抽象类和接口都是从多个子类中抽象出来的共同特征

抽象类主要作为多个类的模板，接口则定义了多类应该遵守的规范

toString()**方法**

A.当直接输出某个类的对象时，实际上是调用该对象的toString()方法 Person p = new Person(); System.out.println(“p”);

B.toString()是Object类中的一个实例方法，用于“自我描述”默认 返回该对象的“类名+@+hashcode”，如果需要自定义其返回 值，可以对它进行重写，格式如下：

public String toString(){return 自定义字符串}

C.可以用对象跟字符串进行连接运算，这时候系统也会调用该对象的 toString()方法的返回值与字符串进行连接

==**与**equals()

A.==用法

a.如果两个变量是基本类型，且都是数值类型，（不一定要求数据类 型严格相同），则只要两个变量的值相等，则返回true

b.对于两个引用类型变量，只有当它们指向同一个对象时才会返回 true，而且不能比较两个类型上没有父子关系的对象，会编译错误

B.equals()

a.是Object类的一个方法，所有对象都能调用它来判断是否跟其他 引用变量相等，这是原始用法，跟==没有区别，同样要求两个变 量指向同一个对象才返回true

b.所以可以通过重写equals()来自定义相等的标准

final

A.final修饰的成员变量必须显式指定初始值

类变量：需在声明该类变量时或在静态初始化块中指定初始值

实例变量；需在声明该实例变量时或在非静态初始化块或构造器中 指定初始值

B.类变量不能在普通初始化块中指定初始值，因为它在类初始化阶段 已经被初始化了，普通初始化块不能对它重新赋值

C.与普通成员变量不同，系统不会给final修饰的变量初始化

D.所以如果要在构造器或初始化块中对成员变量初始化，则不要在初 始化前使用成员变量 例如：

final int a; /\*以下是普通初始化块\*/ { System.out.println(a); }

因为系统不会给a初始化，但只要把final去掉就不会出现编译错误

E. final局部变量只能被赋值一次，所以不能在方法里对final修饰的 形参赋值，因为在调用该方法时，实参给形参传值，这已经是初始 化了

F.用final修饰基本类型变量时，不能对基本类型变量重新赋值，所以 基本类型变量不能改变。但对于引用类型变量而言，它保存的只是 个引用，只要保证这个引用类型变量所引用的地址不变即可，但这 个对象的内容可以改变，这是与基本类型变量的区别。例如一个 final型的数组对象，只要引用一直指向这个数组即可，数组元素的 值可以改变。同样，一个final型的类实例，也可以改变实例的内容

G.final与宏定义

a.当一个final变量在定义时就为其指定初始值，且该初始值可以在 编译时就确定下来(这个初始值可以是直接量，基本算术表达式或 字符串连接运算。因为如果是普通变量或方法，则无法满足在编 译时就确定下来)，那这个final变量本质上就是一个宏变量，编 译器会把程序中所有用到该变量的地方直接替换成该变量的值

b.Java会把使用过的字符串直接量缓存进常量池。例如 String a = “abc”; String b = “abc”系统会让b直接指向常 量池中的字符串，所以a==b会返回true，指向同一个对象

String s1 = “crazy\_java”;

String s2 = “crazy\_”+“java”;

System.out.print,ln(s1 == s2);

String str1 = “crazy\_”;

String str2 = “java”;

String str3 = str1 + str2;

System.out.println(s1 == s3);

代码解析：

s2是用两个字符串直接量连接的，所以编译器可以编译阶段就确 定s2的值为“crazy\_java”，这样一来s2也是指向常量池中缓 存的“crazy\_java”，跟s1相同，所以s1 == s2是true

而s3是str1跟str2连接而成，因为str1跟str2是普通变量，所 以无法在编译时确定s3的值，就无法让s3与s1跟s2一样指向 常量池中的“crazy\_java”，故s1 == s3是false

但如果把str1跟str2定义为final变量，根据G.a的说法，两个 变量本质上就是宏变量，这样s3在编译时就可以确定了，所以 s1 == s3就是true

H.final方法

a.子类不能重写父类中的final方法

b.在继承的时候有子类不能继承父类的private方法，在子类中可 以定义一个跟private方法同名同参数的方法。所以，即使这个 private方法用final修饰时，在子类中也可以定义一个跟 private方法同名同参数的方法。这要跟没有private但有final 的方法区分开来，那种是子类不能定义一个跟final方法同名同 参数的方法，也就是不能重写(参照H.a)

c.final方法仅仅不能被重写，但可以被重载，例如

class A{

public final void A(int a){}

public final void A(int a, int b){}

} //这个程序没有问题

I. final修饰的类不能被继承

**抽象类**

A. 抽象方法和抽象类

a.抽象方法不能有方法体，含有抽象方法的类(包括直接定义了一 个抽象方法、或继承了一个抽象父类，但没有完全实现父类包 含的抽象方法、或实现了一个接口，但没有完全实现接口包含 的抽象方法三种情况)只能被定义成抽象类

b.抽象类不能被实例化，不能用new来调用抽象类中的构造器创 建实例，即使抽象类中不含抽象方法

c.抽象类中可以包含成员变量、方法(普通抽象皆可)、构造器、初 始化块、内部类(接口、枚举)5种成分。抽象类的构造器主要是 用于被子类调用，不能用于创建实例

d.抽象类可以继承具体类，可以有静态main方法

B.抽象方法与空方法的区别是：抽象方法无花括号，无方法体。而 空方法是有花括号，但方法体为空，即花括号里面没内容

C.抽象类只能被继承，不能被实例化。抽象方法必须由子类提供实 现(即重写)，不能被调用。而final类不能被继承，final方法不能 被重写，所以final跟abstract不能同时使用

D.abstract不能修饰变量、构造器

E.static和abstract不能同时修饰某个方法，因为加了abstract， 该方法就不能被调用，当用类名来调用abstract修饰的static方 法时，会出现错误(因为调用了一个没有方法体的方法)

F.抽象方法必须被子类重写才有意义，否则这个方法永远不会有方 法体，故抽象方法不能定义为private权限(private方法不能被继 承)

G.抽象类可以继承接口、实体类

H.抽象类的作用

a.抽象类具有更高层次的抽象，子类在抽象类的基础上进行扩 展、改造。父类提供多个子类的通用方法，并把一个或多个方 法留给子类实现，这是种模板模式

b.抽象父类可以只定义需要使用的某些方法，把不能实现的部分 抽象成抽象方法，留给子类去实现

c.父类中可能包含需要调用其他系列方法的方法，这些被调方法既 可以由父类实现，也可以由子类实现(例如要父类方法中要调用 父类本身的一个抽象方法，这个抽象方法可以由子类去实现)。