

学号 E21714049 专业 计算机科学与技术 姓名 梅世祺  
实验日期 2019.09.25 教师签字                      成绩                     

# 实验报告

【实验名称】 Java 继承和多态

【实验目的】

学习 Java 中的单继承，方法的重写，上转型以及抽象类等知识。

【实验原理】

- 设计一个动物声音“模拟器”，希望模拟器可以模拟许多动物的叫声，要求如下。编写抽象类 `Animal`。`Animal` 抽象类有两个抽象方法 `cry()` 和 `getAnimalName()`，即要求各种具体的动物给出自己的叫声和种类名称
- 编写模拟器类 `Simulator`。该类有一个 `playsound(Animal animal)` 方法，该方法的参数是 `Animal` 类型。即参数 `animal` 可以调用 `Animal` 的子类重写的 `cry()` 方法播放具体动物的声音，调用子类重写的 `getAnimalName()` 方法显示

动物种类的名称。

- 编写 Animal 类的子类:Dog 和 Cat 类。

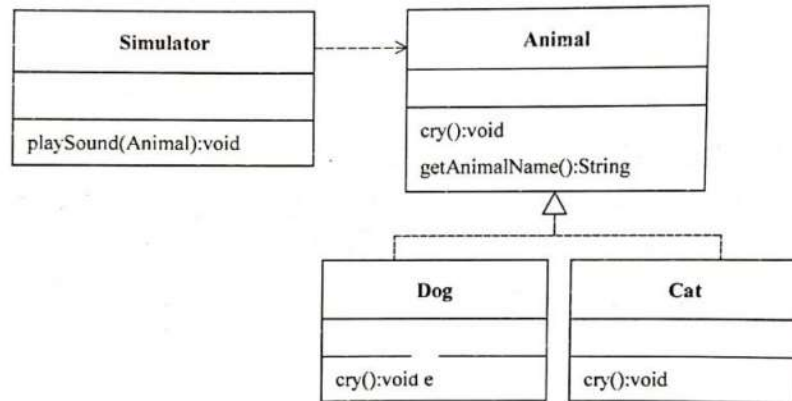


图 5.18 UML 类图

### 【实验内容】

首先编写抽象类 Animal:

```

Animal.java > Animal > getAnimalName()
1  abstract class Animal {
2      abstract void cry();
3      abstract String getAnimalName();
4  }
  
```

接着分别编写 Cat 类和 Dog 类来实现 Animal 类中的抽象方法:

```

Cat.java > Cat > getAnimalName()
1  public class Cat extends Animal {
2      @Override
3      void cry() {
4          System.out.println("Miao...Miao...");
5      }
6      @Override
7      String getAnimalName() {
8          return "Cat";
9      }
10 }
  
```

```
Dog.java > Dog > getAnimalName()
1  public class Dog extends Animal {
2      @Override
3      void cry() {
4          System.out.println("Wang...Wang...");
5      }
6      @Override
7      String getAnimalName() {
8          return "Dog";
9      }
10 }
```

然后编写 Simulator 类，实现 playSound() 实例方法：

```
Simulator.java > Simulator > playSound(Animal)
1  public class Simulator {
2      public void playSound(Animal animal) {
3          System.out.println(animal.getAnimalName());
4          animal.cry();
5      }
6  }
```

最后编写 Application 主类测试代码：

```
Application.java > Application > main(String[])
1  public class Application {
2      Run | Debug
3      public static void main(String[] args) {
4          Simulator simulator = new Simulator();
5          simulator.playSound(new Dog());
6          simulator.playSound(new Cat());
7      }
8  }
```

运行结果：

```

$ cd /home/lolimay/Code/Java/experiment3 ; /usr/lib/jvm/java-11-openjdk-amd64/bin/java -D
cp /home/lolimay/.vscode-server/data/User/workspaceStorage/59d8a148921c2d8f19ed65a34beb5f
/jdt.ls-java-project/bin Application
ls: cannot access '~WRL0001.tmp': No such file or directory
Animal.java           '~WRL0001.tmp'      '~WRL1085.tmp'      '~WRL3318.tmp'
Application.java      '~WRL0003.tmp'      '~WRL1147.tmp'      '~WRL3359.tmp'
Cat.java              '~WRL0005.tmp'      '~WRL1544.tmp'      '~WRL3590.tmp'
Dog.java              '~WRL0103.tmp'      '~WRL1594.tmp'      '~WRL3780.tmp'
Java实验报告3-E21714049-梅世祺.docx '~WRL0291.tmp'      '~WRL1733.tmp'      '~WRL4099.tmp'
Simulator.java        '~WRL0419.tmp'      '~WRL2168.tmp'      第3次作业.pdf
Trans.java            '~WRL0900.tmp'      '~WRL2504.tmp'
'~$va实验报告3-E21714049-梅世祺.docx' '~WRL0965.tmp'      '~WRL2518.tmp'
Dog
Wang...Wang...
Cat
Miao...Miao...

```

运行结果

## 【小结或讨论】

### 1. 抽象类和抽象方法

抽象类不能直接通过 `new` 关键字实例化对象，因为它包含的抽象方法只有声明没有实现。必须要在其子类中实现所有抽象方法后，通过子类实例化。

### 2. 抽象类与接口的区别

接口是对行为的抽象。

- 属性被隐式指定为 `public static final`，即接口中的成员变量全都是全局常量；
- 方法被隐式指定为 `public abstract`，即接口中的方法全都是抽象方法。

抽象类是对类（一类事物）的抽象。

被 `abstract` 关键字修饰的类称为抽象类，抽象类一般包含至少一个抽象方法，抽象方法必须要被子类实现。抽象类可以包含普通实例方法。

### 3. 什么时候用抽象类 什么时候用接口？

门与警报的例子：

```
App.java > Alarm
1 interface Alarm {
2     void alarm();
3 }
4 abstract class Door {
5     abstract void open();
6     abstract void close();
7 }
8 class AlarmDoor extends Door implements Alarm {
9     @Override
10    void open() {
11        System.out.println("open the door");
12    }
13    @Override
14    void close() {
15        System.out.println("close the door");
16    }
17    public void alarm() {
18        System.out.println("alarm!");
19    }
20 }
21 public class App {
22     Run | Debug
23     public static void main(String[] args) {
24         AlarmDoor alarmDoor = new AlarmDoor();
25         alarmDoor.open();
26         alarmDoor.close();
27         alarmDoor.alarm();
28     }
29 }
```

PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL 2:

```
open the door
close the door
alarm!
$
```

如上所示，应该把 `alarm()` 放在接口中，把 `open()` 和 `close()` 放在抽象类 `Door` 中。这样我们就可以通过接口 `Alarm` 和抽象类 `Door` 来定义警报门类（`AlarmDoor`）。