链滴

Java 基础 - 面向对象

作者：Gao-Eason

原文链接：https://ld246.com/article/1649815503222

来源网站：链滴

Java基础-面向对象

本文主要介绍Java OOP 面向对象基础和相关类图。@Eason | Gao

三大特性

封装

利用抽象数据类型将数据和基于数据的操作封装在一起，使其构成一个不可分割的独立实体。数据被护在抽象数据类型的内部，尽可能地隐藏内部的细节，只保留一些对外接口使之与外部发生联系。用无需知道对象内部的细节，但可以通过对象对外提供的接口来访问该对象。

优点:

- **减少耦合: 可以独立地开发、测试、优化、使用、理解和修改**
- **减轻维护的负担: 可以更容易被程序员理解，并且在调试的时候可以不影响其他模块**
- **有效地调节性能: 可以通过剖析确定哪些模块影响了系统的性能**
- **提高软件的可重用性**
- **降低了构建大型系统的风险: 即使整个系统不可用，但是这些独立的模块却有可能是可用的**

以下 Person 类封装 name、gender、age 等属性，外界只能通过 get() 方法获取一个 Person 对的 name 属性和 gender 属性，而无法获取 age 属性，但是 age 属性可以供 work() 方法使用。

注意到 gender 属性使用 int 数据类型进行存储，封装使得用户注意不到这种实现细节。并且在需要改 gender 属性使用的数据类型时，也可以在不影响客户端代码的情况下进行。

```
public class Person {
```

```
    private String name;
    private int gender;
    private int age;

    public String getName() {
        return name;
    }

    public String getGender() {
        return gender == 0 ? "man" : "woman";
    }

    public void work() {
        if (18 <= age && age <= 50) {
            System.out.println(name + " is working very hard!");
        } else {
            System.out.println(name + " can't work any more!");
        }
    }
}
```

继承

继承实现IS-A关系，例如 Cat 和 Animal 就是一种 IS-A 关系，因此 Cat 可以继承自 Animal，从而得 Animal 非 private 的属性和方法。

继承应该遵循里氏替换原则，子类对象必须能够替换掉所有父类对象。

Cat 可以当做 Animal 来使用，也就是说可以使用 Animal 引用 Cat 对象。父类引用指向子类对象为 向上转型 。

```
Animal animal = new Cat();
```

多态

- **理解多态性： 一种事物的多种形态**
- **何为多态性?**

对象的多态性： 父类的的引用指向子类的对象（或者子类的对象吧赋给父类的引用）

多态分为编译时状态和运行时状态：

- **编译时状态主要是指方法的重载**

```
Animal s = new Cat();
```

- **详解：对于编译器来说，编译器只知道s的类型是Animal，所以编译器在检查语法的时候会去Animal.class字节码文件中**

找到move（）方法，找到的话就进行绑定，此次编译通过，绑定成功。（编译阶段属于静态绑定）

- **运行时状态是指程序中定义的对象引用所指向的具体类型在运行时间才会确定。**

● 详解： 运行阶段的时候，实际上会在堆内存中创建Java对象 ， 这个Java对象实际上就是Cat对象 所以在执行

move（）方法的时候，实际上真正参与到move（）方法的真正的对象的是Cat（一只猫），所以运阶段会动态执行

Cat对象的move（）方法。 （运行阶段属于动态绑定）

但是运行的时候和底层堆内存中的实际对象有关 ,真正执行的时候会调用真实对象的相关方法 .

多态总结：编译看左边，执行（运行）看右边。

● 对象的多态性：只适用于方法，不适用于属性。

● 多态性的使用前提：①类的继承关系 ② 方法的重写

运行时多态有三个条件

● 继承

● 覆盖（重写）

● 向上转型

下面的代码中，乐器类(Instrument)有两个子类: Wind 和 Percussion，它们都覆盖了父类的 play() 方法，并且在 main() 方法中使用父类 Instrument 来引用 Wind 和 Percussion 对象。在 Instrum nt 引用调用 play() 方法时，会执行实际引用对象所在类的 play() 方法，而不是 Instrument 类的 法。

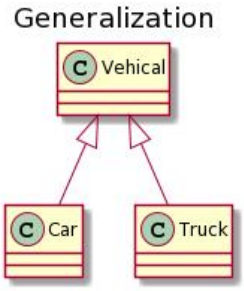
```
public class Instrument {
    public void play() {
        System.out.println("Instument is playing...");
    }
}

public class Wind extends Instrument {
    public void play() {
        System.out.println("Wind is playing...");
    }
}

public class Percussion extends Instrument {
    public void play() {
        System.out.println("Percussion is playing...");
    }
}

public class Music {
    public static void main(String[] args) {
        List<Instrument> instruments = new ArrayList<>();
        instruments.add(new Wind());
        instruments.add(new Percussion());
        for(Instrument instrument : instruments) {
            instrument.play();
        }
    }
```

```
}
```

类图

泛化关系(Generalization)

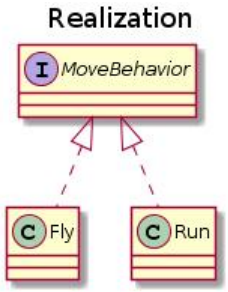
用来描述继承关系，在Java中使用extends关键字




```
@startuml

title Generalization

class Vehical
class Car
class Truck

Vehical <|-- Car
Vehical <|-- Truck

@enduml
```

泛化关系(Generalization)

用来实现一个接口，在Java中使用implements关键字

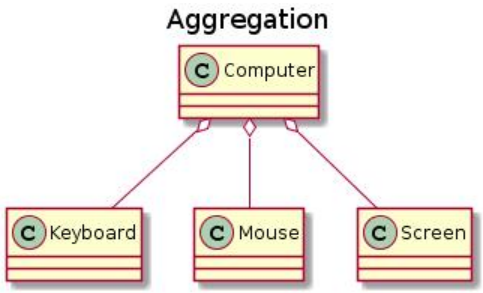



```
@startuml

title Realization

interface MoveBehavior
class Fly
class Run
```

```
MoveBehavior <|.. Fly
MoveBehavior <|.. Run

@enduml
```

聚合关系(Aggregation)

表示整体由部分组成

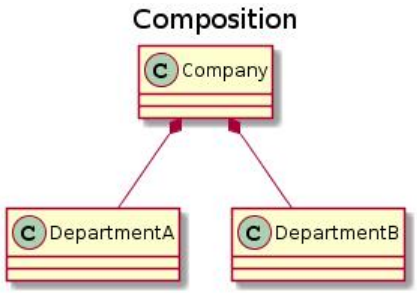



```
@startuml

title Aggregation

class Computer
class Keyboard
class Mouse
class Screen

Computer o-- Keyboard
Computer o-- Mouse
Computer o-- Screen

@enduml
```

组合关系 (Composition)

和聚合不同，组合中整体和部分是强依赖的，整体不存在了，部分也会同时消失。比如公司和部门，司没了部分也不存在了。

但是公司和员工就是属于聚合关系，因为公司没了员工还在

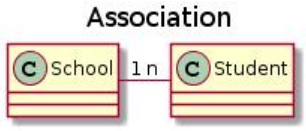



```
@startuml

title Composition
```

```
class Company
class DepartmentA
class DepartmentB

Company *-- DepartmentA
Company *-- DepartmentB

@enduml
```

关联关系 (Association)

表示不同类对象之间有关联，这是一种静态关系，与运行程序的状态无关，在最开始就可以确定。因可以用1对1、多对1、多对多这种关联关系来表示。比如学生和学校就是一种关联关系，一个学校可有很多学生，但是一个学生只属于一个学校，因此这是一种多对1的关系，在运行开始之前就可以确。

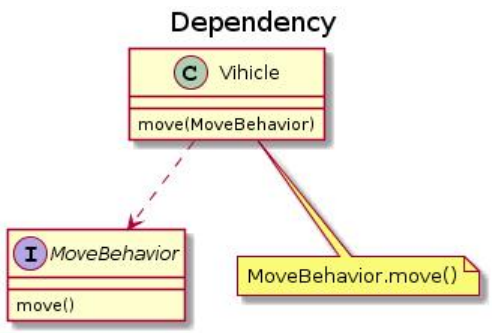



```
@startuml

title Association

class School
class Student

School "1" - "n" Student

@enduml
```

依赖关系 (Dependency)

和 关联关系不同的是，依赖关系是在运行阶段起作用的。A类和B类是依赖关系中主要的三种表现形式：

- **A类是B类中（某种方法的）局部变量。**
- **A类是B类方法的一个参数。**
- **A类向B类发送消息，从而向影响B发生变化。**



```
@startuml

title Dependency

class Vihicle {
    move(MoveBehavior)
}

interface MoveBehavior {
    move()
}

note "MoveBehavior.move()" as N

Vihicle ..> MoveBehavior

Vihicle .. N

@enduml
```

参考资料

- **Java编程思想**
- **看懂UML类图和时序图**
- **面向对象三大特征------封装、继承、多态**
- **Java实现面向对象编程（OOP）**
- **JavaOOP基础知识总结**
- **Java抽象类与OOP三大特征**