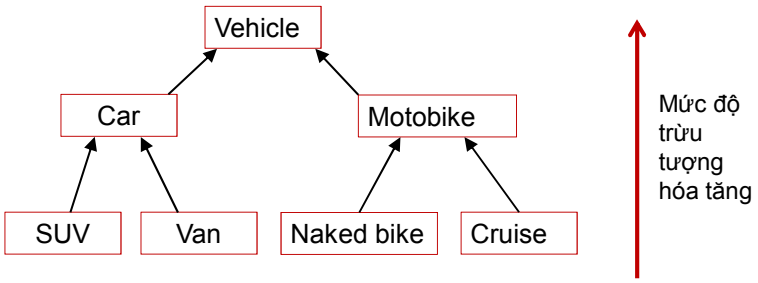
BÀI 5.
TRỪU TƯỢNG VÀ ĐA HÌNH

1. TRỪU TƯỢNG HÓA

Lớp trừu tượng (Abstract class)
Giao diện (Interface)

Trừu tượng hóa

- Loại bỏ đi các thông tin cụ thể, giữ lại các thông tin chung
- Mức độ trừu tượng hóa trên cây kế thừa



3

Lớp trừu tượng

- Khi chưa thể định nghĩa rõ ràng nội dung của một phương thức → cần xây dựng phương thức đó như là phương thức trừu tượng
- Lớp chứa phương thức trừu tượng bắt buộc phải khai báo như lớp trừu tượng

```
package samsung.java.oop.person;
/**The Person class contains some information of someone
*/
public abstract class Person {

   public abstract void displayPerson();
}
```

4

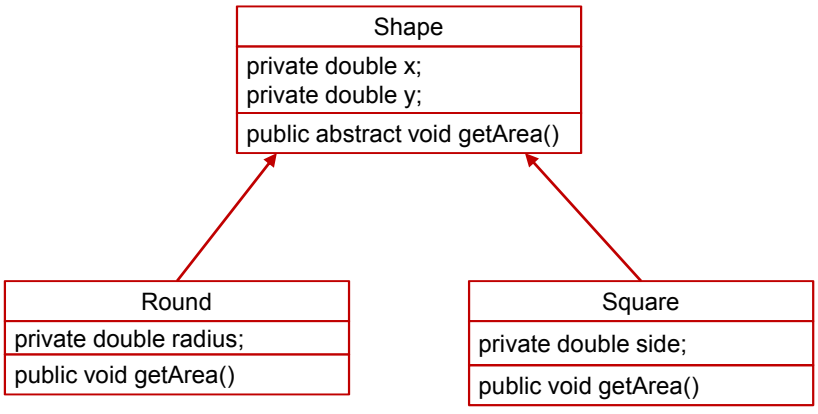
Các quy tắc khi sử dụng lớp trừu tượng

- Phương thức trừu tượng không được phép định nghĩa cụ thể tại lớp cha
- Lớp con kế thừa từ lớp trừu tượng phải định nghĩa nội dung của phương thức trừu tượng
  - Chỉ định truy cập không được chặt hơn lớp cha
  - Nhắc lại mức độ chặt của các chỉ định truy cập:

    public > protected > không chỉ định > private
- Không được tạo đối tượng từ lớp trừu tượng
  - Nhưng lớp trừu tượng vẫn có phương thức khởi tạo

5

Ví dụ về lớp trừu tượng



6

Lớp Shape

```
package samsung.java.oop.shape;
/** The Shape class illutrating a shape has x and y
coordinate and an abstract method */
public abstract class Shape {
	private double x;
	private double y;

	/**
	 * Constructs a new shape
	 */
	public Shape(double initX, double initY){
		this.x = initX;
		this.y = initY;
	}
	public abstract void getArea();
}
```

7

Lớp Round

```
package samsung.java.oop.shape;
/** The Round class presents a round */
public class Round extends Shape {
	private double radius;

	/**
	 * Constructs a new round
	 */
	public Round(double initX, double initY, double
				initRadius){
		super(initX,initY);
		this.radius = initRadius;
	}
	public void getArea(){
		return Math.PI*radius*radius;
	}
}
```

8

Lớp Square

```
package samsung.java.oop.shape;
/** The Square class presents a square */
public class Square extends Shape{
       private double side;

       /**
        * Constructs a new square
        */
       public Square(double initX, double initY, double
                            initSide){
              super(initX,initY);
              this.side = initSide;
       }
       public void getArea(){
              return side*side;
       }
}
```

9

Lớp ShapeTest

```
package samsung.java.oop.shape;
/** The Square class presents a square */
public class ShapeTest{
   public static void main(String arg[]){
       Shape shapeObj = new Shape(1,1); //wrong

       Round roundObj = new Round(1,1,2); //OK
       System.out.println("The area of this round" +
                    roundObj.getArea());

       Square squareObj = new Square(0,1,1); //OK
       System.out.println("The area of this square" +
                            squareObj.getArea());
   }
}
```

10

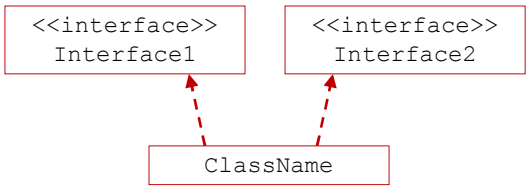
Giao diện

- Java không cho phép đa kế thừa từ nhiều lớp
- Để thực hiện đa kế thừa, Java sử dụng khái niệm giao diện (interface)
- Giao diện chỉ quy định các phương thức phải có, nhưng không định nghĩa cụ thể
  - Cho phép tách rời đặc tả mức trừu tượng và triển khai cụ thể
  - Đảm bảo tính cộng tác trong phát triển phần mềm
- Các giao diện có thể kế thừa nhau
- Cú pháp

```
Modifier interface InterfaceName {
        //Declare constants
        //Declare methods
}
```

11

Triển khai giao diện

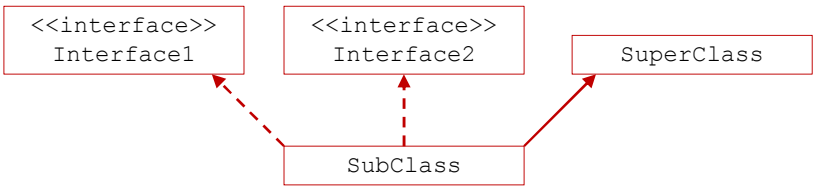



```
Modifier class ClassName implements Interface1, Interface2
{
        //class's body
}
```

- `ClassName` phải định nghĩa mọi phương thức của `Interface1` và `Interface2`

12

Kế thừa và triển khai giao diện đồng thời



- Cú pháp

```
Modifier class SubClass extends SuperClass implements
                          Interface1, Interface2 {
      //class's body
}
```

- `SubClass` kế thừa các phương thức, thuộc tính của `SuperClass` và phải định nghĩa mọi phương thức của `Interface1` và `Interface2`

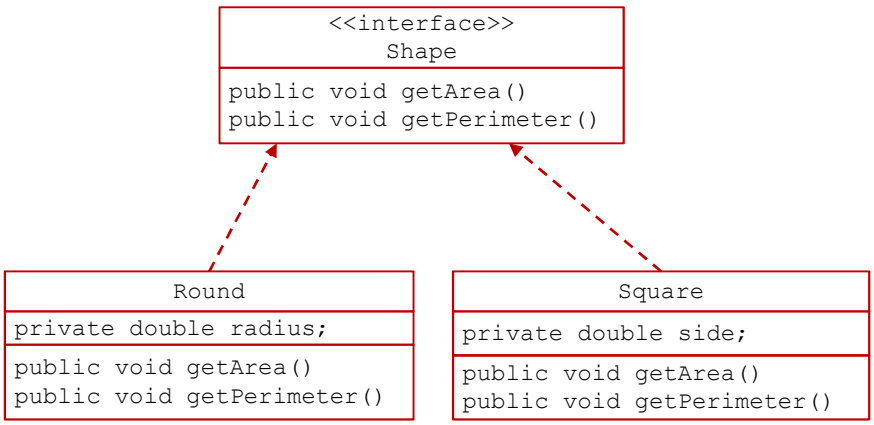
13

Giao diện vs Lớp trừu tượng

Giao diện	Lớp trừu tượng
• Chỉ được phép có thành viên hằng • Mọi phương thức là trừu tượng với chỉ định truy cập `public` • Không có phương thức khởi tạo • Một lớp có thể triển khai nhiều giao diện • Không tái sử dụng mã nguồn	• Có thể có thuộc tính • Ngoài phương thức trừu tượng, có thể có phương thức riêng • Có phương thức khởi tạo • Một lớp chỉ có thể kế thừa từ một lớp trừu tượng • Có tái sử dụng mã nguồn

14

Ví dụ về interface



15

Giao diện Shape

```
package samsung.java.oop.shape;
/** The Shape interfce illutrating a shape */
public interface Shape {
	public void getArea();
	public void getPerimeter();
}
```

16

Lớp Round

```
package samsung.java.oop.shape;
/** The Round class presents a round */
public class Round implements Shape {
	private double radius;

	/** Constructs a new round */
	public Round(double initRadius){
		this.radius = initRadius;
	}
	public void getArea(){
		return Math.PI*radius*radius;
	}
	public void getPerimeter(){
		return 2*Math.PI*radius;
	}
	public double getRadius(){return this.radius;}
}
```

17

2. ĐA HÌNH

Upcasting và downcasting

Chồng phương thức và ghi đè phương thức

18

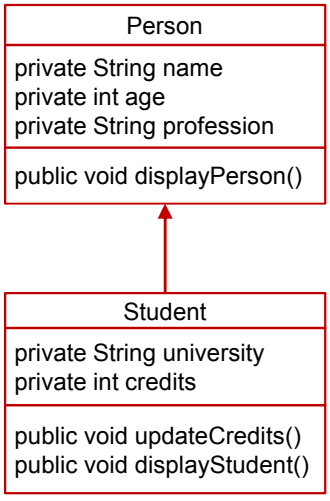
Đa hình (Polymorphism) là gì?

- Đa hình: nhiều hình thức thực hiện một hành vi, nhiều kiểu tồn tại của một đối tượng
- Đa hình trong lập trình:
  - Đa hình phương thức: chồng phương thức, ghi đè phương thức
  - Đa hình đối tượng: nhìn nhận đối tượng theo nhiều kiểu khác nhau
  - Ví dụ: một bạn sinh viên là cán bộ lớp thì có thể nhìn nhận theo 2 góc nhìn

19

Upcasting và Downcasting

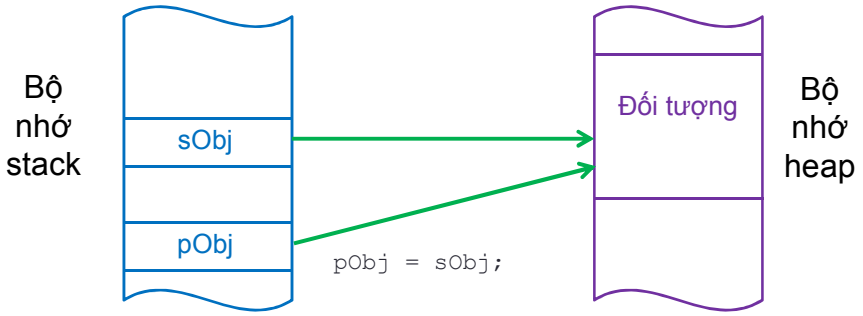
- Upcasting: đối tượng lớp con được nhìn nhận như đối tượng lớp cha
  - Thực hiện tự động
- Downcasting: đối tượng lớp cha được nhìn nhận như đối tượng lớp con
  - Phải ép kiểu



20

Upcasting – Ví dụ

```
Person pObj;
Student sObj = new Student();
pObj = sObj; //upcasting
pObj.displayPerson();//OK
pObj.displayStudent();//wrong
pObj.updateCredits(3);//wrong
```



21

Upcasting và ghi đè phương thức

```
public class Father{
   private int moneyInWallet; //tiền trong ví của cha

   public Father(){
        moneyInWallet = 100;
   }

   public void withdraw(int amount){
       moneyInWallet -= amount;
       System.out.println("The money of father remains: "
                                              + moneyInWallet);
   }
}
```

22

Upcasting và ghi đè phương thức(tiếp)

```
public class Child extends Father{
   private int moneyInWallet; //tiền trong ví của con

   public Child(){
       moneyInWallet = 20;
   }

   //overriding
   public void withdraw(int amount){
       moneyInWallet -= amount;
       System.out.println("The money of child remains: "
                                          + moneyInWallet);
   }
}
```

23

Upcasting và ghi đè phương thức(tiếp)

```
public class Test {
   public static void main(String[] args) {
       Child son = new Child();
       Father father;
       father = son; //upcasting
       father.withdraw(10);
   }
}
```

- Hãy chạy chương trình và xem kết quả
- Giải thích: `father` tham chiếu sang một đối tượng thuộc lớp `Child`. Lúc này, mặc dù đã thực hiện upcasting nhưng lớp con đã ghi đè phương thức `withdraw()` nên `father` thực lời gọi phương thức `withdraw()` của lớp con

→ upcasting không có hiệu lực trên phương thức bị ghi đè

→ Khi phương thức ở lớp cha bị ghi đè ở lớp con thì không có cách nào gọi tới phương thức này qua đối tượng thuộc lớp con nữa

24

Downcasting – Ví dụ

```
Person pObj;
Student sObj = new Student();
pObj = sObj; //upcasting
pObj.displayPerson(); //OK
pObj.updateCredits;//wrong
((Student) pObj).updateCredits(3);//downcasting
```

- Lỗi runtime-error trong trường hợp sau:

```
Person pObj = new Person();
Student sObj = new Student();
pObj.displayPerson(); //OK
pObj.updateCredits;//wrong
((Student) pObj).updateCredits(3);//downcasting
```

25

Toán tử `instanceof`

- Kiểm tra một đối tượng có phải đang là thể hiện của lớp hoặc giao diện nào đó không
- Cú pháp: `objectName` **`instanceof`** `Class`
  `objectName` **`instanceof`** `Interface`
- Kết quả:
  - `true`: đúng
  - `false`: sai

```
if (pObj instanceof Person)
        System.out.println("pObj is a Person");
if (pObj instanceof Student)
        System.out.println("pObj is a Student");
```

26

Chồng phương thức

- Nhắc lại: lớp con có thể viết lại phương thức thừa kế từ lớp cha bằng 2 cách thức:
  - Chồng phương thức(Overloading): giữ tên và giá trị trả về, thay đổi đối số
  - Ghi đè phương thức (Overriding): giữ nguyên tên, giá trị trả về và đối số
- Chồng phương thức có thể thực hiện ngay trong chính 1 lớp:
  - Ví dụ: Viết các phương thức khởi tạo khác nhau

27

Chồng phương thức

- Liên kết lời gọi hàm: xác định địa chỉ trên bộ nhớ của khối mã lệnh thực hiện phương thức khi có lời gọi
- Liên kết tĩnh: Khối mã lệnh của phương thức được xác định khi dịch
- Liên kết động: Khối mã lệnh của hàm được xác định ghi chương trình thực thi
- Chồng phương thức: thực hiện liên kết động

28

Upcasting, downcasting và chồng phương thức

```
public class Father{
   private int moneyInWallet; //tiền trong ví của cha

   public Father(){
        moneyInWallet = 100;
   }

   public void withdraw(int amount){
       moneyInWallet -= amount;
       System.out.println("The money of father remains: "
                                              + moneyInWallet);
   }
}
```

29

Upcasting, downcasting và chồng phương thức (tiếp)

```
public class Child extends Father{
   private int moneyInWallet; //tiền trong ví của con

   public Child(){
       moneyInWallet = 20;
   }

   //overloadding
   public void withdraw(){
       moneyInWallet -= 10;
       System.out.println("The money of child remains: "
                                              + moneyInWallet);
   }
}
```

30

Upcasting, downcasting và chồng phương thức (tiếp)

```
public class Test {
   public static void main(String[] args) {
        Child son = new Child();
        Father father;
        father = son; //upcasting
        father.withdraw(10);
        ((Child)father).withdraw(); //downcasting
   }
}
```

- Hãy chạy chương trình và xem kết quả
- Upcasing và downcasting vẫn giữ hiệu lực trên phương thức được ghi chồng (overloading)

31

Chồng phương thức – Ví dụ

- Giả sử một sản phẩm có giá bán được tính như sau:
  - Giá bán lá giá niêm yết
  - Giá bán khi có khuyến mãi giảm giá = Giá bán * (100%-Tỉ lệ giảm giá)
  - Giá bán khi khách hàng có thẻ thành viên:
    - Hạng 1: giảm giá 10%
    - Hạng 2: giảm giá 20%
    - Hạng 3: giảm giá 30%
- Viết 3 phương thức khác nhau để lấy giá bán

32

Lớp Product

```
package samsung.java.oop.product;
/** The Product class illutrates a product in the store */
public class Product {
   private double price;

   public void getPrice (){
       return this.price;
   }

   public void getPrice(double discount){
       return this.price*(100-discount)/100;
   }

   public void getPrice(double discount, int cardLevel){
       return this.price*(100 - discount -
                          10*cardLevel)/100;
   }
}
```

33

Ghi đè phương thức `equals()`

- Không thể dùng toán tử so sánh == để so sánh 2 đối tượng
- Mọi lớp được kế thừa phương thức `equals` từ lớp Object
- ...nhưng một đối tượng không thể dùng được ngay phương thức `equals()` mà phải định nghĩa lại.
  - Ghi đè phương thức (Overriding)
- Hai đối tượng bằng nhau khi thỏa mãn đồng thời 2 điều kiện:
  - Cùng thuộc một lớp
  - Giá trị của mọi thuộc tính là như nhau
- Hoặc giá trị tham chiếu bằng nhau

34

Ghi đè phương thức `equals()` cho Person

```
   public boolean equals(Object o){
       boolean result = false;

       if (this == o) result = true;
       else if(o != null && o instanceof Person){
          Person other = (Person) o;
          result =  (other.age == this.age)&&
                    (other.name.equals(this.name))&&
                  (other.profession.equals(this.profession));
          }

       return result;
}
```

35

3. LẬP TRÌNH TỔNG QUÁT

36

Lập trình tổng quát

- Thuật toán đã xác định → xây dựng chương trình có thể làm việc với nhiều kiểu dữ liệu khác nhau

```
public class IntBox{
  private Integer data;
  public IntBox(Integer data){
    this.data = data;}
  public Integer getData(){
    return this.data;}
}
```

```
public class FloatBox{
  private Float data;
  public FloatBox(Float data){
    this.data = data;}
  public Float getData(){
    return this.data;}
}
```

```
public class StrBox{
  private String data;
  public StrBox(String data){
    this.data = data;}
  public String getData(){
    return this.data;}
}
```

```
public class AnyBox{
  private AnyClass data;
  public AnyBox(AnyClass data){
    this.data = data;}
  public AnyClass getData(){
    return this.data;}
}
```

37

Lập trình tổng quát

- Thực hiện:
- Sử dụng lớp Object

```
public class ObjBox{
  private Object data;
  public ObjBox(Object data){
    this.data = data;}
  public Object getData(){
    return this.data;}
}
```

- Phải ép kiểu khi sử dụng:

```
ObjBox strBox = new
       ObjBox("Hi");//upcasting
String s = (String)

intBox.getData();//downcasting
```

- Sử dụng lớp hình thức E

```
public class Box<E>{
  private E data;
  public Box(E data){
    this.data = data;}
  public E getData(){
    return this.data;}
}
```

- Không cần ép kiểu:

```
Box<String> strBox = new
              Box<String>("Hi");
String s = strBox.getData();
```

38

Lập trình tổng quát

Sử dụng lớp Object

- Không kiểm soát tương thích dữ liệu khi dịch

```
ObjBox strBox = new
                ObjBox("Hi");
String s = (String)
                intBox.getData();
Integer i = (Integer)
                intBox.getData();
```

- Không xuất hiện lỗi khi dịch nhưng xuất hiện lỗi runtime error khi chạy

Sử dụng lớp hình thức E

- Kiểm soát ngay tương thích dữ liệu khi dịch

```
Box<String> strBox = new
                Box<String>("Hi");
String s = strBox.getData();
Integer i = (Integer)
                strBox.getData();
```

- Báo lỗi khi dịch

39