NGÔN NGỮ LẬP TRÌNH JAVA

Nội dung

Cơ bản về ngôn ngữ lập trình Java

- Lập trình hướng đối tượng
- Biến, từ khoá, kiểu dữ liệu
- Biểu thức, các cấu trúc điều khiển
- Dữ liệu kiểu mảng

Các khía cạnh nâng cao của lập trình hướng đối tượng

- Thiết kế lớp
- Thiết kế lớp nâng cao
- Xử lý ngoại lệ

Xây dựng ứng dụng Java

- Tạo giao diện đồ hoạ
- Xử lý các sự kiện trên giao diện đồ hoạ
- Xây dựng ứng dụng đồ hoạ

Lập trình Java nâng cao

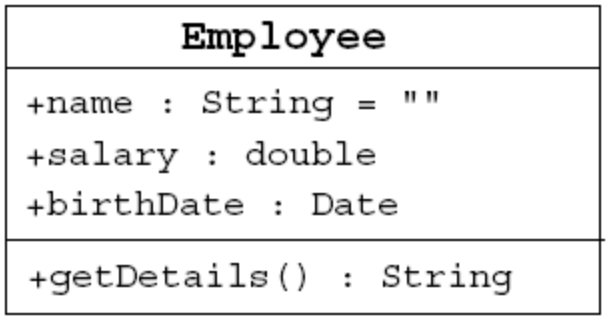
- Luồng
- Vào / Ra
- Lập trình mạng
- Lập trình với CSDL

THIẾT KẾ LỚP

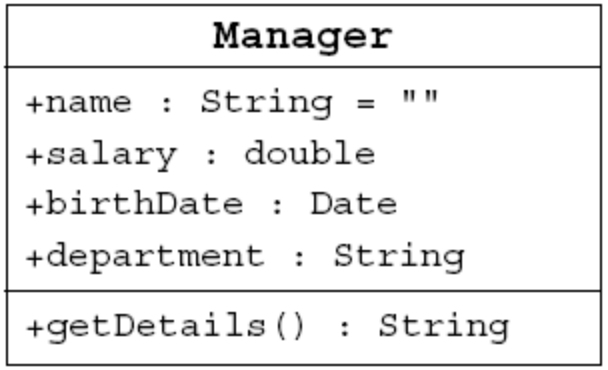
Nội dung

- Định nghĩa `inheritance, polymorphism, overloading, overriding`
- Sử dụng access modifier protected
- Khái niệm hàm khởi tạo và chồng hàm khởi tạo
- Quá trình khởi tạo đối tượng, khởi tạo các thuộc tính của đối tượng.

Lớp con

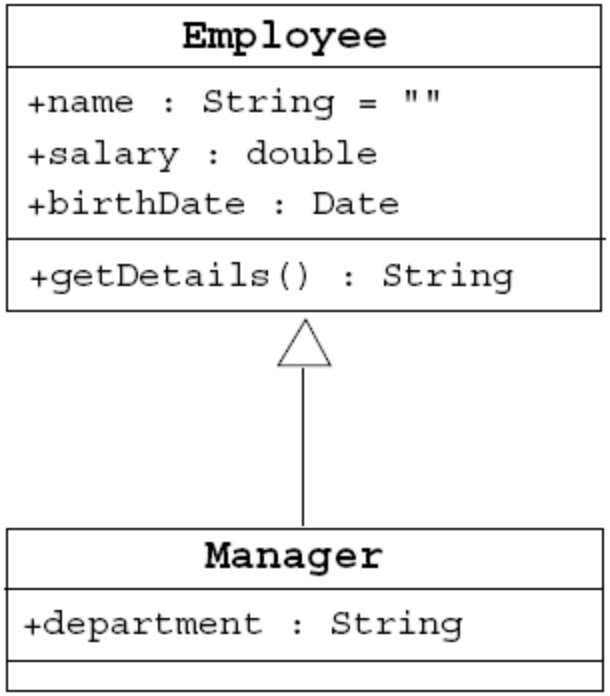



```
public class Employee {
  public String name = "";
  public double salary;
  public Date birthDate;

  public String getDetails() {...}
}
```




```
public class Manager {
  public String name = "";
  public double salary;
  public Date birthDate;
  public String department;

  public String getDetails() {...}
}
```

Lớp con




```
public class Employee {
  public String name = "";
  public double salary;
  public Date birthDate;

  public String getDetails() {...}
}
```

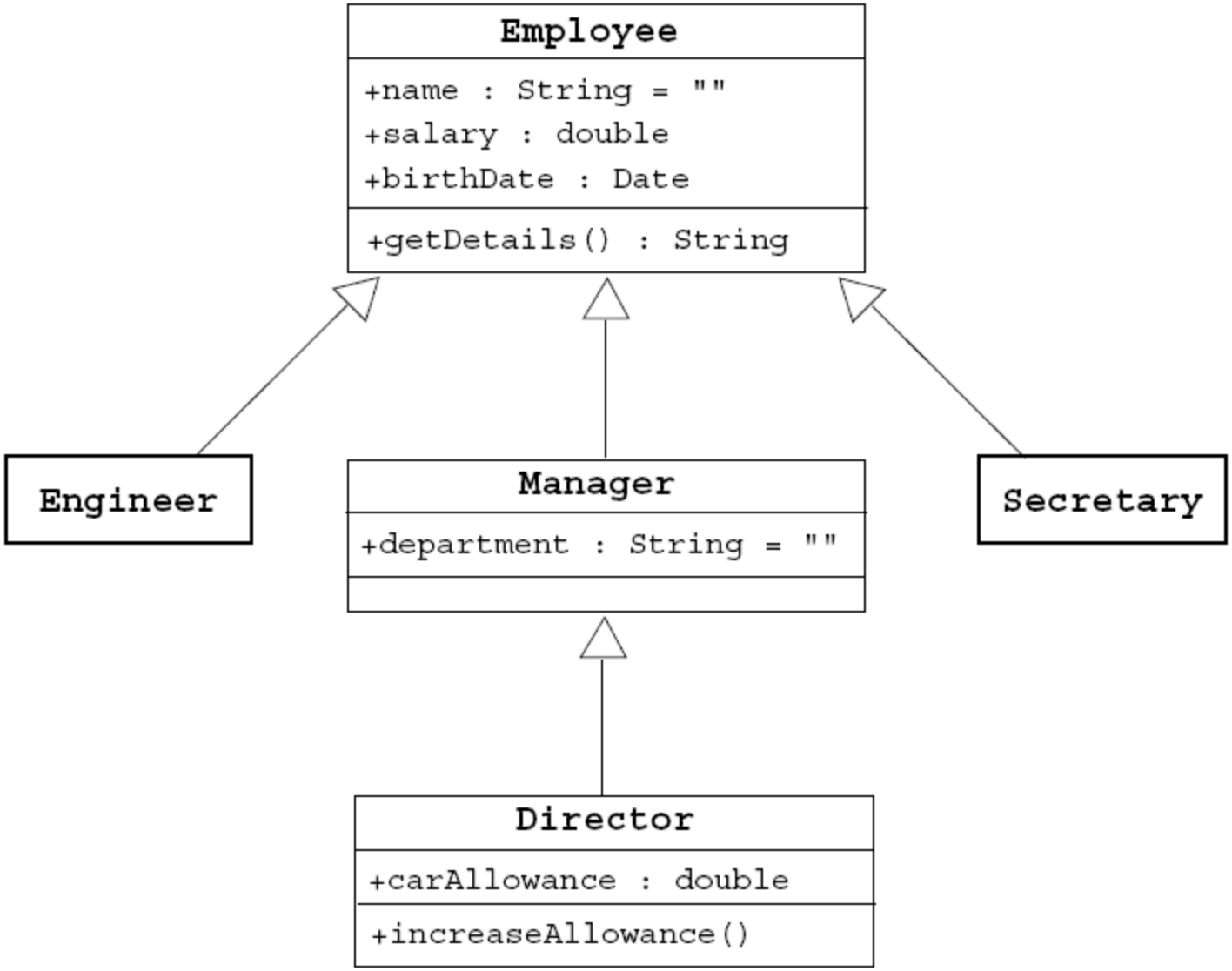
```
public class Manager extends Employee {
  public String department;
}
```

Kỹ thuật đơn thừa kế

- Khi một lớp thừa kế từ một lớp khác, đó là đơn thừa kế
- Interface cung cấp tính năng đa thừa kế cho lớp trong Java
- Cú pháp đơn thừa kế:

```
<modifier> class <name> [extends <superclass>] {
  <declaration>*
}
```

Kỹ thuật đơn thừa kế



Quyền truy cập

Modifier	Same Class	Same Package	Subclass	Universe
`private`	Yes			
default	Yes	Yes		
`protected`	Yes	Yes	Yes	
`public`	Yes	Yes	Yes	Yes

Kỹ thuật đè hàm - overriding

- Một lớp con có thể thay đổi các phương thức ở lớp cha thông qua kỹ thuật đè hàm
- Quy tắc đè hàm:
  - 2 phương thức ở 2 lớp kế thừa có cùng tên
  - cùng tham số
  - cùng kiểu giá trị trả về
  - modifier của phương thức ở lớp con phải rộng hơn hoặc bằng modifier của phương thức ở lớp cha

Kỹ thuật đè hàm

```
1   public class Employee {
2     protected String name;
3     protected double salary;
4     protected Date birthDate;
5
6     public String getDetails() {
7       return "Name: " + name + "\n" +
8              "Salary: " + salary;
9     }
10  }
```

```
1   public class Manager extends Employee {
2     protected String department;
3
4     public String getDetails() {
5       return "Name: " + name + "\n" +
6              "Salary: " + salary + "\n" +
7              "Manager of: " + department;
8     }
9   }
```

Kỹ thuật đè hàm

```
1   public class Parent {
2     public void doSomething() {}
3   }
```

```
1   public class Child extends Parent {
2     private void doSomething() {} // illegal
3   }
```

```
1   public class UseBoth {
2     public void doOtherThing() {
3       Parent p1 = new Parent();
4       Parent p2 = new Child();
5       p1.doSomething();
6       p2.doSomething();
7     }
8   }
```

Tham chiếu super

- Phương thức của lớp con có thể gọi phương thức của lớp cha thông qua tham chiếu super
- Tham chiếu super trỏ tới lớp cha của lớp hiện tại
- Có thể dùng super để truy cập tới thuộc tính, phương thức của lớp cha
- Các phương thức được triệu gọi không nhất thiết phải nằm trong lớp cha, chỉ cần gọi theo đúng mô hình phân cấp là được

Triệu gọi các phương thức overriding

```
1   public class Employee {
2     private String name;
3     private double salary;
4     private Date birthDate;
5
6     public String getDetails() {
7       return "Name: " + name + "\nSalary: " + salary;
8     }
9   }
```

```
1   public class Manager extends Employee {
2     private String department;
3
4     public String getDetails() {
5       // call parent method
6       return super.getDetails()
7              + "\nDepartment: " + department;
8     }
9   }
```

Tính đa hình - polymorphism

- Đa hình là khả năng một đối tượng có nhiều hình thức thể hiện.
- Một đối tượng thì chỉ có một hình thức thể hiện
- Nhưng một biến tham chiếu lại có thể tham chiếu tới các đối tượng với nhiều hình thức thể hiện khác nhau

```
Employee e = new Manager(); // legal

// illegal attempt to assign Manager attribute
e.department = "Sales";
// the variable is declared as an Employee type,
// even though the Manager object has that attribute
```

Ví dụ về tính đa hình

```
public class TaxService {
  public TaxRate findTaxRate(Employee e) {
    // calculate the employee's tax rate
  }
}

// Meanwhile, elsewhere in the application class
TaxService taxSvc = new TaxService();
Manager m = new Manager();
TaxRate t = taxSvc.findTaxRate(m);
```

Phép toán instanceof

```
public class Employee extends Object
public class Manager extends Employee
public class Engineer extends Employee
---------------------------------------

public void doSomething(Employee e) {
  if ( e instanceof Manager ) {
    // Process a Manager
  } else if ( e instanceof Engineer ) {
    // Process an Engineer
  } else {
    // Process any other type of Employee
  }
}
public void doSomething(Employee e) {
  if ( e instanceof Manager ) {
    Manager m = (Manager) e;
    System.out.println("This is the manager of "
                       + m.getDepartment());
  }
  // rest of operation
}
```

Kỹ thuật chồng hàm - overloading

- Là kỹ thuật mà 2 hay nhiều phương thức có cùng tên nhưng khác danh sách tham số

```
public void println(int i)
public void println(float f)
public void println(String s)
```

Kỹ thuật chồng hàm khởi tạo

- Hàm khởi tạo cũng tuân theo quy tắc chồng hàm
- Có thể viết nhiều hàm khởi tạo cho một lớp
- Các hàm khởi tạo cần khác nhau về danh sách tham số
- Có thể sử dụng tham chiếu this để gọi tới một hàm khởi tạo từ một hàm khởi tạo khác

```
public Employee(String name, double salary, Date DoB)
public Employee(String name, double salary)
public Employee(String name, Date DoB)
```

Kỹ thuật chồng hàm khởi tạo

```
1   public class Employee {
2     private static final double BASE_SALARY = 15000.00;
3     private String name;
4     private double salary;
5     private Date   birthDate;
6
7     public Employee(String name, double salary, Date DoB) {
8       this.name = name;
9       this.salary = salary;
10      this.birthDate = DoB;
11    }
12    public Employee(String name, double salary) {
13      this(name, salary, null);
14    }
15    public Employee(String name, Date DoB) {
16      this(name, BASE_SALARY, DoB);
17    }
18    // more Employee code...
19  }
```

Lưu ý về hàm khởi tạo

- Lớp con được thừa kế tất cả các thành phần, phương thức, thuộc tính từ lớp cha trừ HÀM KHỞI TẠO
- Sử dụng super() để gọi tới hàm khởi tạo không tham số của lớp cha
- Sử dụng this() để gọi tới hàm khởi tạo không tham số của lớp hiện tại
- 2 cách để đưa hàm khởi tạo vào lớp:
  - Sử dụng hàm khởi tạo mặc định
  - Viết hàm khởi tạo tường minh

Triệu gọi hàm khởi tạo của lớp cha

```
1   public class Manager extends Employee {
2     private String department;
3
4     public Manager(String name, double salary, String dept) {
5       super(name, salary);
6       department = dept;
7     }
8     public Manager(String name, String dept) {
9       super(name);
10      department = dept;
11    }
12    public Manager(String dept) { // This code fails: no super()
13      department = dept;
14    }
15    //more Manager code...
16  }
```

Phương thức với số tham số không xác định

```
public class Statistics {
  public float average(int... nums) {
    int sum = 0;
    for ( int x : nums ) {
      sum += x;
    }
    return ((float) sum) / nums.length;
  }
}
```

Lớp Object

- Là lớp cha của mọi lớp trong Java
- Khi ta khai báo:

```
public class Employee {
...
}
```

thì cũng tương đương với

```
public class Employee extends Object {
...
}
```

- 2 phương thức quan trọng trong lớp Object
  - `equals`
  - `toString`

Phương thức equals

- Phép toán == trả về true nếu 2 tham chiếu trỏ đến nhau hay nói cách khác, chúng là một
- Phép toán `equals` nói lên rằng 2 object có giá trị bằng nhau nhưng không trỏ đến nhau
- Ta có thể cài đặt lại phương thức `equals` trong mỗi lớp để định nghĩa xem như thế nào là có giá trị bằng nhau

Phương thức equals

```
1   public class MyDate {
2     private int day;
3     private int month;
4     private int year;
5
6     public MyDate(int day, int month, int year) {
7       this.day   = day;
8       this.month = month;
9       this.year  = year;
10    }
```

Phương thức equals

```
11
12    public boolean equals(Object o) {
13      boolean result = false;
14      if ( (o != null) && (o instanceof MyDate) ) {
15        MyDate d = (MyDate) o;
16        if ( (day == d.day) && (month == d.month)
17             && (year == d.year) ) {
18          result = true;
19        }
20      }
21      return result;
22    }
23
24    public int hashCode() {
25      return (day ^ month ^ year);
26    }
27  }
```

Phương thức equals

```
1   class TestEquals {
2     public static void main(String[] args) {
3       MyDate  date1 = new MyDate(14, 3, 1976);
4       MyDate  date2 = new MyDate(14, 3, 1976);
5
6       if ( date1 == date2 ) {
7         System.out.println("date1 is identical to date2");
8       } else {
9         System.out.println("date1 is not identical to date2");
10      }
11
12      if ( date1.equals(date2) ) {
13        System.out.println("date1 is equal to date2");
14      } else {
15        System.out.println("date1 is not equal to date2");
16      }
17
18      System.out.println("set date2 = date1;");
19      date2 = date1;
20
21      if ( date1 == date2 ) {
22        System.out.println("date1 is identical to date2");
23      } else {
24        System.out.println("date1 is not identical to date2");
25      }
26    }
27  }
```

Các lớp Wrapper

Primitive Data Type	Wrapper Class
boolean	Boolean
byte	Byte
char	Character
short	Short
int	Integer
long	Long
float	Float
double	Double

THIẾT KẾ LỚP NÂNG CAO

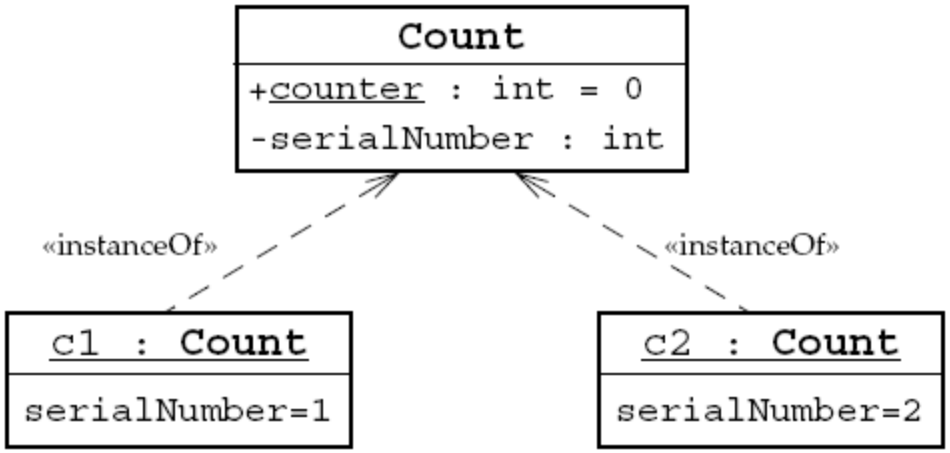
Nội dung

- Từ khoá `static` cho biến, phương thức, lớp.
- Đoạn khởi tạo tĩnh
- Từ khoá `final` cho biến, phương thức, lớp
- Kiểu dữ liệu liệt kê
- Từ khoá `abstract`
- Tạo và sử dụng `interface`

Từ khoá static

- Từ khoá `static` được dùng như một modifier dành cho biến, phương thức và lớp
- Khi khai báo một biến hoặc phương thức là static thì biến, phương thức đó gắn với lớp chứ không gắn với thể hiện của lớp
- Khi đó, những thành phần tĩnh thường được gọi là thành phần của lớp

Thành phần static




```
1   public class Count {
2     private int serialNumber;
3     public static int counter = 0;
4
5     public Count() {
6       counter++;
7       serialNumber = counter;
8     }
9   }
```

Thành phần static

- Nếu thành phần tĩnh là public:

```
1   public class Count1 {
2     private int serialNumber;
3     public static int counter = 0;
4     public Count1() {
5       counter++;
6       serialNumber = counter;
7     }
8   }
```

nó có thể được truy cập từ bên ngoài lớp mà ko cần tạo thể hiện:

```
1   public class OtherClass {
2     public void incrementNumber() {
3       Count1.counter++;
4     }
5   }
```

Ngữ cảnh static

- Trong một ngữ cảnh tĩnh, môi trường tĩnh, không được sử dụng các instance variable

```
1   public class Count3 {
2     private int serialNumber;
3     private static int counter = 0;
4
5     public static int getSerialNumber() {
6       return serialNumber;  // COMPILER ERROR!
7     }
8   }
```

Đoạn khởi tạo tĩnh

- Trong một lớp, có thể có 1 đoạn code tĩnh không thuộc bất cứ phương thức nào
- Đoạn code tĩnh được thực thi duy nhất 1 lần khi class đó được load lên trong JVM
- Thông thường, đoạn code tĩnh được dùng để khởi tạo giá trị cho các thành phần tĩnh trong lớp

Đoạn khởi tạo tĩnh

```
1   public class Count4 {
2     public static int counter;
3     static {
4       counter = Integer.getInteger("myApp.Count4.counter").intValue();
5     }
6   }
```

```
1   public class TestStaticInit {
2     public static void main(String[] args) {
3       System.out.println("counter = "+ Count4.counter);
4     }
5   }
```

The output of the `TestStaticInit` program is:

```
java -DmyApp.Count4.counter=47 TestStaticInit
counter = 47
```

Từ khoá final

- Từ khoá `final` dành cho biến, phương thức và lớp
- Khi một lớp được thiết lập `final`, nó sẽ không thể được kế thừa
- Khi một phương thức là `final`, nó sẽ không thể được cài đè (overriding)
- Một biến được thiết lập `final` thì nó trở thành hằng số
- Ta chỉ được thiết lập final cho biến một lần.
- Hằng số có thể được gán giá trị ngay hoặc 1 lần sau đó.

Hằng số

```
public class Bank {
  private static final double  DEFAULT_INTEREST_RATE = 3.2;
  ... // more declarations
}
```

```
1   public class Customer {
2
3     private final long customerID;
4
5     public Customer() {
6       customerID = createID();
7     }
8
```

Kiểu dữ liệu liệt kê với từ khoá enum

```
1   package cards.domain;
2
3   public enum Suit {
4     SPADES,
5     HEARTS,
6     CLUBS,
7     DIAMONDS
8   }
```

```
1   package cards.domain;
2
3   public class PlayingCard {
4
5     private Suit suit;
6     private int rank;
7
8     public PlayingCard(Suit suit, int rank) {
9       this.suit = suit;
10      this.rank = rank;
11    }
12
13    public Suit getSuit() {
14      return suit;
15    }
```

Kiểu dữ liệu liệt kê với hàm khởi tạo

```
1   package cards.domain;
2
3   public enum Suit {
4     SPADES   ("Spades"),
5     HEARTS   ("Hearts"),
6     CLUBS    ("Clubs"),
7     DIAMONDS ("Diamonds");
8
9     private final String name;
10
11    private Suit(String name) {
12      this.name = name;
13    }
14
15    public String getName() {
16      return name;
17    }
18  }
```

Câu lệnh import tĩnh

- Có thể sử dụng câu `import` tĩnh để import từng thành phần của lớp

```
import static <pkg_list>.<class_name>.<member_name>;
OR
import static <pkg_list>.<class_name>.*;
```

- Ví dụ:

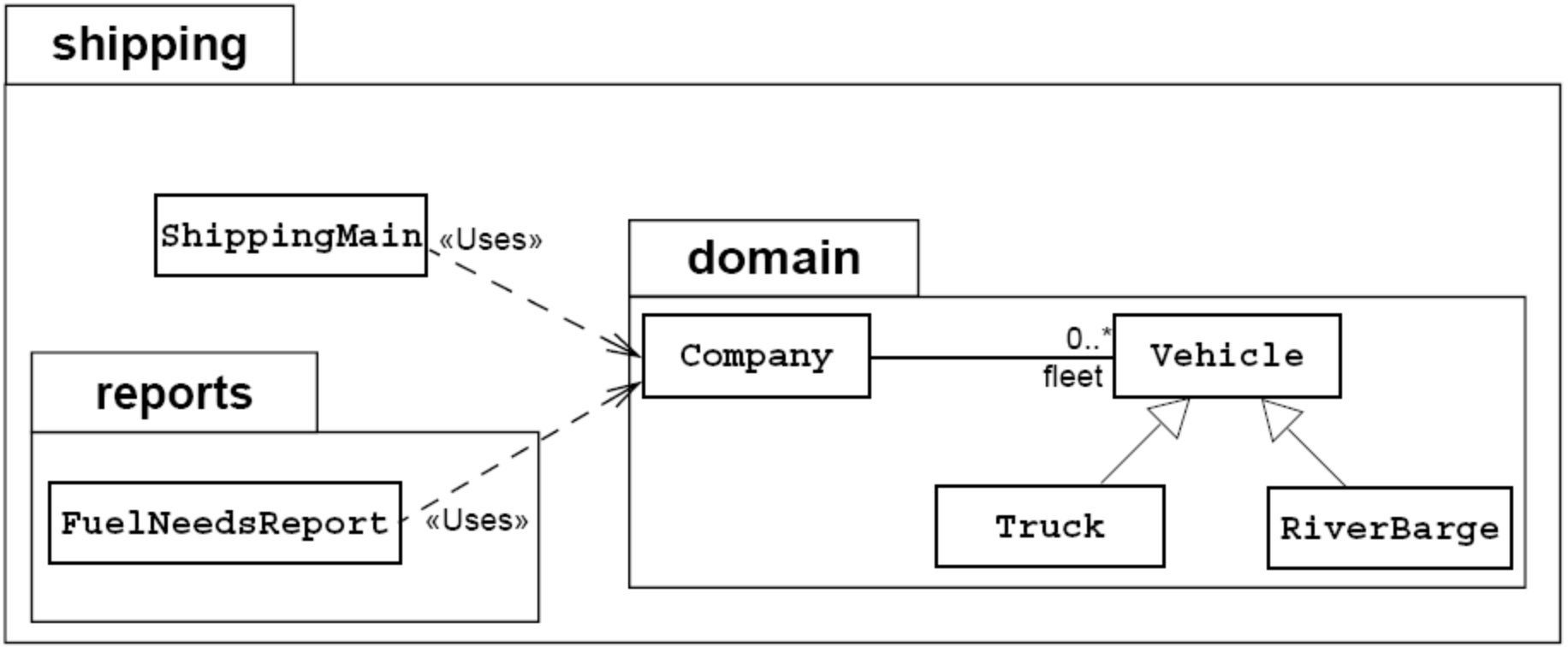
```
import static cards.domain.Suit.SPADES;
OR
import static cards.domain.Suit.*;
```

Ví dụ về import tĩnh

```
1   package cards.tests;
2
3   import cards.domain.PlayingCard;
4   import static cards.domain.Suit.*;
5
6   public class TestPlayingCard {
7     public static void main(String[] args) {
8
9       PlayingCard card1 = new PlayingCard(SPADES, 2);
10      System.out.println("card1 is the " + card1.getRank()
11                         + " of " + card1.getSuit().getName());
12
13      // NewPlayingCard card2 = new NewPlayingCard(47, 2);
14      // This will not compile.
15    }
16  }
```

Từ khoá abstract

- Từ khoá abstract sử dụng cho lớp và phương thức



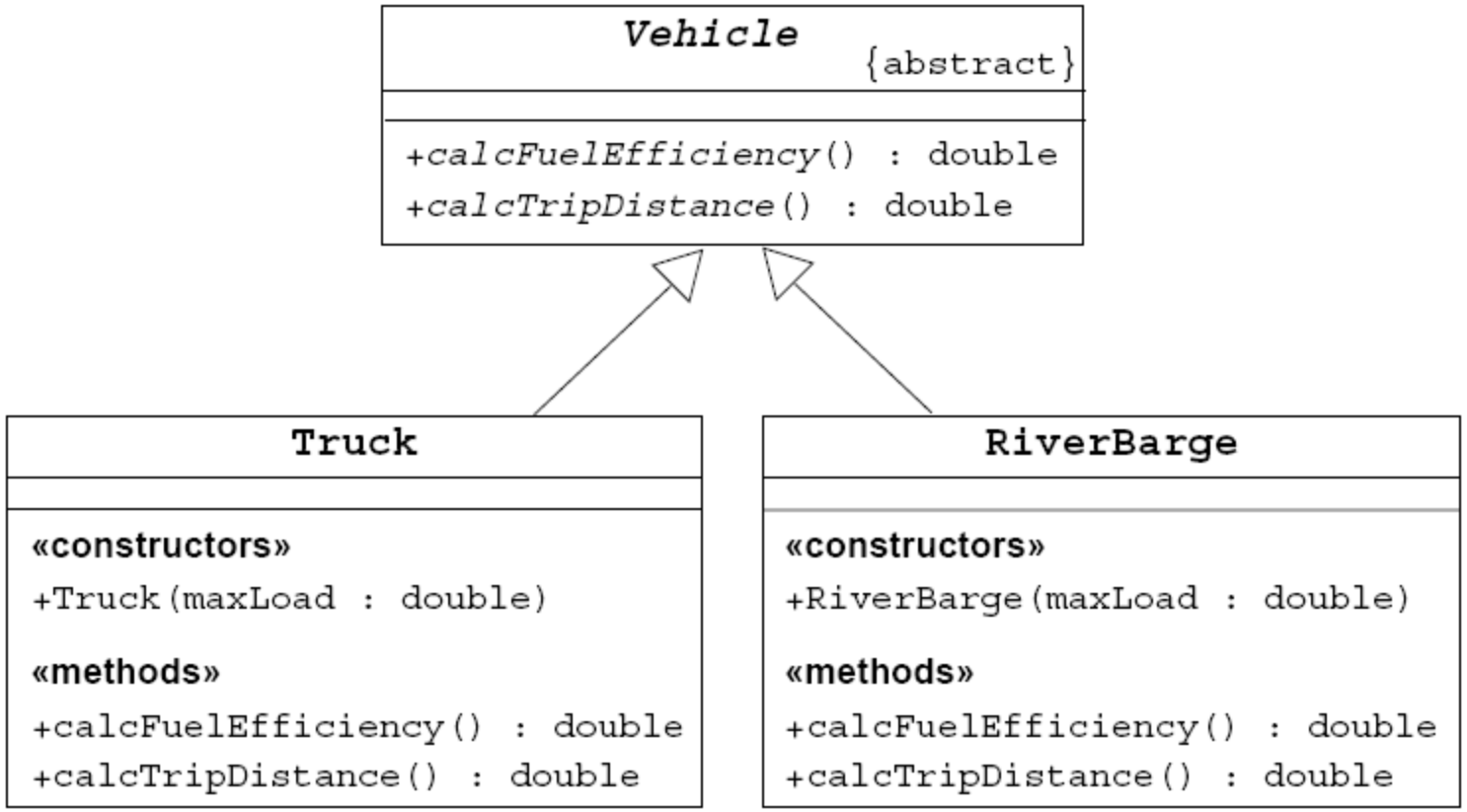
Lớp abstract

```
1   public class ShippingMain {
2     public static void main(String[] args) {
3       Company c = new Company();
4
5       // populate the company with a fleet of vehicles
6       c.addVehicle( new Truck(10000.0) );
7       c.addVehicle( new Truck(15000.0) );
8       c.addVehicle( new RiverBarge(500000.0) );
9       c.addVehicle( new Truck(9500.0) );
10      c.addVehicle( new RiverBarge(750000.0) );
11
12      FuelNeedsReport report = new FuelNeedsReport(c);
13      report.generateText(System.out);
14    }
15  }
```

Lớp abstract

```
1   public class FuelNeedsReport {
2     private Company company;
3
4     public FuelNeedsReport(Company company) {
5       this.company = company;
6     }
7
8     public void generateText(PrintStream output) {
9       Vehicle1 v;
10      double fuel;
11      double total_fuel = 0.0;
12
13      for ( int i = 0; i < company.getFleetSize(); i++ ) {
14        v = company.getVehicle(i);
15
16        // Calculate the fuel needed for this trip
17        fuel = v.calcTripDistance() / v.calcFuelEfficency();
18
19        output.println("Vehicle " + v.getName() + " needs "
20                       + fuel + " liters of fuel.");
21        total_fuel += fuel;
22      }
23      output.println("Total fuel needs is " + total_fuel + " liters.");
24    }
25  }
```

Giải pháp - sử dụng lớp abstract



Giải pháp - sử dụng lớp abstract

```
1   public abstract class Vehicle {
2     public abstract double calcFuelEfficiency();
3     public abstract double calcTripDistance();
4   }
```

```
1   public class Truck extends Vehicle {
2     public Truck(double maxLoad) {...}
3     public double calcFuelEfficiency() {
4       /* calculate the fuel consumption of a truck at a given load */
5     }
6     public double calcTripDistance() {
7       /* calculate the distance of this trip on highway */
8     }
9   }
```

```
1   public class RiverBarge extends Vehicle {
2     public RiverBarge(double maxLoad) {...}
3     public double calcFuelEfficiency() {
4       /* calculate the fuel efficiency of a river barge */
5     }
6     public double calcTripDistance() {
7       /* calculate the distance of this trip along the river-ways */
8     }
9   }
```

Từ khoá abstract

- Khi một phương thức được thiết lập abstract, nó trở thành phương thức trừu tượng
- Phương thức trừu tượng chỉ có nguyên mẫu, không có nội dung. Kết thúc bằng dấu (;)
- Phương thức trừu tượng mặc định là public
- Một lớp chứa một phương thức trừu tượng thì nó là trừu tượng
- Lớp abstract (trừu tượng) thì mặc định là public
- Khi một lớp thừa kế một lớp trừu tượng, nó phải cài đặt lại tất cả các phương thức trừu tượng trong lớp đó.

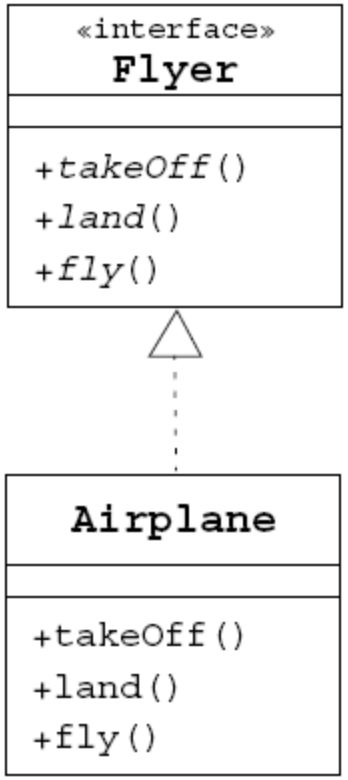
Sử dụng interface

- Interface được coi là giao tiếp giữa client code và class cài đặt interface
- Trong interface chỉ có các nguyên mẫu (prototype) của phương thức. Không có phần cài đặt của phương thức
- Các phương thức trong interface mặc định là public
- Interface mặc định là public
- Nhiều class có thể cài đặt cùng một interface
- Một class có thể cài đặt nhiều interface, điều này tạo nên tính đa thừa kế trong Java

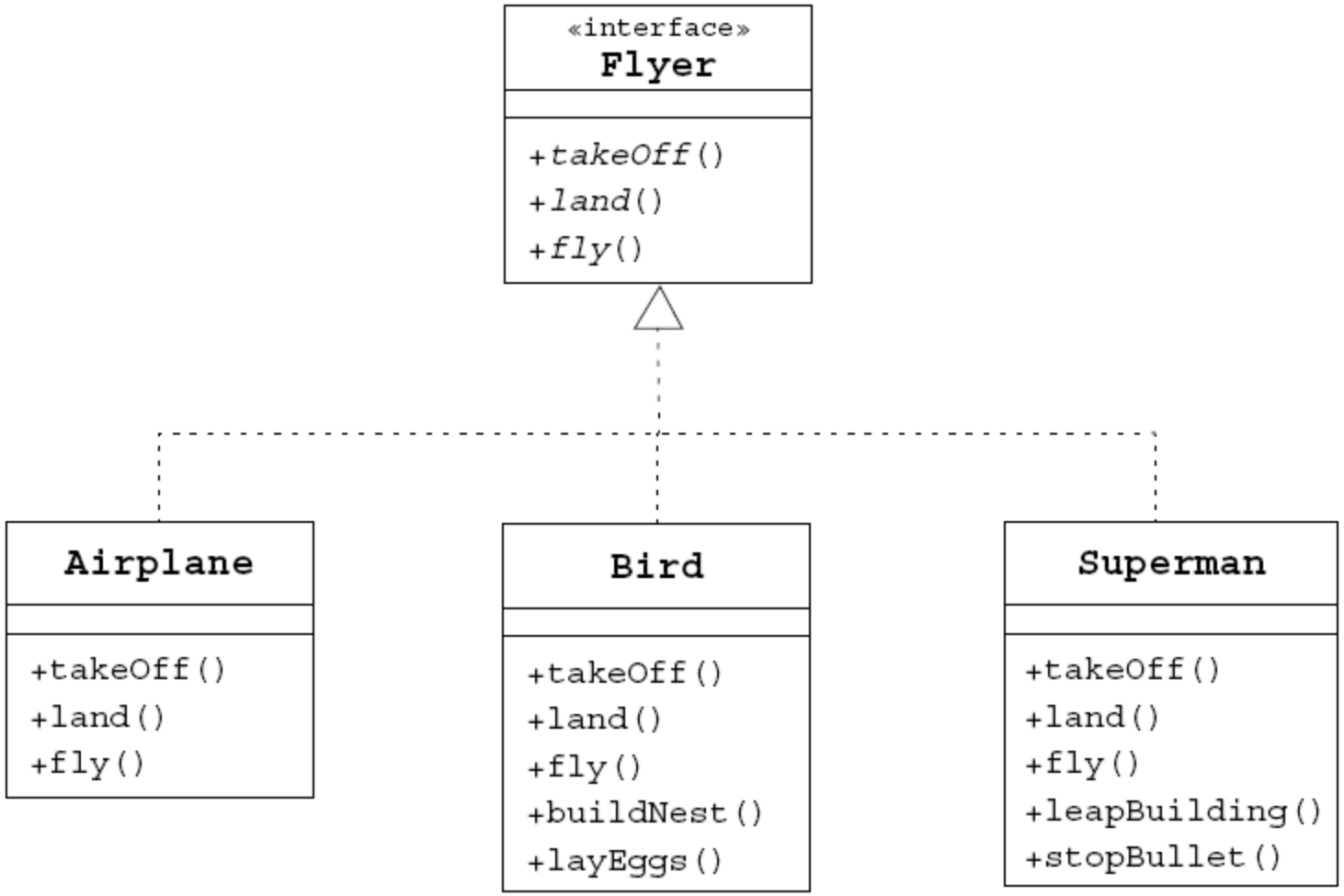
```
<modifier> class <name> [extends <superclass>]
            [implements <interface> [,<interface>]* ] {
  <member_declaration>*
}
```

Sử dụng interface

```
public interface Flyer {
  public void takeOff();
  public void land();
  public void fly();
}

public class Airplane implements Flyer {
  public void takeOff() {
    // accelerate until lift-off
    // raise landing gear
  }
  public void land() {
    // lower landing gear
    // decelerate and lower flaps until touch-down
    // apply brakes
  }
  public void fly() {
    // keep those engines running
  }
}
```



Sử dụng interface



Sử dụng interface