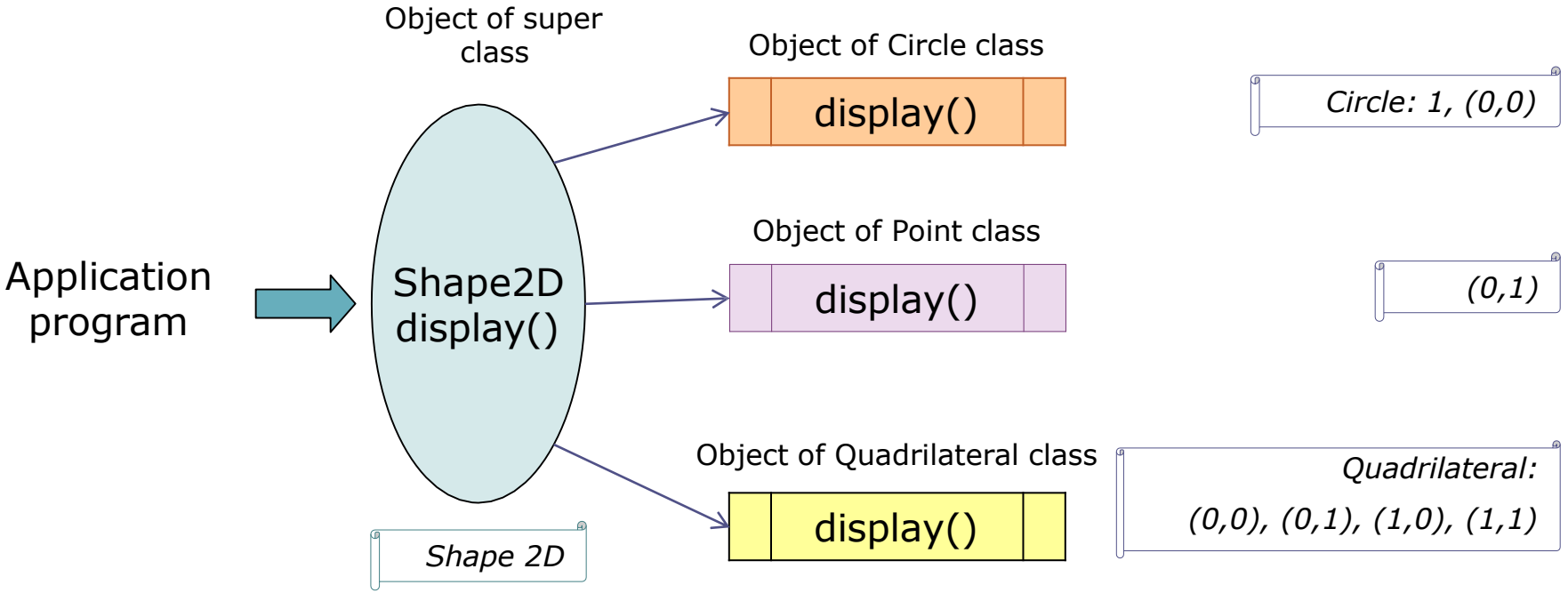
Ví dụ đa hình

```
public class Shape2D {
     public void display() {
        System.out.println("2D Shape");
     }
}

public class Circle extends Shape2D {
     public static final double PI = 3.14;
     private Point p;
     private double r; //radious

     ...
     public void display(){
          System.out.print("Circle: " +  r +
             ",");
          p.display();
          System.out.println();
     }
}
```

```
public class Point extends Shape2D {
     private int x, y;
     ...
     public void display(){
          System.out.print("(" + x + "," + y +
          ")");
     }
}

public class Quadrilateral extends Shape2D {
     private Point p1, p2, p3, p4;
     .....
     public void display(){
          System.out.println("Quadrilateral: ");
          p1.display(); p2.display();
          p3.display(); p4.display();
          System.out.println();
     }
}
```

Đa hình

- Ví dụ: Có nhiều sự lựa chọn một khi một phương thức được gọi thông qua một tham chiếu lớp cha.



Đa hình

- Polymorphism: Nhiều hình thức thực hiện, nhiều kiểu tồn tại
  - Khả năng của một biến tham chiếu thay đổi hành vi theo đối tượng mà nó đang giữ.
- Đa hình trong lập trình
  - Đa hình phương thức:
    - Phương thức trùng tên, phân biệt bởi danh sách tham số.
  - Đa hình đối tượng
    - Nhìn nhận đối tượng theo nhiều kiểu khác nhau
    - Các đối tượng khác nhau cùng đáp ứng chung danh sách các thông điệp có giải nghĩa thông điệp theo cách thức khác nhau.

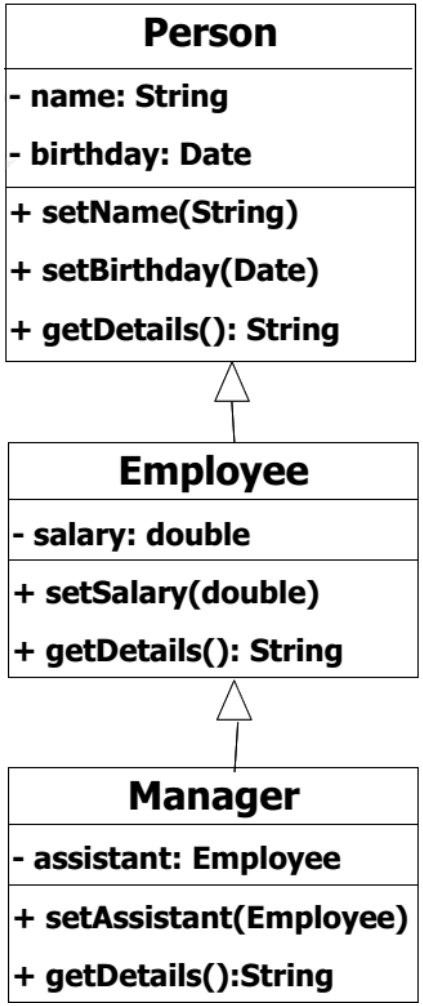
Đa hình

- Polymorphism: gia tăng khả năng tái sử dụng những đoạn mã nguồn được viết một cách tổng quát và có thể thay đổi cách ứng xử một cách linh hoạt tùy theo loại đối tượng
  - 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
- Để thể hiện tính đa hình:
  - Các lớp phải có quan hệ kế thừa với 1 lớp cha nào đó
  - Phương thức được ghi đè (override) ở lớp con

Đa hình

- Ví dụ:
- Các đối tượng khác nhau giải nghĩa các thông điệp theo các cách thức khác nhau
  - Liên kết động (Java)

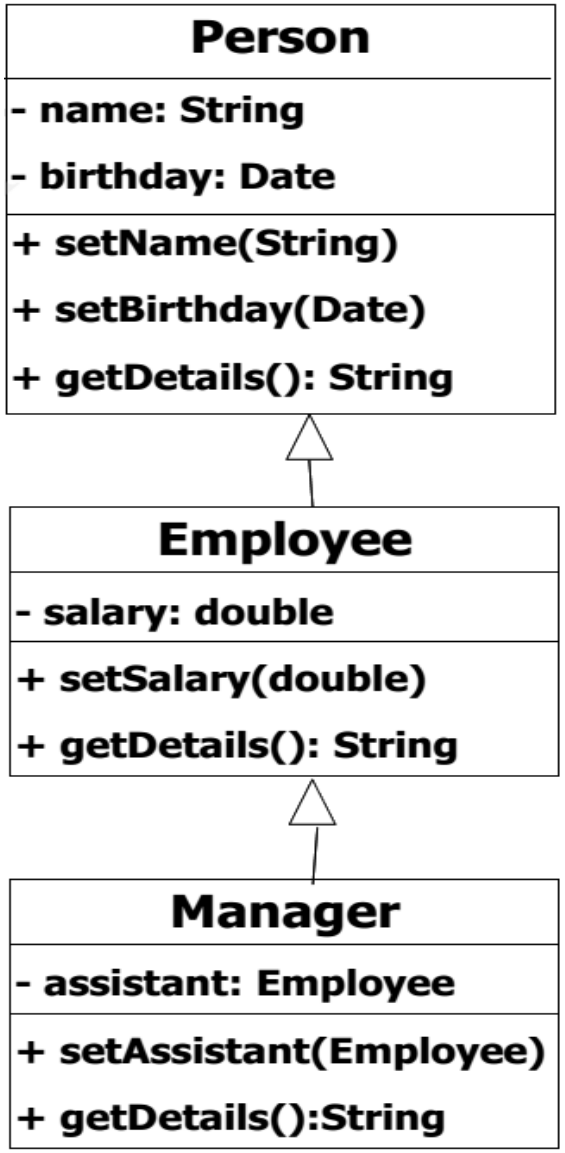
```
Person p1 = new Person();
Person p2 = new Employee();
Person p3 = new Manager();
// ...
System.out.println(p1.getDetail());
System.out.println(p2.getDetail());
System.out.println(p3.getDetail());
```



Đa hình

- Ví dụ:

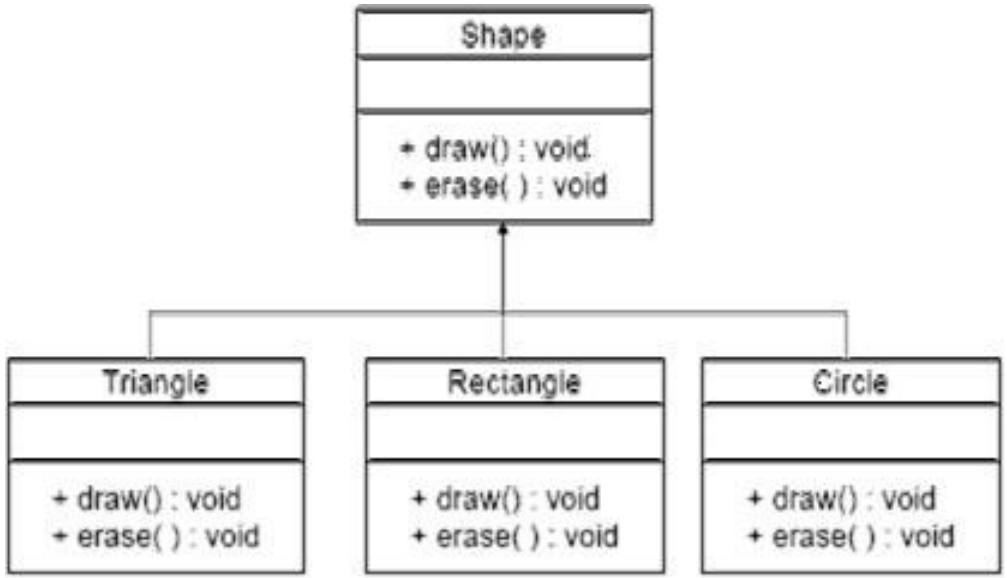
```
class EmployeeList {
     Employee list[];
     ...
     public void add(Employee e) {...}
     public void print() {
        for (int i=0; i<list.length; i++) {
              System.out.println(list[i].
                                    getDetail());
        }
      }
     ...
     EmployeeList list = new EmployeeList();
     Employee e1; Manager m1;
     ...
     list.add(e1);
     list.add(m1);
     list.print();
```



Đa hình

- Ví dụ: Các đối tượng Triangle, Rectangle, Circle đều là các đối tượng Shape

```
...
public static void handleShapes(Shape[] shapes){
  // Vẽ các hình theo cách riêng của mỗi hình
  for( int i = 0; i < shapes.length; ++i) {
    shapes[i].draw();
  }
  ...
  // Gọi đến phương thức xóa
  for( int i = 0; i <
    shapes.length(); ++i) {
       shapes[i].erase();
  }
}
...
```



ĐẠI NAM
UNIVERSITY

Xin cảm ơn!

TRẦN QÚY NAM

Giảng viên Khoa CNTT
namtq@dainam.edu.vn