

一、抽象类（面试重点）

一、快速认识抽象类

```
package com.Abstract;

public class Abstract01 {
    public static void main(String[] args) {

    }
}

abstract class Animal { //这是整个类也要设置为抽象类 abstract + class + 某类名
    private String name;
    public Animal(String name) {
        this.name = name;
    }
    /* 场景：
        Animal范围太广了eat不能具体表明什么动物吃什么的问题
        1.这里的eat方法其实没什么意义
        2.即父类方法不确定性的问题
        3.引出将该方法设计为抽象方法abstract()
        4.所谓抽象方法就是没有实现的方法,(没有实现就是没有方法体)
        5.一般来说,抽象类会被继承, 这些抽象方法就有子类实现*/
    /*public void eat(){
        System.out.println("这是一个动物,但是不知道吃什么。。");
    }*/
    public abstract void eat(); //因此直接将这个方法设置为abstract方法, 整个类就要设置为抽象类
}
```

抽象类介绍

1. 抽象方法不需要‘{}’

```
//错误示范
public abstract void eat(){} //抽象方法不需要 ‘ {} ’
```

2.

抽象类的价值更多作用在于设计,是设计者设计好后, 让子类继承并实现的

3. 抽象类考官喜欢问, 在框架和设计模式使用比较多.

Detail

- 抽象类不能被实例化

```
package com.Abstract.AbstractDetail;

public class Detail01 {
    public static void main(String[] args) {
        //抽象类不可以被实例化(也就是说不能被new了)
        /*A a = new A();*/
    }
}

abstract class A{

}
```

- 抽象类不一定包含abstract方法，也就是说抽象类不一定有抽象方法

```
package com.Abstract.AbstractDetail;

public class Detail02 {
}

//比如这个抽象类B，里头（可以没有）抽象方法
abstract class B{

}
```

- 一旦类包含了抽象方法,则这个类必须声明为abstract
- abstract只能修饰类和方法，不能修饰属性和其它的

```
//抽象类只能修饰类和方法，不能修饰其他的
abstract class C{
    /*private abstract int c;这样会报错*/
}
```

- 抽象类还是类，类可以拥有什么成员，抽象类也能有什么成员
- 如果一个类继承了抽象类,则它必须实现抽象类的所有抽象方法，除非他自己也声明为抽象类

```
//若一个类继承了抽象类
abstract class E{
    public abstract void hi();
}

/*
    1. 则子类必须实现抽象类的(所有)抽象方法
    2. 除非这个子类自己也声明为抽象类
*/

//方法1: 将F这个子类声明为抽象类
abstract class F extends E{

}
```

```
//方法2：实现抽象类的方法
class G extends E{
    @Override//重写父类的方法
    public void hi() { //实例化就是方法体内写功能写东西
        System.out.println("通过实例化父类的抽象方法");
    }
}
```

- **(考点)** .
- 抽象方法不能使用private、final、static来修饰（因为这些关键字都是与重写相违背的）

```
//解释：抽象方法不能使用private、final、static来修饰
abstract class H{
    public abstract void hi();
    /*
    * 1. 若这里用了private
    * private abstract void hi();
    * 就会造成子类没有机会重写方法因为私有只能在本类中使用
    * 2.final(最终)
    * public final abstract void hi();
    * 也会造成子类没办法得到父类的抽象方法，
    * 因为执行final之后这个方法就截止了
    * 3.static
    * public static abstract void hi();
    * abstract 与 static 是不允许组合在一起的，没有这样的写法*/
}
```

二、练习

抽象类

• 课堂练习题 **AbstractExercise01.java 5min练习**

- 1) 题1, 思考: abstract final class A{} 能编译通过吗, why?
- 2) 题2, 思考: abstract public static void test2(); 能编译通过吗, why?
- 3) 题3, 思考: abstract private void test3(); 能编译通过吗, why?
- 4) 编写一个Employee类, 声明为抽象类, 包含如下三个属性: name, id, salary。提供必要的构造器和抽象方法: work()。对于Manager类来说, 他既是员工, 还具有奖金(bonus)的属性。请使用继承的思想, 设计CommonEmployee类和Manager类, 要求类中提供必要的方法进行属性访问, 实现work(), 提示 "经理/普通员工 名字 工作中...."

题1:

- 不能通过, 因为final 与 abstract不能一起用, 类使用了abstract就说明其子类必须要调用该父类(也就是这个抽象类)的方法, 加上了final子类就没办法被继承了。

题2:

- 不可以, abstract 和 static不能合在一起用, 并且static与重写无关。

题3:

- 不能, 因为abstract 和 private冲突, 子类没办法重写到这个方法了

题4:

主函数 Excer01

```
package com.Excer;
/*1.编写一个Employee类,声明为抽象类
 * 2.属性: name、id、salary
 * 3.提供必要的构造器 和 抽象方法方法: work()
 * 4.Manager类既是员工, 还具有奖金bouns属性
 * 5.使用继承的思想,设计CommonEmployee类
 * 和Manager类,要求类中提供必要的方法进行属性访问
 * 实现work()*/

public class Excer01 {
    public static void main(String[] args) {
        Employee[] employees = new Employee[2]; //创建对象数组
        //经理
        employees[0] = new Manager("杨大郎", 100, 20000.12, 5000);
        //普通员工
        employees[1] = new CommonEmployee("杨小郎", 101, 5000);
        //输出
        employees[0].work();
        System.out.println("=====");
        employees[1].work();
    }
}
```

父类 Employee

```
package com.Excer;
/*1.编写一个Employee类,声明为抽象类
 * 2.属性: name、id、salary
 * 3.提供必要的构造器 和 抽象方法方法: work()
 * 4.Manager类既是员工, 还具有奖金bouns属性
 * 5.使用继承的思想,设计CommonEmployee类
 * 和Manager类,要求类中提供必要的方法进行属性访问
 * 实现work()*/
public abstract class Employee {
```

```

public String name;
public int id;
public double salary;
//构造器
public Employee(String name, int id, double salary) {
    this.name = name;
    this.id = id;
    this.salary = salary;
}
public abstract void work(); //提供一个抽象方法
public String getName() {
    return name;
}

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

public int getId() {
    return id;
}

public void setId(int id) {
    this.id = id;
}

public double getSalary() {
    return salary;
}

public void setSalary(double salary) {
    this.salary = salary;
}
}

```

子类 Manager

```

package com.Excer;
/* 4.Manager类既是员工，还具有奖金bouns属性
 * 5.使用继承的思想,设计CommonEmployee类
 * 和Manager类,要求类中提供必要的方法进行属性访问
 * 实现work()*/
public class Manager extends Employee{
    private double bouns; //经理专属拥有的奖金，非抽象类所以可以private
    public Manager(String name, int id, double salary, double bouns){
        super(name, id, salary);
        setBouns(bouns);
    }
    //重写父类的抽象方法work

    @Override
    public void work() {
        System.out.println("经理名字: "+super.name);
        System.out.println("id:"+super.id);
    }
}

```

```

        System.out.println("奖金为: "+ bouns);
        System.out.println("工资为: "+super.salary);
    }

    public double getBouns() {
        return bouns;
    }

    public void setBouns(double bouns) {
        this.bouns = bouns;
    }
}

```

子类 CommonEmployee

```

package com.Excer;
/* 4.Manager类既是员工，还具有奖金bouns属性
 * 5.使用继承的思想,设计CommonEmployee类
 * 和Manager类,要求类中提供必要的方法进行属性访问
 * 实现work()*/
public class CommonEmployee extends Employee{
    public CommonEmployee(String name, int id, double salary) {
        super(name, id, salary);
    }

    @Override
    public void work() {
        System.out.println("普通员名字为: "+super.name);
        System.out.println("id:"+super.id);
        System.out.println("工资为: "+super.salary);
    }
}

```

三、抽象类模板设计模式

老韩需求

- 1) 有多个类，完成不同的任务job
- 2) 要求统计得到各自完成任务的时间
- 3) 请编程实现 **TestTemplate.java**

感情的自然流露

1. 先用最容易想到的方法
2. 分析问题，提出使用模板设计模式

- 思想：在方法中难免会有许多重复用的语句功能，若每次要用就写一遍太麻烦，复用性太差，因此可以将这些语句功能封装到一个方法里头，再按顺序调用会用到的方法，就会大大提高复用性.

代码例子：

1.最笨的方法

```
//最笨的方法
package com.Abstract.DesignModule;
//统计该项目执行速度
public class AA {
    public void job(){
        long start = System.currentTimeMillis();//开始时间
        long num = 0;
        for (long i = 0;i <= 80000000;i++){
            num += i;
        }
        long end = System.currentTimeMillis();//结束时间
        System.out.println("所用时间: ");
    }
}
```

- start、end和输出都复用性不高

2.还行的方法

```
package com.Abstract.DesignModule;
//统计该项目执行速度
public class AA {
    //把计算时间的功能封装起来
    public void calculateTime(){
        long start = System.currentTimeMillis();//开始时间
        job();//在其中间位置调用job()
        long end = System.currentTimeMillis();//结束时间
        System.out.println("所用时间: " + (end - start));
    }

    public void job(){

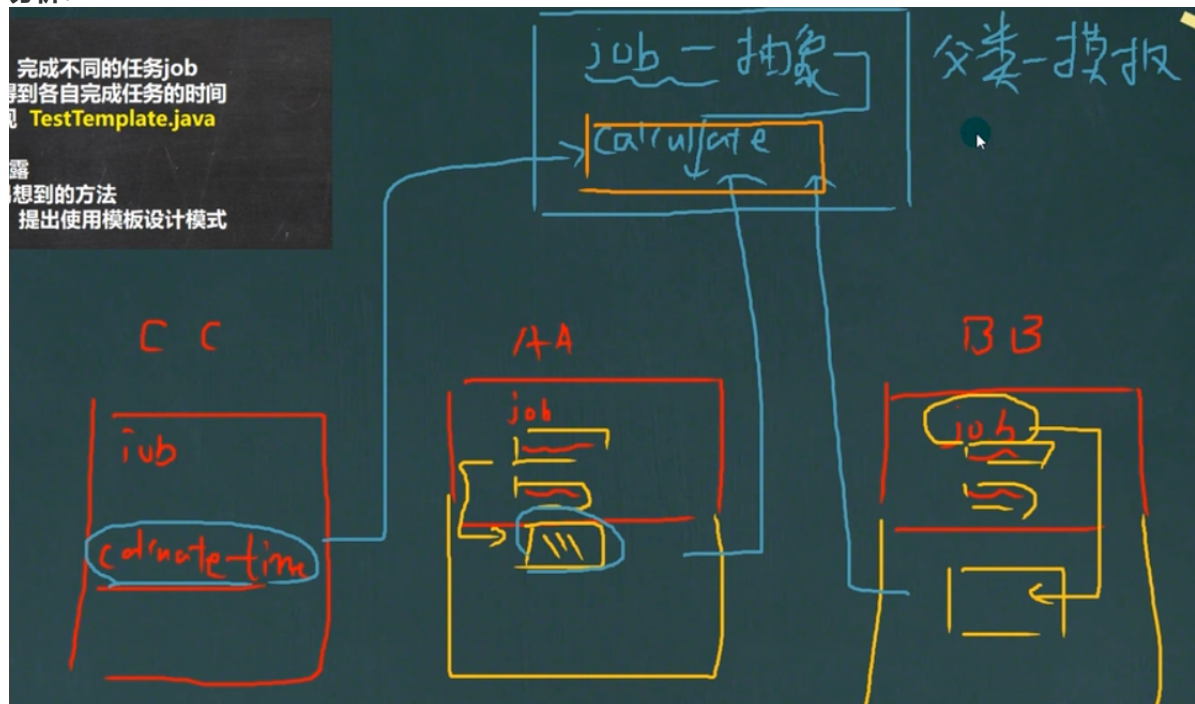
        long num = 0;
        for (long i = 0;i <= 80000000;i++){
            num += i;
        }

    }
}
```


- 此方法稍微提高了代码的复用性，但是其他类需要还得封装多一次

3.最好的设计

分析:



实现理解内功心法 的步骤:

抽象类最佳实践-模板设计模式

● 最佳实践

设计一个抽象类(Template)，能完成如下功能:

- 1) 编写方法calculateTime(),可以计算某段代码的耗时时间
- 2) 编写抽象方法job()
- 3) 编写一个子类Sub,继承抽象类Template,并实现job方法。
- 4) 编写一个测试类TestTemplate,看看是否好用。

```
abstract class Template{ //抽象类
    public abstract void job(); //抽象方法
    public void calculateTime(){ // 统计耗时多久是确定
        //统计当前时间距离 1970-1-1 0:0:0 的时间差, 单位ms
        long start = System.currentTimeMillis();
        job();
        long end = System.currentTimeMillis();
        System.out.println("耗时: " + (end-start));
    }
}
```

。

- 主函数 TestTemplate


```
package com.Abstract.DesignModule;

public class TestTemplate {
    public static void main(String[] args) {
        AA aa = new AA();
        aa.calculateTime();//AA没有calculateTime(),由于继承会到父类找的
    }
}
```

- 抽象父类AbTemplet

```
package com.Abstract.DesignModule;

public abstract class AbTemplet {
    public abstract void job();//抽象方法：目的是为了获得不同子类相应的job功能
    //把计算时间的功能封装起来
    public void calculateTime(){
        long start = System.currentTimeMillis();//开始时间
        job();//在其中位置调用job(),采用动态绑定机制
        long end = System.currentTimeMillis();//结束时间
        System.out.println("所用时间: " + (end - start));
    }
}
```

- 子类AA

```
package com.Abstract.DesignModule;
//统计该项目执行速度
public class AA extends AbTemplet{//继承
    public void job(){//重写(实现了AbTemplet的job方法)
        long num = 0;
        for (long i = 0;i <= 80000000;i++){
            num += i;
        }
    }
}
```

- 若以后还有其他类用上计算时间的方法直接extends，然后重写相应的父类抽象方法。