```
/*
 * final：最终
 * final是一个修饰符
 *
 * final可以修饰：
 * （1）类：包括内部类和外部类
 * （2）方法
 * （3）变量：包括属性和局部变量
 *
 * 1、final修饰类
 * 表示这个类不能被继承，是个太监类，没有子类
 *
 * 2、final修饰方法
 * 表示这个方法可以被子类继承，但是不能被子类重写
 *
 * 3、final修饰变量
 * 表示它是一个常量，值不能被修改
 * 常量名建议大写
 *
 *
 * 学习修饰符：
 * （1）它可以修饰什么？
 * （2）它修饰后有什么不同？
 */
```

```
/*
 * native：原生的，内置的
 * 是一个修饰符，只能修饰方法。
 *
 * native修饰的方法，说明不是用Java 语言实现的方法体，而是由C语言等实现的，用C语言实现后，编译为.dll文件，然后由JAVA代码
 *
 * 对于使用这个方法的人来说，没有区别，和普通的Java方法一样。
 * 该怎么调用怎么调用，子类继承后如果需要重写，就可以重写。
 *
 * 调用方法的规则：
 * （1）静态的用类名，非静态的用对象
 * （2）有形参，传对应个数、类型的实参
 * （3）有返回值就接收，没有返回值就不接收
 *
 */
```

```
/*
 * 1、权限修饰符：
 * public
 * protected
 * 缺省
 * private
 *
 * public/缺省：修饰类（包括内部和外部），方法、属性、构造器
 * protected/private：修饰内部类，方法、属性、构造器
 *
 * 2、static：
 * 修饰：内部类，方法、属性、代码块
 *
 * 3、final：
 * 修饰：修饰类（包括内部和外部），方法、属性、局部变量
 *
 * 4、native：
 * 修饰：方法
 *
 * 总结：
 * 构造器只允许权限修饰符
 */
```

```
 * 总结：
 * 构造器只允许权限修饰符,only public, protected & private are permitted
 * 方法：都可以
 * 局部变量：只能final
 */
```

```
/*
 * abstract：抽象的
 * abstract也是修饰符，它可以修饰：类（包括内部类和外部类）、方法
 *
 * 1、为什么要使用抽象类和抽象方法
 *
 *
 * 需求：
 * 1、设计一个图形类Graphic
 * 2、这个图形类有很多子类：Rectangle矩形、Circle圆形、Triangle三角形等
 * 这些图形，有图形名称，图形的面积，图形的周长
 * 既然所有的子类有共同的特征，那么这些共同的特征应该设计到父类图形类Graphic中，因为父类就是代表所有子类共同的部分。
 *
 * 类：一类具有相同特性的事物的抽象描述。
 */
```

```
* 1、为什么要使用抽象类和抽象方法
*  当我们在设计父类时，发现某些方法是无法给出具体的实现，具体的实现应该在子类中实现。那么这样的方法，在父类中就可以设计为“抽象方法”。
*  包含抽象方法的类，必须是一个“抽象类”
*
*
* 2、抽象类、抽象方法的语法格式
* 【权限修饰符】 abstract class 类名 [extends 父类]{
*      【权限修饰符】 abstract  返回值类型 方法名(【形参列表】);
* }
*
*
* 需求：
* 1、设计一个图形类Graphic
* 2、这个图形类有很多子类：Rectangle矩形、Circle圆形、Triangle三角形等
* 这些图形，有图形名称，图形的面积，图形的周长
* 既然所有的子类有共同的特征，那么这些共同的特征应该设计到父类图形类Graphic中，因为父类就是代表所有子类共同的部分。
*
* 类：一类具有相同特性的事物的抽象描述。
*/
```

3、抽象类的特点

（1）抽象类不能直接new对象

（2）抽象类就是用来被继承的，那么子类继承抽象类后，必须实现抽象类的所有抽象方法，否则子类也得是抽象类。

（3）抽象类的变量可以与子类的对象构成多态引用，执行子类重写的方法

（4）抽象类可以和普通类一样，拥有：属性、构造器、代码块、非抽象的方法等成员，这些给子类可以用。

（5）一个类如果有抽象方法，这个类必须是抽象类，但是一个抽象类也可以没有抽象方法

```
/*
 * 1、权限修饰符
 * public,protected,缺省,private
 * （1）private和abstract不能同时修饰方法，因为private的方法在子类中不可见，无法重写
 */
```

```
* 1、权限修饰符
* public,protected,缺省,private
* （1）private和abstract不能同时修饰方法，因为private的方法在子类中不可见，无法重写
* （2）private和abstract可以同时修饰成员内部类
*
* 2、static
* static修饰属性、方法、代码块、内部类
* abstract修饰类、方法
*
* 修饰内部类：可以
* 修饰方法：static和abstract不能同时修饰方法，因为static的方法不能被重写，并且static的方法直接用“类名.”调用，没有方法体执行
*
* 3、final
```

```
* 3、final
* 修饰方法：final和abstract不能同时修饰方法，final是表示不能被重写，abstract必须被重写
* 修饰类：final和abstract不能同时修饰类，final是表示不能被继承，abstract就是用来继承的。
*
```

```
* 4、native
* 修饰方法：native和abstract不能同时修饰方法，native是表示方法体由C语言实现，abstract表示无方法体，由子类实现
*/
```

3、不能和 abstract 一起使用的关键字

abstract 只能修饰类与方法

* （1）abstract 和 final
* （2）abstract 和 static
* （3）abstract 和 private

abstract 不能修饰属性、构造器、局部变量、代码块

```
/*
 * 模板设计模式（了解）：
 *
 * 模板：定好了框架，格式，结构。具体的内容需要使用者来填写。
 *    简历模板，论文模板，请假条的模板...
 *
 * 在开发中，遇到这样的情况：当解决某个问题时，总体的代码结构是确定的，步骤也是确定的，
 * 只是其中的一小步的具体步骤不确定，那么我们可以把这个不确定的步骤设计为抽象方法，让使用者实现它。
 *
 * 例如：编写一个类，它具有一个功能，可以计算“任意”一段代码的运行时间。
 * 步骤：
 * （1）获取开始时间start
 * （2）执行xxx代码
 * （3）获取结束时间end
 * （4）计算时间差(end-start)
 */
```

提示：System.currentTimeMillis()：得到当前的系统时间，距离1970-1-1,凌晨的毫秒数

```
/*
 * 接口：
 *  代表一种标准。
 *
 * 例如：
 *  1、JDBC：Java Database Connectivity，用Java连接各种数据库。
 * 数据库有很多种：sql server, access,mysql,oracle,redis,mangodb....
 * 这些数据是由不同的公司（产商）来生产，开发的。
 * Java程序：去连接不同的数据库时，使用相同的代码。
 * Java连接和操作mysql数据库的代码，同样可以用来连接和操作oracle数据库，sql server。
 * 希望这个Java代码具有通用性，那么Java就要指定一个标准。
 *
 * 这些标准通常就是一些接口。这些接口公布出来，由各个数据库的厂商来实现它。
 * Connection,Statement,...接口
 * mysql：MysqlConnection,MysqlStatement...
 * oracle:OracleConnection,OracleStatement...
 * 这些实现类，通过jar更换就可以实现，而编程的代码中，面向接口编程。
 *
 * 2、现在我想设计一个数组的工具类MyArrays，这个工具类中有一个方法，可以为任意的对象数组进行排序，从小到大排序。
 * public void sort(Object[] arr)
 *
 */
```

```
/*
 * 对象如何比较大小？
 * 不能直接使用>,<
 *
 * 而且不同的对象比较大小的规则不一样：
 * 例如：两个学生如何比较大小，可能按身高，可能按年龄，可能按照成绩比较
 *      两个圆对象如何比较大小，可能按照半径比较，可能按照面积比较
 *      两个员工对象如何比较大小，可能按照编号，可能按照薪资
 *
 * 我就可以为两个对象比较大小指定一个标准，接口
 * Java中确实提供了这样的接口：java.lang.Comparable
 *                          java.util.Comparator
 *
 */
```

```
/*
 * Ctrl+1：快速修复
 *
 * 接口和类是同一个级别的概念。
 * Java的数据类型：基本数据类型和引用数据类型
 * 引用数据类型：类、接口、数组、枚举...
 *
 * 1、如何声明一个接口
 * 语法格式：
 * 【修饰符】 interface 接口名{
 * }
 *
```

```
* 2、接口的成员有哪些？
* JDK1.8之前：
* （1）全局的静态的常量：public static final
* （2）公共的抽象的方法：public abstract
*
* JDK1.8之后：
* 。。。
*
* 说明：接口是没有构造器，代码块，除了全局的静态的常量以外的普通的属性等。
*
*/
```

```
* 3、如何实现接口？
* 用来被实现的。
* 【修饰符】class 实现类 implements 接口们{
* }
*/
```

```
* 4、接口的特点：
* （1）实现类在实现接口时，必须实现接口的所有的抽象方法，否则这个实现类就必须是个抽象类。
* （2）一个类可以同时实现多个接口
* （3）接口不能直接创建对象
* （4）接口可以与实现类的对象构成多态引用
* （5）一个类可以同时继承父类，又实现接口，但是要求，继承在前实现在后。
*
*
* 理解：
* 父类：亲生父亲，只有一个
* 接口：干爹，可以同时有很多个
*/
```

```
/*
* 类与类之间：继承extends
* 类与接口之间：实现 implements
* 接口与接口之间：继承extends
*/
```

```
* 4、接口的特点：
* （1）实现类在实现接口时，必须实现接口的所有的抽象方法，否则这个实现类就必须是个抽象类。
* （2）一个类可以同时实现多个接口
* （3）接口不能直接创建对象
* （4）接口可以与实现类的对象构成多态引用
* （5）一个类可以同时继承父类，又实现接口，但是要求，继承在前实现在后。
* （6）接口与接口之间支持多继承
*
*
* 理解：
* 父类：亲生父亲，只有一个
* 接口：干爹，可以同时有很多个
*/
```