

课程回顾

1 继承：子类沿用父类非私有的属性和方法的一种编码形式

2 继承：优化多个类存在的冗余代码

3 继承：同名的属性、同名的方法、子类调用父类构造方法

课后作业第四题：参考答案

- 父类：Che

```
1 package cn.kgc.exercise;
2
3 /**
4  * @Author: lc
5  * @Date: 2022/3/25
6  * @Description: 父类
7  * @Version: 1.0
8  */
9 public class Che {
10
11     private int count;
12     private String color;
13
14     public int getCount() {
15         return count;
16     }
17
18     public void setCount(int count) {
19         if (count > 0) {
20             this.count = count;
21         } else {
22             System.out.println("非机动车必须有轮子");
23         }
24     }
25
26     public String getColor() {
27         return color;
28     }
29
30     public void setColor(String color) {
31         this.color = color;
32     }
33
34     public Che(int count, String color) {
35         this.count = count;
36         this.color = color;
37     }
38
39     /**
40     * 跑的行为
41     */
42 }
```

```

42     public void run(){
43         System.out.println("蹬着"+getCount()+"个轮子的"+getColor()+"色的自行车
跑");
44     }
45 }

```

- 子类: 自行车

```

1  package cn.kgc.exercise;
2
3  /**
4   * @Author: lc
5   * @Date: 2022/3/25
6   * @Description: cn.kgc.exercise
7   * @Version: 1.0
8   */
9  public class ZiXingChe extends Che{
10     //子类只需要提供特有的属性和行为
11
12     public ZiXingChe(int count, String color) {
13         super(count, color);
14     }
15
16     @Override
17     public void run() {
18         //System.out.println("蹬着"+getCount()+"个轮子的"+getColor()+"色的自行车
跑");
19         super.run();
20     }
21 }

```

- 子类: 电动车

```

1  package cn.kgc.exercise;
2
3  /**
4   * @Author: lc
5   * @Date: 2022/3/25
6   * @Description: cn.kgc.exercise
7   * @Version: 1.0
8   */
9  public class DianDongChe extends Che{
10     private boolean hasDian;
11
12     public boolean isHasDian() {
13         return hasDian;
14     }
15
16     public void setHasDian(boolean hasDian) {
17         this.hasDian = hasDian;
18     }
19
20     public DianDongChe(int count, String color, boolean hasDian) {
21         super(count, color);
22         this.hasDian = hasDian;
23     }

```

```

24
25     @Override
26     public void run() {
27         if (isHasDian()) {
28             System.out.println("骑着"+getCount()+"个轮子的"+getColor()+"色的电
电动车走");
29         }else{
30             //创建自行车对象时，具体的轮子的个数和颜色应该是从电动车对象里面获取的
31             //new ZixingChe(this.getCount(),this.getColor()).run();
32             //System.out.println("蹬着"+getCount()+"个轮子的"+getColor()+"色的
自行车跑");
33             super.run();
34         }
35     }
36 }

```

- 测试类：电动车是否有电

```

1  package cn.kgc.exercise;
2
3  /**
4   * @Author: lc
5   * @Date: 2022/3/25
6   * @Description: cn.kgc.exercise
7   * @Version: 1.0
8   */
9  public class Tester {
10     public static void main(String[] args) {
11         DianDongChe che = new DianDongChe(2, "黑色", true);
12         che.setHasDian(true); //优点
13         che.run();
14
15         //修改电池量
16         che.setHasDian(false); //没电
17         che.run();
18         //new DianDongChe(2, "黑色", false).run();
19     }
20 }

```

课程目标

1 Object类常用的方法 ===== 理解

2 抽象类和抽象方法 ===== 掌握

3 final关键字的使用 ===== 掌握

4 static关键字的使用 ===== 理解

课程实施

1 抽象类和抽象方法

1-1 抽象方法

抽象方法：也是方法。与普通的方法不同之处：没有方法体

```
1 | 访问修饰符 abstract 返回值类型 方法名(形参列表);
```

抽象方法调用没有意义！！！抽象方法所在的类，不允许创建对象。

1-2 抽象类

一般来说，抽象方法所在的类一定是抽象类

抽象类特点：不能创建对象通常专业的说成：不能实例化

抽象类一旦被子类继承，子类必须重写父类中所有的抽象方法。

```
1 | public abstract class 类名{
2 |
3 |
4 | }
```

```
    Che che1 = new Che( count: 2, color: "red");
}
```

'Che' is abstract; cannot be instantiated

抽象类不能实例化

Implement methods Alt+Shift+Enter

More actions... Alt+Enter

1-4 抽象的意义

抽象类一定是父类！！！父类中如果出现了抽象的方法，**强制**子类方法的重写（重写所有的抽象方法）。一个子类继承抽象类，而不重写父类的抽象方法，该子类也只能是抽象类，永远不能创建对象！！

抽象更多编程思想！！

1-5 小结

抽象方法一定在抽象类中！！子类继承抽象类后，必须重写父类所有的抽象方法。否则子类不能实例化！！

思考：

抽象类中一定有抽象方法吗？？不一定。

抽象类中没有抽象方法时，定义抽象类的目的，希望拿子类继承！！

1-3 课堂案例

需求：

父类：员工类（姓名 岗位 方法：

工作（方案一：输出：是人就要工作。方案二：抽象方法））

子类：

程序员类: 姓名 岗位 方法: 工作 (输出: java程序员李四在写代码)

教师类: 姓名 岗位 方法: 工作 (输出: java教员王五在备课)

参考代码

- 父类

```
1 package cn.kgc.demo1;
2
3 /**
4  * @Author: lc
5  * @Date: 2022/3/25
6  * @Description: 案例一: 父类--员工类
7  * @Version: 1.0
8  */
9 public abstract class Employee {
10     private String name;
11     private String job;
12
13     public String getName() {
14         return name;
15     }
16
17     public void setName(String name) {
18         this.name = name;
19     }
20
21     public String getJob() {
22         return job;
23     }
24
25     public void setJob(String job) {
26         this.job = job;
27     }
28
29     public Employee(String name, String job) {
30         this.name = name;
31         this.job = job;
32     }
33
34     /**
35      * 抽象方法
36      */
37     public abstract void work();
38 }
```

- 子类: 程序员

```
1 package cn.kgc.demo1;
2
3 /**
4  * @Author: lc
5  * @Date: 2022/3/25
6  * @Description: Employee的子类: IT程序员
7  * @Version: 1.0
8  */
```

```

9 public class IT extends Employee{//继承: 子类可以无条件拥有父类非私有的属性和方法
10     public IT(String name, String job) {
11         super(name, job);
12     }
13     //work()是否可以视而不见, 不重写, 程序也不会出错?? 抽象方法在子类中必须重写!!!
14     @Override
15     public void work() {
16         System.out.println(getJob()+getName()+"写代码");
17     }
18 }

```

- 子类: 教师类

```

1 package cn.kgc.demo1;
2
3 /**
4  * @Author: lc
5  * @Date: 2022/3/25
6  * @Description: 员工类子类的教师
7  * @Version: 1.0
8  */
9 public class Teacher extends Employee{
10     public Teacher(String name, String job) {
11         super(name, job);
12     }
13
14     @Override
15     public void work() {
16         System.out.println(getJob()+getName()+"在备课");
17     }
18 }

```

- 测试类: 测试工作方法

```

1 package cn.kgc.demo1;
2
3 /**
4  * @Author: lc
5  * @Date: 2022/3/25
6  * @Description: cn.kgc.demo1
7  * @Version: 1.0
8  */
9 public class Tester {
10     public static void main(String[] args) {
11         //1.创建对象
12         IT it = new IT("张三丰", "java开发工程师");
13         //2.对象.方法名
14         it.work();//先找自己的, 没有再找父类的
15
16         //创建老师对象并调用work()方法
17         Teacher tea = new Teacher("老王", "java教员");
18         tea.work();
19     }
20 }
21 //执行后, 输出的结果是
22 java开发工程师张三丰写代码

```

2 final的使用

最终的，最后的

2-1 修饰变量

final修饰的变量称为：常量

常量的特点是：一旦赋值，就不能再修改

```
1 final 变量===》常量
2 final 数据类型 常量名=值；
```

2-2 修饰方法

final修饰的方法称为：最终方法

最终方法的特点：子类不能重写

final和abstract不能同时使用

```
1 public final 返回值类型 方法名(形参列表){
2     //方法体
3 }
```

2-3 修饰类

final修饰的类称为：最终类

最终类的特点是：不能被其他类继承

```
1 public final class 类名{
2     //属性
3
4     //方法
5 }
```

3 Object类

Object：对象

OOP:Object Oriented Programme

java中所有的类直接或者间接的继承Object类。

万物皆对象。所有的类 is a Object 一种

3-1 Object的意义

java定义所有的类，都拥有Object提供的非私有的属性和方法！！！！

```
public class Object
```

类 Object 是类层次结构的根类。每个类都使用 Object 作为超类。所有对象（包括数组）都实现这个类的方法。

3-2 Object学习方法

API文档！！Java 的API (API: **A**pplication(应用) **P**rogramming(程序) **I**nterface(接口))

3-3 Object的方法

protected Object	clone() 创建并返回此对象的一个副本。
boolean	equals(Object obj) 指示其他某个对象是否与此对象“相等”。
protected void	finalize() 当垃圾回收器确定不存在对该对象的更多引用时，由对象的垃圾回收器调用此方法。
Class(?)	getClass() 返回此 Object 的运行类。
int	hashCode() 返回该对象的哈希码值。
void	notify() 唤醒在此对象监视器上等待的单个线程。
void	notifyAll() 唤醒在此对象监视器上等待的所有线程。
String	toString() 返回该对象的字符串表示。
void	wait() 在其他线程调用此对象的 notify() 方法或 notifyAll() 方法前，导致当前线程等待。
void	wait(long timeout) 在其他线程调用此对象的 notify() 方法或 notifyAll() 方法，或者超过指定的时间量前，导致当前线程等待。
void	wait(long timeout, int nanos) 在其他线程调用此对象的 notify() 方法或 notifyAll() 方法，或者其他某个线程中断当前线程，或者已超过某个实际时间量前，导致当前线程等待。

3-5 toString()方法

String	toString() 返回该对象的字符串表示。	Object提供的toString()的作用
--------	-----------------------------------	------------------------

作用：原本父类Object设计toString()是为了获取一个字符串 类名@哈希码。

toString()怎么调用？？sout(对象)自动调用，程序员不用写调用代码sout(对象.toString())

在自己的程序里面，一般建议重写toString()，希望sout(对象名)不是输出哈希地址，而是直接输出对象的属性值

参考案例

- Student类

```
1 package cn.kgc.demo2;
2
3 /**
4  * @Author: lc
5  * @Date: 2022/3/25
6  * @Description: cn.kgc.demo2
7  * @Version: 1.0
8  */
9 public class Student {
10     public String name;
11     public int age;
12     public char sex;
13     //重写：改造父类的不合适的方法体！！让方法变得更适合子类使用
14     //alt+insert
15     @Override
16     public String toString() {
17         return "Student{" +
18             "name='" + name + '\'' +
```



```

19         ", age=" + age +
20         ", sex=" + sex +
21         '}'';
22     }
23 }

```

- Tester类

```

1  package cn.kgc.demo2;
2
3  import org.junit.Test;
4
5  /**
6   * @Author: lc
7   * @Date: 2022/3/25
8   * @Description: cn.kgc.demo2
9   * @Version: 1.0
10  */
11  public class TestStudent {
12      @Test
13      public void testToString(){
14          Student student = new Student();
15          student.name="jack";
16          System.out.println(student.toString());//jack$$$$$
17          //Object提供的toString()
18          //String s=student.toString();
19          //输出toString()执行的结果
20          //System.out.println(s);//s是toString()执行的结果
21          //System.out.println("=====");
22          //cn.kgc.demo2.Student@17d10166
23          //System.out.println(student.toString());//student是对象的哈希地址
24          //sout(对象名)实际上jvm执行时，靠对象.toString()获取输出结果
25          //思考：toString()什么时候会调用？？sout(对象)就会自己调用toString()
26          //toString()是父类Object的？父类提供的输出哈希码，子类使用有没有用？没用
27          //改造toString():sout(对象)想看到对象所有的属性值，不用再自己写showInfo()
28          //结论：toString()就是showInfo()真身！！
29      }
30  }

```

3-6 重点!!! equals()方法*****

boolean	<u>equals</u> (Object obj) 指示其他某个对象是否与此对象“相等”。
---------	--

作用：Object提供equals()比较两个对象内存地址是否相等

程序员在自己的类中重写equals()原因是：希望比较对象时，不比较地址，而是比较两个对象的属性值是否一样!!!

The `equals` method for class `Object` implements the most discriminating possible equivalence relation on objects; that is, for any non-null reference values `x` and `y`, this method returns `true` if and only if `x` and `y` refer to the same object (`x == y` has the value `true`).

Note that it is generally necessary to override the `hashCode` method whenever this method is overridden, so as to maintain the general contract for the `hashCode` method, which states that equal objects must have equal hash codes.

Params: `obj` – the reference object with which to compare.

Returns: `true` if this object is the same as the `obj` argument; `false` otherwise.

See Also: `hashCode()`, `java.util.HashMap`

```
public boolean equals(Object obj) {
```

```
    return (this == obj);
```

Object比较对象是否相等，使用`==`。
`==`和`equals`其实是一回事！！

```
}
```

- Student类

```
1 package cn.kgc.demo2;
2
3 /**
4  * @Author: lc
5  * @Date: 2022/3/25
6  * @Description: cn.kgc.demo2
7  * @Version: 1.0
8  */
9 public class Student {
10     public String name;
11     public int age;
12     public char sex;
13
14     public Student(String name, int age, char sex) {
15         this.name = name;
16         this.age = age;
17         this.sex = sex;
18     }
19
20     //判断两个对象是否相等???
21     //判断两个学生对象是否相等? true---是同一个对象 false---不是同一个对象
22     //什么同一个对象? 内存地址是一样的!! Object提供equals()判断依据哈希地址, ==比较
    对象是就是比地址
23
24     //日常开发, 判断两个对象是否一样, 客户看值还是看内存地址???
25     //重写: 改造父类的不合适的方法体!! 让方法变得更适合子类使用
26     //alt+insert自动生成的!!
27     @Override
28     public boolean equals(Object o) {
29         if (this == o) return true;
30         if (o == null || getClass() != o.getClass()) return false;
31
32         Student student = (Student) o;
33
34         if (age != student.age) return false;
35         if (sex != student.sex) return false;
36         return name != null ? name.equals(student.name) : student.name ==
null;
```

```

37     }
38
39     @Override
40     public int hashCode() {
41         int result = name != null ? name.hashCode() : 0;
42         result = 31 * result + age;
43         result = 31 * result + (int) sex;
44         return result;
45     }
46
47     //alt+insert
48     @Override
49     public String toString() {
50         return "Student{" +
51             "name='" + name + '\'' +
52             ", age=" + age +
53             ", sex=" + sex +
54             '}';
55     }
56 }

```

- Tester测试类

```

1  public class TestStudent {
2      @Test
3      public void testEquals(){
4          //内存什么时候分配哈希地址
5          Student s1 = new Student("张无忌",21,'男');
6          Student s2 = new Student("张无忌",21,'男');
7          //如果两个人连身份证号码都一样？人类认定是一个人！！
8          //Student不重写equals()就是比较两个对象的内存地址。Student重写equals()后就
按照重写的方案比较对象
9          System.out.println("s1和s2是同一个对象吗？"+s1.equals(s2));//true
10         //==判断两个值是否相等
11         System.out.println("s1和s2是同一个对象吗？"+(s1==s2));//false ==永远比地
址！！！！
12     }
13 }

```

课堂练习:

需求: Person类 (属性: 身份证号码ID--String 姓名--String 性别-char)

全参构造方法

重写toString()

重写equals和hashCode(),比较依据: ID

测试类: 创建两个对象, 只要对象ID一样, 对象就是相等。

参考代码:

- Person类

```

1  package cn.kgc.exercise;
2
3  /**

```

```
4  * @Author: lc
5  * @Date: 2022/3/25
6  * @Description: cn.kgc.exercise
7  * @Version: 1.0
8  */
9  public class Person {
10     private String ID;
11     private String name;
12     private char sex;
13
14     public String getID() {
15         return ID;
16     }
17
18     public void setID(String ID) {
19         this.ID = ID;
20     }
21
22     public String getName() {
23         return name;
24     }
25
26     public void setName(String name) {
27         this.name = name;
28     }
29
30     public char getSex() {
31         //可以流程控制语句!!
32         return sex;
33     }
34
35     public void setSex(char sex) {
36         if (sex=='男' || sex=='女') {
37             this.sex = sex;
38         }else{
39             System.out.println("性别只能是男或女");
40         }
41     }
42
43     public Person() {
44     }
45
46     public Person(String ID, String name, char sex) {
47         this.ID = ID;
48         this.name = name;
49         this.sex = sex;
50     }
51
52     //一个类通常都会重写toString()
53
54     @Override
55     public String toString() {
56         return "Person{" +
57             "ID='" + ID + '\'' +
58             ", name='" + name + '\'' +
59             ", sex=" + sex +
60             '}';
61     }
```

```

62 //是否需要重写equals():比较对象, 且比对象的属性
63
64 @Override
65 public boolean equals(Object o) {
66     if (this == o) return true;
67     if (o == null || getClass() != o.getClass()) return false;
68
69     Person person = (Person) o;
70
71     return ID != null ? ID.equals(person.ID) : person.ID == null;
72 }
73
74 @Override
75 public int hashCode() {
76     return ID != null ? ID.hashCode() : 0;
77 }
78 }

```

- 测试类

```

1 package cn.kgc.exercise;
2
3 /**
4  * @Author: lc
5  * @Date: 2022/3/25
6  * @Description: cn.kgc.exercise
7  * @Version: 1.0
8  */
9 public class TestPerson {
10     //单元测试里面不能加Scanner
11     public static void main(String[] args) {
12         Person p1 = new Person("123", "jack", '男');
13
14         Person p2 = new Person("123", "jack34324535", '男');
15         System.out.println(p1); //自动调用toString()
16         System.out.println(p2); //自定调用toString()
17
18         System.out.println("p1和p2是同一个哈希地址: " + (p1==p2));
19         System.out.println("p1和p2内容是否一样: " + p1.equals(p2));
20     }
21 }
22 //测试结果如下所示:
23 Person{ID='123', name='jack', sex=男}
24 Person{ID='123', name='jack', sex=男}
25 p1和p2是同一个哈希地址: false
26 p1和p2内容是否一样: true

```

4 static关键字的使用

当属性是boolean类型时，生成的取值方法是以is开始，而不是get开始

```
public class DianDongChe extends Che{
    private boolean hasDian;

    public boolean isHasDian() {
        return hasDian;
    }

    public void setHasDian(boolean hasDian) {
        this.hasDian = hasDian;
    }
}
```

最终方法不能被子类重写

```
public void showInfo() {
}
```

'showInfo()' cannot override 'showInfo()' in 'cn.kgc.demo1.Employee': overridden method is final

Make 'Employee.showInfo' not final Alt+Shift+Enter More actions... Alt+Enter 不能重写final修饰的方法

最终类不能被重写

```
public class WebIT extends IT{
    public WebIT(String name,
        super(name, job);
}
```

Cannot inherit from final 'cn.kgc.demo1.IT'

Make 'IT' not final Alt+Shift+Enter More actions... Alt+Enter

cn.kgc.demo1
public final class IT
extends Employee

最终类不能被继承

课程总结

OOP梳理！！

抽象！！方法 类

final关键字的作用

Object面试

作业安排

预习安排

多态：

向上转型 向下转型代码体现方式

instanceof运算符有什么作用？

父类作为形参和父类作为返回值

接口