2、具体内容

单例设计模式（多例设计模式）主要是一种控制实例化对象产生个数的设计操作。

■单例设计

如果说现在有一个程序类，假设该程序类的定义如下：

```
class Singleton {
	public void print() {
		System.out.println("www.mldn.cn") ;
	}
}
public class JavaDemo {
	public static void main(String args[]) {
		Singleton instanceA = new Singleton() ;
		Singleton instanceB = new Singleton() ;
		Singleton instanceC = new Singleton() ;
		instanceA.print() ;
		instanceB.print() ;
		instanceC.print() ;
	}
}
```

但是由于某些要求，现在要求Singleton这个类只允许提供有一个实例化对象。那么此时首先应该控制的就是构造方法，因为所有的新的实例化对象产生了，那么一定要调用构造方法，如果构造方法“没有了”，那么自然就无法产生实例化对象。

范例：构造方法私有化

```
class Singleton {
	private Singleton() {}	// 构造方法私有化了
	public void print() {
		System.out.println("www.mldn.cn") ;
	}
}
```

```
public class JavaDemo {
    public static void main(String args[]) {
        Singleton instance = null ;     // 声明对象
        instance = new Singleton() ;//错误：Singleton()可以在Singleton中访问private
    }
}
```

但是现在是有一个严格要求的：必须产生有一个实例化对象，所以现在就必须想办法产生一个实例化对象交给客户端去调用。那么这个时候的分析如下

1、private访问权限的主要特点在于：不能再类外部访问，但是可以在类本身调用，所以现在可以考虑在类的内部调用构造；

```
class Singleton {
    private Singleton instance= new Singleton() ;
    private Singleton() {}      // 构造方法私有化了
    public static Singleton getInstance() {
        return INSTANCE ;
    }
    public void print() {
        System.out.println("www.mldn.cn") ;
    }
}
```

2、此时Singleton类内部的instance属于一个普通的属性，而普通属性是在有实例化对象产生之后才会被调用的，那么这个时候外部无法产生实例化对象，所以这个属性就不能够访问到了，那么就必须考虑如何在没有实例化对象的时候获取此属性，那么只有static属性可以访问。

```
class Singleton {
    static Singleton instance = new Singleton() ;
    private Singleton() {}      // 构造方法私有化了
    public void print() {
        System.out.println("www.mldn.cn") ;
    }
}
public class JavaDemo {
    public static void main(String args[]) {
        Singleton instance = null ;     // 声明对象
        instance = Singleton.instance();
        instance.print() ;
    }
}
```

3、类中的属性应该封装后使用，所以理论上此时的instance需要被封装起来，那么就需要通过一个static方法获得。

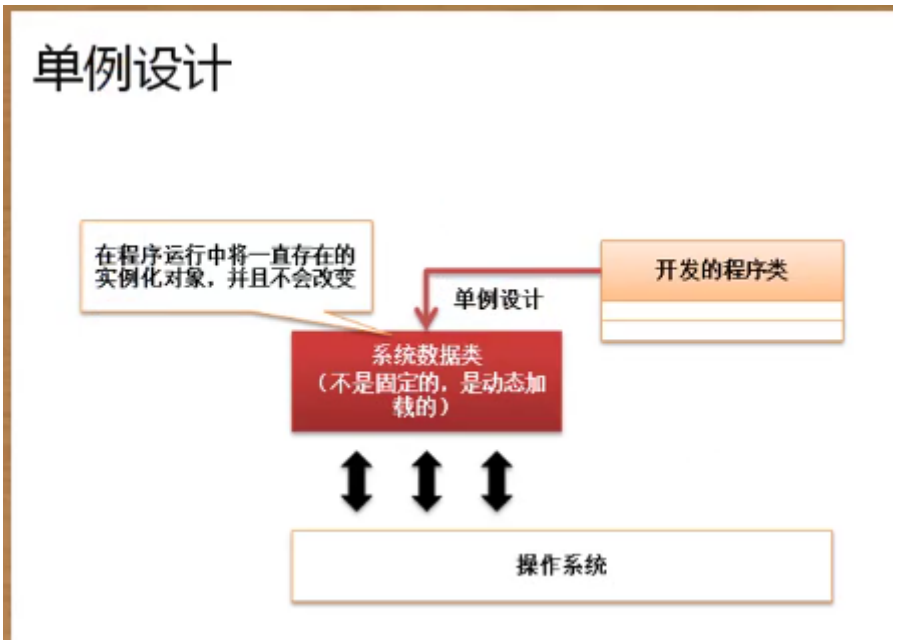
```
class Singleton {
    private static Singleton instance = new Singleton() ;
    private Singleton() {}      // 构造方法私有化了
    public static Singleton getInstance() {
        return INSTANCE ;
    }
```

```
    public void print() {
        System.out.println("www.mldn.cn") ;
    }
}
public class JavaDemo {
    public static void main(String args[]) {
        Singleton instance = null ;     // 声明对象
        instance = Singleton.getInstance();
        instance.print() ;
    }
}
```

4、整个代码从头强调的是只有一个实例化对象，这个时候虽然提供有static的实例化对象，但是这个对象可以被重新实例化。所以需要保证此时Singleton类内部的instance无法再次实例化，那么应该使用final定义。

```
class Singleton {
    private static final Singleton INSTANCE = new Singleton() ;
    private Singleton() {}      // 构造方法私有化了
    public static Singleton getInstance() {
        return INSTANCE ;
    }
    public void print() {
        System.out.println("www.mldn.cn") ;
    }
}
public class JavaDemo {
    public static void main(String args[]) {
        Singleton instance = null ;     // 声明对象
        instance = Singleton.getInstance();
        instance.print() ;
    }
}
```

在很多情况下有些类时不需要重复产生对象的，例如：如果一个程序启动，那么现在肯定需要有一个类负责保存有一些程序加载的数据信息。



对于单例设计模式也分为两种：懒汉式、饿汉式。在之前所定义的都属于饿汉式。在系统加载类的时候就会自动提供有Singleton类的实例化，而还有一种懒汉式，在第一次使用的时候进行实例化对象处理。

范例：将单例修改为懒汉式

```
class Singleton {
	private static Singleton instance ;
	private Singleton() {}	// 构造方法私有化了
	public static Singleton getInstance() {
		if (instance == null) {	// 第一次使用
			instance = new Singleton() ; // 实例化对象
		}
		return instance ;
	}
	public void print() {
		System.out.println("www.mldn.cn") ;
	}
}
public class JavaDemo {
	public static void main(String args[]) {
		Singleton instance = null ;	// 声明对象
		instance = Singleton.getInstance();
		instance.print() ;
	}
}
```

面试题：请编写一个Singleton程序，并说明其主要特点？

- 代码如上，可以把懒汉式（后面需要考虑到线程同步问题）和饿汉式都写上；
- 特点：构造方法私有化，类内部提供static方法获取实例化对象，这样不管外部如果操作永远都只有一个实例化对象提供。

■多例设计

与单例设计对应的还有一个称为多例设计，单例设计指的是值保留有一个实例化对象，而多例设计指的是可以保留有多个实例化对象，例如：如果现在要定义一个描述性别的类，那么该对象只有两个：男、女。或者描述颜色的基色的类，可以使用：红色、绿色、蓝色。这种情况下就可以利用多例设计来解决。

范例：实现多例设计

```
class Color { // 定义描述颜色的类
	private static final Color RED = new Color("红色") ;
	private static final Color GREEN = new Color("绿色") ;
	private static final Color BLUE = new Color("蓝色") ;
	private String title ;
	private Color(String title) {	// 构造方法私有化
		this.title = title ;
	}
	public static Color getInstance(String color) {
		switch(color) {
			case "red": return RED ;
			case "green": return GREEN ;
			case "blue": return BLUE ;
			default: return null ;
		}
	}
	public String toString() {
		return this.title ;
	}
}
public class JavaDemo {
	public static void main(String args[]) {
		Color c = Color.getInstance("green") ;
		System.out.println(c) ;
	}
}
```

多例设计与单例设计的本质是相同的，一定都会在内部提供有static方法以返回实例化对象。