

BÀI 3. CÁC NGUYÊN LÝ LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG TRONG JAVA

Cơ bản về kế thừa và kết tập trong OOP

Kế thừa và kết tập trong Java

Lớp lồng nhau

Trừu tượng và đa hình

1

1

Kế thừa và kết tập

1. Tái sử dụng mã nguồn
2. Kết tập (Aggregation)
3. Kế thừa (Inheritance)

2

2

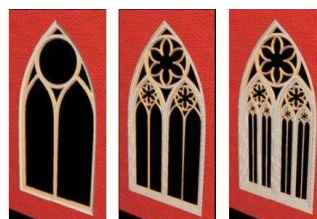
1. TÁI SỬ DỤNG MÃ NGUỒN

3

3

Tái sử dụng mã nguồn là gì?

- Sử dụng lại các mã nguồn đã viết
- Lập trình cấu trúc: chương trình con
- Lập trình hướng đối tượng: nhiều loại đối tượng có thuộc tính, hành vi tương tự nhau → tái sử dụng các lớp đã viết
 - Trong một lớp vẫn tái sử dụng phương thức
- Ưu điểm:
 - Giảm chi phí
 - Nâng cao khả năng bảo trì
 - Nâng cao khả năng mô hình hóa
 - ...



4

4

Các cách thức tái sử dụng mã nguồn

- Sao chép lớp cũ thành 1 lớp khác
 - Hạn chế: Dư thừa, khó quản lý khi có thay đổi
- Kết tập: Lớp mới là tập hợp hoặc sử dụng các lớp đã có
 - Chúng ta đã từng viết hàm main() trong đó có khai báo các đối tượng của một lớp. Nhưng đó không phải là kết tập
- Kế thừa: Lớp mới phát triển thêm các thuộc tính hoặc phương thức từ lớp đã có

5

5

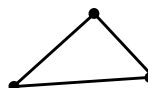
2. KẾT TẬP (AGGREGATION)

6

6

Kết tập là gì


- Thành phần lớp mới chứa các đối tượng của lớp cũ
- Lớp mới: Lớp chứa/Lớp toàn thể
- Lớp cũ: Lớp thành phần
- Ví dụ:
 - Lớp cũ: Điểm (Point)
 - Lớp mới: Tam giác (Triangle) có 3 điểm
- Lớp chứa tái sử dụng các thuộc tính và phương thức của lớp thành phần thông qua đối tượng

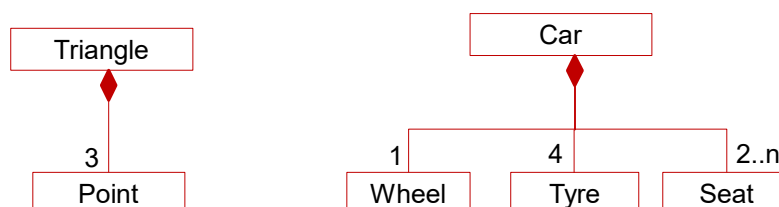


7

7

Biểu diễn kết tập trên biểu đồ thiết kế

- Lớp chứa  Lớp thành phần
- Sử dụng bội số quan hệ:
 - 1 số nguyên dương(1, 2, 3...)
 - Dải số (0..1, 1..n)
 - Bất kỳ giá trị nào: *
 - Không ghi: mặc định là 1



8

8

Minh họa trên Java – Lớp Point

```
package java.oop.basic.aggregation;
/** The Point class presents a point in the system Oxy */
public class Point {
    private double x, y;

    /** The constructor method
     * @param initX : the x coordinate
     * @param initY : the y coordinate
     */
    public void Point(double initX, double initY){
        this.x = newX;
        this.y = newY;
    }
}
```

9

9

Minh họa trên Java – Lớp Point (tiếp)

```
/** The X setter method*/
public void setX(double newX){
    this.x = newX;
}
/** The Y setter method*/

/** The X getter method*/
public double getX(){
    return this.x;
}
/** The Y getter method*/

/** Display the coordinates of a point*/
public void displayPoint(){
    System.out.printf("(%f,%f\n)",this.x, this.y);
}
}
```

10

10

Minh họa trên Java – Lớp Triangle

```
package java.oop.basic.aggregation;
/** The Triangle class presents a triangle in the system
Oxy */
public class Triangle {
    private Point vertex1, vertex2, vertex3;

    /** The constructor method
    *@param vertex1 : the 1st coordinate
    *@param vertex2 : the 2nd coordinate
    *@param vertex3 : the 3nd coordinate
    */
    public Triangle(Point vertex1, Point vertex2,
                    Point vertex3) {
        this.vertex1 = vertex1;
        this.vertex2 = vertex2;
        this.vertex3 = vertex3;
    }
}
```

11

11

Minh họa trên Java – Lớp Triangle (tiếp)

```
/** The setter methods*/
/** The getter method*/

public void displayTriangle() {
    System.out.println("The triangle has three
        vertices:");
    vertex1.displayPoint();
    vertex2.displayPoint();
    vertex3.displayPoint();
}
}
```

12

12

Ví dụ

- Xây dựng một trò chơi xúc xắc. Cách chơi như sau:
 - Mỗi hạt xúc xắc được gieo sẽ có giá trị ngẫu nhiên 1..6
 - Hai người lần lượt gieo 1 hạt xúc xắc
 - Sau mỗi lượt gieo, số điểm của lượt đó được tích lũy vào số điểm của người chơi
 - Sau các lượt gieo theo quy định, người thắng cuộc là người có tổng số điểm lớn hơn

13

13

Phát hiện lớp

- Trò chơi cần có 3 lớp: Die (xúc xắc), Player (người chơi), Match (trận đấu)
- Lớp Die:
 - Thuộc tính: face
 - Phương thức: roll() thiết lập một giá trị ngẫu nhiên cho face
- Lớp Player:
 - Thuộc tính: name, point
 - Phương thức: throwDie(Die) chờ nhấn phím bất kỳ để thực hiện gieo xúc xắc

Die
private int face
public roll() public int getFace()

Player
private String name private int point
public void throwDie(Die) public void setPoint(int) public int getPoint()

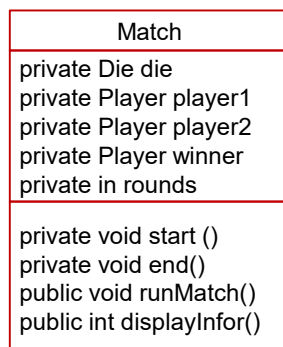
14

14

Phát hiện lớp (tiếp)

- Lớp Match:

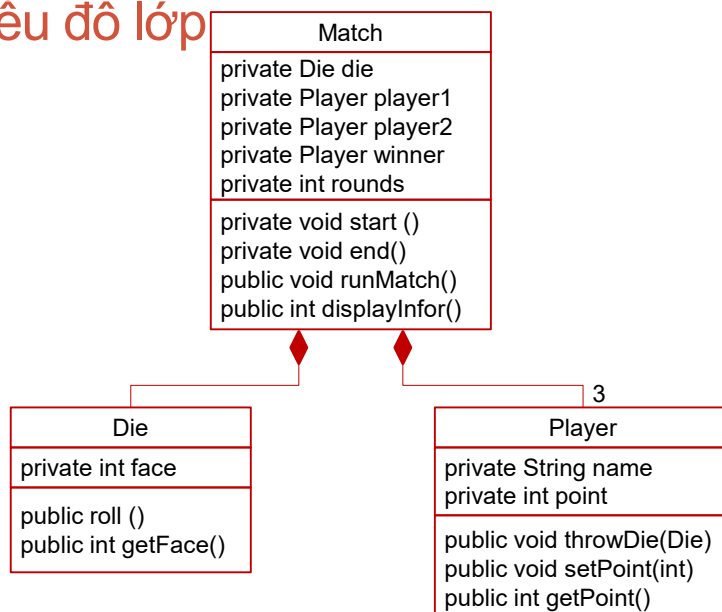
- Thuộc tính: die, player1, player2, winner, rounds (số lượt gieo)
- Hành vi: start(), end(), runMatch(), displayInfor()



15

15

Biểu đồ lớp



16

16

Lớp Die

```
package java.oop.die.game;

import java.util.Random;

/**
 * The Die class presents the die in game
 */
public class Die {
    private int face;

    /**
     * Constructs a new die
     */
    public Die(){
        this.face = 1;
    }
}
```

17

17

Lớp Die (tiếp)

```
/**
 * Generate randomly a face
 * @return The face of the dice after rolling
 */
public int roll(){
    Random rand = new Random();
    this.face = rand.nextInt(5) + 1;
    return this.face;
}

/**
 * Get the current face of the die
 * @return The current face
 */
public int getFace(){
    return this.face;
}
}
```

18

18

Lớp Player

```
package java.oop.die.game;
import java.io.IOException;
import java.util.Scanner;

/** This class describes a player in game */
public class Player {
    private String name;
    private int point;
    private Scanner pressKey;

    /**Constructs a new player with his name
     * @param initName: The player's name */
    public Player(String initName){
        this.name = new String(initName);
        this.point = 0;
    }
}
```

19

19

Lớp Player (tiếp)

```
/**Player throw a die
 * @param die: The Die object */
public void throwDie(Die die){
    int currentThrow;
    pressKey = new Scanner(System.in);
    System.out.print("Press Enter to throw your die!");
    try {
        System.in.read();
    } catch (IOException e) {
        e.printStackTrace();
    }
    pressKey.nextLine();
    currentThrow = die.roll();
    this.point += currentThrow;
    System.out.println(currentThrow + " points");
}
```

20

20

Lớp Player (tiếp)

```

/**Set a new point for player after he threw die
 * @param newPoint: The new point after throwing
 */
public void setPoint(int newPoint){
    this.point = newPoint;
}
/**Get the current point of the player
 * @return: The current point*/
public int getPoint(){
    return this.point;
}
/**Get the name of player
 * @return The name of player */
public String getName(){
    return this.name;
}
}

```

21

21

Lớp Match

```

package java.oop.die.game;
public class Match {
    private Die die;
    private Player player1, player2;
    private Player winner;
    private int rounds;

    /** Constructs a new match with initial values
     * @param initPlayer1: The 1st player
     * @param initPlayer2: The 2nd player
     * @param initRounds: The number of rounds
     */
    public Match(String initPlayer1, String initPlayer2, int
                  initRounds){
        this.player1 = new Player(initPlayer1);
        this.player2 = new Player(initPlayer2);
        this.die = new Die();
        this.rounds = initRounds;
    }
}

```

22

22

Lớp Match (tiếp)

```

/** Running the match */
public void runMatch(){
    this.start();
    this.stop();
    this.displayInfor();
}

/** Start the match*/
private void start(){
    System.out.println("Start!");
    for(int i = 1; i<= this.rounds; i++){
        System.out.println("-----Round " + i + "-----");
        System.out.println(player1.getName() + " throw!");
        player1.throwDie(die);
        System.out.println(player2.getName() + " throw!");
        player2.throwDie(die);
    }
}

```

23

23

Lớp Match (tiếp)

```

/** Stop the match */
private void stop(){
    int pointPlayer1, pointPlayer2;
    pointPlayer1 = player1.getPoint();
    pointPlayer2 = player2.getPoint();
    if(pointPlayer1 > pointPlayer2)
        this.winner = this.player1;
    else if(pointPlayer2 > pointPlayer1)
        this.winner = this.player2;
}

/** Display the information of the game */
public void displayInfor(){System.out.println("---RESULT---");
    System.out.println(player1.getName() +" has " +
        player1.getPoint() +" points");
    System.out.println(player2.getName() +" has " +
        player2.getPoint() +" points");
    if(this.winner!=null)
        System.out.println("The winner is " + winner.getName());
    else System.out.println("You are draw");
}

```

24

24

Lớp Game để thử nghiệm

```
package java.oop.die.game;
import java.util.Scanner;
public class Game {
    public static void main(String[] args) {
        Match match;
        String player1, player2;
        int rounds;
        Scanner inputData = new Scanner(System.in);
        System.out.print("Enter the name of the 1st player: ");
        player1 = inputData.next();
        System.out.print("Enter the name of the 2nd player: ");
        player2 = inputData.next();
        System.out.print("Enter the number of rounds: ");
        rounds = inputData.nextInt();

        match = new Match(player1, player2, rounds);
        match.runMatch();
    }
}
```

25

25

Bài tập

- Lớp Player có thêm thuộc tính wonMatch ghi lại số ván thắng.
- Giả sử 2 người chơi phải chơi 3 ván đấu. Nếu người chơi nào thắng trước 2 ván thì người chơi đó thắng cả trận đấu.
- Hãy viết lại các lớp để thỏa mãn yêu cầu trên

26

26

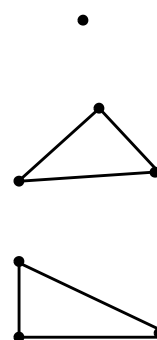
3. KẾ THỪA (INHERITANCE)

27

27

Kế thừa là gì?

- Tạo lớp mới bằng cách phát triển từ lớp đã có
- Lớp mới kế thừa những thành viên đã có trong lớp cũ
 - **Lưu ý: Không kế thừa giá trị**
- Lớp cũ: Lớp cha (superclass), lớp cơ sở (baseclass)
- Lớp mới: Lớp con (subclass), lớp dẫn xuất (derived class)
- Ví dụ:
 - Lớp cũ: Điểm (Point)
 - Kết tập: Tam giác (Triangle) có 3 điểm
 - Kế thừa: Tam giác vuông (Right Triangle)



28

28

Kế thừa vs Kết tập

Kế thừa

- Tái sử dụng mã nguồn thông qua lớp
- Quan hệ “là một loại”
- Ví dụ: Tam giác vuông là một loại tam giác

Kết tập

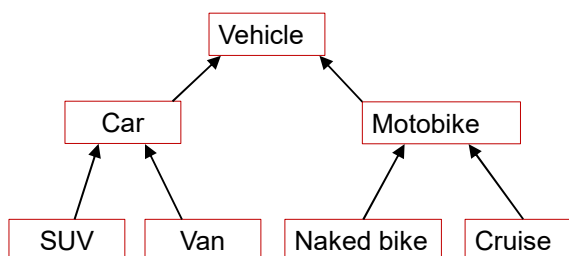
- Tái sử dụng mã nguồn thông qua đối tượng.
- Quan hệ “là một phần”
- Ví dụ: Tam giác có 3 đỉnh

29

29

Biểu diễn kế thừa trên biểu đồ thiết kế

- Lớp cha ←——— Lớp con
- Cây phân cấp kế thừa: Biểu diễn mối quan hệ kế thừa giữa các lớp
- Lớp con kế thừa các thành viên của các lớp tổ tiên theo chỉ định truy cập
- SUV kế thừa trực tiếp từ Car
- SUV kế thừa gián tiếp từ Vehicle



30

30

Kế thừa trong Java

- Cú pháp

```
class SubClass extends SuperClass{
    //SubClass body
}
```

- Lớp con truy cập tới thành viên lớp cha qua từ khóa **super**
- Mọi lớp trong Java đều kế thừa từ lớp tổng quát Object
- Lớp Object cung cấp một số phương thức `toString()`, `equals()`
- Java chỉ cho phép đơn kết thừa: một lớp chỉ có thể kế thừa từ duy nhất 1 lớp khác

31

31

Chỉ định truy cập và kế thừa

- Chỉ định truy cập lớp:
 - **public**: cho phép lớp con kế thừa nằm ở bất kỳ đâu
 - **Không chỉ định**: chỉ cho phép lớp con kế thừa nằm cùng gói
- Chỉ định truy cập thành viên:
 - **public, protected**: cho phép lớp con ở bất kỳ đâu được kế thừa thuộc tính/phương thức này, được truy cập vào thuộc tính/thành viên tương ứng trên lớp cha
 - **Không chỉ định**: chỉ cho phép lớp con ở cùng gói được kế thừa và được truy cập tới thuộc tính/thành viên tương ứng của lớp cha.
 - **private**: lớp con không được kế thừa thuộc tính/phương thức này, không được truy cập vào thuộc tính/thành viên tương ứng trên lớp cha

32

32

Chỉ định truy cập và kế thừa – ví dụ

```
package java.oop.public.inheritance;

/** This is a public superclass*/
public class PublicClass {
    private int privateValue;
    protected int protectedValue;
    int noModifierValue;
    public float publicValue;

    private void privateMethod(){};
    protected int protectedMethod(){};
    String noModifierMethod(){};
    public float publicMethod(){};
}
```

33

33

Chỉ định truy cập và kế thừa – ví dụ

```
package java.oop.public.inheritance;

/** The subclass is in the same package*/
public class AnySubClass extends PublicClass{
    super.privateValue = 0; //wrong
    super.protectedValue = 0; //OK
    super.noModifierValue = 0; //OK
    super.publicValue = 0; //OK

    //Similarly with methods
}
```

34

34

Chỉ định truy cập và kế thừa – ví dụ

```
package java.oop.public.inheritance.other;
import package java.oop.public.inheritance.*

/** The subclass is in the other package*/
public class OtherSubClass extends PublicClass{
    super.privateValue; //wrong
    super.protectedValue; //OK
    super.noModifierValue; //wrong
    super.publicValue; //OK

    //Similarly with methods
}
```

35

35

Chỉ định truy cập và kế thừa – ví dụ

```
package java.oop.restrict.inheritance;

/** This is a superclass without modifier*/
class RestrictClass {
    private int privateValue;
    protected int protectedValue;
    int noModifierValue;
    public publicValue

    private void privateMethod(){};
    protected int protectedMethod(){};
    String noModifierMethod(){};
    public float publicValue(){};
}
```

36

36

Chỉ định truy cập và kế thừa – ví dụ

```
package java.oop.restrict.inheritance;

/** The subclass is in the same package*/
public class AnySubClass extends RestrictClass{
    super.privateValue = 0; //wrong
    super.protectedValue = 0; //OK
    super.noModifierValue = 0; //OK
    super.publicValue = 0; //OK

    //Similarly with methods
}
```

37

37

Chỉ định truy cập và kế thừa – ví dụ

```
package java.oop.restrict.inheritance.other;
import package java.oop. restrict.inheritance.*

/** The subclass is in the other package*/
public class OtherSubClass extends RestrictClass{ //wrong
}
```

- Lớp cha `RestrictClass` không cho phép lớp con nằm bên ngoài gói

38

38

Khởi tạo đối tượng trong kế thừa

- Lớp con không kế thừa phương thức khởi tạo của lớp cha
 - Lớp cha phải được khởi tạo trước lớp con
 - Các phương thức khởi tạo của lớp con luôn gọi phương thức khởi tạo của lớp cha
 - Tự động gọi (không tường minh – không cần thể hiện bằng câu lệnh gọi): nếu lớp cha có phương thức khởi tạo mặc định
 - Phải gọi trực tiếp (tường minh): nếu lớp cha có phương thức khởi tạo khác mặc định
- Cú pháp: `super(parameterList)`

39

39

Khởi tạo đối tượng trong kế thừa

```
public class Base{
    public Base(){
        System.out.println("Base");
    }
}
```

```
public class Sub extends Base{
    public Sub(){
        System.out.println("Sub");
    }
}
```

```
public class Test{
    public static void main(String[] args){
        Sub subObj = new Sub();
    }
}
```

Kết quả khi chạy Test:

```
Base
Sub
```

40

40

Khởi tạo trong kế thừa – Ví dụ

```
package java.oop.inheritance.construct;

/** This is a any superclass*/
public class AnyClass {
    private int supValue;

    /**Constructs a new AnyClass object*/
    public AnyClass(int initSupValue){
        this.supValue = initSupValue;
    }
}
```

41

41

Khởi tạo trong kế thừa – Ví dụ

```
package java.oop.inheritance.construct;

/** This is a any subclass*/
public class AnySubClass extends AnyClass{
    private int subValue;

    /**Constructs a new AnySubClass object*/
    public AnySubClass(int initSubValue){
        this.subValue = initSubValue;
    }/*wrong because don't calls the constructor of the
    superclass*/

    /**Constructs a new AnySubClass object without parameter*/
    public AnySubClass(){
        super(0);
    }
}
```

42

42

```

/**Constructs a new AnySubClass object with initial value
    for superclass*/
public AnySubClass(int initSupValue){
    super(initSupValue);
}

/**Constructs a new AnySubClass object with initial values
    for both*/
public AnySubClass(int initSubValue, int initSupValue){
    this.subValue = initSubValue;
    super(initSupValue);
}/*wrong because don't firstly calls superclass's
    constructor*/

/**Constructs a new AnySubClass object with initial values
    for both*/
public AnySubClass(int initSubValue, int initSupValue){
    super(initSupValue);
    this.subValue = initSubValue;
}

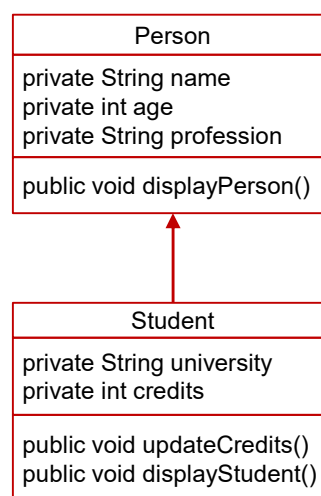
```

43

43

Kế thừa – Ví dụ đầy đủ

- **Lớp Person:**
 - name: **tên**
 - age: **tuổi**
 - profession: **nghề nghiệp**
 - displayPersion(): **hiển thị thông tin**
- **Lớp Student kế thừa lớp Person:**
 - university: **trường học**
 - credits: **số tín chỉ đã tích lũy**
 - updateCredits(int): **cập nhật số tín chỉ đã tích lũy**
 - displayStudent(): **hiển thị thông tin.**



44

44

Lớp Person

```
package java.oop.person;

/**The Person class contains some information of someone */
public class Person {
    private String name;
    private int age;
    private String profession;

    /** Construct a new Person object with name and age
     * @param initName: Initial name
     * @param initAge: Initial age
     */
    public Person(String initName, int initAge){
        this.name = new String(initName);
        this.age = initAge;
        this.profession = new String("Unemployed");
    }
}
```

45

45

Lớp Person (tiếp)

```
/**Set new profession for a person
 * @param newProfession: New profession
 */
public void setProfession(String newProfession){
    this.profession = new String(newProfession);
}

/** Display the information of someone
 */
public void displayPerson(){
    System.out.println("Full name: " + this.name);
    System.out.println("Age: " + this.age);
    System.out.println("Profession: " + this.profession);
}
}
```

46

46

Lớp Student

```
package java.oop.student;
import java.oop.person.Person;
/**The Student class contain the information of the student.
 * This class inherit from the Peerson class.*/
public class Student extends Person {
    private String university;
    private int credits;
    /** Construct a new student with name, age and the
     university where he studies. The cumulated credits of
     the student initiated by zero */
    public Student(String initName, int initAge, String
                    initUniversity){
        super(initName, initAge);
        setProfession("Student");
        this.university = initUniversity;
        credits = 0;
    }
```

Sử dụng setter method để truy cập vào các thuộc tính private của lớp cha

47

47

Lớp Student (tiếp)

```
/**Update the cumulated credits of the student after he
 completed a course
 * @param moreCredits: The more credits cumulated by
 student */
public void updateCredits(int moreCredits){
    this.credits += moreCredits;
}
/** Display the information of the student */
public void displayStudent(){
    displayPerson();
    System.out.println("University: " + this.university);
    System.out.println("Cumulated credits: " +
        this.credits);
}
}
```

Gọi phương thức của lớp cha

48

48

Lớp StudentManagement

```

java.oop.student;
import java.oop.person.Person;
public class StudentManagement {
    public static void main(String[] args) {
        Person someone = new Person("Nguyen Ha Dong",18);
        someone.displayPerson();

        System.out.println("\nDong becomes a student at HUST");
        String name = someone.getName();
        int age = someone.getAge();

        Student bkStudent = new Student(name, age, "HUST");
        bkStudent.displayStudent();

        System.out.println("\nDong has just passed the Java
            Programming course");

        bkStudent.updateCredits(3);
        bkStudent.displayStudent();
    }
}

```

lấy giá trị thuộc tính của một đối tượng lớp cha cho một đối tượng lớp con

49

49

Che thuộc tính

- Trong lớp con khai báo một thuộc tính có tên giống lớp cha thì trên lớp con thuộc tính của lớp cha bị che đi.
- Để truy cập tới thuộc tính trên lớp cha dùng từ khóa `super`
- Để phân biệt trên lớp con, dùng từ khóa `this`
- Ví dụ

```

package java.oop.override.field
public class Parents {
    int intData; //no modifier
    float floatData; //no modifier
}

```

50

50

Ví dụ - Che thuộc tính (tiếp)

```
package java.oop.override.field
public class Children extends Parent {
    private int intData; //overrides intData
    public void displayData(){
        intData = 1; //Integer data in Children
        super.intData = -1; //Overriden integer data in Parent
        floatData = 0.0f; //Non-overriden float data in Parent

        System.out.println("Integer in Chidlren:" + intData);
        System.out.println("Integer in Parents:" +
                            super.intData);
        System.out.println("Float in Parents:" +
                            super.floatData);
    }
}
```

51

51

Ví dụ - Che thuộc tính (tiếp)

```
package java.oop.override.field
public class OverridenTest {
    public static void main(String[] args) {
        Children child = new Children();
        child.displayData();
    }
}
```

• Kết quả thực hiện chương trình:

```
Integer in Children: 1
Integer in Parents: -1
Float in Parents: 0.0
```

- Lớp con vẫn kế thừa thuộc tính intData của cha, nhưng chỉ có thể truy cập nếu sử dụng từ khóa super tương minh **super.intData = -1;**

52

52

Chồng phương thức và ghi đè phương thức

- Lớp con có thể định nghĩa lại các phương thức kế thừa được từ lớp cha:
 - Ghi đè (overriding)
 - Chồng phương thức (overloading)
- Ghi đè (override): Giữ nguyên tên phương thức và danh sách đối số. Khi đó phương thức của lớp cha sẽ bị che đi
 - Truy cập tới phương thức lớp cha qua từ khóa `super`
- Chồng phương thức (overloading): Giữ nguyên tên phương thức và thay đổi danh sách đối số. Phương thức lớp cha được gọi bình thường.
- *Lưu ý: luôn phải giữ nguyên kiểu dữ liệu trả về*

53

53

Ghi đè phương thức – Ví dụ

```
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);
    }
}
```

54

54

Ghi đè phương thức – Ví dụ

- Chúng ta sẽ ghi đè phương thức `withdraw()` tại lớp con

```
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);
    }
}
```

55

55

Ghi đè phương thức – Ví dụ

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

- Hãy chạy chương trình và xem kết quả
- Giải thích: Lớp `Child` đã có phương thức `withdraw(int amount)` để rút tiền từ ví của mình. Do đó khi thực hiện lời gọi `son.withdraw(10)` thì đối tượng tham chiếu bởi `son` được nhìn nhận như là đối tượng thuộc lớp `Child`, và thực hiện rút tiền từ ví của mình

56

56

Chồng phương thức – Ví dụ

- Chúng ta sẽ ghi chồng phương thức `withdraw()` tại lớp con

```
public class Child extends Father{
    private int moneyInWallet; //tiền trong ví của con
    public Child(){
        moneyInWallet = 20;
    }

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

57

57

Chồng phương thức – Ví dụ

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

- Hãy chạy chương trình và xem kết quả
- Giải thích:
 - Trong lời gọi `son.withdraw(10)`; đối tượng tham chiếu bởi `son` được xem như là đối tượng lớp `Father` nên rút tiền từ ví của cha
 - Trong lời gọi `son.withdraw()`; đối tượng tham chiếu bởi `son` được xem như là đối tượng lớp `Child` nên rút tiền từ ví của mình

58

58

Cấm ghi đè

- Trong một số trường hợp cần cấm ghi đè phương thức khi kế thừa:
 - Đảm bảo tính đúng đắn: việc ghi đè phương thức có thể gây ra sự sai lệch về ý nghĩa
 - Tính hiệu quả: giảm thời gian xử lý lời gọi phương thức
- Cấm ghi đè: định nghĩa phương thức với từ khóa `final`

```
/** Display the information of someone
 */
public final void displayPerson() {
    System.out.println("Full name: " + this.name);
    System.out.println("Age: " + this.age);
    System.out.println("Profession: " + this.profession);
}
```

59

59

Từ khóa `final`

- Khai báo một thành viên hằng: không thể bị che trong lớp con
- Khai báo một phương thức: không thể bị ghi đè trong lớp con
- Khai báo một lớp: không cho phép kế thừa

```
package java.oop.person;
/**The Person class contains some information of someone */
public final class Person {
    //class's body
}
```

60

60

4. TRỪU TƯỢNG HÓA

Lớp trừu tượng (Abstract class)

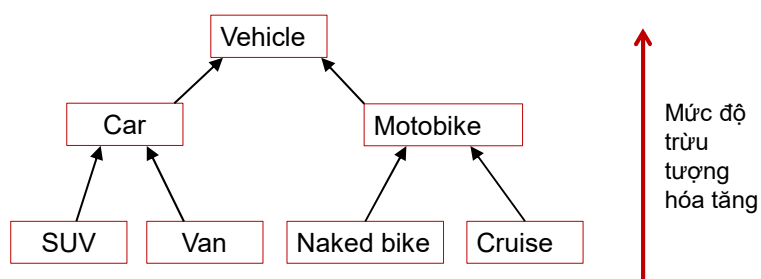
Giao diện (Interface)

61

61

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



62

62

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

    public abstract void displayPerson();
}
```

63

63

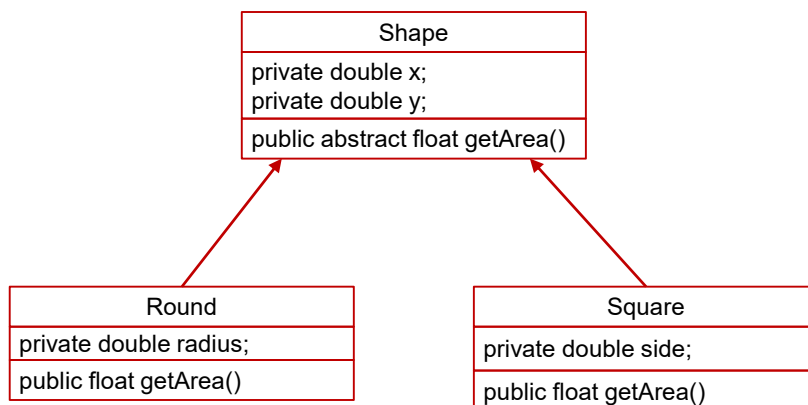
Các quy tắc khi sử dụng lớp trừu tượng

- 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

64

64

Ví dụ về lớp trừu tượng



65

65

Lớp Shape

```

package java.oop.shape;

/** The Shape class illustrating 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();
}
  
```

66

66

Lớp Round

```
package 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;
    }
}
```

67

67

Lớp Square

```
package 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;
    }
}
```

68

68

Lớp ShapeTest

```
package 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());
    }
}
```

69

69

Giao diện

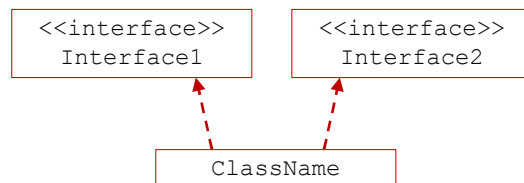
- Java không cho phép đa kế thừa từ nhiều lớp
- Để thực hiện đa kế thừa, Java sử dụng khái niệm giao diện (interface)
- Giao diện chỉ quy định các phương thức phải có, nhưng không định nghĩa cụ thể
 - Cho phép tách rời đặc tả mức trừu tượng và triển khai cụ thể
 - Đảm bảo tính cộng tác trong phát triển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

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

70

70

Triển khai giao diện



```

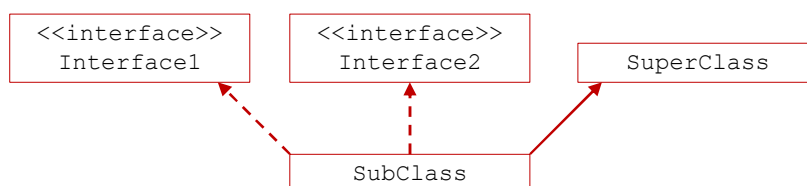
Modifier class ClassName implements Interface1, Interface2
{
    //class's body
}
  
```

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

71

71

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**

72

72

Giao diện vs Lớp trừu tượng

Giao diện

- 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

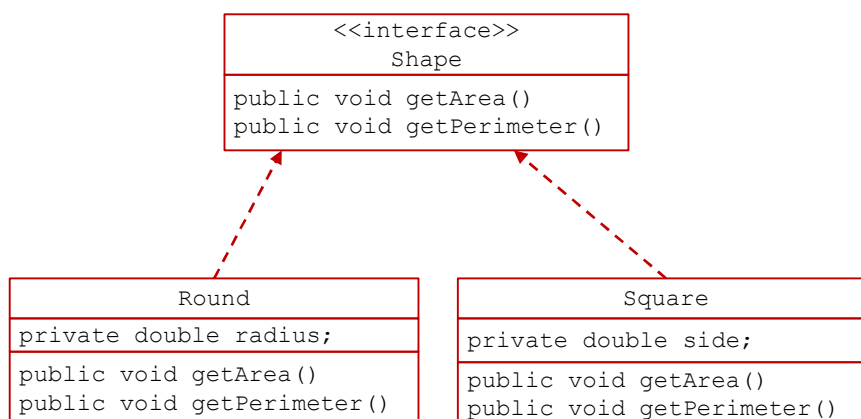
Lớp trừu tượng

- 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

73

73

Ví dụ về interface



74

74

Giao diện Shape

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

75

75

Lớp Round

```
package 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;}
}
```

76

76

5. ĐA HÌNH

Upcasting và downcasting

Chồng phương thức và ghi đè phương thức

77

77

Đa hình (Polymorphism) là gì?

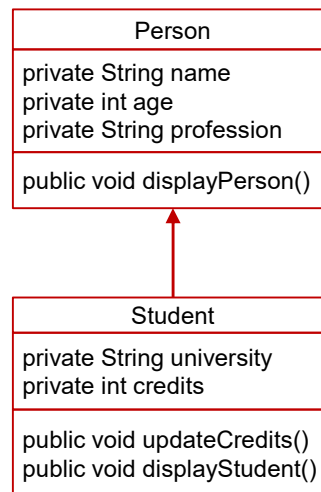
- Đa hình: nhiều hình thức thực hiện một hành vi, nhiều kiểu tồn tại của một đối tượng
- Đa hình trong lập trình:
 - Đa hình phương thức: chồng phương thức, ghi đè phương thức
 - Đa hình đối tượng: nhìn nhận đối tượng theo nhiều kiểu khác nhau
 - Ví dụ: một bạn sinh viên là cán bộ lớp thì có thể nhìn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

78

78

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



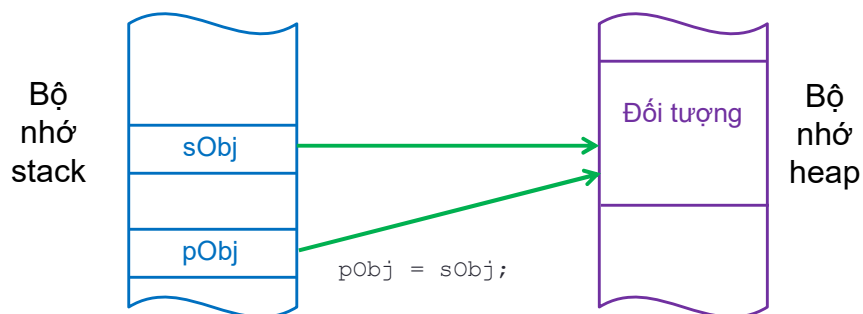
79

79

Upcasting – Ví dụ

```

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



80

80

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);
    }
}
```

81

81

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);
    }
}
```

82

82

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

83

83

Downcasting – Ví dụ

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

84

84

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");
```

85

85

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

86

86

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

87

87

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);
    }
}
```

88

88

Upcasting, downcasting và chồng phương thức (tiếp)

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

    public Child(){
        moneyInWallet = 20;
    }

    //overloading
    public void withdraw(){
        moneyInWallet -= 5;
        System.out.println("The money of child remains: "
                            + moneyInWallet);
    }
}
```

89

89

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
- Upcasting và downcasting vẫn giữ hiệu lực trên phương thức được ghi chồng (overloading)

90

90

Chồng phương thức – Ví dụ

- Giả sử một sản phẩm có giá bán được tính như sau:
 - Giá bán là 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

91

91

Lớp Product

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

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

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

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

92

92

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

93

93

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;
}
```

94

94

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;}
}
```

95

95

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)
    strBox.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();
```

96

96

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)
    strBox.getData();
Integer i = (Integer)
    strBox.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

97