Java基础

zcx

一、面向对象编程

1. **对象**
   - 对象是一种特殊的数据结构，可以用来记住一个事物的数据，从而代表该事物
   - 如何创建对象
     a. 设计对象模板，即类，包含这个对象的基本信息，如属性和方法

```
public class Star{
String name;
int age;
double height;
double weight;
}
```

     b. 创建对象，即实例化对象，使用类名加new关键字，创建对象实例

```
Star s1 = new Star();
```

     c. 使用对象

```
s1.name="张三";
s1.age=30;
s1.gender="女";
s1.height=1.58;
s1.weight=50;
```

2. **面向对象使用案例——封装思想**
   - 目标：用面向对象方法存储两个同学的信息，包含姓名，性别，语文成绩，数学成绩，并打印出每个同学的总成绩和平均成绩
   -

```
public class student { 定义学生类
String name;
String gender;
double ygrade;
double mgrade;
}
```

   -

```
student s1 = new student(); main函数内创建对象实例
s1.name="张三";
```

```
s1.gender="女";
s1.ygrade=100;
s1.mgrade=100;
student s2 = new student();
s2.name="李四";
s2.gender="男";
s2.ygrade=59;
s2.mgrade=100;
System.out.println("张三的总成绩为："+ (s1.ygrade+ s1.mgrade));
System.out.println("李四的总成绩为："+ (s2.ygrade+ s2.mgrade));
System.out.println("张三的平均成绩为："+ (s1.ygrade+ s1.mgrade)/2);
System.out.println("李四的平均成绩为："+ (s2.ygrade+ s2.mgrade)/2); 这四行打印过于重复，可以封装到类中
```

- ```
  public class student { 重新封装的学生类
  String name;
  String gender;
  double ygrade;
  double mgrade;
  public void sum_grade_print(){ 封装求总成绩方法
  System.out.println(name+"的总成绩为："+(ygrade+mgrade));
  }
  public void average_grade_print(){ 封装求平均成绩方法
  System.out.println(name+ "的平均成绩为："+((ygrade+mgrade)/2));
  }}
  ```

- ```
  student s1 = new student(); main函数内创建对象实例
  s1.name="张三";
  s1.gender="女";
  s1.ygrade=100;
  s1.mgrade=100;
  s1.sum_grade_print(); 调用封装的方法
  s1.average_grade_print(); 非常简洁
  student s2 = new student();
  s2.name="李四";
  s2.gender="男";
  s2.ygrade=59;
  s2.mgrade=100;
  s2.sum_grade_print();
  s2.average_grade_print();
  ```

二、类的基本语法

1. **构造器**
   - 构造器就是构造函数，用来创建对象实例，构造器名称与类名相同，构造器没有返回值。
   - ```
     public class student {
     //构造器，即构造函数
     public student(){} //无参数构造器
     //有参数构造器
     public student(String name,int age){}
     }
     ```
   - **特点：创建对象实例时，会自动调用构造器，构造器可以重载**
   - ***应用场景：创建对象时，可以同时完成对对象成员变量(属性)的初始化赋值***
   - **注意事项：类自带无参构造器，但是若自己写了一个有参构造器，类中自带的无参构造器无法使用**
2. **this关键字**
   - this是一个变量，可以用在方法中，来拿到当前对象
   - 用途：解决变量名冲突的问题
   - ***可以把this理解为当前对象本身，this.name就是当前对象的name属性***
   - **可以解决当成员变量名和传参名相同时，一个用 this.name，一个用name来解决**
   - ```
     public class student {
     String name;
     int age;
     String sex; //对象的属性
     public void print(){ 对象的方法
     System.out.println(this.name);
     System.out.println(this.age);
     System.out.println(this.sex);
     }
     public student(String name,int age,String sex){ 标准的有参构造器，使用this关键字
     this.name=name;
     this.age=age;
     this.sex=sex;
     }//相同的属性名，this.name表示当前对象的name属性，name表示传参的name参数
     }
     ```
3. **封装**
   - 就是用类设计对象处理某一个事物的数据时，应该要把处理的数据以及处理这些数据的方法，设计到一个对象中去
   - 面向对象的三大特征：封装、继承、多态
   - 类就是一种封装

- 封装的设计要求：**合理隐藏，合理暴露**
- 如何合理隐藏：***用private关键字修饰变量***,这样类内可以访问但是外部对象不可访问
- ```
  public class student {
  private String name;
  private int age;
  private double chinese;
  private double math;
  public void printallscore(){
  System.out.println(name+"的总成绩为："+(chinese+ math));
  }
  public void printallscore2(){
  System.out.println(name+"的平均成绩为："+(chinese+ math)/2);
  }
  }
  ```
- 如何合理暴露：***用public修饰的get和set方法*** 合理暴露成员变量的取值和赋值
- ```
  public void setage(int a){//暴露赋值
  if(a>0&&a<=100)
  {
  age=a;
  }
  }
  public int getage(){ //暴露取值
  return age;
  }
  可以在主函数中：
  public static void main(String[] args) {
  student s1 = new student();
  s1.setage(19); //进行赋值
  System.out.println(s1.getage()); //进行取值
  }
  ```

4. **Javabean**
   - 是一种特殊类，叫做实体类，满足以下要求：
     - 类中成员变量全部私有，并提供public修饰的getter、setter方法
     - 类中需要提供一个无参数构造器，有参数构造器可选
   - ```
     public class student {
     private String name;
     private int age;
     private double chinese;
     private double math;
     public student(String name, int age, double chinese, double math) {有参数构造器
     ```
     ```

```
this.name = name;
this.age = age;
this.chinese = chinese;
this.math = math;
}
public student() { 无参数构造器
}
public String getName() {
return name;
}
public int getAge() {
return age;
}
public double getChinese() {
return chinese;
}
public double getMath() {
return math;
}
public void setName(String name) {
this.name = name;
}
public void setAge(int age) {
this.age = age;
}
public void setChinese(double chinese) {
this.chinese = chinese;
}
public void setMath(double math) {
this.math = math;
}
}
```

  - 右键空白处，点击Generate，点击setter或getter选项，按住shift键，鼠标点击最后一行，即可自动生成
  - 右键空白处，点击construction，按住shift键，鼠标点击最后一行，点OK，可生成有参数构造器
  - 右键空白处，点击construction，选择select none，即可生成无参数构造器
- 实体类的基本作用：创建对象，存取数据(封装数据)
- ```
  public static void main(String[] args) {
  student s1 = new student();
  ```
  ```

```
s1.setName("CXZ");
s1.setAge(18);
s1.setChinese(100);
s1.setMath(100);
System.out.println(s1.getName());
System.out.println(s1.getAge());
System.out.println(s1.getChinese());
System.out.println(s1.getMath()); // 上述是无参构造器生成的对象，需要一个一个存
student s2 = new student("CXZ",18,100,100);
System.out.println(s2.getName());
System.out.println(s2.getAge());
System.out.println(s2.getChinese());
System.out.println(s2.getMath()); 上述是有参构造器生成的对象，一起存进去
}
```

- 实体类的应用场景
  - **分层思想：实体类的对象只负责数据存取，而数据的处理交给其他类的对象完成，实现数据和业务处理的分离**
  - ```
    student_service ss = new student_service(); 主函数内
    ss.print_sum_grade(s1);
    ss.print_average_grade(s1);
    ss.print_sum_grade(s2);
    ss.print_average_grade(s2);
    public class student_service { 另一个类中
    //打印学生总成绩
    public void print_sum_grade(student s){
    System.out.println(s.getName()+"的总成绩是："+(s.getChinese()+s.getMath()));
    }
    //打印学生平均成绩
    public void print_average_grade(student s){
    System.out.println(s.getName()+"的平均成绩是："+
    (s.getChinese()+s.getMath())/2);
    }
    }
    ```
  - ```
    student_service ss = new student_service(s2);主函数内，这是另一种传入数据的方法
    ss.print_sum_grade();
    ss.print_average_grade();
    public class student_service {另一个类中
    private student s; 私有化对象
    public student_service(student s) { 有参数构造器
    ```

```
this.s = s;
}
//打印学生总成绩
public void print_sum_grade(){
System.out.println(s.getName()+"的总成绩是："+(s.getChinese()+s.getMath()));
}
//打印学生平均成绩
public void print_average_grade(){
System.out.println(s.getName()+"的平均成绩是："+
(s.getChinese()+s.getMath())/2);
}
}
```

5. **static**
   - 叫静态，可以修饰成员变量、成员方法
   - 成员变量有静态变量(类变量，有static修饰)，和实例变量(对象的变量，无static修饰)之分
   - 静态变量：有static修饰，属于类，在计算机里只有一份，会被类的全部对象共享，各个类中没有这个
   - 实例变量：无static修饰，属于每个对象的，各个对象都有
   - ```
     public class student {
     static String name; //静态成员变量
     int age; //实例成员变量
     }
     ```
   - 静态变量的访问：
   - ```
     student.name="CXZ";
     System.out.println(student.name);
     ```
     直接用类名去赋值和访问，每个类共享这个，如果多个对象修改了这个值，就会覆盖直到最后一个对象赋的值
   - 实例变量的访问：
   - ```
     student s1 = new student();
     s1.age=18;
     student s2 = new student();
     s2.age=19;
     System.out.println(s1.age);
     System.out.println(s2.age);
     ```
     每个对象各自持有
   - 静态变量的应用场景
     - 在开发中，如果某个数据只需要一份，且希望能被共享(访问、修改)，则该数据可以定义为类变量来记住
     - 比如：系统启动后，要求用户类记住自己创建了多少个用户对象，则可用静态变量，因为结果只有一份
     ```

```
public class user {
public static int count=0;
public user(){ 构造器
count++;
}
}
public class test2 {
public static void main(String[] args) {
new user();
new user();
new user();
System.out.println(user.count); 打印结果是3，因为创建了3个对象
}
}
```

- static修饰方法：静态方法和实例方法
- 静态方法：有static修饰的成员方法，属于类
- 实例方法：无static修饰的成员方法，属于对象

```
public class student {
private double score;
public static void printhelloworld(){ 静态方法
System.out.println("hello world");
System.out.println("hello world");
System.out.println("hello world");
}
public void printpass(){ 动态方法
System.out.println(score>=60?"通过":"不通过");
}
public void setscore(double score){
this.score=score;
}
}
public class test { test类中
public static void main(String[] args) {
student.printhelloworld(); 静态方法调用
//
student s1 = new student();
student s2 = new student();
s1.setscore(60); 动态方法调用
s2.setscore(50);
s1.printpass();
```

```
s2.printpass();
}
}
```

- ***如果这个方法只是为了做一个功能，不需要访问对象的数据，则这个方法可为静态方法***
- ***如果这个方法是对象的行为，需要访问对象的数据，则必须定义为实例方法***
- 静态方法的应用场景：**做工具类**
- 工具类：类中的方法全是一些静态方法，每个方法都是用来完成一个功能的，以便直接使用，调用方便，代码复用，提高了开发效率。(要是用实例方法就得创建对象调用，浪费内存)
- ```
  public class util {
  //静态类验证码方法
  public static String getCode(int length){
  String code = "";
  for (int i = 0; i < length; i++) {
  int num = (int)(Math.random()*10);
  code += num;
  }
  return code;
  } ***比如这个验证码方法，可以设置为静态方法，这样登录类和注册类都可以直接调用这个工具类的方法，避免重复书写***
  }
  ```

- 工具类没有创建对象的需求，故可以把构造器私有(不加也行其实)
- ```
  private util(){
  }
  ```

- 注意事项：
  - ***静态方法中可以直接访问静态成员，不可以直接访问实例成员，但是可以间接访问，即在静态方法中声明一个对象然后访问***
  - ***实例方法中既可以直接访问静态成员，也可以直接访问实例成员***
  - ***实例方法中可以出现this关键字，静态方法中不可以***