day15 【单例模式、多例模式、枚举、工厂模式】

今日目标

- 单例模式---->必须掌握
  - 饿汉式单例
  - 懒汉式单列
- 多例模式---->必须掌握
- 枚举---->必须掌握
  - 定义枚举
  - 使用枚举
- 工厂模式---->必须掌握
- lombok插件

第一章 单例设计模式

1.1 单例设计模式的概述

引入

```
public class Person{}
public class Test{
    public static void main(String[] args){
        Person p1 = new Person();
        Person p2 = new Person();
        Person p3 = new Person();
        ...
    }
}
```

单例设计模式的作用

单例模式，是一种常用的软件设计模式。通过单例模式可以保证系统中，应用该模式的这个类只有一个实例。即一个类只有一个对象实例。

单例设计模式实现步骤

1. 将构造方法私有化，使其不能在类的外部通过new关键字实例化该类对象。
2. 在该类内部创建一个唯一的对象
3. 定义一个静态方法返回这个唯一对象。

例设计模式的类型

根据实例化对象的时机单例设计模式又分为以下两种:

1. 饿汉单例设计模式
2. 懒汉单例设计模式

1.2 饿汉式单例设计模式

- 概述: 饿汉单例设计模式就是使用类的时候已经将对象创建完毕，不管以后会不会使用到该类的唯一对象，先创建了再说。很着急的样子，故被称为“饿汉模式”。
- 代码如下:

```
// 饿汉式单例设计模式
public class Person {
    // 概述:  饿汉单例设计模式就是使用类的时候已经将对象创建完毕，
    // 不管以后会不会使用到该类的唯一对象，先创建了再说。很着急的样子，故被称为“饿汉模式”。

    // 1.将构造方法私有化,防止外界通过new直接创建该类的对象
    private Person(){

    }

    // 2.定义一个私有的成员变量,用来存储该类的唯一对象
    private static final Person P = new Person();

    // 3.提供一个公共的静态方法用来获取该类的唯一对象
    public static Person getInstance(){
        return P;
    }

    public static void method(){}

}
```

```
public class Test {
    public static void main(String[] args) {
        Person.method();// 就已经创建了Person类的唯一对象

        // 获取该类的唯一对象
        System.out.println(Person.getInstance());
        System.out.println(Person.getInstance());
        System.out.println(Person.getInstance());
        System.out.println(Person.getInstance());
        System.out.println(Person.getInstance());
    }
}
```

1.3 懒汉式单例设计模式

- 概述: 懒汉单例设计模式就是调用getInstance()方法时对象才被创建，先不急着创建出对象，等要用的时候才创建对象。不着急，故称为“懒汉模式”。
- 代码如下:

```
// 懒汉式单列设计模式
public class Person {
    // 概述:  懒汉单例设计模式就是调用getInstance()方法时对象才被创建,
    // 先不急着创建出对象，等要用的时候才创建对象。不着急，故称为“懒汉模式”。

    // 1.将构造方法私有化,防止外界通过new调用构造方法创建该类对象
    private Person(){

    }

    // 2.定义一个私有的静态成员变量,用来存一下该类唯一的对象
    private static Person p; // 初始值为null

    // 3.提供一个公共的静态方法给外界获取该类的唯一对象
    public static synchronized Person getInstance(){
        // 判断: 如果是第一次调用getInstance方法,就创建该类的唯一对象,否则就返回第一次
创建的唯一对象
        if (p == null){
            // 说明是第一次调用getInstance()方法
            p = new Person();
        }
        return p;
    }

    public static void method(){}

}
```

```
public class Test {
    public static void main(String[] args) {
        Person.method();

        // 使用该类的唯一对象
        System.out.println(Person.getInstance());
        System.out.println(Person.getInstance());
        System.out.println(Person.getInstance());
        System.out.println(Person.getInstance());
    }
}
```

- **注意:** 懒汉式单列设计模式在多线程的情况下很容易出现创建多个对象,所以getInstance方法需要加锁

第二章 多例设计模式

2.1 多例设计模式

多例设计模式的作用

多例模式，是一种常用的软件设计模式。通过多例模式可以保证系统中，应用该模式的类有**固定数量**的对象产生。

说白了,多例设计模式就是保证使用该模式的类会有固定数量的该类对象产生

实现步骤

1.创建一个类, **将构造方法私有化**，使其不能在类的外部通过new关键字实例化该类对象。

2.在该类内部产生固定数量的对象

3.提供一个静态方法来随机获取一个该类的对象

实现代码

```
// 多例设计模式: 保证该类只有3个对象产生
public class Person {
    // 1.将构造方法私有化,防止外界通过new调用构造方法创建对象
    private Person() {

    }

    // 2.在该类的内部创建固定数量的该类对象
    private static ArrayList<Person> list = new ArrayList<>();

    static {
        for (int i = 0; i < 3; i++) {
            // 创建该类的对象
            Person p = new Person();
            // 添加到集合中
            list.add(p);
        }
    }

    // 3.提供一个公共的静态方法随机返回一个该类的对象
    public static Person getInstance(){
        // 创建Random对象
        Random r = new Random();

        // 生成一个随机索引
        int index = r.nextInt(list.size());

        // 根据索引取对象,返回
        return list.get(index);
    }

}
```

```
public class Test {
    public static void main(String[] args) {
        System.out.println(Person.getInstance());
        System.out.println(Person.getInstance());
        System.out.println(Person.getInstance());
        System.out.println(Person.getInstance());
        System.out.println(Person.getInstance());
        System.out.println(Person.getInstance());
        System.out.println(Person.getInstance());
        System.out.println(Person.getInstance());
```

```
        System.out.println(Person.getInstance());
    }
}
```

运行结果：

```
com.itheima.demo3_多例设计模式.Person@74a14482
com.itheima.demo3_多例设计模式.Person@1540e19d
com.itheima.demo3_多例设计模式.Person@1540e19d
com.itheima.demo3_多例设计模式.Person@677327b6
com.itheima.demo3_多例设计模式.Person@677327b6
com.itheima.demo3_多例设计模式.Person@74a14482
com.itheima.demo3_多例设计模式.Person@1540e19d
com.itheima.demo3_多例设计模式.Person@677327b6
com.itheima.demo3_多例设计模式.Person@74a14482
```

第三章 枚举

3.1 枚举的定义和使用--重点

不使用枚举存在的问题

假设我们要定义一个人类，人类中包含姓名和性别。通常会将性别定义成字符串类型，效果如下：

```
public class Person {
    private String name;
    private String sex;

    public Person() {
    }

    public Person(String name, String sex) {
        this.name = name;
        this.sex = sex;
    }

    // 省略get/set/toString方法
}
```

```
public class Demo01 {
    public static void main(String[] args) {
        Person p1 = new Person("张三", "男");
        Person p2 = new Person("张三", "abc"); // 因为性别是字符串,所以我们可以传入任意字符串
    }
}
```

不使用枚举存在的问题：可以给性别传入任意的字符串，导致性别是非法的数据，不安全。

枚举的概念

枚举是一种引用数据类型,java中枚举的底层是一个有固定个数对象的"特殊类"。**所以如果某种类型的数据有固定个值,就可以定义为枚举类型。比如性别, 季节，方向。**

定义枚举的格式

- 格式:

```
public enum 枚举名{
    枚举值,枚举值,枚举值,...
}
// 枚举值一般都是所有字母大写
```

- 案例:

```
public enum Sex {
    BOY,
    GIRL,
    YAO
}

public enum Season {
    SPRING,
    SUMMER,
    AUTUMN,
    WINTER
}

public enum Direction {
    UP,
    DOWN,
    LEFT,
    RIGHT
}
```

枚举的使用

- 格式: `枚举名.枚举值`
- 案例:

```
public class Person {
    private String name;
    private Sex sex;

    public Person(String name, Sex sex) {
        this.name = name;
        this.sex = sex;
    }

    public Person() {
    }

    public Sex getSex() {
        return sex;
    }

    public void setSex(Sex sex) {
        this.sex = sex;
```

```
    }

    public String getName() {
        return name;
    }

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

    @Override
    public String toString() {
        return "Person{" +
                "name='" + name + '\'' +
                ", sex=" + sex +
                '}';
    }
}


public class Test {
    public static void main(String[] args) {
        // 格式: 枚举类型.枚举值
        // 创建Person对象
        Person p1 = new Person();
        p1.setName("张三");
        p1.setSex(Sex.BOY);

        // 创建Person对象
        Person p2 = new Person("李四", Sex.GIRL);

        // 打印输出
        System.out.println("p1:" + p1);
        System.out.println("p2:" + p2);
    }
}
```

3.2 枚举中的其他内容(听听就好)

- 枚举的本质是一个使用了多例设计模式的类，所以枚举中还可以有**成员变量，成员方法,构造方法等。**
- 枚举的本质是一个类，我们刚才定义的Sex枚举最终效果如下：

```
// 枚举本质其实就是一个使用了多例设计模式的类,所以枚举中还可以有成员变量，成员方法,构造方法等。
public enum Sex {
    BOY,GIRL,YAO;

    // 成员变量
    public int num = 10;

    // 构造方法
    private Sex(){
```

```
    }

    // 成员方法
    public void show(){
        System.out.println("show....");
    }

}

public class Test {
    public static void main(String[] args) {
        // 使用Sex枚举
        // 1.定义一个Sex枚举类型的变量
        Sex sex1 = Sex.GIRL;

        // 2.使用sex1访问成员变量或者成员方法
        System.out.println(sex1.num);// 10
        sex1.show();// show...
    }
}
```

第四章 工厂设计模式

4.1 工厂模式的概述

工厂模式的介绍

工厂模式（Factory Pattern）是 Java 中最常用的设计模式之一。这种类型的设计模式**属于创建型模式**，它提供了一种创建对象的最佳方式。之前我们创建类对象时, 都是使用new 对象的形式创建, 除new对象方式以外, 工厂模式也可以创建对象.

耦合度: 类与类之间的关系,如果关系比较强,高耦合, 如果关系比较弱,低耦合(开发)

需求: 有10个类,需要在10个测试类中,分别创建这10个类的对象

以前: 直接通过new 来创建 --->每一个测试类都需要和这10个类进行关联--(耦合度高)

工厂模式: 定义一个工厂类,专门用来创建这10个类的对象, 并提供获取对象的方法,那这个时候测试类只需要跟工厂类关联即可----> 低耦合

工厂模式的作用

将获取对象的代码与要创建对象的代码进行分开，获取对象的代码不需要直接创建对象，也就不需要关心创建对象时需要的数据。只需要通过工厂类获取对象即可。

- 解决类与类之间的耦合问题

案例演示

需求

1. 编写一个Car接口, 提供run方法

2. 编写一个Falali类实现Car接口,重写run方法
3. 编写一个Benchi类实现Car接口,重写run方法

   提供一个工厂类,可以用来生产汽车对象

实现代码

1.编写一个Car接口, 提供run方法

```
public interface Car {
    void run();
}
```

2.编写一个Falali类实现Car接口,重写run方法

```
public class Falali implements Car {
    @Override
    public void run() {
        System.out.println("法拉利正在以300迈的速度行驶...");
    }
}
```

3.编写一个Benchi类实现Car接口

```
public class Benchi implements Car {

    @Override
    public void run() {
        System.out.println("奔驰正在以200迈的速度行驶...");
    }
}
```

4.提供一个CarFactory(汽车工厂),用于生产汽车对象

```
public class CarFactory {

    /**
     * 用来创建汽车对象的方法
     * @param carType
     * @return 汽车对象
     */
    public static Car createCar(String carType){
        if ("Falali".equalsIgnoreCase(carType)){
            // 返回Falali对象
            return new Falali();

        }else if ("Benchi".equalsIgnoreCase(carType)){
            // 返回Benchi对象
            return new Benchi();
        }else{
            return null;
        }
    }
```

```
}
```

5.定义CarFactoryTest测试汽车工厂

```
public class Test1 {
    public static void main(String[] args) {
       /* // 创建Falali汽车对象
        Falali fll = new Falali();
        fll.run();

        // 创建Benchi汽车对象
        Benchi bc = new Benchi();
        bc.run();*/

        // 创建Falali汽车对象
        Car car1 = CarFactory.createCar("Falali");
        car1.run();

        // 创建Benchi汽车对象
        Car car2 = CarFactory.createCar("Benchi");
        car2.run();


    }
}
```

第五章 Lombok【自学扩展】

5.1 Lombok的使用

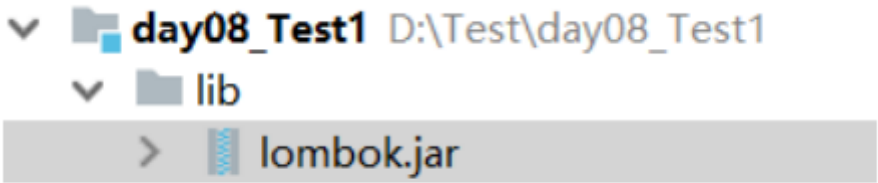
lombok介绍

- lombok可以**使用注解的方式让一些代码变的简洁 方便**
- 实体类中有一些固定的代码：构造方法，getter/setter、equals、hashcode、toString方法都是固定的，写出来看着比较麻烦。而Lombok能通过注解的方式，在编译时自动为属性生成这些代码。

lombok使用

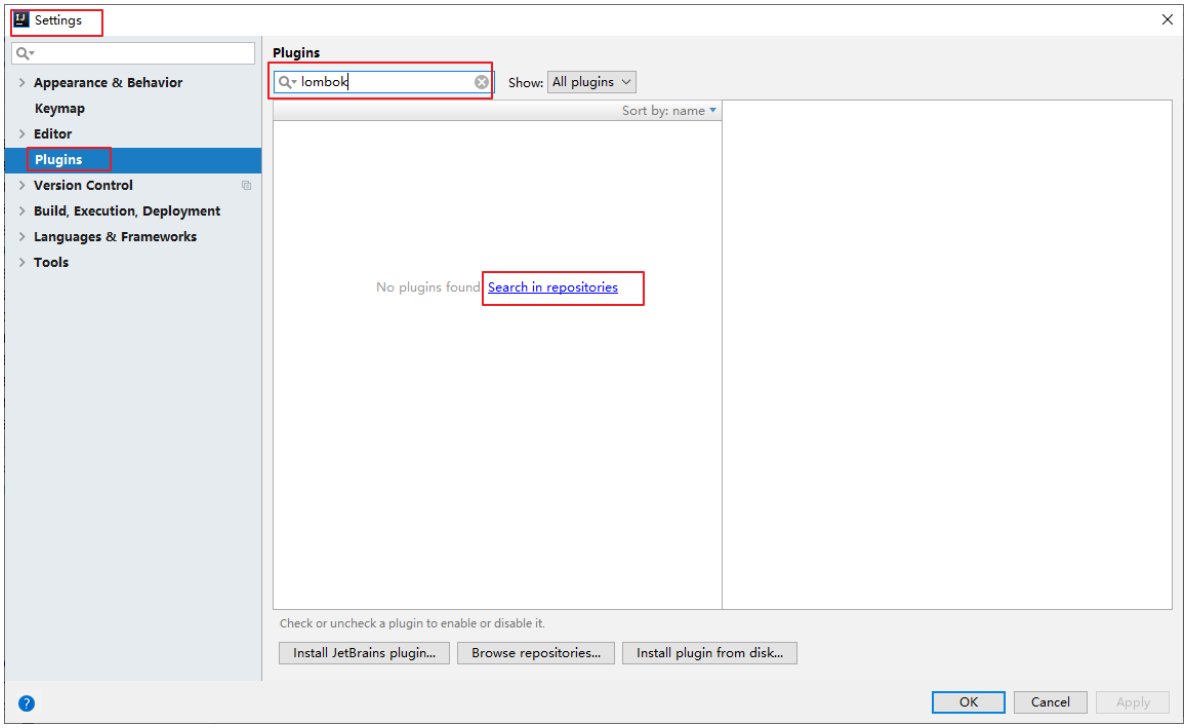
1. 添加lombox的jar包：

将lombok.jar(本例使用版本：1.18.10)，添加到模块目录下，并添加到ClassPath

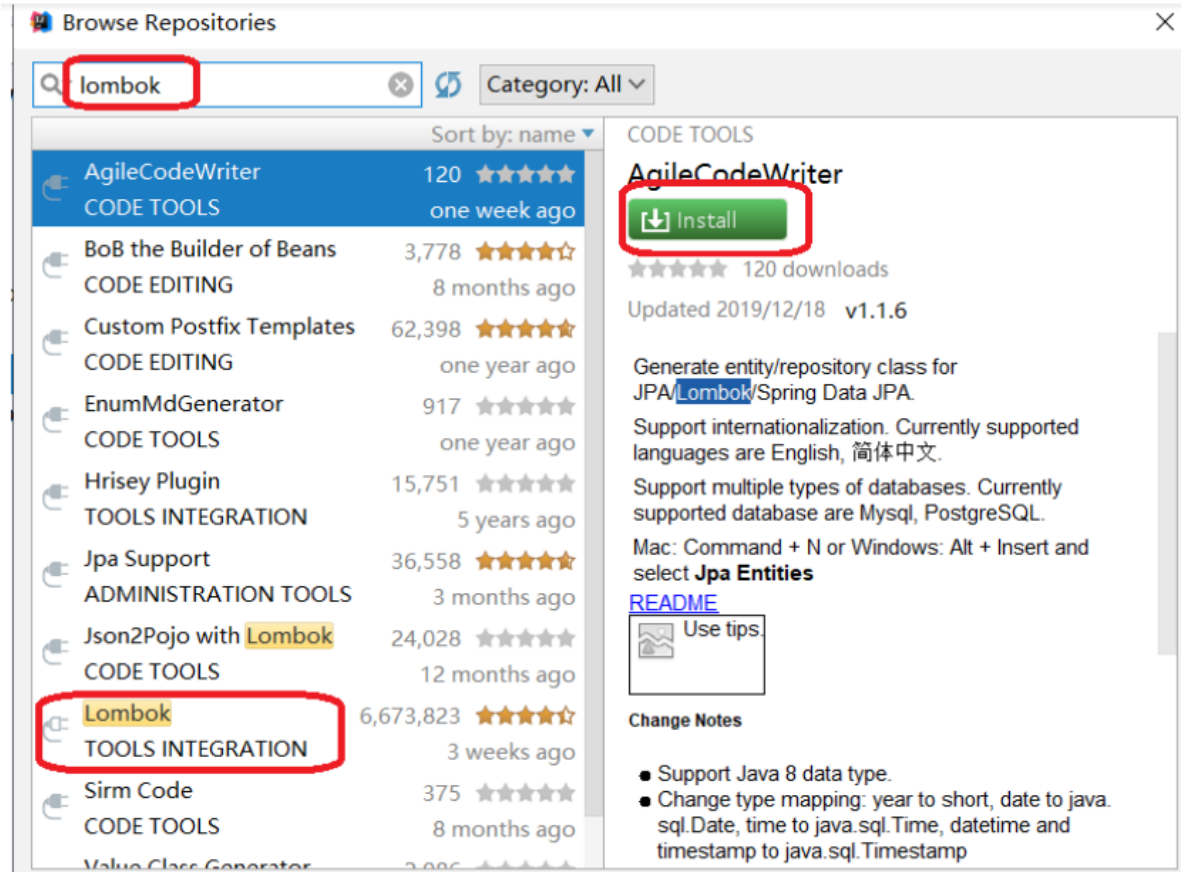


2. 为IDEA添加lombok插件（**连接网络使用**）

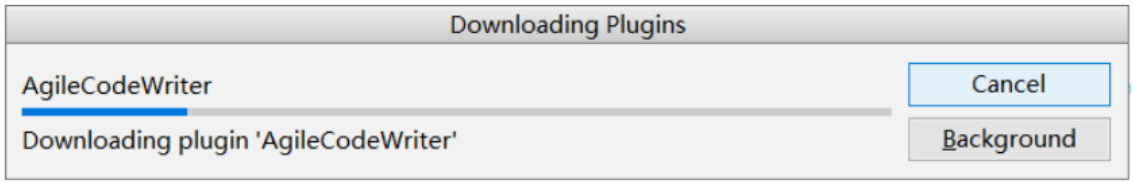
- 第一步



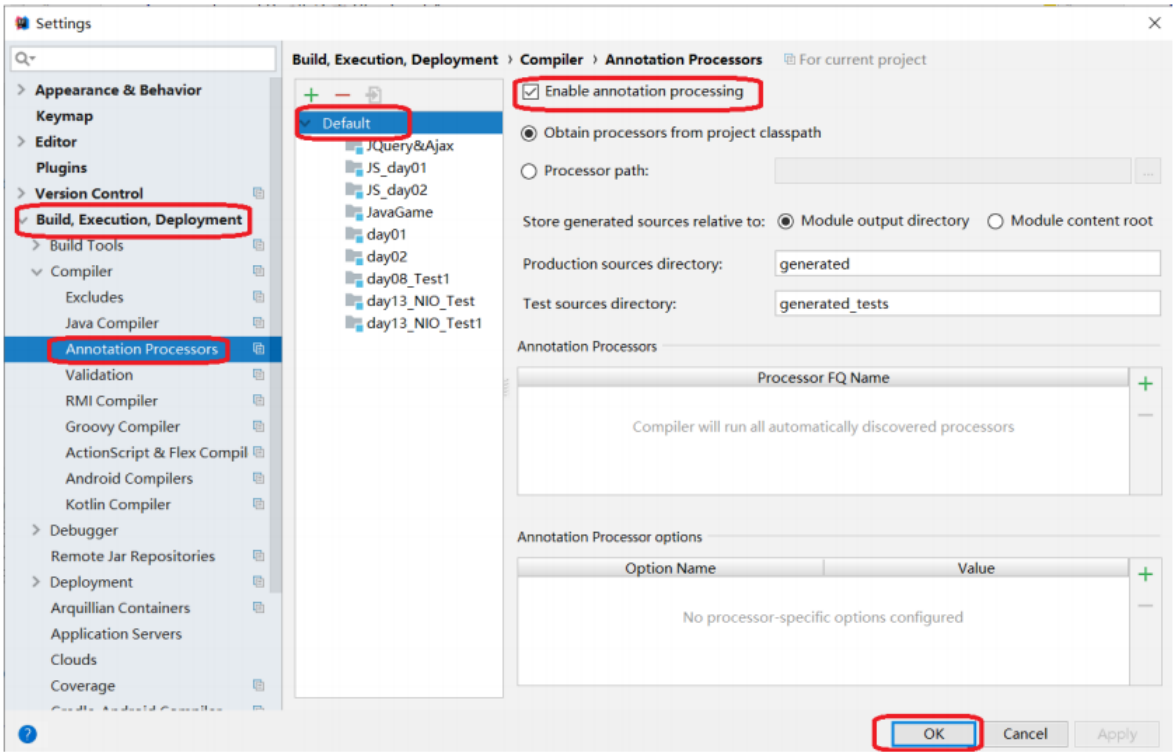
- 第二步：



- 第三步：



- 第四步：



1. **安装完毕后，重启IDEA。**
2. 新建一个类：Student

```java
import lombok.Data;

@Data
public class Student {
    private String name;
    private int age;
    private String sex;
}

```

```java
Student stu = new Student();
stu.setName("张三");
stu.setAge(21);
stu.setSex("男");
System.out.println(stu);
```

lombok常用注解

- @Getter和@Setter
  - 作用：生成成员变量的get和set方法。
  - 写在成员变量上，指对当前成员变量有效。
  - 写在类上，对所有成员变量有效。
  - 注意：静态成员变量无效。
- @ToString：
  - 作用：生成toString()方法。
  - 该注解只能写在类上。
- **@NoArgsConstructor和@AllArgsConstructor**
  - @NoArgsConstructor：无参数构造方法。
  - @AllArgsConstructor：满参数构造方法。
  - 注解只能写在类上。
- @EqualsAndHashCode
  - 作用：生成hashCode()和equals()方法。
  - 注解只能写在类上。
- **@Data**
  - 作用： 生成setter/getter、equals、hashCode、toString方法，如为final属性，则不会为该属性生成setter方法。
  - 注解只能写在类上。
- 案例:

```
@Data
@NoArgsConstructor
```

```java
@AllArgsConstructor
public class Person {
    private String name;
    private int age;

    // 构造方法--空参,满参
    // set\get
    // toString
    // equals and hashCode

}


public class Test {
    public static void main(String[] args) {
        // 创建Person对象
        Person p1 = new Person();
        p1.setName("张三");
        p1.setAge(18);
        System.out.println(p1.getName() + "," + p1.getAge());
        System.out.println("p1:" + p1);

        Person p2 = new Person("张三",18);
        System.out.println(p1.equals(p2));// true
        System.out.println(p1.hashCode());// 45721950
        System.out.println(p2.hashCode());// 45721950
    }
}
```

总结

必须练习:

1.单例设计模式---->1.2 1.3
2.多例设计模式---->2.1
3.定义和使用枚举-->3.1
4.工厂设计模式---->4.1

- 能够说出单例设计模式的好处
  保证使用该模式设计的类只有1个对象产生
  步骤:
  1.将构造方法私有化
  2.在类的内部创建该类的唯一对象
  3.提供公共静态方法用来获取该类的唯一对象

- 能够说出多例模式的好处
  保证使用该模式设计的类只有固定数量对象产生
  步骤:
  1.将构造方法私有化
  2.在类的内部创建该类固定数量的对象
  3.提供公共静态方法用来获取该类中创建的任意对象

- 能够定义枚举
  格式:

```
public enum 枚举名{
    枚举值,枚举值,...
}
注意:枚举值一般所有字母大写
```

使用: 枚举名.枚举值

- 能够使用工厂模式编写java程序
  定义一个类,提供一个静态方法,在静态方法中创建类的对象并返回