《Java语言及网络编程》作业二

学号	姓名	班级
05191643	许万鹏	信息安全19-01班

1 第一题

1.1 题目

读程序，写结果

```
package com.wanpengxu.homework2;

class A {
    public String Show(D obj) {
        return ("A and D");
    }

    public String Show(A obj) {
        return ("A and A");
    }
}

class B extends A {
    public String Show(B obj) {
        return ("B and B");
    }

    public String Show(A obj) {
        return ("B and A");
    }
}

class C extends B {
    public String Show(C obj) {
        return ("C and C");
    }

    public String Show(B obj) {
        return ("C and B");
    }
}

class D extends B {
    public String Show(D obj) {
        return ("D and D");
    }
```

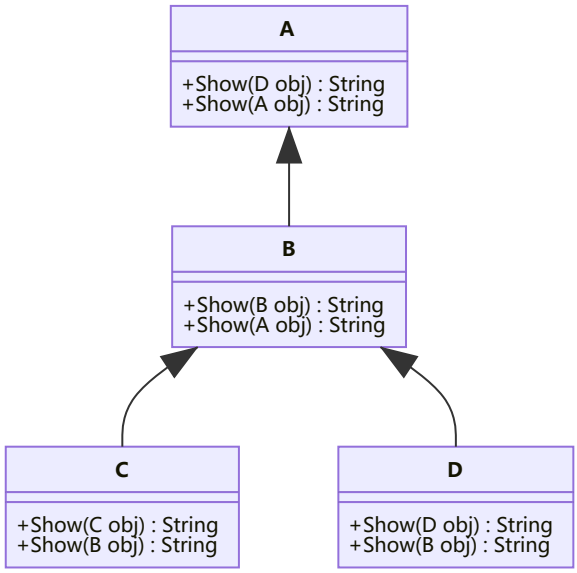
```
    public String Show(B obj) {
        return ("D and B");
    }
}

public class mainTest {
    public static void main(String[] args) {
        A a1 = new A();
        A a2 = new B();
        B b = new B();
        C c = new C();
        D d = new D();
        System.out.println(a1.Show(b));
        System.out.println(a1.Show(c));
        System.out.println(a1.Show(d));
        System.out.println(a2.Show(b));
        System.out.println(a2.Show(c));
        System.out.println(a2.Show(d));
        System.out.println(b.Show(b));
        System.out.println(b.Show(c));
        System.out.println(b.Show(d));
    }
}
```

1.2 答案

```
A and A
A and A
A and D
B and A
B and A
A and D
B and B
B and B
A and D
```

1.3 分析

先来看一下类图



继承链：从某个特定的类到其祖先的路径。

在类图中继承链便是从某个类沿向上箭头直到最初的祖先的路径。

动态绑定方法调用的优先问题，优先级由高到低依次为：一级调度this.show(O)、二级调度super.show(O)、三级调度this.show((super)O)、 四级调度super.show((super)O)

首先是a1对象的调用，由`A a1 = new A();`知a1的编译时类型是A，运行时类型是**A**，那么a1中的方法有：

1. 自己定义的方法

```
public String Show(D obj) {
        return ("A and D");
    }
```

2. 自己定义的方法

```
public String Show(A obj) {
        return ("A and A");
    }
```

a1.Show(b)：a1中不含有参数为B的方法，由继承链知B可向上转型至A，那么调用`2`，输出`A and A`

a1.Show(c)：a1中不含有参数为C的方法，由继承链知C可向上转型至A，那么调用`2`，输出`A and A`

a1.Show(d)：a1中含有参数为D的方法，那么调用`1`，输出`A and D`

接着是a2对象的调用，由`A a2 = new B();`知a2的编译时类型是A，运行时类型是**B**，那么a2中的方法有：

1. 继承自A的方法

```
    public String Show(D obj) {
            return ("A and D");
        }
```

2. 继承自A但被自己覆盖的方法

```
    public String Show(A obj) {
            return ("B and A");
        }
```

3. 自己定义的方法

```
    public String Show(B obj) {
            return ("B and B");
        }
```

这里可以使用反射机制进行验证，代码如下：

```
import java.lang.reflect.Method;
...
        System.out.println(a2.getClass());
        Class<? extends A> testa2 = a2.getClass();
        Method[] methods = testa2.getMethods();  // getDeclaredMethods()
        for (Method m : methods) {
            System.out.println(m);
        }
```

其中:

1. `getClass()`用于获取变量运行时类型的类名;
2. `getMethods()`用于获取Class变量中的所有方法（从父类继承的和自己声明的）
3. `getDeclaredMethods()`用于获取Class变量自己声明的方法。

```
class com.wanpengxu.homework2.B
public java.lang.String com.wanpengxu.homework2.B.Show(com.wanpengxu.homework2.B)
public java.lang.String com.wanpengxu.homework2.B.Show(com.wanpengxu.homework2.A)
public java.lang.String com.wanpengxu.homework2.A.Show(com.wanpengxu.homework2.D)
public final void java.lang.Object.wait(long,int) throws java.lang.InterruptedException
public final void java.lang.Object.wait() throws java.lang.InterruptedException
public final native void java.lang.Object.wait(long) throws java.lang.InterruptedException
public boolean java.lang.Object.equals(java.lang.Object)
public java.lang.String java.lang.Object.toString()
public native int java.lang.Object.hashCode()
public final native java.lang.Class java.lang.Object.getClass()
public final native void java.lang.Object.notify()
public final native void java.lang.Object.notifyAll()
```

但是，由于a2的编译时类型是A，所以在代码中只能调用`1`和`2`，否则会报错，即使你的实例中真的有`3`这个方法。

a2.Show(b)：a2中含有参数为B的方法，但不能调用，由继承链知B可向上转型至A，那么调用`2`，输出`B and A`

a2.Show(c)：a2中不含有参数为C的方法，由继承链知C可向上转型至A，那么调用`2`，输出`B and A`

a2.Show(d)：a2中含有参数为D的方法，那么调用`1`，输出`A and D`

最后是b对象的调用，由`B b = new B();`知b的编译时类型是B，运行时类型是**B**，那么b中的方法有：

1. 继承自A的方法

```
public String Show(D obj) {
        return ("A and D");
    }
```

2. 继承自A但被自己覆盖的方法

```
public String Show(A obj) {
        return ("B and A");
    }
```

3. 自己定义的方法

```
public String Show(B obj) {
        return ("B and B");
    }
```

b.Show(b)：b中含有参数为B的方法，那么调用`3`，输出`B and B`

b.Show(c)：b中不含有参数为C的方法，由继承链知C可向上转型至B，那么调用`3`，输出`B and B`

b.Show(d)：b本类不中含有参数为D的方法，但其超类中含有参数为D的方法，那么调用`1`，输出`A and D`

2 第二题

2.1 题目

读程序，写结果

```
package com.wanpengxu.homework2;

class Base {
    private String name = "base";

    public Base() {
        tellName();
    }
```

```
    public void tellName() {
        System.out.println("Base tell name: " + name);
    }
}

public class Derived extends Base {
    private String name = "derived";

    public Derived() {
        tellName();
    }

    public void tellName() {
        System.out.println("Derived tell name: " + name);
    }

    public static void main(String[] args) {
        new Derived();
    }
}
```

2.2 答案

```
Derived tell name: null
Derived tell name: derived
```

2.3 分析

流程如下:

1. 从`main`方法进入，执行`new Derived();`语句。
2. 调用Derived的构造方法`Derived()`。
3. 因`class Derived extends Base`，所以`Derived()`的第一句是由编译器添加的`Base()`，那么调用其父类的构造方法`Base()`。
4. Base类的构造方法初始化自己的属性，首先执行`private String name = "base";`，使Base.name="base"，随后调用`tellName()`方法，因为运行时类型是Derived，所以根据继承链，优先匹配Derived中的`tellName()`方法，但因为此时还没有到Derived初始化，所以this.name=null，打印出的结果为

   ```
   Derived tell name: null
   ```

   随后`Base()`方法结束,继续执行`Derived()`。
5. 此时`Derived()`初始化自己的属性，执行`private String name = "derived";`，使this.name="derived"，随后调用`tellName()`方法，同样的原因，匹配Derived中的`tellName()`方法，但此时this.name="derived"，打印出的结果为

```
Derived tell name: derived
```

随后`Derived()`方法结束，`new Derived();`语句完成，`main`方法结束，程序结束。

3 第三题

3.1 题目

生成动物，要求如下：

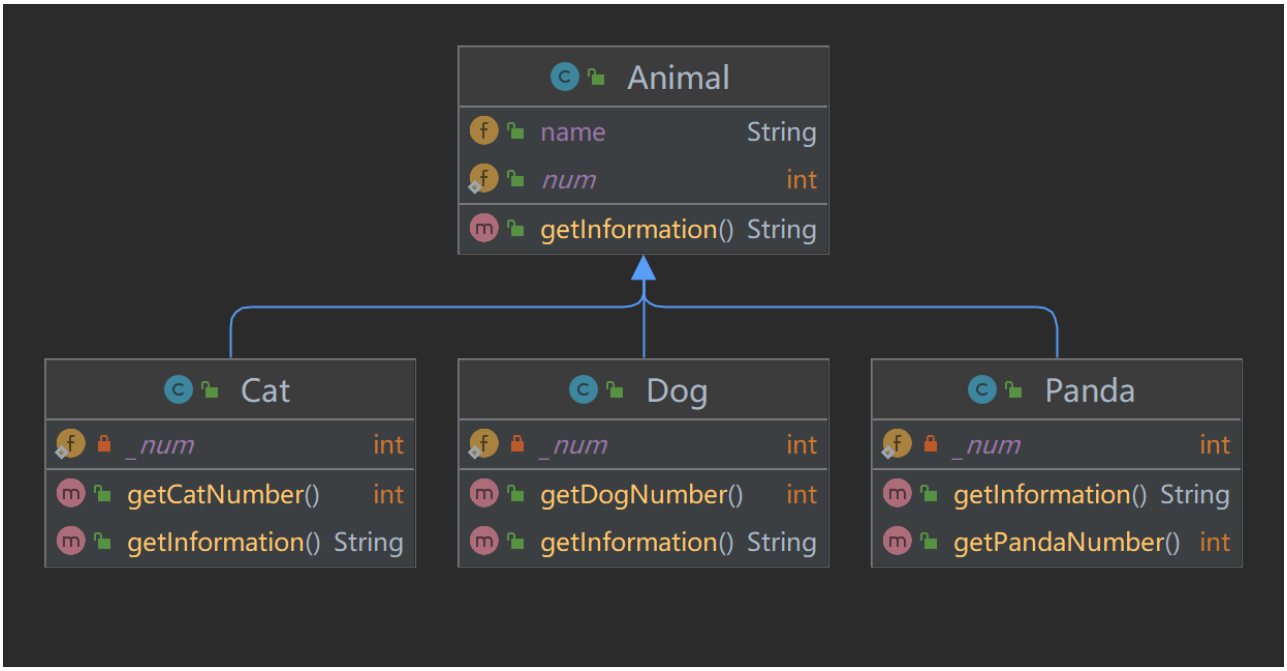
1. 循环通过标准输入端输入需要生成的动物，当遇到结束标志，则结束程序运行。
2. 每次生成动物，通过标准输出端显示动物的信息。
3. 动物的信息包括：目前所有动物的总数，当前这一类动物的总数。
4. 整个程序结构用工厂模式设计，保证将来动物园有新的动物加入时，程序可扩展。

3.2 答案

3.2.1 项目结构树

```
D:.
│  IdeaProjects.iml
│
├─.idea
│  ...
├─out
│  ...
└─src
    └─com
        └─wanpengxu
            └─homework2
                │  Derived.java
                │  mainTest.java
                │
                └─third
                    │  Main.java
                    │
                    ├─factory
                    │      AnimalFactory.java
                    │
                    └─product
                            Animal.java
                            Cat.java
                            Dog.java
                            Panda.java
```

3.2.2 类图



3.2.3 代码

1. Animal.java

```
package com.wanpengxu.homework2.third.product;

public class Animal {
    public static int num = 0;
    public String name;

    public Animal(String name) {
        this.name = name;
        num += 1;
        System.out.println("现在共有" + num + "只动物！");
    }

    public String getInformation() {
        return "\"" + name + "\"";
    }
}
```

2. Cat.java

```
package com.wanpengxu.homework2.third.product;

public class Cat extends Animal {
    private static int _num = 0;

    public Cat(String name) {
        super(name);
        Cat._num += 1;
```

```
            System.out.println("你获得了1只猫！现在已有" + getCatNumber() + "只猫！");
        }

        @Override
        public String getInformation() {
            return "\"" + name + "\": 一只猫";
        }

        public int getCatNumber() {
            return Cat._num;
        }
    }
```

3. Dog.java

```
    package com.wanpengxu.homework2.third.product;

    public class Dog extends Animal {
        private static int _num = 0;

        public Dog(String name) {
            super(name);
            Dog._num += 1;
            System.out.println("你获得了1只狗！现在已有" + getDogNumber() + "只狗！");
        }

        @Override
        public String getInformation() {
            return "\"" + name + "\": 一只狗";
        }

        public int getDogNumber() {
            return Dog._num;
        }
    }
```

4. Panda.java

```
    package com.wanpengxu.homework2.third.product;

    public class Panda extends Animal {
        private static int _num = 0;

        public Panda(String name) {
            super(name);
            Panda._num += 1;
```

```
            System.out.println("你获得了1只熊猫！现在已有" + getPandaNumber() + "只熊
猫！");
        }

        @Override
        public String getInformation() {
            return "\"" + name + "\": 一只熊猫";
        }

        public int getPandaNumber() {
            return Panda._num;
        }
    }
```

5. AnimalFactory.java

```
    package com.wanpengxu.homework2.third.factory;

    import com.wanpengxu.homework2.third.product.*;

    public class AnimalFactory {
        public static Animal addAnimal(String type, String name) {
            return switch (type.toLowerCase()) {
                case "cat" -> new Cat(name);
                case "dog" -> new Dog(name);
                case "panda" -> new Panda(name);
                default -> null;
            };
        }
    }
```

6. Main.java

```
    package com.wanpengxu.homework2.third;

    import com.wanpengxu.homework2.third.factory.AnimalFactory;
    import com.wanpengxu.homework2.third.product.Animal;

    import java.util.ArrayList;
    import java.util.Scanner;

    public class Main {
        public static void main(String[] args) {
            ArrayList<Animal> animals = new ArrayList<>();
            System.out.println("欢迎来到动物工厂！工厂现在可以生产以下动物：Cat，Dog，
Panda！输入Quit退出！");
```

```
        try (Scanner sc = new Scanner(System.in)) {  // try-with-resources 类似try-
finally
            while (true) {
                System.out.print("请输入想生产的动物类型: ");
                String type = sc.next();
                if (type.equalsIgnoreCase("Quit"))
                    break;
                System.out.print("请输入它的名字: ");
                String name = sc.next();
                Animal animal = AnimalFactory.addAnimal(type, name);
                if (animal == null)
                    System.out.println("请输入可生产的动物! ");
                else
                    animals.add(animal);
            }
        }
        System.out.println("生成完毕! 现在你有: ");
        for (Animal animal : animals)
            System.out.println(animal.getInformation());
    }
}
```

3.2.4 运行截图

```
欢迎来到动物工厂！工厂现在可以生产以下动物：Cat, Dog, Panda！输入Quit退出！
请输入想生产的动物类型：Panda
请输入它的名字：萌萌
现在共有1只动物！
你获得了1只熊猫！现在已有1只熊猫！
请输入想生产的动物类型：panDA
请输入它的名字：滚滚
现在共有2只动物！
你获得了1只熊猫！现在已有2只熊猫！
请输入想生产的动物类型：cat
请输入它的名字：毛毛
现在共有3只动物！
你获得了1只猫！现在已有1只猫！
请输入想生产的动物类型：sheep
请输入它的名字：喜羊羊
请输入可生产的动物！
请输入想生产的动物类型：quit
生成完毕！现在你有：
"萌萌"：一只熊猫
"滚滚"：一只熊猫
"毛毛"：一只猫
```

3.3 分析

题目要求使用工厂模式，那么按照工厂模式的实现，有：

1. 要有一个父类Animal，这里因为动物本身有一些公用的属性和方法，所以最好不要定义为abstract类；
2. 要有一些继承自Animal的具体的类作为某种动物，如Cat, Dog, Panda；
3. 需要一个工厂类AnimalFactory生成基于给定信息的实体类的对象，这里我是用增强型switch实现的，当然也可以用普通switch或if-else if-else实现；
4. 需要一个使用该工厂的类（用户类）来操作工厂完成生产任务。我加入了一点功能：生成动物时为其命名，并在退出后显示已生成动物名和其类型。另外，使用了可以自动关闭资源的try-with-resources语法，它的功能类似于try-finally，但语法十分精简。

因为采用了工厂模式，所以拓展动物种类的操作也很简单：

1. 复制一份动物类，将其类名重构为新增的动物，方法名同理；
2. 更改一些字符串以与新增动物相对应；
3. 在AnimalFactory类中新增一行case。