ĐẠI NAM
UNIVERSITY

CHƯƠNG 4
KẾ THỪA (INHERITANCE)

Giảng viên: Phạm Văn Tiệp

Bài toán cần giải quyết

Xây dựng chương trình quản lý việc tính lương cho nhân viên.

- Nhân viên có các thuộc tính là: mã số nhân viên, họ và tên, ngày sinh, số ngày làm việc.
- Lương của nhân viên được tính bằng công thức: số ngày làm việc nhân với 10 USD
- Nhân viên bán hàng còn có thêm thuộc tính số sản phẩm đã bán trong tháng.
- Lương nhân viên bán hàng được tính theo công thức: số ngày làm việc nhân với 5 USD + số sản phẩm nhân với 2 USD

Giải quyết bài toán – cách 1

- **Copy & Paste**

```
class NhanVien
{
    private string maNV;
    private string tenNV;
    private Date ngaySinh;
    private int songaylamviec;

    public int TinhLuong()
    {
        return songaylamviec* 10;
    }
}
```

```
Class NhanVienBH
{
    private string maNV;
    private string tenNV;
    private Date ngaySinh;
    private int songaylamviec;
    private int sosanpham;

    public int TinhLuong()
    {
        return songaylamviec * 5 + sosanpham * 2;
    }
}
```

Mã nguồn phải viết lại nhiều lần. Khó trong việc bảo trì, nâng cấp sau này

Giải quyết bài toán – cách 2

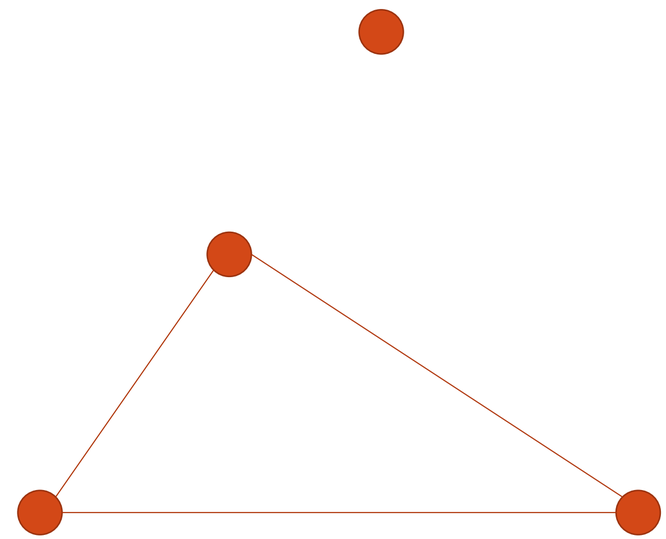
- **Double Duty**

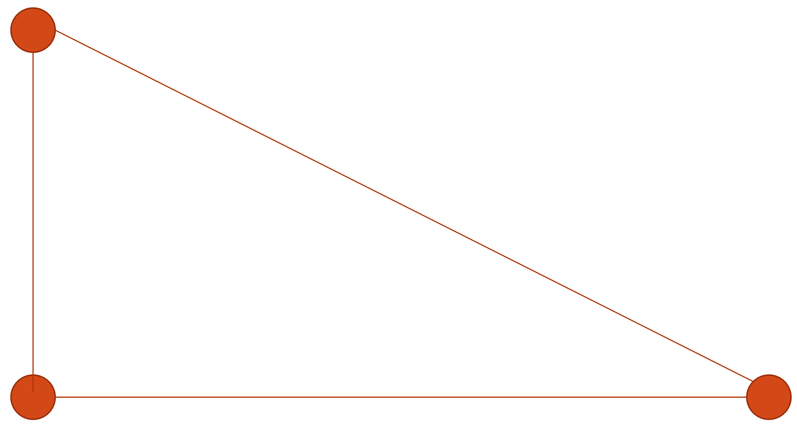
```
class NhanVien
{
    private string maNV;
    private string tenNV;
    private Date ngaySinh;
    private int songaylamviec;
    private int sosanpham;
    private bool isSaler;

    public int TinhLuong()
    {
        if(isSaler)
            return songaylamviec* 5 + sosanpham * 2;
        return songaylamviec * 10;
    }
}
```

Đoạn code sẽ khó quản lý do 1 lớp đối tượng quản lý 2 nhiệm vụ

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)



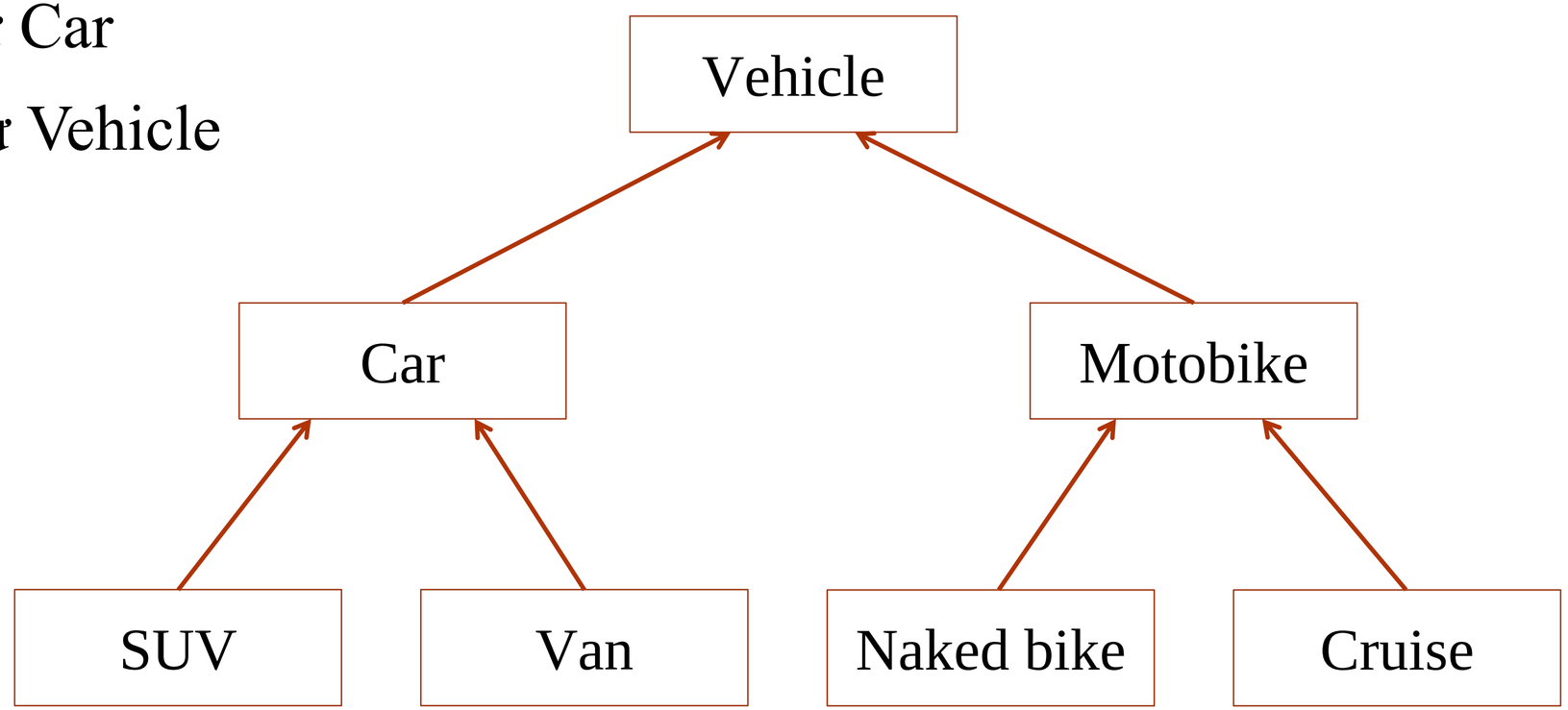


So sánh Kế thừa và Kết tập

Kế thừa	Kết tập
▪ Tái sử dụng mã nguồn thông qua lớp ▪ Quan hệ "là một loại" ▪ Ví dụ: Tam giác vuông là một loại tam giác	▪ Tái sử dụng mã nguồn thông qua đối tượng ▪ Quan hệ "là một phần" ▪ Ví dụ: Tam giác có 3 đỉnh

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



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 kế thừa: một lớp chỉ có thể kế thừa từ duy nhất 1 lớp khác

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

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(){};
}
```

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
}
```

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
}
```

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(){};
}
```

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
}
```

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

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
  - **Gọi trực tiếp** (tường minh): nếu lớp cha có phương thức khởi tạo khác mặc định

Cú pháp: super(parameterList)

Ví dụ

```
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
```

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;
   }
}
```

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);
    }
```

Ví dụ

```
/**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;
}
```

Đối tượng cha và con – Ví dụ

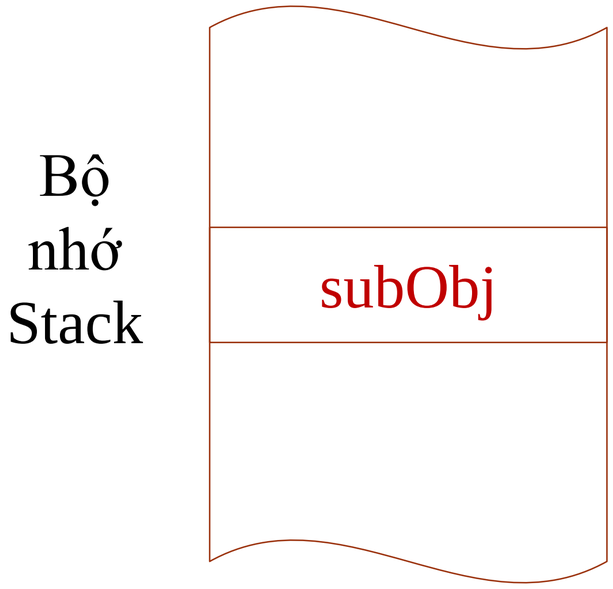
```
public class Base{
   public String pubData;
   private String prvData;
   public Base(){
      System.out.println(“Base”);
      prvData = “private”;
   }
   public String getPrvData(){
      return prvData;
   }
}
```

```
public class Sub extends Base{
   public Sub(){
      //truy cập tới thành viên
      //của cha trong lớp con
      pubData = “public”;
      System.out.println(“Sub”);
   }
}
```

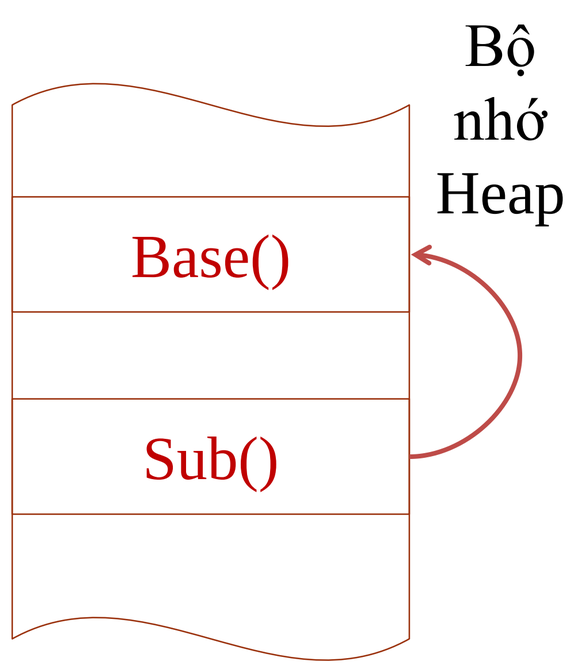
```
public class Test{
   public static void main(String[] args){
      Sub subObj = new Sub();
      //truy cập tới thành viên của cha qua đối tượng con
      System.out.println(subObj.pubData);
      System.out.println(subObj.getPrvData());
   }
}
```

Giải thích

- Khi khởi tạo đối tượng con, đối tượng cha được tạo ra và độc lập với đối tượng con này.
- Tuy nhiên, không thể truy cập từ ngoài tới đối tượng cha vì đối tượng cha là không tường minh.
- Đối tượng con có tham chiếu tới đối tượng cha qua từ khóa super (tham chiếu này là private)



Sub subObj = new Sub();



Giải thích

- Trong lớp con, nếu truy cập tới thuộc tính/phương thức của cha thì truy cập đó có được là qua từ khóa super

```
//từ khóa super được dùng không tường minh
pubData = “public”;
// lời gọi tương đương khi dùng từ khóa super tường minh
super.pubData = “public”;
```

- **Bản chất của kế thừa**: đối tượng con có thể truy cập tới cha của nó qua từ khóa super (tường minh, hoặc không tường minh)
  - Kế thừa không có nghĩa là truyền thuộc tính/phương thức từ sở hữu của cha sang sở hữu của con
  - Trong ví dụ, lớp con không có thuộc tính pubData và phương thức getPrvData()

Giải thích

- Khi truy cập tới một phương thức của lớp cha qua đối tượng con kế thừa, thì đối tượng con đó đã được nhìn nhận như là một đối tượng thuộc lớp cha (upcasting)
- Ví dụ: đối tượng mà subObj tham chiếu tới được nhìn nhận như là đối tượng thuộc lớp Base

  System.out.println(subObj.pubData);

  System.out.println(subObj.getPrvData());
- Có thể hiểu lời gọi trên là

  subObj.super.pubData

  subObj.super.getPrvData()

Mặc dù khi lập trình, nếu viết như vậy sẽ bị báo lỗi (vì tham chiếu super ở lớp con là private).

Kế thừa – Một số ví dụ khác

```
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);
   }
}
```

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

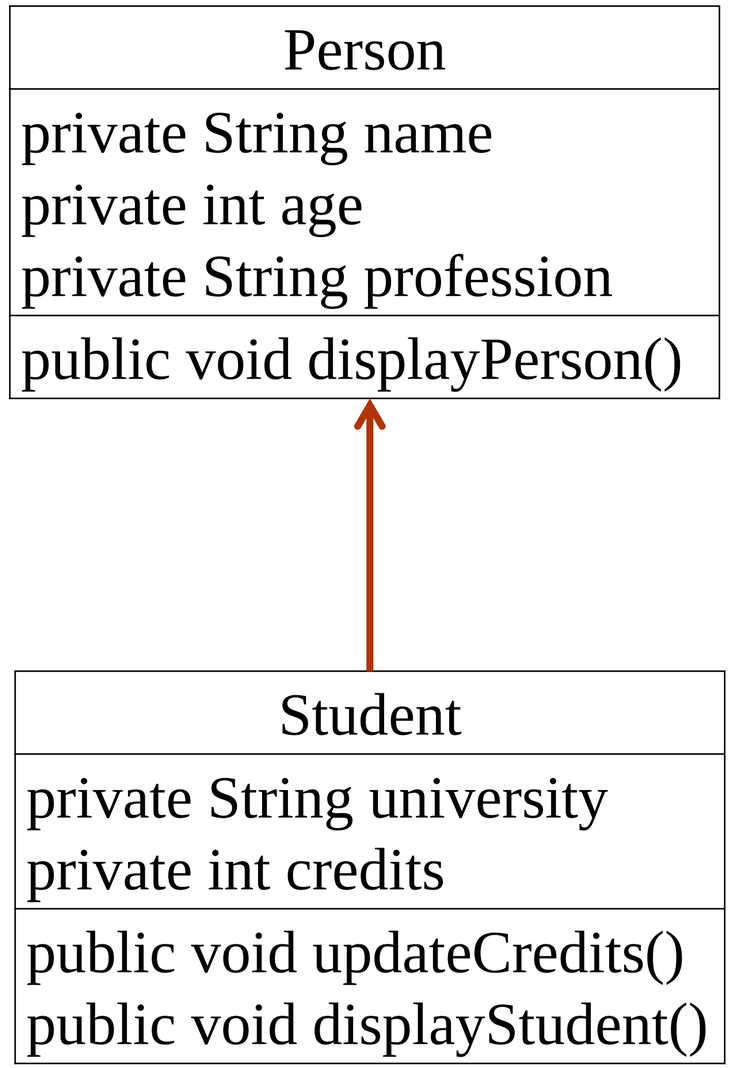
Kế thừa – Một số ví dụ khác

```
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 không có phương thức withdraw() để rút tiền từ ví của mình. Do đó khi thực hiện lời gọi son.withdraw() thì đối tượng được tham chiếu bởi son được tự động nhìn nhận như là đối tượng thuộc lớp cha Father, và thực hiện rút tiền từ ví của cha

Kế thừa – Ví dụ đầy đủ

- Lớp Person:
  - name: tên
  - age: tuổi
  - profession: nghề nghiệp
  - displayPerson(): 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.



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");
}
```

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);
  }
}
```

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);
   super.setProfession("Student");
   this.university = initUniversity;
   credits = 0;
}
```

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

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(){
     super.displayPerson();
     System.out.println("University: " + this.university);
     System.out.println("Cumulated credits: " + this.credits);
   }
}
```

Gọi phương thức của lớp cha để hiển thị giá trị các thuộc tính của lớp cha

Lớp StudentManagement

```
java.oop.student;
import java.oop.person.Person;
public class StudentManagement {
    public static void main(String[] args) {
        Person someone = new Person("Le Van Son",18);
        someone.displayPerson();
        System.out.println("\nSon becomes a student at DNU");
        String name = someone.getName();
        int age = someone.getAge();
        Student dnStudent = new Student(name, age, "DNU");
        dnStudent.displayStudent();
        System.out.println("\nSon has just passed the Java Programming course");
        dnStudent.updateCredits(3);
        dnStudent.displayStudent();
    }
}
```

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

Lớp Student – Không sử dụng kế thừa

```
public class Student {
    private String name;
    private int age;
    private String profession;
    private String university;
    private int credits;
    public Student(String initName, int initAge, String  initUniversity){
        name = initName;
        age = initAge;
        profession = "Student";
        this.university = initUniversity;
        credits = 0;
}
```

Lớp Student – Không sử dụng kế thừa (tiếp)

```
public void displayStudent(){
    System.out.println("Full name: " + this.name);
    System.out.println("Age: " + this.age);
    System.out.println("Profession: " + this.profession);
    System.out.println("University: " + this.university);
    System.out.println("Cumulated credits: " + this.credits);
    }
}
```

- Dễ thấy lớp Student đã không còn tái sử dụng những gì đã sẵn có của lớp Person
- Kế thừa là "tái sử dụng"
- Kế thừa không phải là "tái sở hữu": không chuyển những gì lớp cha có sang lớp con

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
}
```

Ví dụ - Che thuộc tính (tiếp)

```
package java.oop.override.field
public  class  Children  extends  Parents{
   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);
   }
}
```

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 tường minh super.intData = -1;

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

Ví dụ

- Chúng ta sẽ viết lại phương thức để hiển thị thông tin trong lớp Student

```
/**Display the information of the student by overriding the method
displayPerson() of the superclass*/
public void displayPerson(){
  super.displayPerson();
  System.out.println("University: " + this.university);
  System.out.println("Cumulated credits: " + this.credits);
  }
}
```

Trở lại với ví dụ về Father với Child

- 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);
  }
}
```

Father và Child

```
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() để rút tiền từ ví của mình. Do đó khi thực hiện lời gọi son.withdraw() 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

Father và Child – Chồng phương thức

- Chúng ta sẽ 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);
  }
}
```

Father và Child – Chồng phương thức

```
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

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 lạc 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);
   }
}
```

Từ khoá 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
}
```