抽象类的意义

```
public class DemoAbstractClass {
    public static void main(String[] args) {
        // 1.抽象 abstract
        // 1.1 抽象方法: 没有具体实现 public void show(){代码} public abstract void show();
        // 1.2.抽象类: 使用abstract修饰 public abstract class Employee
        // 1.3.定义抽象:类上加 abstract 含有抽象方法。
        // Employee employee=new Employee(); 错误
        // 1.4.创建子类，再创建对象
        Teacher teacher = new Teacher();
        teacher.work();
        // 【意义】1.可以进一步抽取子类的共性。
        // 2.有的类不想让它他对象。给个这类加abstract
        // InputStream input=new InputStream();
        // 3.可以子类必须实现的方法，设计子类的共性。
    }
}
public abstract class Employee {
    private String id;
    private String name;
    //set/get
    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    //进一步抽取
    public abstract void work();
}
public class Teacher extends Employee{
    @Override
    public void work() {
        System.out.println("备课");
        System.out.println("讲课");
    }
}
```

抽象类作用-定义子类必须实现方法

```
public abstract class Car {
   public abstract void engine();

}

public class Suv extends Car{
    @Override
    public void engine() {
        System.out.println("suv 发动机");
    }

}
```

接口

```
public interface MyInteface {
```

```
    // 定义显示功能
    public abstract void show();

    public abstract void show1();
    public void show2();
    void show3();
    public static final double PI=3.14;
    public static final double A=3.14;
}
/*
 * public interface cn.itcast.demo01.MyInteface {
  public static final double PI;
  public static final double A;
  public abstract void show();
  public abstract void show1();
  public abstract void show2();
  public abstract void show3();
}
*/



public class MyInterfaceImpl implements MyInteface {

    @Override
    public void show() {
     System.out.println("MyInterfaceImpl.show");
    }

    @Override
    public void show1() {
        System.out.println("MyInterfaceImpl.show1");

    }

    @Override
    public void show2() {
        System.out.println("MyInterfaceImpl.show2");

    }

    @Override
    public void show3() {
        System.out.println("MyInterfaceImpl.show3");
    }

}

public class DemoInterface {
    public static void main(String[] args) {
        // 2.接口:接口是功能的集合
        // 2.1.可以包含多个功能
        // 2.2.功能必须是抽象方法（跟类区别开）
        // 【接口的意义】比抽象类更抽象的类
        // 3.完全抽象
        // 类:用来完全具体描述事物。 每个功能都有实现细节。
        // 接口:用来完全抽象地描述事物。 每个功能都没有具体实现，都是抽象的。
        // 抽象类:用来不完全抽象地描述事物。方法可以是抽象的，也可以不是。
        // 4.定义接口
        // 4.1.interface 接口.功能的集合
        // 4.2.每个方法都必须是抽象方法。
        // 4.3. public interface 接口名称{
        // 抽象方法1;
        // 抽象方法2;
        // }
        //5.注意事项:接口里面的抽象方法默认是public abstract 返回值  方法名();
        //5.1.方法必须是公有
        //5.2.方法必须是抽象
        //5.3.变量必须是常量
        //6.接口也是类，但是不是从源码角度，而是编译结果来看。 bin/.class
        //7.接口不能直接new 对象，必须先实现，再创建对象。
```

```
        //7.1.implements 实现。
        //7.2.实现:就是给定抽象方法以具体细节。
        //7.3.一般情况，需要给所有的方法以具体细节。  实现后的方法 有{代码} 没有;
        MyInterfaceImpl impl=new MyInterfaceImpl();
        impl.show();
        //7.4.特殊情况. 实现类是抽象类  此时应该把implements"理解为" 继承
    }
}


public abstract class MyAbstractClassImpl implements MyInteface{
    // 定义显示功能
    public abstract void show();
    public abstract void show1();
    public abstract void show2();
    public  abstract void show3();
}
```

接口 练习案例 辑毒狗

```
public abstract class Dog {
    private String name;
    private int age;

    // set/get
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    // 抽象的方法用来定义没有具体细节的功能
    public abstract void voice();

    public abstract void eat();

}
public class ShapiDog extends Dog {
    @Override
    public void voice() {
        System.out.println("ShapiDog 叫");

    }
    @Override
    public void eat() {
        System.out.println("ShapiDog 骨头");
    }

}


public interface JiDu  {
    //抽象方法
  public abstract void jidu();
}


public class JiduDog extends Dog implements JiDu {
```

```
    @Override
    public void voice() {

        System.out.println("JiduDog. 叫");
    }

    @Override
    public void eat() {

        System.out.println("JiduDog. 狗粮");
    }

    @Override
    public void jidu() {
        System.out.println("实现接口中的额外功能  辑毒");

    }

}



public class DemoInterfaceEx {
    public static void main(String[] args) {
        // 8. 犬 沙皮犬 辑毒犬 抽象类 继承 接口 实现
        // 8.1.创建狗类
        // 8.2.创建沙皮 继承可以使用父类资源避免代码重复编写
        ShapiDog dog = new ShapiDog();
        dog.eat();
        // 8.3.辑毒犬
        JiduDog dog2 = new JiduDog();
        dog2.eat();
        dog2.voice();
        // 额外功能
        dog2.jidu();
        // 应用场景:相同功能用抽象类 , 额外功能使用接口。
    }

}
```

接口特点

```
public interface A {
  public abstract void a();
}



public interface B extends A {
    public abstract void b();
    // public abstract void a();
}
public class BImpl implements B {

    @Override
    public void a() {


    }

    @Override
    public void b() {


    }

}
```

```
public class DemoInterface {

    public static void main(String[] args) {
        // 9.特点 (功能的集合,比抽象类更抽象的类  额外功能)
        // 9.1. 接口可以继承接口(继承的结果就是添加父接口的功能)
        // 9.2. 多实现:Java支持一个类同时实现多个接口 实现多个额外功能
        // 9.3.一个接口 可以继承多个接口。public class Ipad implements
        // PlayVideo,PlayMusic，A,B,C,D,E
        // 创建一个新接口 然后 N合一。达到implements后面的代码简洁
        // 9.4 类可以在继承一个类的同时，实现多个接口。 额外功能 案例辑毒犬
        JiduDanceDog dog = new JiduDanceDog();
        dog.dance();
        dog.jidu();
        // 9.5.接口与父类的功能可以重复，均代表要具备某种功能，并不冲突。      功能的集合 Set
        //小结:项目中不会考虑这么多， 面试题 。
        //10.区分
        //抽象类 :重用代码使用抽象类   接口  添加额外功能使用接口
    }
}
```

接口可以继承接口

接口可以继承接口(继承的结果就是添加父接口的功能)

```
public interface B extends A {
    public abstract void b();
    // public abstract void a();
}

public interface A {
  public abstract void a();
}

public class BImpl implements B {

    @Override
    public void a() {


    }

    @Override
    public void b() {


    }

}
```

同时实现多个接口

```

public class Impl implements A,B {

    @Override
    public void a() {
        System.out.println("Impl.a");
    }

    @Override
    public void b() {
        System.out.println("Impl.b");
    }

}

public interface A {
  public abstract void a();
```

```
}

public interface B {
 public abstract void b();
}
```

案例2

```
public class Ipad implements PlayVideo,PlayMusic {

    @Override
    public void playMusic() {

        System.out.println("mp3");
    }

    @Override
    public void playMp4() {

        System.out.println("mp4");
    }

}

public interface PlayMusic {
  public abstract void playMusic();
}
public interface PlayVideo {
 public abstract void playMp4();
}
```

一个接口 可以继承多个接口

```
public interface Play extends PlayMusic,PlayVideo {

}
public class Ipad2 implements Play{

    @Override
    public void playMusic() {
        System.out.println("Ipad2 .playMusic");

    }

    @Override
    public void playMp4() {
        System.out.println("Ipad2 .playMp4");

    }

}
```

接口与父类的功能可以重复

```
public interface A {
 public abstract void a();
 public abstract void show();
}
public interface B {
    public abstract void b();
    public abstract void show();
}
public class Impl implements A, B {

    @Override
    public void b() {
```

```
    }

    @Override
    public void a() {

    }

    @Override
    public void show() {

    }

}
```

不是四个方法，最后是实现三个方法

多态

```
public class DemoDuoTai {

    public static void main(String[] args) {
        // 1.多态(一个对象可以选择不同的形态)
        // 11.事物:一个事物的多种形态。 张三是一个人 同时张三也是一个学生。
        // 12.继承关系中，对象也可是父类形态 也可是子类形态
        Person p = new Student();
        Student p2 = new Student();
        // 13.接口实现类关系中，对象 也可是接口形态也可是实现类形态。
        MyImpl impl = new MyImpl();
        impl.show();
        MyInterface impl2 = new MyImpl();
        impl2.show();


    }

}
```

继承关系

```
//父类
public class Person {
    private int age;
    private String name;

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

}

//子类
public class Student  extends Person {
    public void study() {
        System.out.println("学生学习");
    }
}
```

实现关系

```
public interface MyInterface {
    public abstract void show();
}
public class MyImpl implements MyInterface {

    @Override
    public void show() {
        System.out.println("实现 show方法");

    }

}
```

多态意义

```
public class Fu {
    public void show() {
        System.out.println("Fu");
    }
}

public class Zi extends Fu {
    public void show() {
        System.out.println("Zi.show");
    }
    public void add() {
        System.out.println("Zi.add");
    }
}
public class Zi2 extends Fu {
    public void show() {
        System.out.println("Zi3.show");
    }

    public void add() {
        System.out.println("Zi3.add");
    }
}

public class DemoDuoTai {

    public static void main(String[] args) {
        // 13 多态:
        // 追求扩展性 尽量使用父类形态
        // Zi zi=new Zi();
        // Zi2 zi=new Zi2();
        // Zi3 zi=new Zi3();
        //Fu obj = new Zi3();
        // 多态的弊端
        // 父类形态不能使用子类特有方法
         Fu obj=new Zi();
         obj.show();
         Zi zi= (Zi)obj;
         zi.add();
        // Zi zi = new Zi();
        // zi.add();
        // zi.show();
    }
}
```

类型转换

```
public abstract class Animal {

    private String name;
    private int age;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }

    public abstract void eat();
}


public class Cat extends Animal {
    @Override
    public void eat() {
        System.out.println("Cat.eat");
    }
}
public class Dog extends Animal {

    @Override
    public void eat() {
        System.out.println("Dog .eat");

    }

}

public class DemoCast {

    public static void main(String[] args) {
        // 13.类型
        // 基础数据类型 类型转换。 类型1 obj=(类型1)obj2;
        int a = 1;
        double b = a;
        int c = (int) b;
        // 13.1上下转换关系(继承关系或者实现类关系)
        // 13.1.1 向上转换:向父类转换 默认转
        Animal animal = new Dog();
        animal.eat();
        // 13.1.2.向下转换 :向子类转换
        // Dog dog = (Dog) animal;
        // dog.eat();
        // 向上转不需要添加括号，向下转需要添加括号
        if (animal instanceof Cat) {
            Cat cat = (Cat) animal;
            cat.eat();
        } else {
            System.out.println("animal对象  不是Cat");
        }

        if (animal instanceof Dog) {
            Dog dog = (Dog) animal;
            dog.eat();
        }
        //【注意事项】 继承 关系中，在父类子类之间，同级的兄弟类不能转换ClassCastException
        //13.2.instanceof  判断是哪种类型  true是  false不是   为强制类型转换提供安全判断

    }
```

```
}
```

电脑案例

```
public class Computer {
    // 开电源
    public void powerOn() {
        System.out.println("Computer.powerOn");
    }

    public void use(USBInterface obj) {
        System.out.println("使用外设: " + obj);
        obj.work();
    }

    // 电源关闭
    public void powerOff() {
        System.out.println("Computer powerOff ");
    }
}


public class KeyBoard implements USBInterface{
    public void work() {
        System.out.println("键盘正常工作");
    }
}


public class Mouse implements USBInterface{
    public void work() {
        System.out.println("Mouse正常运行");
    }
}

//功能的集合
public interface USBInterface {
    public abstract void work();
}

public class DemoComputer {

    public static void main(String[] args) {
        // 14.电脑
        // 外设 鼠标 键盘
        // 根据事物对象应类 创建出类 Computer Mouse keyBoard
        Computer computer = new Computer();
        // Mouse mouse=new Mouse();
        // 创建键盘实例
        // KeyBoard keyboard=new KeyBoard();
        // 向上转型
        USBInterface mouse = new Mouse();
        USBInterface keyboard = new KeyBoard();
        // 多态:一个对象的多种形态. 存在继承关系或者实现关系。
        computer.powerOn();
        computer.use(mouse);
        computer.use(keyboard);
        // computer.use(keyboard);报错 原因是通用性不强，因为没有按照统一的规则进行生产。
        // 接口:功能的集合
        computer.powerOff();

    }

}
```