一、类变量和类方法

1、类变量-简介

类变量也叫静态变量/静态属性，是该类的所有对象共享的变量，任何一个该类的对象去访问它时，取到的都是相同的值，同样任何一个该类的对象去修改它时，修改的也是同一个变量。

2、类变量-语法

```
访问修饰符 static 数据类型 变量名;
static 访问修饰符 数据类型 变量名;
```

3、类变量-调问

```
类名.类变量名
对象名.类变量名
```

4、类变量-细节

4.1、使用场景：当我们需要让某个类的所有对象都共享一个变量时，就可以考虑使用类变量(静态变量)
4.2、类变量和实例变量区别：类变量是该类的所有对象共享的，而实例变量是每个对象独享的
4.3、加上static称为类变量或静态变量，否则称为实例变量/普通变量/非静态变量
4.4、类变量可以通过 类名.类变量名 或者 对象名.类变量名 来访问，但Java设计者推荐我们使用类名.类变量名方式访问
4.5、实例变量不能通过 类名.类变量名 方式访问
4.6、类变量是在类加载时就初始化了，也就是说，即使你没有创建对象，只要类加载了，就可以使用类变量
4.7、类变量的生命周期是随类的加载开始，随着类消亡而销毁

5、类方法-简介

类方法也叫静态方法。当方法中不涉及到任何和对象相关的成员，则可以将方法设计成静态方法，提高开发效率。

6、类方法-语法

```
访问修饰符 static 数据返回类型 方法名() {}
static 访问修饰符 数据返回类型 方法名() {}
```

7、类方法-调用

```
类名.类方法名
对象名.类方法名
```

8、类方法-细节

8.1、类方法和普通方法都是随着类的加载而加载，将结构信息存储在方法区：
8.2、类方法可以通过类名调用，也可以通过对象名调用
8.3、普通方法和对象有关，需要通过对象名调用，如 对象名.方法名(参数)
8.4、类方法中不允许使用和对象有关的关键字，如this和super
8.5、类方法(静态方法)中只能访问静态变量或静态方法
8.6、普通成员方法，既可以访问 非静态成员，也可以访问静态成员

总结：静态方法，只能访问静态的成员；非静态方法，可以访问静态成员和非静态成员

二、理解main方法语法

1、语法

```
public static void main(String[] args) {}
```

2、深入理解

2.1、main方法是虚拟机调用
2.2、Java虚拟机需要调用类的main()方法，所以该方法的访问权限必须是publi
2.3、Java虚拟机在执行main()方法时不必创建对象，所以该方法必须是static
2.4、该方法接收String类型的数组参数，该数组中保存执行Java命令时传递给所运行的类的参数

3、细节

3.1、在main()方法中，我们可以直接调用main方法所在类的静态方法或静态属性
3.2、但是，不能访问该类中的非静态成员，必须创建该类的一个实例对象后，才能通过这个对象去访问类中的非静态成员

三、代码块

1、简介

代码块又称为初始化块，属于类中的成员（即类的一部分），类似于方法，将逻辑语句封装在方法体中，通过{}包围起来。

2、语法

```
[修饰符] {
    代码
}

注意：
    1) 修饰符可选，要写的话，也只能写static
    2) 代码块分为两类，使用static 修饰的叫静态代码块；没有static修饰的，叫普通代码块/非静态代码块
    3) 逻辑语句可以为任何逻辑语句（输入、输出、方法调用、循环、判断等）
    4) ;号可以写，也可以忽略
```

3、优点

```
3.1、相当于另外一种形式的构造器，可以做初始化的操作
3.2、如果多个构造器中都有重复的语句，可以抽取到初始化块中，提高代码的重用性
```

4、细节

```
4.1、static代码块是类加载时执行，并且只会执行一次
4.2、类什么时候被加载？
    1) 创建对象实例时
    2) 创建子类对象实例，父类也会被加载
    3) 使用类的静态成员时（静态属性、静态方法）
4.3、普通代码块是在创建对象时调用的，创建一次，调用一次
4.4、创建对象时，一个类的调用顺序：
    1) 调用静态代码块和静态属性初始化
    2) 调用普通代码块和普通属性的初始化
    3) 调用构造器（构造方法）
4.5、构造器（构造方法）的最前面其实隐含了super() 和 调用普通代码块
4.6、创建子类对象时，一个类的调用顺序：
    1) 父类的静态代码块和静态属性
    2) 子类的静态代码块和静态属性
    3) 父类的普通代码块和普通属性初始化
    4) 父类的构造方法
    5) 子类的普通代码块和普通属性初始化
    6) 子类的构造方法
4.7、静态代码块只能直接调用静态成员（静态属性和静态方法），普通代码块可以调用任意成员
```

四、单例设计模式

1、简介

```
    类的单例设计模式，就是采取一定的方法保证在整个软件系统中，对某个类只能存在一个对象实例，并且该类只提供一个取得其对象实例的方法。
```

2、饿汉式步骤

2.1、构造器私有化
2.2、在类的内部直接创建对象（该对象是static）
2.3、提供一个公共的static方法，返回对象
2.4、在类的外部直接调用该方法

3、懒汉式步骤

3.1、构造器私有化
3.2、定义一个static静态属性对象
3.3、提供一个public的static方法，可以返回一个对象
3.4、在类的外部直接调用该方法

4、饿汉式懒汉式区别

4.1、创建时机不同：饿汉式是在类加载就创建了对象实例，懒汉式是在使用时才创建
4.2、饿汉式问题：在类加载时就创建，可能存在资源浪费问题
4.3、懒汉式问题：线程安全问题

五、final关键字

1、简介

final，可以修饰类、属性、方法和局部变量

2、使用场景

2.1、当不希望类被继承时，可以使用final修饰
2.2、当不希望父类的某个方法被子类覆盖/重写时，可以使用final修饰
2.3、当不希望类的某个属性的值被修改，可以使用final修饰
2.4、当不希望某个局部变量被修改，可以使用final修饰

3、细节

3.1、final修饰的属性又叫常量，一般用 xx_xx_xx来命名
3.2、final修饰的属性在定义时，必须赋初始值，并且以后不能再修改，可以在如下位置
    1）定义时：如 public final double TAX_RATE=0.08;
    2）在构造器中
    3）在代码块中
3.3、如果final修饰的属性是静态的，则初始化的位置只能是
    1）定义时
    2）在静态代码块，不能在构造器中赋值
3.4、final类不能继承，但可以实例化对象
3.5、如果类不是final类，但是含有final方法，则该方法虽然不能重写，但是可以被继承
3.6、当一个类已经是final类时，就没有必要再将方法修饰成final方法
3.7、final不能修饰构造器（构造方法）
3.8、final和static往往搭配使用，效率更高，不会导致类加载

3.9、包装类（Integer, Double Float, Boolean等都是final），String也是final类

六、抽象类

1、简介

当父类的某些方法需要声明，但是又不确定如何实现时，可以将其声明为抽象方法，那么这个类就是抽象类。

2、细节

2.1、抽象类不能被实例化
2.2、抽象类不一定要包含abstract方法。也就是说，抽象类可以没有abstract方法
2.3、一旦类包含了abstract方法，则这个类必须声明为abstract
2.4、abstract只能修饰类和方法，不能修饰属性和其他的
2.5、抽象类可以有任意成员，但抽象类还是类
2.6、抽象方法不能有主题，即不能实现
2.7、如果一个类继承了抽象类，则它必须实现抽象类的抽象方法，除非它自己页声明为abstract类
2.8、抽象方法不能使用private、final、static修饰，因为这些关键词都是和重写相违背的

3、模板设计模式

3.1、将需要实现的核心功能写在一个类中
3.2、定义多个选项功能，并创建不同的类
3.3、再核心功能类中调用选项功能，实现模板shi'ji

七、接口

1、简介

接口就是给出一些没有实现的方法，封装到一起，到某个类要使用的时候，再根据具体情况把这些方法写出来

2、语法

```
interface 接口名 {
    //属性
    //方法
}
class 类名 implements 接口 {
    //自己属性;
    //自己方法;
    必须实现的接口的抽象方法
}
```

3、应用场景

3.1、需求：制造战斗机。专家只需要把飞机的功能/规格定下来即可，然后让别的人具体实现
3.2、需求：开发软件。为了控制和管理软件，项目经理可以定义一些接口，然后程序员具体实现

4、细节

4.1、接口不能被实例化
4.2、接口中所有的方法是public方法，接口中抽象方法，可以不用abstract修饰
4.3、一个普通类实现接口，就必须将该接口的所有方法都实现
4.4、抽象类实现接口，可以不用实现接口的方法
4.5、一个类同时可以实现多个接口
4.6、接口中的属性，只能是final的，而且是 public static final 修饰符
4.7、接口中属性的访问形式：接口名.属性名
4.8、接口不能继承其他的类，但是可以继承多个别的接口
4.9、接口的修饰符只能是public和默认，这点和类的修饰符是一样的

5、实现接口 vs 继承类

5.1、接口和继承解决的问题不同
- 继承：解决代码的复用性和可维护性
- 接口：设计，设计好各种规范（方法），让其他类去实现这些方法

5.2、接口比继承更加灵活
- 接口比继承更加灵活，继承是满足is a的关系，接口只需满足like a的关系

5.3、接口在一定程度上实现代码解耦

6、接口的多态

6.1、接口的多态参数(Interface usb)
6.2、接口页可以使用多态数组
6.3、接口存在多态传递现象（实现了某个接口，当某个接口继承了另一个接口时，此时父类接口中的方法也可以实现）

八、内部类

1、简介

一个类的内部又完整地嵌套了另一个类结构。被嵌套的类称为内部类(inner class)，嵌套其他类的类称为外部类(outer class)。内部类最大的特点就是可以直接访问私有属性。

2、语法

```
class Outer {       // 外部类
    class Inner {   // 内部类
    }
}
class Other {       // 外部其他类
}
```

3、分类

```
定义在外部类局部位置上（方法/代码块内）：
1) 局部内部类（有类名）
2) 匿名内部类（没有类名）

定义在外部类的成员位置上：
1) 成员内部类（没有static修饰）
2) 静态内部类（使用static修饰）
```

4、局部内部类-简介

```
4.1、局部内部类定义在方法/代码块中
4.2、作用域在方法体或者代码块中
4.3、本质仍然是一个类
```

5、局部内部类-语法

```
class Outer {
    public void method() {
        class Inner {
        }
    }
}
```

6、局部内部类-细节

```
5.1、局部内部类 访问 外部类的成员，直接访问
5.2、外部类 访问 局部内部类，创建对象，再访问
5.3、外部其他类 不能访问 局部内部类。因为局部内部类是一个局部变量
5.4、如果外部类和局部类的成员重名时，默认遵循就近原则。如果想访问外部类的成员，可以使用（外部类名.this.成员）去访问
```

7、匿名内部类-简介

```
- 匿名内部类是定义在外部类的局部位置，比如方法中，并且没有类名
6.1、本质是类
6.2、内部类
6.3、该类没有名字
6.4、同时还是一个对象
```

8、匿名内部类-语法

```
new 类或接口(参数列表){
    类体
}
```

9、匿名内部类-细节

9.1、匿名内部类既是一个类的定义，同时本身也是一个对象
9.2、可以直接访问外部类的所有成员，包含私有的
9.3、不能添加访问修饰符，因为它的地位就是一个局部变量
9.4、作用域：仅仅在定义它的方法或代码块中
9.5、匿名内部类 访问 外部类成员，直接访问
9.6、外部其他类 不能访问 匿名内部类，因为匿名内部类地位是一个局部变量
9.7、如果外部类和匿名内部类的成员重名时，匿名内部类访问的话，默认遵循就近原则。如果想访问外部类的成员，则可以使用（外部类名.this.成员）进行访问

10、成员内部类-简介

成员内部类是定义在外部类的成员位置，并且没有static修饰

11、成员内部类-语法

```
class Outer {
    class Inner {
    }
}
```

12、成员内部类-细节

12.1、可以直接访问外部类的所有成员，包含私有的
12.2、可以添加任意访问修饰符（public、protected、默认、private），因为它的低位就是一个成员
12.3、作用域：整个类体
12.4、成员内部类 访问 外部类成员，直接访问
12.5、外部类 访问 成员内部类，创建对象，再访问
12.6、外部其他类 访问 成员内部类
    1) outer08.new Inner08(); 将new Inner08()当作是outer08的成员
    2) 在外部类中编写一个方法，可以返回 Inner08对象
12.7、如果外部类和内部类的成员重名时，内部类访问的话，默认遵循就近原则，如果想访问外部类的成员，则可以使用（外部类名.this.成员）去访问

13、静态内部类-简介

静态内部类是定义在外部类的成员位置，并且有static修饰

14、静态内部类-语法

```
class Outer {
    static class Inner {
    }
}
```

15、静态内部类-细节

15.1、可以直接访问外部类的所有静态成员，包含私有的，但不能直接访问非静态成员

15.2、可以添加任意访问修饰符(public、protected、默认、private)，因为它的地位就是一个成员

15.3、作用域：同其他的成员，为整个类体

15.4、静态内部类 访问 外部类，直接访问所有静态成员

15.5、外部类 访问 静态内部类，创建对象，再访问

15.6、外部其他类 访问 静态内部类

1) `Outer.Inner inner = new Outer.Inner();` 通过类名直接访问

2) `Outer.Inner inner = outer.getInner();` 编写一个方法，返回类型为静态内部类

15.7、如果外部类和静态内部类的成员重名时，静态内部类访问时，默认遵循就近原则，如果想访问外部类的成员，则可以使用（外部类名.成员）去访问