LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

NỘI DUNG TRONG CHƯƠNG

- *Lập trình hướng đối tượng: khái niệm, tính chất của lập trình hướng đối tượng*
- *Một số khái niệm cơ bản về lớp, đối tượng, thuộc tính*
- *Phương thức khởi tạo*
- *Phương thức Getter và Setter*
- *So sánh hai đối tượng*
- *Mảng đối tượng*
- *Một số lớp sẵn có*
- *Tính kế thừa, tính đa hình, tính trừu tượng trong lập trình hướng đối tượng*

Chương 5 đề cập đến một nội dung quan trọng trong lập trình Java là lập trình hướng đối tượng. Chương này sẽ trình bày các khái niệm cơ bản nhất về lập trình hướng đối tượng như lớp, thuộc tính, thể hiện, phương thức khởi tạo. Các tính chất của lập trình hướng đối tượng như tính kế thừa, tính trừu tượng, tính đa hình hay tính đóng gói cũng được trình bày để người đọc hiểu rõ được cách thức triển khai hướng đối tượng trong Java được thực hiện như thế nào.

KHÁI NIỆM HƯỚNG ĐỐI TƯỢNG (OOP)

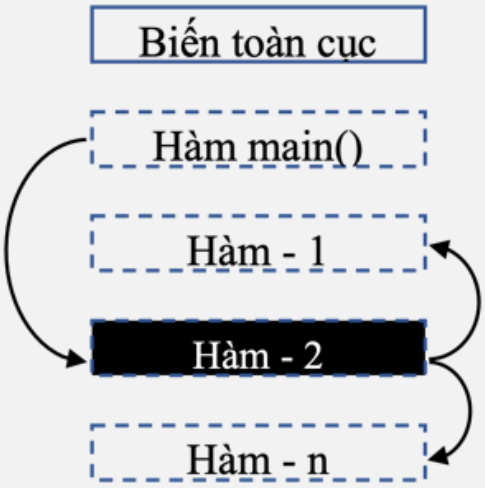
Lập trình hướng đối tượng là gì

Lập trình hướng đối tượng (**OOP** - **Object Orientted Programming**) là một phương pháp phát triển ứng dụng dựa trên khái niệm về lớp và đối tượng:

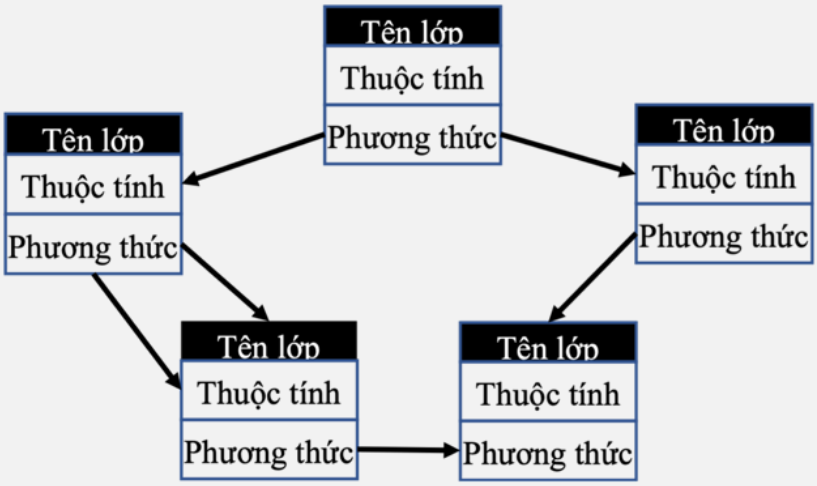
- **Đối tượng**: Một thực thể có trạng thái và hành vi. Ví dụ như xe đạp, bàn, ghế,... Nó có thể mang tính vật lý hoặc logic.
- **Lớp**: Một tập hợp các đối tượng. Nó là một thực thể logic.

Hình 5-1 Các đối tượng được tạo ra từ một khuôn mẫu là lớp car

Lập trình hướng đối tượng và hướng thủ tục có nhiều điểm khác nhau:



Hình 5-2 Lập trình hướng thủ tục



Hình 5-3 Lập trình hướng đối tượng

Lợi thế của OOP khi so với ngôn ngữ **lập trình hướng thủ tục** (procedure-oriented) đó là OOP giúp việc thiết kế, phát triển và bảo trì dễ dàng hơn trong khi với lập trình hướng thủ tục thì việc quản lý mã nguồn là khá khó khăn, đặc biệt là đối với các dự án lớn. Điều này làm tăng hiệu quả cho quá trình phát triển phần mềm. Ngoài ra, OOP cung cấp khả năng để mô phỏng các sự kiện trong thế giới thực từ đó khiến cho việc phát triển phần mềm được dễ dàng hơn.

	Hướng thủ tục	Hướng đối tượng
Hướng tiếp cận	Từ trên xuống	Từ dưới lên
Chia chương trình	Thành các hàm	Thành các lớp
Chế độ truy cập thực thể	Không có đặc tả truy cập.	Xác định truy cập bằng "public", "private", "protected" và "default"
Chồng phương thức và đa hình	Không có	Có chồng hàm, có tính đa hình
Thừa kế	Không hỗ trợ thừa kế	Thừa kế được hỗ trợ ở ba trạng thái "public", "private", "protected"
Chia sẻ dữ liệu	Dữ liệu được chia sẻ giữa các phương thức trong chương trình	Dữ liệu được chia sẻ giữa các đối tượng nếu được phép
Các ngôn ngữ thường sử dụng	TC, VB, Pascal...	C ++, JAVA, C #, Ruby...

Các tính chất của OOP

Hình 5-4 Minh họa các đối tượng trên một trò chơi điện tử

- **Tính kế thừa**: Khi một đối tượng đạt được các thuộc tính và các hành vi của đối tượng cha, thì đó là tính kế thừa. Điều này làm tăng tính tái sử dụng cho code.
- **Tính đa hình**: Tính đa hình thể hiện một đối tượng có thể ở nhiều hình thái/dạng (form) khác nhau, từ đó thực hiện các hành vi khác nhau.
- **Tính trừu tượng**: Tính trừu tượng giúp loại bỏ những thứ phức tạp, không cần thiết của đối tượng nó chỉ tập trung vào khả năng thực hiện, chứ không quan tâm đến cách thức cụ thể công việc đó được thực hiện như nào.
- **Tính đóng gói**: Là việc liên kết các đoạn mã vào trong một đơn vị đơn (Lớp), việc đóng gói ngăn chặn việc truy cập từ bên ngoài.

Một số khái niệm về lớp, đối tượng, thuộc tính và phương thức

- **Lớp** (Class) : là nơi định nghĩa thông tin về các đối tượng. Cũng có thể hiểu lớp là một kiểu dữ liệu mà biến của kiểu dữ liệu này được coi như một đối tượng.
- **Phương thức** (Method): là hành vi (hành động) của đối tượng

Xem đoạn chương trình sau, trong đó định nghĩa một lớp Student với từ khóa class:

```
1  class Student {
2     String name;
3     int age;
4     public Student(String name, int age) {
5         this.name = name;
6         this.age = age;
7     }
8     public void display() {
9         System.out.println("Name: " + name);
10        System.out.println("Age: " + age);
11    }
12    public static void main(String[] args) {
```

```
        Student s1 = new Student("Linh", 24);
        s1.display();
    }
}
```

Kết quả:

```
Name: Linh
Age: 24
```

Cú pháp để khai báo một lớp như sau:

```
class <tên lớp> {
    // Khai báo danh sách các thuộc tính
    <kiểu_dữ_liệu> <tên thuộc tính>;
    ...
    // Khai báo danh sách các phương thức
    public <kiểu trả về> <tên phương thức>(<kiểu tham số> <tên tham số>, ...) {
            ...
        }
}
```

Tạo lớp Student với hai thuộc tính là name (tên), age (tuổi), hai phương thức getInformation() và display() như bảng sau:

Hình 5-5 Khai báo các thuộc tính và phương thức cho đối tượng Student

```
class Student {
    String name;
    int age;
    public void getInformation() {
    }
    public void display() {
    }
}
```

Dùng từ khóa **new** để khai báo đối tượng của một lớp. Khai báo một đối tượng thuộc lớp **Student**:

```
Student s1 = new Student("Linh", 24);
s1.display();
```

Dùng toán tử **"."** để truy xuất tới các thuộc tính và phương thức của một đối tượng.

```
public class Program {
    public static void main(String[] args) {
        Circle c = new Circle(7);
        System.out.println(c.getArea());
        System.out.println(c.getCircumference());
    }
}
class Circle {
    private double radius;
```

```
    public Circle(double radius) {
        this.radius = radius;
    }
    public double getArea() {
        return 3.14 * radius * radius;
    }
    public double getCircumference() {
        return 3.14 * 2 * radius;
    }
}
```

Kết quả:

```
153.86
43.96
```

SO SÁNH HAI ĐỐI TƯỢNG

Xét đối tượng Student:

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

Lớp Chương trình với phương thức **main**:

```
public class Program {
    public static void main(String[] args) {
        Student s1 = new Student("Viet", "Bac Ninh");
        Student s2 = new Student("Viet", "Bac Ninh");
        System.out.println(s1 == s2);
    }
}
```

Kết quả chạy là:

```
False
```

Hai đối tượng **s1** và **s2** được khai báo và khởi tạo giống nhau, cùng tên và địa chỉ nhưng kết quả **s1 == s2** là **false**. Do toán tử == trong java là toán tử so sánh địa chỉ nơi đối tượng được cấp phát. Toán tử này không so sánh các thuộc tính của đối tượng.

Để so sánh giá trị phải tự viết phương thức so sánh hoặc sử dụng phương thức **equals()** mà lớp **Object** đã cung cấp sẵn và cần ghi đè lại phương thức như chương trình sau:

```java
public class Student {
    public String name;
    public String address;
    public Student(String name, String address) {
        this.name = name;
        this.address = address;
    }
    @Override
    public boolean equals(Object obj) {
        return this.name == ((Student) obj).name && this.address == ((Student) obj).address;
    }
}
```

Lớp Chương trình với phương thức main:

```java
public class Program {
    public static void main(String[] args) {
        Student s1 = new Student("Viet", "Bac Ninh");
        Student s2 = new Student("Viet", "Bac Ninh");
        System.out.println(s1.equals(s2));
    }
}
```

Kết quả:

```
True
```

CHỈ ĐỊNH TRUY CẬP

Chỉ định truy cập (**Access modifie**r) là các "từ" dùng trước các khai báo của một class, biến hay method (phương thức) để thể hiện khả năng truy cập của class, biến hay method đó ở các class khác.

Với class ta có 2 loại Access modifier là public và default nhưng với biến và method (phương thức) thì ta có 4 access modifier (public, protected, default, private).

- **default** (khi không khai báo Access Modifier): Truy cập trong nội bộ package.
- **private:** Truy cập trong nội bộ lớp.
- **public**: Thành phần công khai, truy cập tự do từ bên ngoài.
- **protected**: Thành phần được bảo vệ, bị hạn chế truy nhập từ bên ngoài.

Bảng tổng hợp:

Bảng 5-1 Bảng tổng hợp các chỉ định truy cập

Chỉ định truy cập	Cùng lớp	Cùng package	Lớp con ngoài package	Lớp khác ngoài package
private	Có	Không	Không	Không

default	Có	Có	Không	Không
protected	Có	Có	Có	K
public	Có	Có	Có	Có

PHƯƠNG THỨC GETTER VÀ PHƯƠNG THỨC SETTER

Các thuộc tính của lớp nếu là public thì có thể được truy xuất trực tiếp từ nơi khác. Nhằm hạn chế tối đa các biến có phạm vi truy cập public, bảo mật các thông tin cần thiết bên trong đối tượng, cần để phạm vi truy xuất của các thuộc tính là private và truy xuất tới các thuộc tính này thông qua các phương thức public (phương thức setter và getter).

- Phương thức **getter** được sử dụng để lấy giá trị của một thuộc tính (thường là private)
- Phương thức **setter** được sử dụng để gán giá trị cho một thuộc tính (thường là private)

Phương thức Getter

Phương thức **getter** là phương thức cung cấp "**quyền truy cập**" đọc cho một biến.

```
1  public class Student {
2      public String name;
3      public int age;
4      public double gpa;
5  }
```

Phương thức getter thường có tên theo quy tắc "getPropertyName" với "PropertyName" là tên của thuộc tính mà nó đang truy xuất.

```
1  public String getName(){
2      return name;
3  }
```

Phương thức Setter

Phương thức **setter** cung cấp "**quyền ghi**" vào một biến private, cho phép các lớp khác có thể gán một giá trị mới cho biến.

Phương thức setter thường có tên theo quy tắc "setPropertyName" với "PropertyName" là tên của thuộc tính mà nó đang cập nhật

```
1  public void setName(String name){
2      this.name = name;
3  }
```

PHƯƠNG THỨC KHỞI TẠO (CONSTRUCTOR)

Phương thức khởi tạo là gì?

- Phương thức khởi tạo (**constructor**) là một phương thức đặc biệt, phương thức này sẽ tự động đ+ược gọi khi khởi tạo một đối tượng
- **Constructor** có đặc điểm là không được định nghĩa kiểu trả về và có tên trùng với tên lớp
- Với các phương thức thông thường thì cần dùng toán tử **"."** để gọi tới. Gọi phương thức khởi tạo thì sử dụng toán tử new

Các đối tượng được tạo bằng toán tử **new**. Ví dụ một chương trình khởi tạo đối tượng từ lớp Student:

```
1  Student s1; //Khai báo một biến kiểu Student
2  s1 = new Student(); //Xây dựng một đối tượng mới và lưu trữ một tham chiếu đến nó.
```

Trong ví dụ này, "new Student" là một biểu thức cấp phát bộ nhớ cho đối tượng, khởi tạo biến thể hiện của đối tượng, sau đó trả về một tham chiếu đến đối tượng. Sau khi câu lệnh gán thực hiện, s1 tham chiếu đến đối tượng mới được tạo. Một phần của biểu thức này "Student()" giống như một lệnh gọi phương thức. Trên thực tế, đó là lời gọi đến phương thức đặc biệt gọi là phương thức khởi tạo. Mọi lớp đều có ít nhất một phương thức khởi tạo. Nếu người lập trình không viết định nghĩa trong một lớp, thì hệ thống sẽ cung cấp một phương thức khởi tạo mặc định cho lớp đó.

Quy tắc tạo phương thức khởi tạo:

- Tên phương thức khởi tạo phải **giống** với tên class
- Một phương thức khởi tạo **không có kiểu trả về**
- Một phương thức khởi tạo Java không thể **abstract**, **static**, **final** và **synchronized**
- Phương thức khởi tạo vẫn có thể khai báo quyền truy cập **public**, **private** và **protected**
- Đặc biệt, một phương thức khởi tạo không thể được khai báo là phương thức tĩnh **static**
- Phương thức khởi tạo trong Java được chia làm hai loại là **không tham số** và **có tham số**

Phương thức khởi tạo không tham số

Phương thức khởi tạo không tham số là phương thức sẽ được tự động tạo ra đối với mỗi lớp. Ví dụ phương thức Student():

```
1  public class Student{
2     String name;
3     int age;
4     public Student(){
5     }
6  }
```

Phương thức khởi tạo không tham số không cần viết, hệ thống sẽ tự động tạo ra khi lớp đó không có phương thức khởi tạo nào.

Phương thức khởi tạo có tham số

Phương thức khởi tạo có tham số là phương thức sẽ truyền thêm các giá trị vào trong quá trình khởi tạo đối tượng. Ví dụ:

```
public class Student{
    String name;
    int age;
    public Student(String name, int age) {
        this.name = name;
        this.age = age;
    }
}
```

Lưu ý khi đã khai báo một phương thức khởi tạo, thì phương thức khởi tạo mặc định sẽ tự động biến mất. Nếu vẫn muốn sử dụng thì phải khai báo lại phương thức khởi tạo này.

MẢNG ĐỐI TƯỢNG

Việc khai báo và sử dụng mảng các đối tượng không giống với khai báo và sử dụng mảng các kiểu dữ liệu cơ sở: **int**, **double**. Để sử dụng được các phần tử trong mảng cần phải khởi tạo cho từng phần tử.

Ví dụ Student là lớp được định nghĩa có mảng kiểu Student[]. Với mảng kiểu Student[], mỗi phần tử của mảng là một biến kiểu Student. Để lưu trữ thông tin cho 30 sinh viên, sử dụng một mảng:

```
public static void main(String[] args) {
    // Khai báo mảng students vớii 30 phần tử
    Student[] students = new Student[30];
    for (int i = 0; i < 30; i++) {
        // Khởi tạo các phần tử trong mảng students
        students[i] = new Student();
    }
}
```

Mảng có 30 phần tử: students[0], students[1], ...students[29]. Khi thực hiện xong, mỗi student[i] trỏ đến một đối tượng kiểu Student.

```
public class Student {
    String name;
    int age;
    public Student(String name, int age) {
        this.name = name;
        this.age = age;
    }
    public void display() {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
    }
}
```

Lớp Program:

```
public class Program {
    public static void main(String[] args) {
        // Khai báo mảng students với 30 phần tử
        Student[] students = new Student[3];
        students[0] = new Student("Thai", 12);
        students[1] = new Student("Tuan", 13);
        students[2] = new Student("Duc", 11);
        for (int i = 0; i < 3; i++) {
            students[i].display();
        }
    }
}
```

Kết quả là:

```
Name:Thai
Age: 12
Name:Tuan
Age: 13
Name: Duc
Age: 11
```

MỘT SỐ LỚP SẴN CÓ

Lớp java.lang.StringBuilder

java.lang.StringBuilder (Lớp StringBuilder nằm trong gói tiêu chuẩn java.lang) giúp xây dựng một chuỗi dài từ phần nhỏ hơn một cách hiệu quả. Đầu tiên cần tạo ra một đối tượng thuộc StringBuilder.Lớp

```
StringBuilder builder =  new StringBuilder();
```

Giống như một chuỗi, một **StringBuilder** chứa một chuỗi các ký tự. Một chuỗi dài có thể được tạo trong StringBuilder bằng cách sử dụng một chuỗi append(). Khi hoàn tất phương thức builder.toString() sẽ trả về một bản sao của chuỗi trong trình tạo dưới dạng giá trị bình thường kiểu String.

Ví dụ so sánh thời gian ghép chuỗi:

```
public class Program {
    private static final String STR = "A";
    public static String stringConcatenation(){
        String s = STR;
        for (int i = 0; i < 100000; i++)
            s = s + STR;
        return s;
    }
    public static String stringBufferConcatenation(){
        StringBuilder sb = new StringBuilder(STR);
        for (int i = 0; i < 100000; i++)
            sb.append(STR);
        return sb.toString();
    }
    public static void main(String[] args) {
```

```
16          long start = System.currentTimeMillis();
17          stringConcatenation();
18          long end = System.currentTimeMillis();
19          System.out.println("Thời gian dùng để ghép chuỗi String: " + (end - start) +
"ms");
20          start = System.currentTimeMillis();
21          stringBufferConcatenation();
22          end = System.currentTimeMillis();
23          System.out.println("Thời gian dùng để ghép chuỗi dùng StringBuilder: "+ (end
- start) + "ms");
24      }
25  }
```

Kết quả:

```
1  Thời gian dùng để ghép chuỗi String: 3542ms
2  Thời gian dùng để ghép chuỗi dùng StringBuilder: 2ms
```

Lớp java.util.Date

java.util.Date được sử dụng để biểu thị thời gian. Khi một đối tượng Date được xây dựng không có tham số, kết quả sẽ đại diện cho ngày và giờ hiện tại:

```
1  public static void main(String[] args) {
2      System.out.println(new Date());
3  }
```

Kết quả hiển thị như sau:

```
Sun Sep 12 07:58:44 ICT 2021
```

Lớp java.lang.Math

Lớp **java.lang.Math** chứa nhiều các phương thức tĩnh để thực hiện các phương thức toán học, ví dụ: sin, cos, tan, các phương thức làm tròn, các phương thức sinh số ngẫu nhiên. Ví dụ:

```
double x = Math.random()
```

Ví dụ: Tạo class PairOfDice, đối tượng của lớp này đại diện cho một cặp xúc xắc chứa hai biến toàn cục để hiển thị các số trên xúc xắc. Khi một PairOfDice được tạo, viên xúc xắc được khởi tạo thành các giá trị ngẫu nhiên.

```
1  public class PairOfDice {
2      public int dice1, dice2;
3      public PairOfDice() {
4          roll();
5      }
6      public void roll() {
7          dice1 = (int) (Math.random() * 6) + 1;
8          dice2 = (int) (Math.random() * 6) + 1;
```

```
 9         System.out.println("Dice 1 = " + dice1);
10         System.out.println("Dice 2 = " + dice2);
11     }
12     public static void main(String[] args) {
13         PairOfDice pairOfDice = new PairOfDice();
14     }
15 }
```

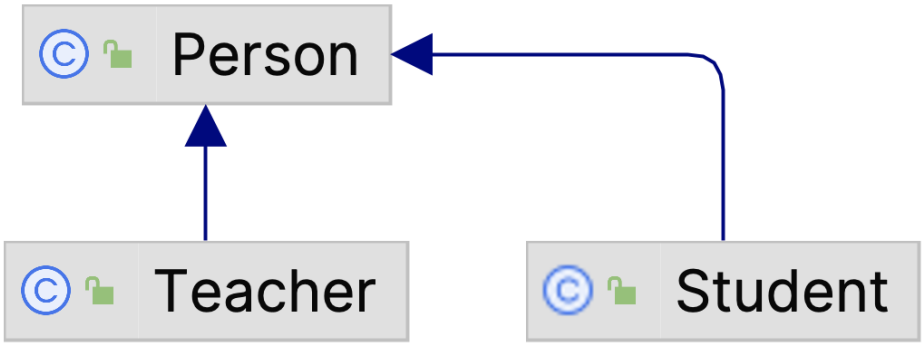
Kết quả hiển thị:

```
1 Dice 1 = 5
2 Dice 2 = 6
```

Chú ý cũng có thể sử dụng lớp java.util.Random để tạo ra đối tượng Random, từ đó tạo ra các số ngẫu nhiên

TÍNH KẾ THỪA

Trong lập trình hướng đối tượng, Kế thừa là thừa hưởng lại những thuộc tính và phương thức của một lớp. Lớp được thừa hưởng những thuộc tính và phương thức từ lớp khác được gọi là **lớp dẫn xuất** hay lớp con và lớp bị lớp khác kế thừa gọi là **lớp cơ sở** hoặc lớp cha.



Bảng 5-2 Lớp Teacher và Student cùng kế thừa từ lớp Person

Trong Java để kế thừa một lớp, dùng từ khoá **extends** .

```
1 class B extends A{
2 //Bổ sung và sửa đổi
3 //Nội dung được kế thừa từ lớp A
4 }
```

Ví dụ:

```
1 public class Student extends Person {
2 }
```

```
1 public class Teacher extends Person {
2 }
```

Lớp **Student** và **Teacher** thừa hưởng các thuộc tính chung từ lớp Person. Lớp Person là lớp cha (lớp cơ sở), lớp Teacher và Student là hai lớp con (lớp dẫn xuất).

Lớp con không được thừa hưởng các thuộc tính và phương thức **private** từ lớp cha. Trong kế thừa, các lớp con chỉ có thể thừa hưởng được các thuộc tính và phương thức có phạm vi truy cập public, protected và default (trong trường hợp lớp con và lớp cha cùng package). Để truy xuất tới thuộc tính private của lớp cha từ lớp con phải thông qua phương thức setter và getter. Một lớp sẽ không thể được kế thừa nếu như nó được khai báo với từ khóa **final**

GHI ĐÈ PHƯƠNG THỨC

Trong kế thừa, có thể ghi đè lại phương thức của lớp cha như đoạn chương trình sau :

```
class SuperClass{
    public void display() {
            System.out.println("Hello from SuperClass");
    }
}
class SubClass extends SuperClass{
    @Override
    public void display() {
            System.out.println("Hello from SubClass");
    }
}
class Entry {
    public static void main(String[] args) {
            SubClass s = new SubClass();
            s.display();
    }
}
```

Kết quả hiển thị:

```
Hello from SubClass
```

Phương thức **display()** của lớp **SuperClass** đã bị ghi đè bởi phương thức **display()** trong lớp **SubClass**. Trong ví dụ trên **@Override** là một annotation để chú thích rằng **display()** là một phương thức ghi đè phương thức từ lớp cha.

Việc sử dụng **@Override** annotation là không bắt buộc nhưng trong mọi trường hợp nên dùng vì

- Nếu một phương thức được chú thích với annotation **@Override** thì chương trình sẽ kiểm tra xem phương thức này có thực sự ghi đè phương thức của lớp cha không, nếu không thì sẽ báo lỗi.
- Giúp người khác khi nhìn vào code sẽ hiểu được đây là phương thức được ghi đè từ lớp cha.
- Một phương thức được khai báo kèm **final** là một phương thức không cho ghi đè

LỚP JAVA.LANG.OBJECT

Mọi lớp trong Java đều kế thừa từ lớp **java.lang.Object**. Nếu tạo một lớp và không khai báo nó thuộc lớp con của một lớp, thì lớp đó sẽ tự động trở thành lớp con của lớp **java.lang.Object.**

Các phương thức của lớp Object:

Phương thức	Mô tả
public final Class **getClass()**	Trả về đối tượng Class chứa thông tin về lớp của đối tượng đó.
public int **hashCode()**	Trả về mã băm (hashcode) của đối tượng.
public boolean **equals**(Object obj)	So sánh hai đối tượng
protected Object **clone**() throws CloneNotSupportedException	Tạo và trả về một bản sao của đối tượng
public String **toString**()	Trả về chuỗi mô tả đối tượng
public final void **notify**()	Đánh thức một Tiểu trình.
public final void **notifyAll**()	Đánh thức tất cả các tiểu trình
public final void **wait**(long timeout) throws InterruptedException public final void **wait**() throws InterruptedException	"Ngủ" chờ một tiến trình khác đánh thức hoặc sau một khoảng thời gian thì tự "dậy"
protected void **finalize**() throws Throwable	Hàm sẽ được gọi trước khi đối tượng bị hủy bởi Bộ dọn rác (garbage collector)

Tạo lớp với phương thức **toString**() mới cho lớp đó, phương thức này thay thế phiên bản kế thừa. Ví dụ thêm phương thức vào lớp **PairOfDice** từ phần trước:

```
1  public class PairOfDice {
2     public int dice1, dice2;
3     public PairOfDice() {
4         roll();
5     }
6     public String toString () {
7         if (dice1 == dice2)
8             return "Double " + dice1;
9         else
10            return  dice1 + " and " + dice2;
11    }
12    public void roll() {
13        dice1 = (int) (Math.random() * 6) + 1;
14        dice2 = (int) (Math.random() * 6) + 1;
15    }
16    public static void main(String[] args) {
17        PairOfDice pairOfDice = new PairOfDice();
18        System.out.println(pairOfDice.toString());
19    }
20 }
```

ĐỐI TƯỢNG THIS VÀ ĐỐI TƯỢNG SUPER

Đối tượng this

this từ khóa rất quan trọng trong Java dùng để tham chiếu ngược lại đến đối tượng, giúp truy xuất đến các giá trị của đối tượng đó.

Sử dụng từ khóa **this** để phân biệt đâu là biến và đâu là thuộc tính của lớp khi chúng trùng tên với nhau. Điều này thể hiện rõ ràng nhất thông qua phương thức khởi tạo (**constructor** method)

```
1  public class Circle {
2     private double radius;
3     public Circle(double radius) {
4         radius = radius;
5     }
6     public static void main(String[] args) {
7         Circle circle = new Circle(3.12);// Khởi tạo một Circle có radius = 4;
8         System.out.println("radius = " + circle.radius);
9     }
10 }
```

Kết quả hiển thị:

```
radius = 0
```

Khởi tạo một phương thức khởi tạo (constructor) có tham số **radius** để khởi tạo bán kính cho hình tròn nhưng tham số này lại trùng tên với thuộc tính **radius** trong lớp Circle. phương thức **main**() sẽ khởi tạo đối tượng của lớp Circle với radius bằng 3.12 và in giá trị của radius ra ngoài màn hình.

Giá trị nhận được là 0 (giá trị mặc định biến khi khởi tạo). Lý do vì trình biên dịch không phân biệt được đâu là tham số và đâu là thuộc tính của lớp. Vì vậy cần dùng một this để phân biệt tham số và thuộc tính của class.

Chạy chương trình sau:

```
1  public class Circle {
2     private double radius;
3     public Circle(double radius) {
4         this.radius = radius;
5     }
6     public static void main(String[] args) {
7         Circle circle = new Circle(3.12);// Khởi tạo một Circle có radius = 4;
8         System.out.println("radius = " + circle.radius);
9     }
10 }
```

Kết quả chạy:

```
radius = 3.12
```

Đối tượng super

super được dùng để loại bỏ sự nhầm lẫn giữa lớp cha và lớp con có các phương thức trùng tên. Khi một lớp con chứa một phương thức thể hiện với một phương thức trong lớp cha của nó, phương thức từ lớp cha sẽ bị ẩn theo cùng một cách. Khi đó phương thức trong lớp con sẽ ghi đè phương thức từ lớp cha. Tuy nhiên, **super** có thể được sử dụng để truy cập phương thức từ lớp cha.

Xem xét lớp Animal:

```
class Animal { // Superclass (parent)
    public void animalSound() {
        System.out.println("The animal makes a sound");
    }
}
class Dog extends Animal { // Subclass (child)
    public void animalSound() {
        super.animalSound(); // Call the superclass method
        System.out.println("The dog says: bow wow");
    }
}
```

Chương trình chính:

```
public static void main(String[] args) {
    Animal myDog = new Dog(); // Create a Dog object
    myDog.animalSound(); // Call the method on the Dog object
}
```

Kết quả chạy:

```
The animal makes a sound
The dog says: bow wow
```

GIAO DIỆN

Giao diện (**interface**) được dùng để lưu trữ các phương thức trừu tượng và các biến hằng số. Một số đặc điểm của interface:

- Giao diện được tạo ra bởi từ khóa **interface.**
- Giống với lớp trừu tượng, ta không thể khởi tạo được đối tượng của interface.
- Interface chỉ được sử dụng để triển khai (**implements)** ra các lớp con hoặc giao diện con.
- Tất cả các phương thức trong interface phải là phương thức **abstract** (trừu tượng) hoặc phương thức **default** (mặc định).
- Tất cả các biến trong interface đều được trình biên dịch hiểu là các hằng số.
- Từ khóa abstract có thể không cần khi khai báo phương thức trừu tượng, tương tự từ khóa static final không cần khi khai báo hằng số.

Ví dụ, nếu khai báo **interface** IAnimal như sau:

```
public interface IAnimal {
    default void move() {
        System.out.println("Moving");
    }
    void sound();
    int numberOfAnimal = 0;
}
```

Một lớp có thể kế thừa nhiều interface. Vì Java là ngôn ngữ được thiết kế với mục đích đơn giản nên không hỗ trợ đa kế thừa với class, nhưng do bản chất interface chỉ chứa các phương thức rỗng nên Java cho phép một lớp triển khai từ nhiều interface.

```
interface IFlyable {
    void fly();
}
interface IEatable {
    void eat();
}
class Bird implements IFlyable, IEatable {
    @Override
    public void fly() {
        System.out.println("Bird flying");
    }
    @Override
    public void eat() {
        System.out.println("Bird eats");
    }
}
```

Chương trình chính:

```
public class Program {
    public static void main(String[] args) {
        Bird bird = new Bird();
        bird.eat();
        bird.fly();
    }
}
```

Kết quả khi chạy chương trình:

```
Bird eats
Bird flying
```

Lưu ý: Để triển khai interface dùng từ khóa **implements** thay vì **extends**, Khi nhìn vào interface thì thứ duy nhất mà bạn nhìn thấy đó là các phương thức trừu tượng (các tính năng), do đó sử dụng interface được coi là trừu tượng hoàn toàn. Interface thường được đặt tên với chữ cái I viết hoa ở đầu, theo sau thường là một danh từ hoặc tính từ.

TÍNH ĐA HÌNH

Tính đa hình (Polymorphism) là một đối tượng có thể thực hiện một tác vụ theo nhiều cách khác nhau

Ví dụ lớp **Animal:**

```
class Animal {
    public void sound() {
        System.out.println("some sound");
    }
}
//Lớp Dog kế thừa Animal
class Dog extends Animal {
    public void sound() {
        System.out.println("bow bow");
    }
}
//Lớp Cat kế thừa Animal
class Cat extends Animal {
    public void sound() {
        System.out.println("meow meow");
    }
}
//Lớp Duck kế thừa Animal
class Duck extends Animal {
    public void sound() {
        System.out.println("quack quack");
    }
}
```

Chương trình chính:

```
public class Program {
    public static void main(String[] args) {
        Animal animal = new Animal();
        animal.sound();
        animal = new Dog();
        animal.sound();
        animal = new Cat();
        animal.sound();
        animal = new Duck();
        animal.sound();
    }
}
```

Kết quả khi chạy:

```
some sound
bow bow
meow meow
quack quack
```

Ở đây có thể nói đối tượng Animal tồn tại ở nhiều hình thái khác nhau và ở mỗi hình thái, đối tượng lại có những thể hiện khác nhau.

Để kiểm tra một đối tượng có phải thể hiện của một lớp nào đó không, sử dụng từ khóa **instanceof** . Ví dụ kiểm tra animal có phải là một đối tượng của lớp Dog:

```
1  public class Program {
2     public static void main(String[] args) {
3          Animal animal = new Animal();
4          animal.sound();
5          animal = new Dog();
6          if(animal instanceof Dog){
7            System.out.println("animal is instance of Dog class")
8          }
9     }
10 }
```

TÍNH TRỪU TƯỢNG

Tính trừu tượng là một tính chất mà chỉ tập trung vào những tính năng của đối tượng và ẩn đi những thông tin không cần thiết. Để thực hiện tính trừu tượng trong Java có thể sử dụng lớp trừu tượng (abstract class) và interface.

Một lớp được khai báo là lớp trừu tượng bằng từ khóa **abstract** sẽ không thể khởi tạo được đối tượng của lớp đó mà chỉ có thể sử dụng để kế thừa.

```
1  abstract class Animal{
2  }
3  class Cat extends Animal{
4  }
```

Chương trình trên sẽ không báo lỗi nhưng nếu thay **Animal a = new Cat()** thành **Animal a = new Animal()** thì chương trình sẽ báo lỗi (do không thể khởi tạo được đối tượng của lớp trừu tượng). Lớp trừu tượng có thể có các phương thức trừu tượng. phương thức trừu tượng là phương thức mà chỉ có phần khai báo, không có phần thân.

```
1  abstract class Animal{
2     public abstract void makeSound();
3  }
```

Nếu một lớp được kế thừa từ lớp trừu tượng thì lớp đó phải ghi đè tất cả các phương thức trừu tượng.

```
1  class Cat extends Animal {
2     @Override
3     public void makeSound() {
4         System.out.println("meow meow");
5     }
6  }
```

Khi đó đối tượng **Cat** có thể được tạo và gọi phương thức **makeSound**()

```
public class Program {
    public static void main(String[] args) {
        Animal a = new Cat();
        a.makeSound();
    }
}
```

Kết quả

```
meow meow
```

THỰC HÀNH

Tính kế thừa

Tạo lớp Person

```
public class Person {
    private int id;
    private String name;
    private int age;
    private String address;
    public Person(int id, String name, int age, String address) {
        this.id = id;
        this.name = name;
        this.age = age;
        this.address = address;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public String getAddress() {
        return address;
    }
    public void setAddress(String address) {
        this.address = address;
```

```
    }
}
```

Tạo lớp Student kế thừa từ lớp Person:

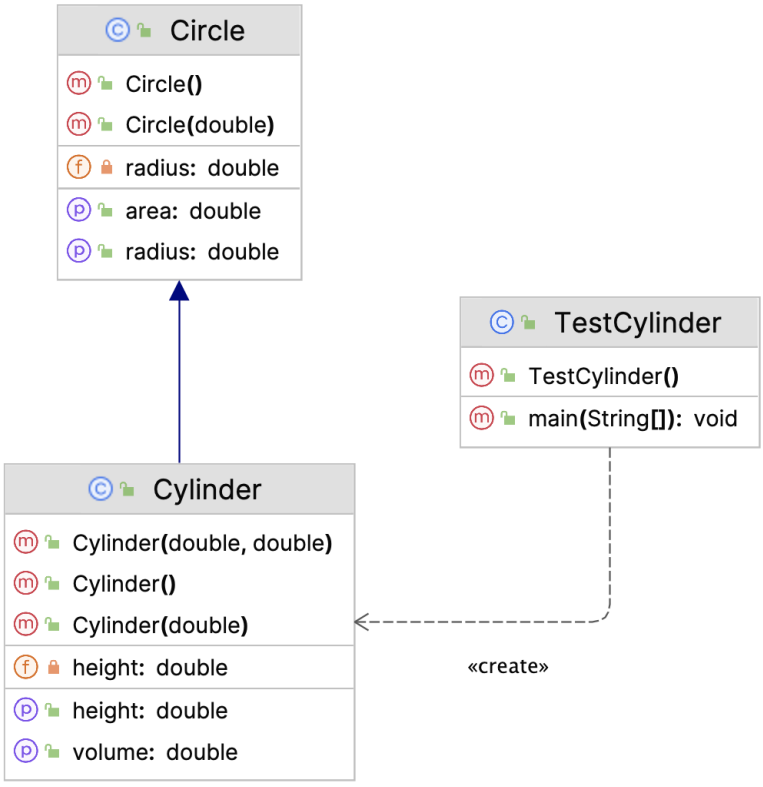
```
public class Student extends Person {
  int gpa;
   public Student(int id, String name, int age, String address, int gpa) {
       super(id, name, age, address);
       this.gpa = gpa;
   }
   public int getGpa() {
       return gpa;
   }
   public void setGpa(int gpa) {
       this.gpa = gpa;
   }
}
```

Tạo lớp Teacher kế thừa từ lớp Person:

```
public class Teacher extends Person {
   private int salary;
   public Teacher(int id, String name, int age, String address, int salary) {
       super(id, name, age, address);
       this.salary = salary;
   }
   public int getSalary() {
       return salary;
   }
   public void setSalary(int salary) {
       this.salary = salary;
   }
}
```

Tính kế thừa

Tạo một lớp cơ sở tên là Circle và một lớp khác có tên là lớp Cylinder kế thừa Circle. Sau đó, viết một chương trình để sử dụng các lớp này. Sơ đồ cho chương trình như sau:



Tạo lớp Circle như sau:

```java
public class Circle {
   private double radius;
   private String color;
   // 1st constructor, which sets both radius and color to default
   public Circle() {
       radius = 1.0;
       color = "red";
   }
   // 2nd constructor with given radius, but color default
   public Circle(double r) {
       radius = r;
       color = "red";
   }
   // A public method for retrieving the radius
   public double getRadius() {
       return radius;
   }
   // A public method for computing the area of circle
   public double getArea() {
       return radius * radius * Math.PI;
   }
}
```

Tạo lớp Cylinder kế thừa từ lớp Circle:

```java
public class Cylinder extends Circle {
   private double height;
   // Constructor with default color, radius and height
   public Cylinder() {
       super(); // call superclass no-arg constructor Circle()
       height = 1.0;
```

```
    }
    // Constructor with default radius, color but given height
    public Cylinder(double height) {
        super(); // call superclass no-arg constructor Circle()
        this.height = height;
    }
    // Constructor with default color, but given radius, height
    public Cylinder(double radius, double height) {
        super(radius); // call superclass constructor Circle(r)
        this.height = height;
    }
    // A public method for retrieving the height
    public double getHeight() {
        return height;
    }
    // A public method for computing the volume of cylinder
// use superclass method getArea() to get the base area
    public double getVolume() {
        return getArea() * height;
    }
}
```

Tạo lớp TestCylinder để kiểm tra các lớp đã tạo

```
public class TestCylinder { // save as "TestCylinder.java"
    public static void main(String[] args) {
// Declare and allocate a new instance of cylinder
// with default color, radius, and height
        Cylinder c1 = new Cylinder();
        System.out.println("Cylinder:"
                + " radius=" + c1.getRadius()
                + " height=" + c1.getHeight()
                + " base area=" + c1.getArea()
                + " volume=" + c1.getVolume());
// Declare and allocate a new instance of cylinder
// specifying height, with default color and radius
        Cylinder c2 = new Cylinder(10.0);
        System.out.println("Cylinder:"
                + " radius=" + c2.getRadius()
                + " height=" + c2.getHeight()
                + " base area=" + c2.getArea()
                + " volume=" + c2.getVolume());
// Declare and allocate a new instance of cylinder
// specifying radius and height, with default color
        Cylinder c3 = new Cylinder(2.0, 10.0);
        System.out.println("Cylinder:"
                + " radius=" + c3.getRadius()
                + " height=" + c3.getHeight()
                + " base area=" + c3.getArea()
                + " volume=" + c3.getVolume());
    }
}
```

Thực hiện chương trình. Kết quả khi chạy chương trình như sau:

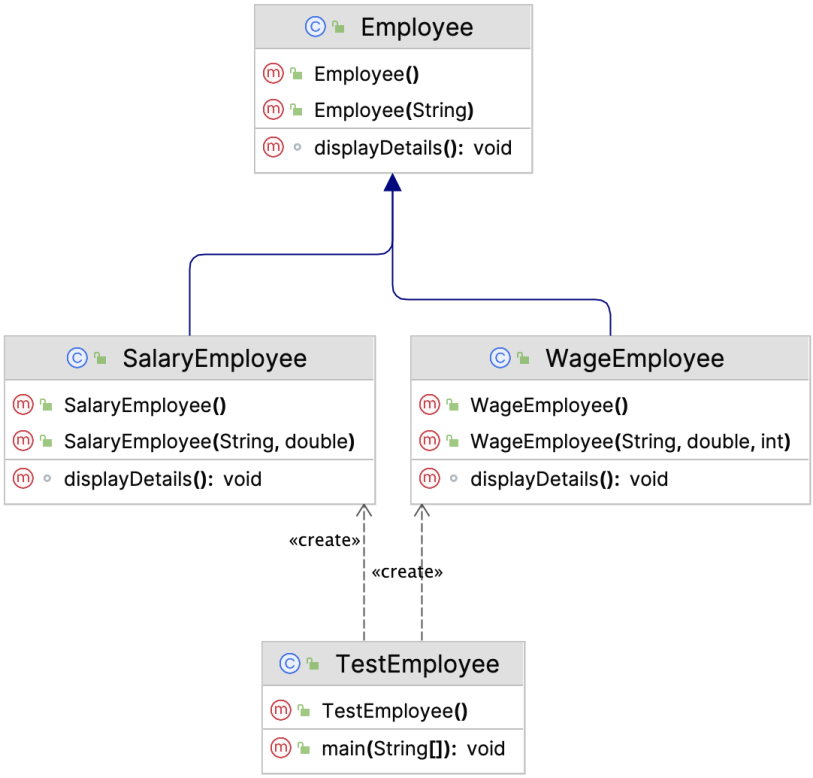
```
Cylinder: radius=1.0 height=1.0 base area=3.141592653589793 volume=3.141592653589793
Cylinder: radius=1.0 height=10.0 base area=3.141592653589793 volume=31.41592653589793
Cylinder: radius=2.0 height=10.0 base area=12.566370614359172 volume=125.66370614359172
```

Bài toán quản lý nhân viên

Tạo chương trình để làm việc với 2 đối tượng nhân viên:

- **Nhân viên biên chế**: có mức lương cố định
- **Nhân viên hợp đồng**: có mức lương và giờ làm việc

Sơ đồ lớp của chương trình như sau:



Tạo lớp Employee theo các bước như sau:

```
public class Employee {
}
```

Khai báo biến để lưu trữ tên nhân viên:

```
String empName;
```

Tạo 2 phương thức khởi tạo:

```
public Employee() {
}
public Employee(String name) {
```

```
4     empName = name;
5  }
```

Tạo phương thức để hiển thị tên:

```
1  void displayDetails() {
2     System.out.printf("Employee Name: %s", empName);
3  }
```

Tạo lớp SalaryEmployee. Tạo lớp SalaryEmployee kế thừa từ lớp Employee:

```
1  public class SalaryEmployee extends Employee {
2  }
```

Khai báo biến salary để lưu vào lương nhân viên

```
double salary;
```

Khai báo các phương thức khởi tạo

```
1  public SalaryEmployee() {
2  }
3  public SalaryEmployee(String name, double currentSalary){
4     super(name);
5     salary = currentSalary;
6  }
```

Phương thức để hiển thị chi tiết lương của nhân viên

```
1  void displayDetails() {
2     super.displayDetails();
3     System.out.printf("\n" + empName + "'s Salary: %.2f\n", salary);
4  }
```

Tạo lớp WageEmployee. WageEmployee kế thừa từ lớp Employee:

```
1  public class WageEmployee extends Employee {
2  }
```

Khai báo thêm các thuộc tính cho lớp WageEmployee:

```
1  double rate;
2  int hours;
```

Khai báo thêm 2 phương thức khởi tạo:

```
1  public WageEmployee() {
2  }
3  public WageEmployee(String name, double wageRate, int wageHours) {
4     super(name);
5     rate = wageRate;
```

```
6     hours = wageHours;
7  }
```

Tạo phương thức để hiển thị chi tiết lương nhân viên

```
1  void displayDetails() {
2     super.displayDetails();
3     System.out.printf("\n" + empName + "'s Salary: %.2f\n", rate * hours);
4  }
```

Tạo lớp **TestEmployee** với phương thức main để thực hiện chương trình chính:

```
1  public class TestEmployee {
2     public static void main(String[] args) {
3         Employee objJohn = new SalaryEmployee("John", 2300.50);
4         objJohn.displayDetails();
5         Employee objDavid = new WageEmployee("David", 34.50, 10);
6         objDavid.displayDetails();
7     }
8  }
```

Thực hiện chương trình và in kết quả đầu ra:

```
1  Employee Name: John
2  John's Salary: 2300.50
3  Employee Name: David
4  David's Salary: 345.00
```

CÂU HỎI ÔN TẬP LÝ THUYẾT

1. Từ khóa super được sử dụng để

- A. Tham chiếu tới giao diện
- B. Tham chiếu tới lớp con
- C. Tham chiếu tới lớp cha
- D. Tham chiếu tới siêu đối tượng

2. Để kiểm tra một đối tượng có phải là thể hiện của một lớp hay không, dùng

- A. instanceof
- B. typeof
- C. isclass

D. classof

3. Từ khóa abstract được sử dụng để

A. Trừu tượng hóa một khối lệnh

B. Tạo lớp hoặc phương thức trừu tượng

C. Ngăn cản việc kế thừa

D. Không cho phép truy cập thuộc tính

4. Để một lớp kế thừa một lớp khác, sử dụng từ khóa

A. extends

B. implements

C. abstract

D. inherit

5. Muốn một thuộc tính không được truy cập bên ngoài lớp sử dụng từ khóa

A. private

B. protected

C. lockaccess

D. default

6. Chỉ định truy cập protected dùng để

A. Bảo vệ một thuộc tính khỏi sự thay đổi

B. Cho phép lớp kế thừa có thể truy cập

C. Ngăn việc truy cập bởi các lớp trong cùng package

D. Ngăn việc truy cập bởi các lớp con

7. Phương thức getter và setter được dùng để

A. Lấy và thiết lập giá trị cho đối tượng

B. Thiết lập quyền truy cập

C. Ngăn không cho truy cập vào biến

D. Tùy chỉnh quyền truy cập thuộc tính

8. Đâu là phát biểu sai về phương thức khởi tạo

A. Phương thức khởi tạo có tên giống với tên lớp

B. Phương thức khởi tạo không cho kế thừa

C. Phương thức khởi tạo phải luốn đi kèm phương thức hủy

D. Một lớp có thể có nhiều phương thức khởi tạo

9. Tính đa hình thể hiện việc

A. Một đối tượng chỉ có thể có một hành vi duy nhất

B. Một đối tượng có thể kế thừa nhiều lần

C. Một đối tượng có thể ở nhiều hình thái khác nhau

D. Một đối tượng có thể liên quan tới nhiều đối tượng khác

10. Hãy cho biết kết quả hiển thị khi chạy chương trình sau:

```
1  class Person {
2      private String name;
3      public String getName() {
4          if(name.length()>16)
5              return "Name is too long!";
```

```
6           else
7               return name;
8       }
9       public void setName(String name) {
10          if(name!=null && name.length()>2)
11              this.name = name;
12      }
13  }
14  public class Program {
15      public static void main(String[] args) {
16          Person person = new Person();
17          person.setName("Nguyen Van Thanh");
18          System.out.println(person.getName());
19      }
20  }
```

- A. Xảy ra một ngoại lệ
- B. Xảy ra một lỗi biên dịch
- C. Màn hình hiển thị Name is too long
- D. Màn hình hiển thị Nguyen Van Thanh

11. Hãy cho biết điều gì xảy ra khi biên dịch/chạy đoạn mã sau:

```
1  interface MyInterface {
2      int x = 1;
3  }
4  public class MyClass {
5      public static void main(String[] args) {
6          MyInterface.x = 2;
7          System.out.println(MyInterface.x);
8      }
9  }
```

- A. In ra giá trị 1
- B. In ra giá trị 2
- C. Lỗi khi biên dịch
- D. Lỗi khi chạy

12. Hãy cho biết kết quả thực hiện đoạn mã sau:

```
class Parent {
    private String name;
    private int age;
    public Parent() {}
}

class Child extends Parent {
    public Child() {
        name = "John";
        age = 42;
    }

    public static void main(String[] args) {
        Child child = new Child();
        System.out.println(child);
    }
}
```

- A. Lỗi biên dịch
- B. John 42
- C. {name:"John", age:42}
- D. Child@xxxxxx trong đó xxxxxx là một dãy số

13. Hãy cho biết kết quả thực hiện đoạn mã sau:

```
public class Program {
    int field1, field2;
    String field3;
    public Program(int i, int j, String k) {
        field1 = i;
        field2 = j;
        field3 = k;
    }
    public static void main(String[] args) {
        Program foo1 = new Program(0, 0, "bar");
        Program foo2 = new Program(0, 0, "bar");
        System.out.println(foo1.equals(foo2)); // prints false
    }
}
```

- A. false
- B. true
- C. Lỗi biên dịch
- D. Chương trình đưa một ngoại lệ

14. Hãy cho biết kết quả thực hiện đoạn mã sau:

```
public interface MyInterface {
    void method1();
    int method2();
    String TEXT = "Hello" ;
    int ANSWER = 42;
    static void main(String[] args) {
        System.out.println(TEXT);
    }
}
```

- A. In ra chữ Hello
- B. Lỗi biên dịch
- C. Chương trình đưa một ngoại lệ
- D. In ra chữ TEXT

BÀI TẬP TỰ THỰC HÀNH

Thực hành về trừu tượng

Toàn bộ Mã nguồn viết trong 1 lớp có tên Media, trong đó có phương thức main và các lớp Item, MP3 và Book:

Tạo lớp Item:

- Là một lớp trừu tượng
- Có thuộc tính chứa tên name (public) kiểu String
- Có thuộc tính mô tả description (private) kiểu String có phương thức get set tương ứng
- Có thuộc tính chứa ID (protected) kiểu String
- Có thuộc tính giá price (default) kiểu float hoặc int
- Có 1 phương thức trừu tượng void showInfo()
- Có phương thức khởi tạo với tham số là các thuộc tính nói trên

Tạo lớp MP3:

- Kế thừa (extend) lớp Item

- Có thêm thuộc tính duration
- Có phương thức khởi tạo với đầy đủ thuộc tính

Tạo lớp Book:

- Tạo lớp Book kế thừa lớp Item
- Có thêm các thuộc tính author (String), numberOfPages(int), genre (String).
- Có phương thức khởi tạo với đầy đủ thuộc tính

Hàm chương trình chính

- Viết phương thức main() tạo ra 1 đối tượng MP3 và 1 đối tượng Book
- Gọi phương thức showInfo() của mỗi đối tượng để hiện thị thông tin thuộc tính của đối tượng

Thực hành về đa hình

Viết mã lệnh trong lớp trừu tượng **Shape**, chú ý hoàn thiện class bằng cách bổ sung mã lệnh vào một số phương thức.

```
public abstract class Shape {
   private String color;
   private boolean filled;
   Shape() {
   }
   Shape(String color, boolean filled) {
   }
   String getColor() {
   }
   void setColor(String color) {
   }
   boolean isFilled() {
   }
   void setFilled(boolean filled) {
   }
   abstract double getArea();
   abstract double getPerimeter();
   public abstract String toString();
}
```

Viết mã lệnh trong lớp **Circle**, chú ý hoàn thiện class bằng cách **bổ sung mã lệnh** vào một số phương thức.

```
public class Circle extends Shape {
   private double radius;
   Circle() {
   }
   Circle(double radius) {
   }
   Circle(double radius, String color, boolean filled) {
```

```
    }
    public double getRadius() {
    }
    void setRadius(double radius) {
    }
    double getArea() {
    }
    double getPerimeter() {
    }
    public String toString() {
    }
}
```

Viết mã lệnh trong lớp **Rectangle**, chú ý hoàn thiện class bằng cách **bổ sung mã lệnh** vào một số phương thức.

```
public class Rectangle extends Shape {
    private double width;
    private double length;
    Rectangle() {
    }
    @Override
    double getArea() {
    }
    @Override
    double getPerimeter() {
    }
    @Override
    public String toString() {
    }
    public double getWidth() {
    }
    public void setWidth(double width) {
    }
    public double getLength() {
    }
    public void setLength(double length) {
    }
}
```

Viết mã lệnh trong lớp **Square**, chú ý hoàn thiện class bằng cách **bổ sung mã lệnh** vào một số phương thức. Bổ sung mã lệnh cho lớp Square

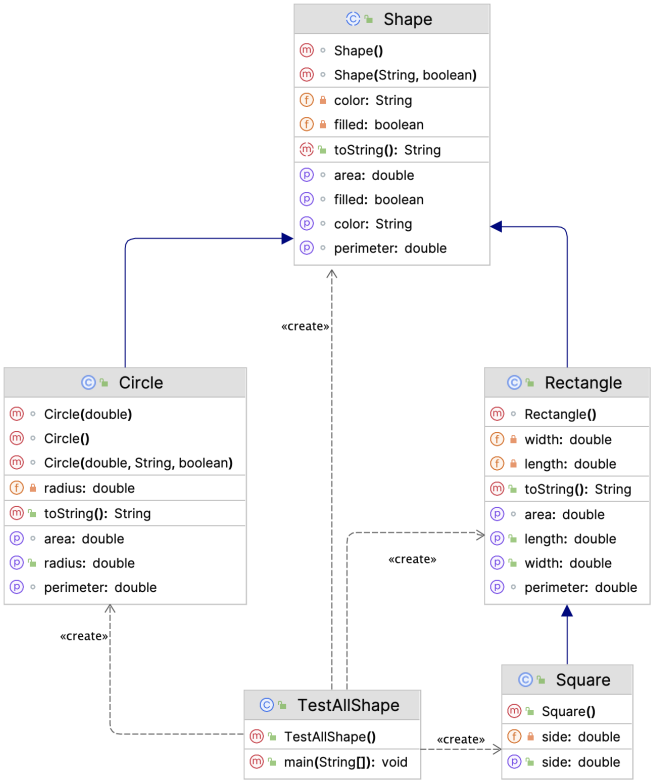
```
public class Square extends Rectangle {
}
```

Sử dụng đoạn mã dưới đây để kiểm tra tất cả các lớp được tạo ở trên.Trong các đoạn mã này có một số lỗi, hãy giải thích, sửa lỗi và chạy chương trình.

```
public class TestAllShape {
    public static void main(String[] args) {
        Shape s1 = new Circle(5.5, "RED", false);
```

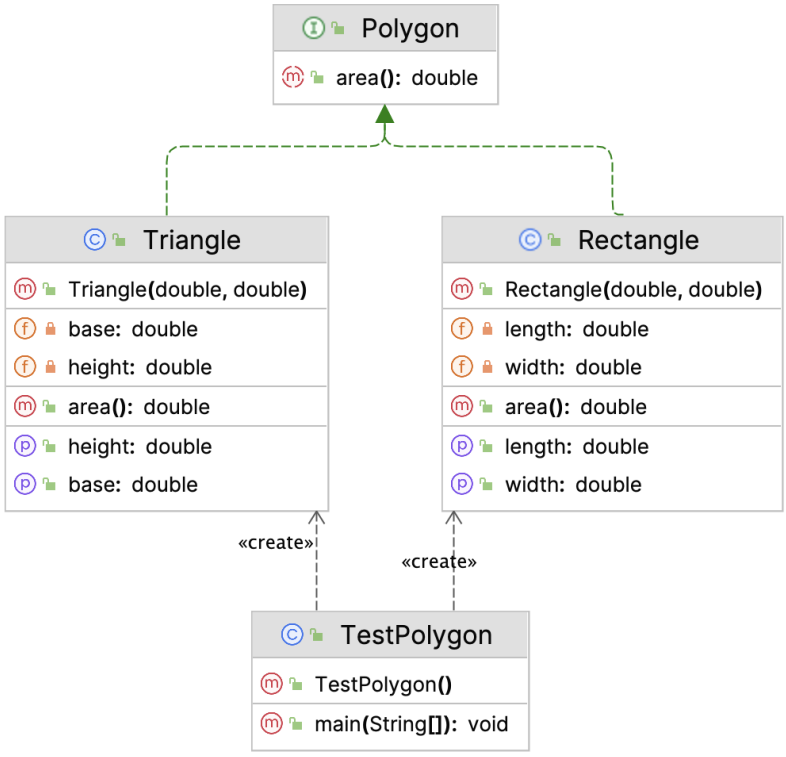
```
        System.out.println(s1);
        System.out.println(s1.getArea());
        System.out.println(s1.getPerimeter());
        System.out.println(s1.getColor());
        System.out.println(s1.isFilled());
        System.out.println(s1.getRadius());
        Circle c1 = (Circle)s1;
        System.out.println(c1);
        System.out.println(c1.getArea());
        System.out.println(c1.getPerimeter());
        System.out.println(c1.getColor());
        System.out.println(c1.isFilled());
        System.out.println(c1.getRadius());
        Shape s2 = new Shape();
        Shape s3 = new Rectangle(1.0, 2.0, "RED", false);   // Upcast
        System.out.println(s3);
        System.out.println(s3.getArea());
        System.out.println(s3.getPerimeter());
        System.out.println(s3.getColor());
        System.out.println(s3.getLength());
        Rectangle r1 = (Rectangle)s3;   // downcast
        System.out.println(r1);
        System.out.println(r1.getArea());
        System.out.println(r1.getColor());
        System.out.println(r1.getLength());
        Shape s4 = new Square(6.6);     // Upcast
        System.out.println(s4);
        System.out.println(s4.getArea());
        System.out.println(s4.getColor());
        System.out.println(s4.getSide());
        Rectangle r2 = (Rectangle)s4;
        System.out.println(r2);
        System.out.println(r2.getArea());
        System.out.println(r2.getColor());
        System.out.println(r2.getSide());
        System.out.println(r2.getLength());
        Square sq1 = (Square)sq1;
        System.out.println(sq1);
        System.out.println(sq1.getArea());
        System.out.println(sq1.getColor());
        System.out.println(sq1.getSide());
        System.out.println(sq1.getLength());
    }
}
```

Tham khảo sơ đồ lớp sau:



Thực hành về Interface

Thiết kế một interface Polygon, định nghĩa các hành vi cơ bản để các lớp khác triển khai. Sơ đồ lớp sẽ như sau:



Thực hành về lập trình hướng đối tượng nói chung

Viết mã lệnh để tạo ra class Shape, với các biến, phương thức khởi tạo và phương thức như sau:

```
1  private String color;
2  public Shape(String color) {
3     this.color = color;
4  }
```

```
 5  public String toString() {
 6     return "Shape of color=\"" + color + "\"";
 7  }
 8  public double getArea() {
 9     System.err.println("Không biết hình gì, không thể tính diện tích");
10     return 0; // Need a return to compile the program
11  }
```

Tạo lớp Rectangle kết thừa từ lớp Shape:

```
1  public class Rectangle extends Shape{
2
3  }
```

Khai báo thêm các thuộc tính cho lớp Rectangle

```
1  private int length;
2  private int width;
```

Viết phương thức khởi tạo:

```
 1  public Rectangle(String color, int length, int width) {
 2     super(color);
 3     this.length = length;
 4     this.width = width;
 5  }
```

Hai phương thức override từ các phương thức ở lớp cha

```
 1  @Override
 2  public String toString() {
 3     return "Rectangle of length=" + length + " and width=" + width + ", subclass of "
+ super.toString();
 4  }
 5  @Override
 6  public double getArea() {
 7     return length*width;
 8  }
```

Tạo lớp Triangle kết thừa từ lớp Shape:

```
1  public class Triangle extends Shape {
2  }
```

Khai báo thêm các thuộc tính cho lớp Triangle

```
1  private int base;
2  private int height;
```

Viết phương thức khởi tạo

```
public Triangle(String color, int base, int height) {
   super(color);
   this.base = base;
   this.height = height;
}
```

Hai phương thức override từ các phương thức ở lớp cha

```
@Override
public String toString() {
   return "Triangle of base=" + base + " and height=" + height + ", subclass of " +
super.toString();
}
@Override
public double getArea() {
   return 0.5*base*height;
}
```

Tạo lớp TestShape là chương trình chính, viết mã lệnh cho phương thức main()

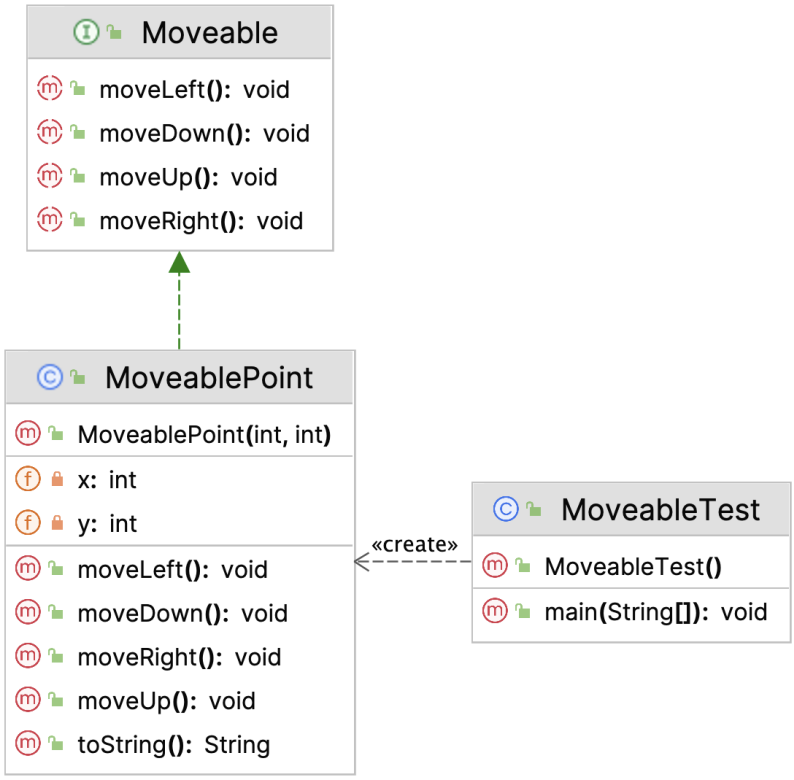
```
Shape s;
s = new Rectangle("red", 4, 5);
System.out.println(s);
System.out.println("Area is " + s.getArea());
s = new Triangle("blue", 4, 5);
System.out.println(s);
System.out.println("Area is " + s.getArea());
s = new Shape("green");
System.out.println(s);
System.out.println("Area is " + s.getArea());
```

Chạy chương trình chính và kiểm tra kết quả:

```
Rectangle of length=4 and width=5, subclass of Shape of color="red"
Area is 20.0
Triangle of base=4 and height=5, subclass of Shape of color="blue"
Area is 10.0
Shape of color="green"
Area is 0.0
```

Thực hành về Interface, chồng phương thức

Cho sơ đồ lớp của chương trình được hiển thị dưới đây:



Tạo Interface có tên Moveable

```
1  interface Moveable {
2  }
```

Khai báo các phương thức trừu tượng trong interface Moveable

```
1  void moveUp();
2  void moveDown();
3  void moveLeft();
4  void moveRight();
```

Tạo lớp MovablePoint triển khai từ Movable Interface

```
 1  public class MoveablePoint implements Moveable {
 2      @Override
 3      public void moveUp() {
 4      }
 5      @Override
 6      public void moveDown() {
 7      }
 8      @Override
 9      public void moveLeft() {
10      }
11      @Override
12      public void moveRight() {
13      }
14  }
```

Lập trình cho lớp MovablePoint. Khai báo biến (x, y) - tọa độ của điểm

```
private int x;
private int y;
```

Phương thức khởi tạo

```
public MoveablePoint(int x, int y) {
   this.x = x;
   this.y = y;
}
public String toString() {
   return "Point at (" + x + "," + y + ")";
}
```

Viết mã thực hiện các phương thức trừu tượng xác định Movable

```
@Override
public void moveUp() {
   y--;
}
@Override
public void moveDown() {
   y++;
}
@Override
public void moveLeft() {
   x--;
}
@Override
public void moveRight() {
   x++;
}
```

Tạo lớp chương trình chính với phương thức main:

```
public static void main(String[] args) {
   Moveable m1 = new MoveablePoint(5, 5);
   System.out.println(m1);
   m1.moveDown();
   System.out.println(m1);
   m1.moveRight();
   System.out.println(m1);
}
```

Chạy chương trình chính, kết quả:

```
Point at (5,5)
Point at (5,6)
Point at (6,6)
```

TÀI LIỆU THAM KHẢO

[1] Core Java: Fundamentals (2021) , Cay Horstmann (Oracle Press Java)

[2] Head First Java: A Brain-Friendly Guide (2022), Kathy Sierra, O'Reilly Media

[3] Java OOP Done Right: Create object oriented code you can be proud of with modern Java Paperback (2019), Mr Alan Mellor, Mellor Books

[4] Murach's Java Programming (5th Edition) (2017), Joe Murach, Mike Murach & Associates

[5]. Java for Absolute Beginners Learn to Program the Fundamentals the Java 9+ Way

[6]. Modern Java Recipes: Simple Solutions to Difficult Problems in Java 8 and 9 (2017), by Ken Kousen, O'Reilly Media

[7] Effective Java (2018), Joshua Bloch, Addison-Wesley Professional