Java编程思想_面向对象之多态

课程概要

学习目标

理解多态的概念、好处和弊端

能够通过抽象类和接口定义多态关系并使用

理解多态关系中方法的调用过程

理解抽象类的概念

能够定义并正确使用抽象类

理解接口的概念

能够定义接口和实现类并使用

能够区分接口和抽象类的区别

能够说出接口、抽象类、普通类之间的继承和实现关系

多态概述

什么是多态?

多种状态，同一对象在不同情况下表现出不同的状态或行为

水在不同的温度下有不同的形态:

液态水

水蒸气

固态冰

米奇在不同的场景下有不同的角色：

乐手

棒球手

糕点师

Java中实现多态的步骤

要有继承（或实现）关系

要有方法重写

父类引用指向子类对象（is a关系）

```
public class Test {
    public static void main(String[] args) {
        //父类引用指向子类对象
        Animal a = new Dog();
    }
}
```

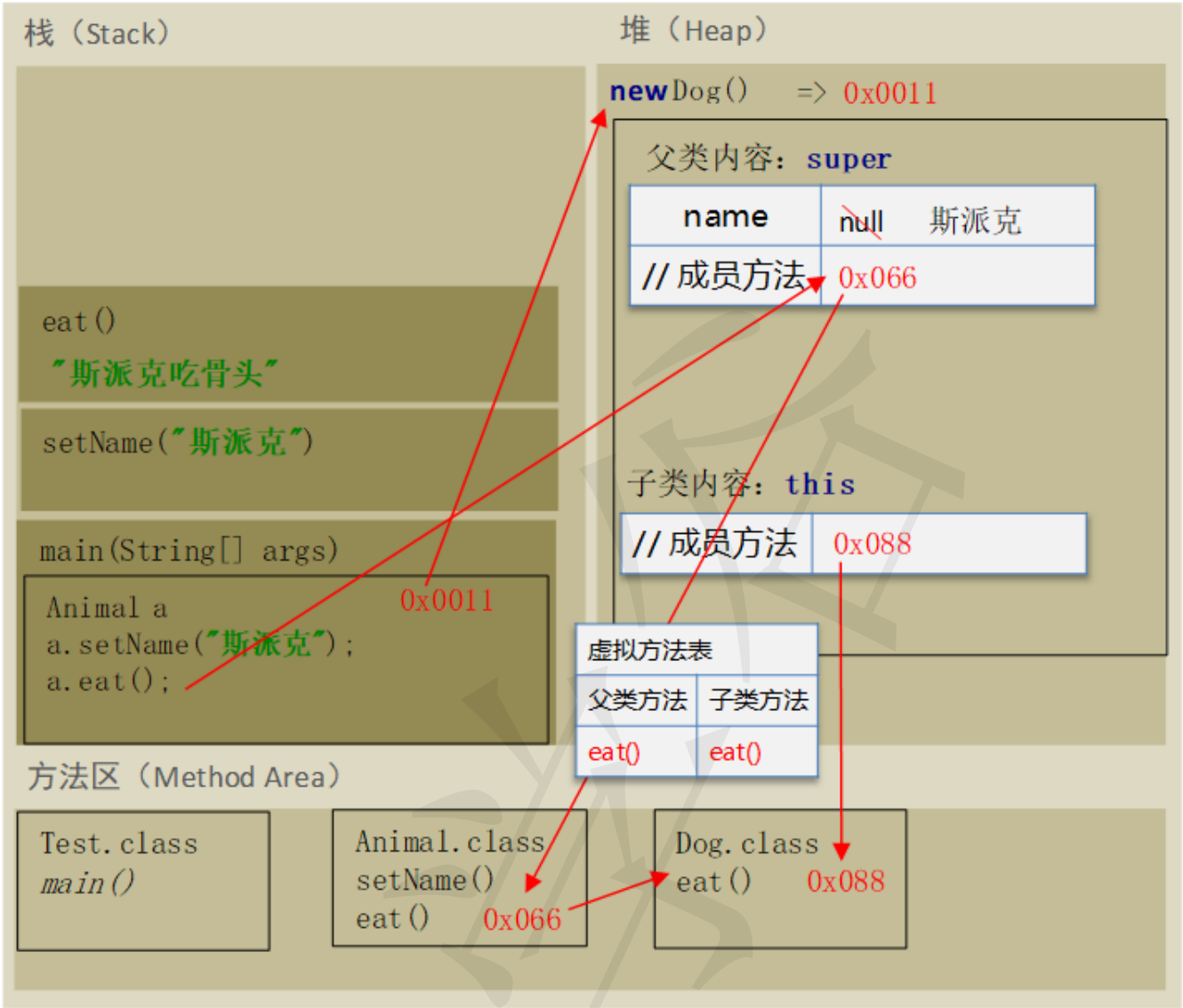
为什么父类引用可以指向子类对象?

因为二者满足 **子类 *is a* 父类** 的关系，所以任何一个Dog都可以Animal的形式使用

父类引用指向子类对象的内存图

在多态关系中，父类引用可以指向子类对象，因为它们满足“is a”的关系。此时，子类对象如何创建的，这个过程与父类有什么关系？通过父类引用调用方法时，执行过程是怎样的？如果子类重写了父类方法，又是如何调用的呢？

```
public class Test {
    public static void main(String[] args) {
        Animal a = new Dog();//父类引用指向子类对象
        a.setName("斯派克");
        a.eat();
    }
}
public class Animal {
    private String name;
    public void eat() {
        System.out.println("吃饭");
    }
    // getter,setter
}
public class Dog extends Animal {
    public void eat() {
        System.out.println(getName()+"吃骨头");
    }
}
```



加载类：

创建子类对象时，先加载父类，再加载子类

构造方法：

先执行父类的构造方法，初始化子类对象的*父类成员部分*然后再初始化子类成员部分

成员方法：

类的成员方法在*方法区*开辟空间并有一个地址值，该类的每一个对象都会记录方法区中的地址

在类的加载过程中，创建*虚拟方法表*，记录了子父类方法重写关系的信息。通过父类引用调用方法时，会查找虚拟方发表，看该方法是否被重写，如果表中记录了重写信息，则执行子类的重写方法。

多态的效果演示

需求：父类型变量作为参数时，可以接收任意子类对象

分析：

A：定义方法，参数类型为父类型Animal： showAnimal(Animal animal) B：分别创建Dog类和Mouse类的对象
C：调用showAnimal方法演示效果

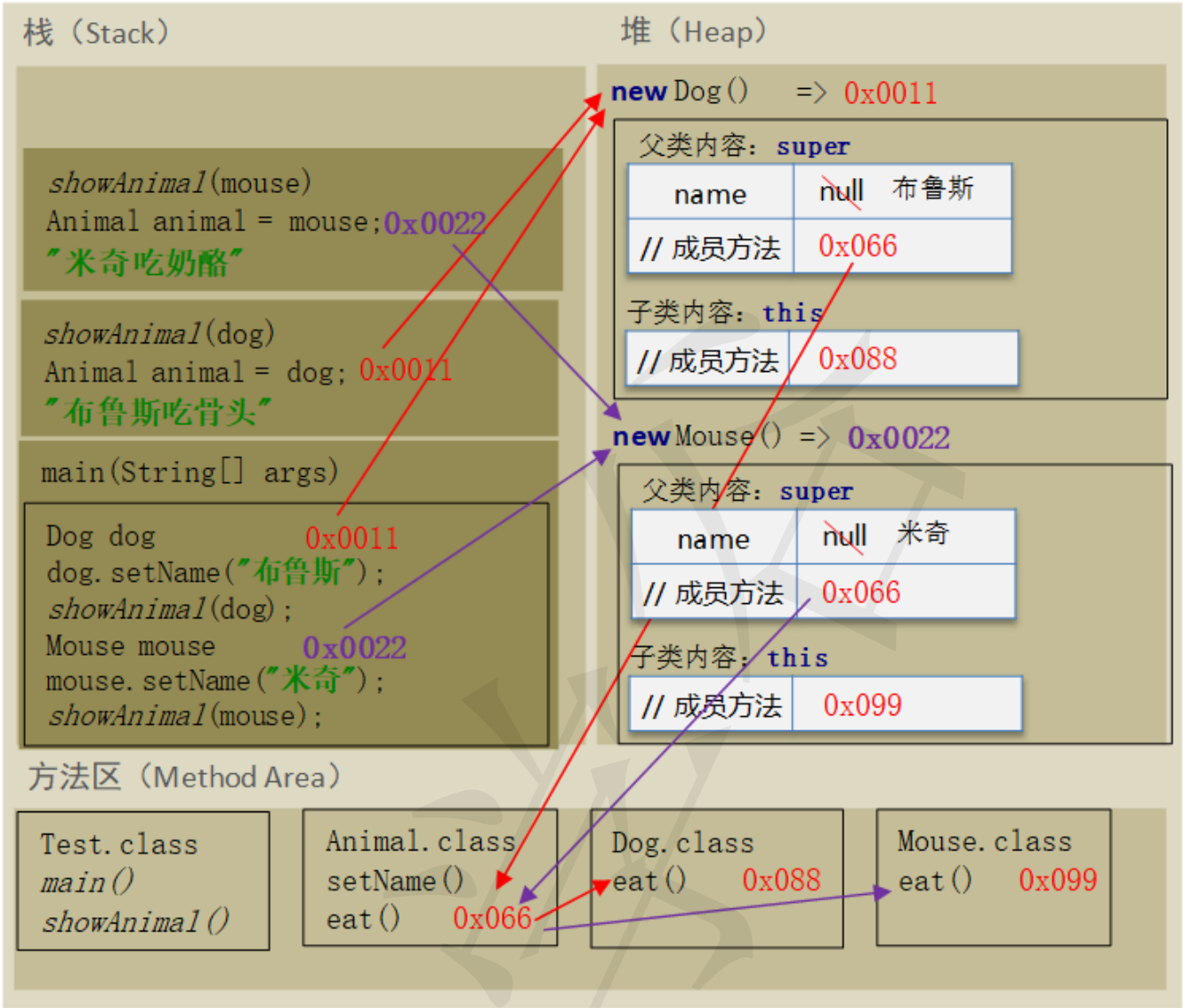
```
public static void showAnimal(Animal animal) {
    animal.eat();
}
```

内存图

```
public static void main(String[] args) {
    Dog dog = new Dog();
    dog.setName("布鲁斯");
    showAnimal(dog); // 输出：布鲁斯吃骨头

    Mouse mouse = new Mouse();
    mouse.setName("米奇");
    showAnimal(mouse); // 输出：米奇吃奶酪
}

public static void showAnimal(Animal animal) {
    // Animal animal = dog;
    // Animal animal = mouse;
    animal.eat();
}
```



同一个对象Animal在不同的情况下（传递进来的子类对象不同）表现出了不同的行为：吃骨头、吃奶酪

多态关系中成员变量的使用

需求：子父类中定义了同名的成员变量，如何调用?

```
public class Test {
    public static void main(String[] args) {
        Animal animal = new Dog();
        System.out.println(animal.name);
        Dog dog = new Dog();
        System.out.println(dog.name);
    }
}
public class Animal {
    String name = "Animal";
}
public class Dog extends Animal {
    String name = "Dog";
}
```

分析：

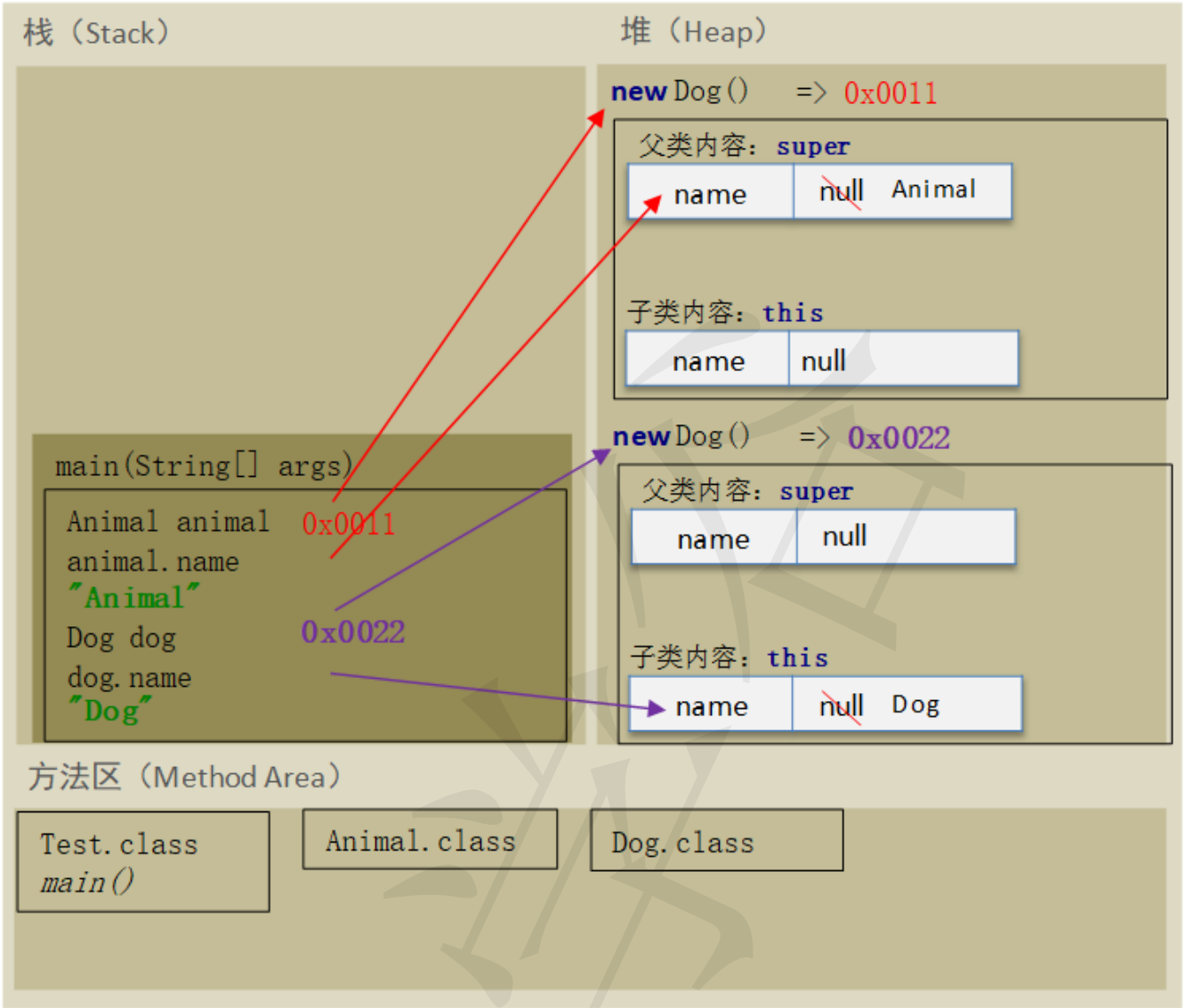
A：子父类中定义同名属性name并分别初始化值： String name；B：在测试类中以多态的方式创建对象并打印name属性值：Animal animal = new Dog()；C：在测试类中以普通方式创建对象并打印name属性值：Dog dog = new Dog();

结论：

成员变量不能重写

成员变量的使用，取决于调用该变量的类型，成员变量不能重写

内存图：



多态的好处和弊端

多态的好处

可维护性：

基于继承关系，只需要维护父类代码，提高了代码的复用性，大大降低了维护程序的工作量

为什么要学习多态：

从程序的模块化和复用性来解释

封装：隐藏数据的实现细节，让数据的操作模块化，提高代码复用性

继承：复用方法，从对象的行为这个层面，提高代码的复用性

多态：复用对象，程序运行时同一个对象表现出不同的行为

可扩展性：

把不同的子类对象都当作父类看待，屏蔽了不同子类对象间的差异，做出通用的代码，以适应不同的需求，实现了向后兼容

多态的弊端

不能使用子类特有成员

解决办法：

向下转型（前提：必须准确知道该父类引用指向的子类类型）

类型转换

当需要使用子类特有功能时，需要进行类型转换

```
public static void main(String[] args) {
    Dog dog = new Dog();
    dog.setName("布鲁斯");
    showAnimal(dog); // 输出：布鲁斯吃骨头
}

public static void showAnimal(Animal animal) {
    if(animal instanceof Dog) {
        Dog dog = (Dog)animal;
        dog.watch();
    }
    animal.eat();
}
```

向上转型（自动类型转换）

```
// 子类型转换成父类型
Animal animal = new Dog();
```

向下转型（强制类型转换）

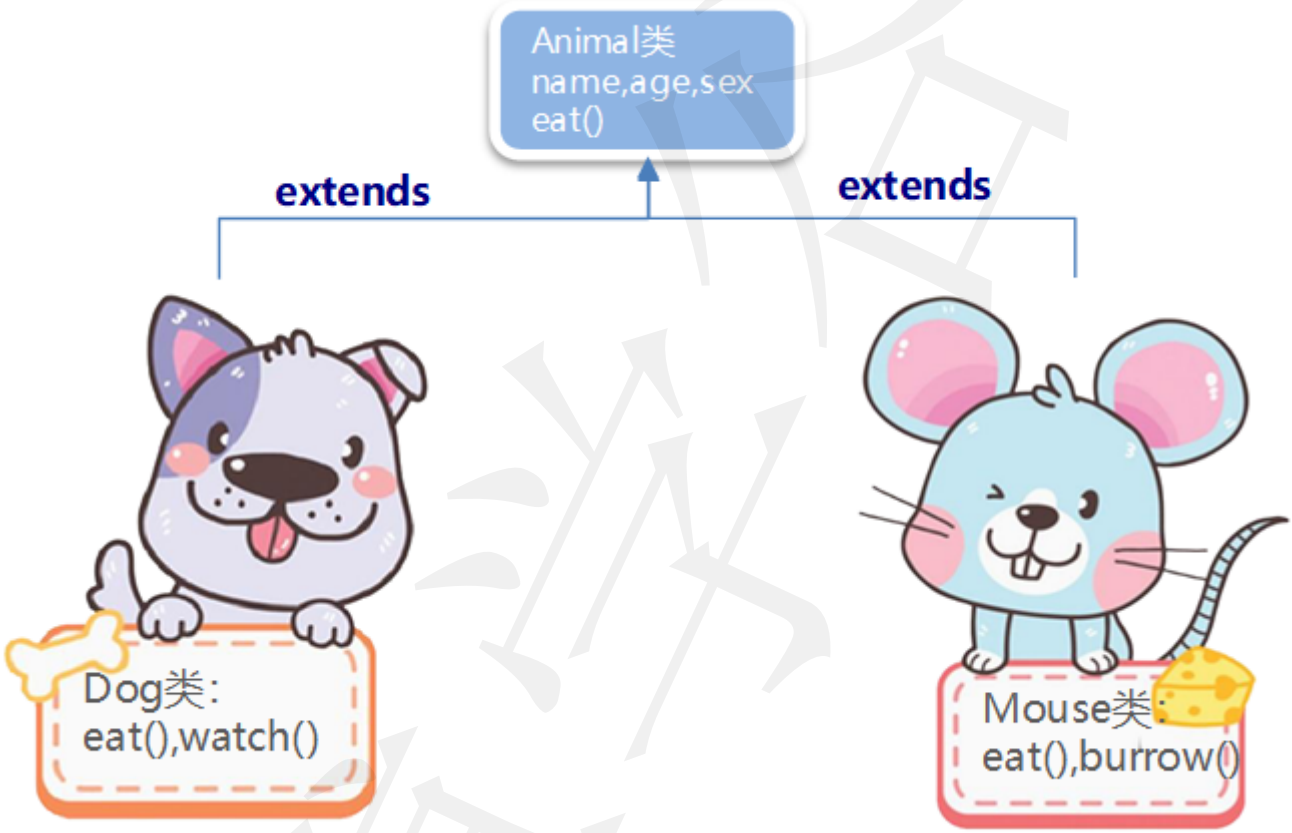
```
// 父类型转换成子类型
Dog dog = (Dog)animal;
```

注意事项

只能在继承层次内进行转换，否则可能造成异常（`ClassCastException`） 将父类对象转换成子类之前，使用`instanceof`进行检查

抽象类概述

抽象类的由来



父类默认的eat方法已过时，所有子类都需要重写，所以将父类eat的方法体没有意义；

但是每个动物都必须拥有eat功能，所以将父类eat定义成抽象方法。

```java
// 将类也定义为抽象的：abstract
public abstract class Animal {
    private String name;
    // 将eat方法定义为抽象的：abstract
    public abstract void eat();

    // getter,setter
}
```

```java
public class Mouse extends Animal {
    public void eat() {
        System.out.println(getName() + "吃奶酪");
    }
}
public class Dog extends Animal {
    public void eat() {
        System.out.println(getName() + "吃骨头");
    }
}
```

抽象类的概念

包含抽象方法的类。用abstract修饰

抽象方法的概念

只有方法声明，没有方法体的方法。用abstract修饰

抽象方法的由来

当需要定义一个方法，却不明确方法的具体实现时，可以将方法定义为abstract，具体实现延迟到子类

抽象类的特点

抽象类的特点

1. 修饰符：必须用abstract关键字修饰

   修饰符 abstract class 类名 {}

   修饰符 abstract 返回类型 方法名 {}

2. 抽象类不能被实例化，只能创建子类对象
3. 抽象类子类的两个选择

   重写父类所有抽象方法

   定义成抽象类

抽象类成员的特点

成员变量：

可以有普通的成员变量 也可以有成员常量（final）

成员方法：

可以有普通方法，也可以有抽象方法 抽象类不一定有抽象方法，有抽象方法的类一定是抽象类（或接口）

构造方法：

像普通类一样有构造方法，且可以重载

案例：定义员工之间的继承关系并使用

需求：

开发团队中有程序员和经理两种角色，他们都有姓名、工资、工号等属性，都有工作的行为，经理还有奖金属性。请使用继承思想设计出上述需求中的类，并分别创建对象使用。

分析：

A：经理和程序员都是员工，把他们共同的属性和行为定义在父类Employee中，由于并不明确工作的具体内 容，所以父类中工作的方法定义为抽象的：

name，salary，id；work();

B：定义经理类Manager，继承Employee，属性和行为：

bonus; work();

C：定义程序员类Coder，继承Employee，属性和行为：

work();

D：在测试类中分别创建对象并使用

final关键字

final的概念

最终的、最后的

final的作用

用于修饰类、方法和变量

修饰类：

该类不能被继承

String, System

修饰方法：

该方法不能被重写 不能与abstract共存

修饰变量：

最终变量，即常量，只能赋值一次 不建议修饰引用类型数据，因为仍然可以通过引用修改对象的内部数据，意义不大

static关键字

static的概念

静态的

static的作用

用于修饰类的成员： 成员变量：类变量 成员方法：类方法

调用方式

类名. 成员变量名; 类名. 成员方法名(参数);

static修饰成员变量

特点:

被本类所有对象共享

需求：定义研发部成员类，让每位成员进行自我介绍

```
"C:\Program Files\Java\jdk-11.0.1\bin\java.exe" ...
我是研发部的小黑，我的工作内容是写代码
我是研发部的媛媛，我的工作内容是鼓励师

Process finished with exit code 0
```

```
"C:\Program Files\Java\jdk-11.0.1\bin\java.exe" ...
我是开发部的小黑，我的工作内容是写代码
我是开发部的媛媛，我的工作内容是鼓励师

Process finished with exit code 0
```

分析：

A：研发部成员统称为开发者，定义类Developer。 B：每位开发者所属部门相同，所以属性departName用static修饰： public static String departName = "研发部"; C：Developer类的普通属性和行为： name, work; selefIntroduce(); D：在测试类中创建对象并使用 E：修改部门名称为“开发部”，测试效果

注意事项

随意修改静态变量的值是有风险的，为了降低风险，可以同时用final关键字修饰，即公有静态常量（注意命名的变化）：

```
public final static String DEPART_NAME = "研发部";
```

static修饰成员方法

静态方法：

静态方法中没有对象this，所以不能访问非静态成员

静态方法的使用场景

只需要访问静态成员 不需要访问对象状态，所需参数都由参数列表显示提供

需求：定义静态方法，反转数组中的元素

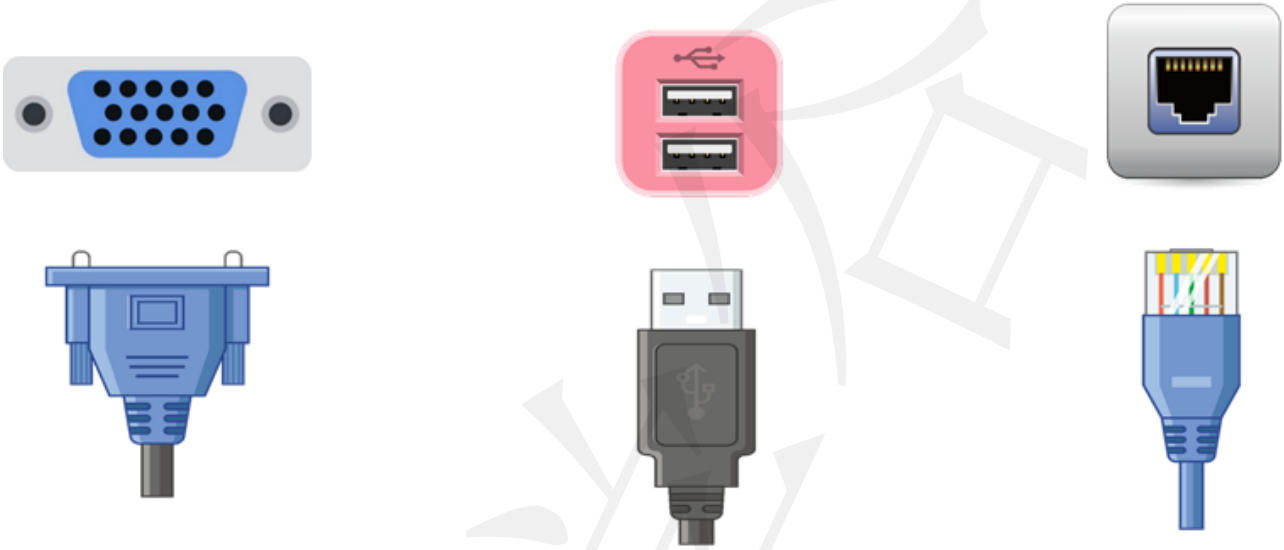
```
"C:\Program Files\Java\jdk-11.0.1\bin\java.exe" ...
反转前: 6, 2, 4, 1, 5, 9
反转后: 9, 5, 1, 4, 2, 6


Process finished with exit code 0
```

分析：

A：先明确定义方法的三要素： 方法名：reverse（反转） 参数列表：int[] arr 返回值类型：void B：遍历数组，交换数组索引为i和length-1-i的元素： arr[i] <=> arr[arr.length -1 - i] C：当索引 i >= (length – 1 – i) 时，停止交换元素 D：在测试类中创建对象并使用

接口概述



接口的概念

接口技术用于描述类具有什么功能，但并不给出具体实现，类要遵从接口描述的统一规则进行定义，所以，接口是对外提供的一组规则、标准。

接口的定义

定义接口使用关键字interface

```
interface 接口名 {}
```

类和接口是实现关系，用implements表示

```
class 类名 implements 接口名
```

接口创建对象的特点

1. 接口不能实例化

   通过多态的方式实例化子类对象

2. 接口的子类（实现类）

   可以是抽象类，也可以是普通类

接口继承关系的特点

1. 接口与接口之间的关系

   继承关系，可以多继承，格式：

   ```
   接口 extends 接口1，接口2，接口3...
   ```

2. 继承和实现的区别

   继承体现的是“is a”的关系，父类中定义共性内容

   实现体现的是“like a”的关系，接口中定义扩展功能

接口成员的特点

接口成员变量的特点

接口没有成员变量，只有公有的、静态的常量：

```
public static final 常量名 = 常量值;
```

接口成员方法的特点

JDK7及以前，公有的、抽象方法：

```
public abstract 返回值类型 方法名();
```

JDK8以后，可以有默认方法和静态方法：

```
public default 返回值类型 方法名() {}
```

```
static 返回值类型 方法名() {}
```

JDK9以后，可以有私有方法：

```
private 返回值类型 方法名() {}
```

接口构造方法的特点

接口不能够实例化，也没有需要初始化的成员，所以接口没有构造方法