面向对象下

一、包装类

1.基本数据类型的包装类

Java为8种基本数据类型提供了对应的包装类

基本数据类型	包装类
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

- Java为包装类提供了**自动装箱**和**自动拆箱**的功能：可以把一个基本数据类型变量直接赋给对应的包装类变量或Object类变量，也可以把一个包装类变量直接赋给对应的基本数据类型变量
- 可以利用包装类的parseXxx(String s)静态方法（除了Character类没有该方法）或valueOf(String s)静态方法将字符串转换为基本类型值
- 可以利用String类的valueOf()静态方法将基本数据类型变量转换为字符串
- 包装类变量是引用变量，但是可以直接与基本类型值比较大小。此外，包装类变量之间比较大小只有引用指向同一个对象时才为true
- 除了常量池会缓存字符串直接量，系统也存在一个cache数组缓存-128~127之间的所有整数，所以自动装箱后数值相同的Integer变量指向同一个对象

二、Object类处理对象的方法

1.toString()方法

Object类提供了toString()实例方法，用于表示对象的基本信息，其总是返回该对象实现类的“类名+@+hashCode”值，因此实际使用中需要重写该方法

2.==运算符和equals()方法

- ==运算符：用于比较两个基本数据类型变量时，只要数值相等，就返回true，用于比较引用类型变量时，它们指向同一个对象时才返回true，不具有继承关系的变量不能进行比较
- equals()方法：Object类中提供的实例方法，与==运算符没有区别。String类改写了该方法，用于比较字符串对象包含的字符序列是否相同，实际使用中需要重写该方法

注意常量池的问题：ConstantPoolTest.java

三、static修饰符

1.理解类成员

在Java类中允许包含5中类成员：构造器、初始化块、成员变量、方法、内部类，使用static修饰的成员是类成员，系统底层是通过类来调用类成员的

2.单例类

只允许同时存在一个实例的类被称为单例类
单例类构造方法：该类所有构造器使用private修饰，提供一个public static方法用于调用构造器创建对象，提供一个private static成员变量用于缓存该实例
单例类的构造：SingletonTest.java

四、final修饰符

final关键字可以修饰类、变量、方法，表示这些类、变量、方法不可改变

1.final成员变量

final修饰的成员变量一旦被赋值就不能再改变，因此final修饰的成员变量在定义时就必须显示定义初始值。

- 类成员变量：必须在静态初始化块或声明该变量时指定初始值
- 实例成员变量：必须在非静态初始化块、构造器或声明该变量时指定初始值

2.final局部变量

final修饰的局部变量一旦被赋值就不能再改变，而系统不会对局部变量自动执行初始化操作，因此final局部变量要么在声明时指定初始值，要么在后面代码中指定初始值

3.final修饰基本类型变量和引用类型变量

- final修饰基本类型变量：该变量一旦指定初始值就不能再改变
- final修饰引用类型变量：final只能保证该变量指向同一个对象，但不能保证该对象不改变

4.执行"宏替换"的final变量

对于一个final变量来说，不管是类变量、实例变量或是局部变量，只要在声明该变量时指定了初始值，且该初始值在编译时就能确定下来，那么这个变量相当于一个常量，编译器会将所有用到该变量的地方替换成对应常量

5.final方法

final修饰的方法不能被重写，如果希望父类方法不被重写，可以使用final修饰

- 如果final修饰的方法是private方法，子类依然可以定义方法名、形参列表都相同的新方法
- final修饰的方法不能被重写，依然可以被重载

6.final类

final修饰的类不能被继承，如果希望一个类不被继承，可以使用final修饰这个类

7.不可变类

不可变类是指创建该类的实例后，该实例的实例变量是不可改变的。8个包装类和String类都是不可变类。

- 不可变类就是使用final修饰该类的实例变量，但是当final引用变量时，引用的对象是可变的，就会出现问题，此时需要注意构造器的构造方式

8.缓存实例的不可变类

如果不可变类的实例被经常使用，那么可以考虑缓存该不可变类的实例，减少系统开销
缓存实例的不可变类：CacheImmutableTest.java

五、abstract修饰符

1.抽象方法和抽象类

使用抽象类和抽象方法后，编译时类型变量不需要强制类型转换就能调用运行时类型的方法

- 使用abstract修饰的类和方法称为抽象类和抽象方法，抽象类不能创建实例，抽象方法不能有方法体
- 抽象类中不一定有抽象方法，但是含有抽象方法（包括自定义抽象方法、从父类继承抽象方法却没有完全实现、实现一个接口却没有完全实现接口中的抽象方法）的类一定是抽象类
- 抽象类可以包含构造器、成员变量、方法、内部类、初始化块，但其构造器不能被new关键字调用以创建实例，主要用于被子类方法调用
- 抽象方法区别于空方法体方法，用分号代替花括号

2.抽象类的作用

利用抽象类可以使用模板模式

六、接口

1.理解接口

接口是从多个相似类中抽象出来的规范，接口不提供任何实现。接口体现的是规范和实现相分离的哲学

2.定义接口

定义接口的格式：

```
[修饰符] interface 接口名 extends 父接口1,父接口2...{
//零到多个常量定义...
//零到多个抽象方法定义...
//零到多个内部类,接口,枚举定义
//零到多个私有方法,默认方法或类方法定义...
}
```

- 修饰符：可以是public或省略
- 接口名：由多个有意义单词连缀而成，每个单词首字母大写，其余字母小写，单词之间没有任何分隔符
- 接口是一种规范，因此接口中不能定义初始化块和构造器，只能定义成员变量、方法、内部类

- 接口中的成员变量只能是静态常量，必须使用public static final修饰，且必须在声明时指定初始值
- 接口中的方法可以是类方法、抽象实例方法、默认方法（非抽象实例方法）、私有方法。类方法一定使用public static方法，抽象实例方法一定是public abstract方法，非抽象实例方法一定是public default方法，私有方法一定是private方法。类方法可以直接通过接口来调用，私有方法主要做默认方法和类方法的工具方法，可以是类方法或实例方法
- 接口可以看作一种特殊的类，一个Java源文件种最多只能有一个public接口，且该源文件必须与该public接口同名

3.继承接口

接口支持多继承，一个接口可以有多个直接父接口，子接口能够继承父接口的所有静态常量和抽象方法

4.使用接口

接口不能创建实例，其主要作用是被实现类实现。接口可以定义引用变量，但只能引用到其实现类的实例上
一个类可以实现一个或多个接口，继承使用extends关键字，而实现则使用implements关键字
类实现接口的格式：

```
[修饰符] class 类名 extends 父类 implements 接口1，接口2...{
//类体
}
```

- 类实现接口，可以获得接口的所有静态常量和方法（默认方法和抽象方法），这个类必须完全实现接口中的抽象类，否则其必须定义为抽象类
- Java利用类实现多个接口模拟C++的多继承

5.接口和抽象类

接口和抽象类都具有以下特征：

- 接口和抽象类都不能被实例化，都用于被其他类实现或继承
- 接口和抽象类都可以包含有抽象方法，实现接口或继承抽象类的普通子类都必须实现这些抽象方法
- 接口是一种规范，类似于总纲，而抽象类也是一种规范，类似于一种中间体

6.面向接口编程

略

七、内部类

1.理解内部类

定义在外部类内部的类称为内部类

- 内部类定义格式于外部类基本相同，但内部类修饰符可以是：public、private、protected、final、abstract、static或者省略
- 内部类可以看作外部类的一个成员，内部类可以访问外部类的私有成员，但是外部类成员不能访问内部类的成员
- 内部类可以是成员内部类（主要是成员内部类）、局部内部类、匿名内部类（不是类成员）
- 成员内部类的.class文件名为： `OuterClass$InnerClass.class`

2.非静态内部类

不使用static修饰的成员内部类称为非静态内部类,非静态内部类不能拥有静态成员
如果外部类成员变量、内部类成员变量与内部类的局部变量同名,则可以通过外部类类名.this,this作为限定来区分

- 非静态内部类能够访问外部类的所有成员,但是非静态内部类成员只在非静态内部类里是可知的,不能被外部类成员直接使用.外部类成员只能显式创建内部类实例来调用其成员
- 外部类的静态成员不能直接使用非静态内部类,包括使用非静态内部类定义变量、创建实例等

3.静态内部类

使用static修饰的内部类称为静态内部类,其属于外部类本身,静态内部类可以拥有静态成员和非静态成员

- 静态内部类只能访问外部类的静态成员,而外部类的成员不能直接访问内部类，外部类成员只能显式创建内部类实例来调用其实例成员或使用内部类类名调用其类成员
- 可以在接口中定义内部类，接口中的内部类必须使用public static修饰

4.成员内部类总结

成员内部类分静态内部类和非静态内部类，他们都是外部类的成员。
外部类的成员不能直接访问内部类的成员，内部类的成员可以直接访问外部类成员
非静态内部类不能有静态成员，可以访问所有外部类成员，只有非静态外部类成员才能使用非静态内部类，通过创建非静态内部类的实例来调用
静态内部类可以有非静态成员，只能访问静态外部类成员，所有外部类成员都可以使用静态内部类，通过创建静态内部类实例或通过静态内部类类名调用

5.使用内部类

（1）在外部类内部使用内部类

在外部类内部使用内部类时，与使用普通类没有区别

（2）在外部类外部使用非静态内部类

- 定义内部类变量：OuterClass.InnerClass varName
- 创建非静态内部类实例：OuterInstance.new InnerConstructor()
- 非静态内部类可以创建子类，该子类可以是内部类或外部类，但是该子类的构造器只能通过其父类的外部类对象来调用父类构造器，格式： `外部类类名.super()`
- 内部类实例和其子类实例都需要保存一个指向外部类对象的引用

（3）在外部类外部使用静态内部类

- 定义内部类变量：OuterClass.InnerClass varName
- 创建静态内部类实例：new OuterClass.InnerConstructor()
- 静态内部类的子类可以是外部类或内部类，子类构造器调用父类构造器时不需要通过外部类对象来调用，方便很多

6.局部内部类

在方法中定义的内部类被称为局部内部类，局部内部类只在方法中有效
局部内部类的.class文件格式为： `OuterClass$NInnerClass.class` ，数字N用于区分不同的局部内部类

7.匿名内部类

匿名内部类用于创建一次使用的类。匿名内部类在定义时就创建一个实例，然后匿名内部类消失，不能重复使用
定义匿名内部类格式：

```
new 接口()|父类构造器(参数列表){
//类体
```

```
}
```

- 匿名内部类必须继承一个父类或者实现一个接口
- 匿名内部类不能是抽象类，因为其要创建实例，所以必须完全实现接口或父类中的抽象方法
- 匿名内部类不能定义构造器，因为匿名内部类没有类名。但匿名内部类可以定义初始化块完成初始化过程
- 创建匿名内部类时，可以不传入参数，代表调用父类无参数构造器；也可以传入参数，代表调用父类带参数构造器。匿名内部类也可以重写父类方法。
- 如果局部变量被匿名内部类访问，那么该局部变量相当于自动使用了final修饰

八、Lambda表达式

1.理解Lambda表达式

Lambda表达式主要用于代替匿名内部类。
Lambda表达式的组成部分：

- 形参列表：形参列表允许省略形参类型，只有一个参数时可以省略圆括号
- 箭头：英文中划线和大于符号组成
- 代码块：如果代码块只有一条语句，那么花括号可以省略。

Lambda表达式需要返回值，且只能有一条return语句。当花括号中只有一条语句时，可以省略return语句，Lambda表达式返回该条语句的值

2.Lambda表达式与函数式接口

Lambda表达式的类型，被称为目标类型，Lambda表达式的目标类型必须式“函数式接口”。函数式接口指只包含一个抽象方法的接口，可以包含多个默认方法，类方法。

Lambda表达式的使用限制：

- Lambda表达式的目标类型必须是明确的函数式接口
- Lambda表达式只能为函数式接口创建对象。

为保证Lambda表达式的目标类型是明确的函数式接口，有三种实现方式：

- 将Lambda表达式赋值给函数式接口类型的变量
- 将Lambda表达式作为函数式接口类型的参数传给某个方法
- 使用函数式接口对Lambda表达式进行强制类型转换

Lambda表达式的目标类型是可变的，唯一的要求是Lambda表达式实现的方法与抽象方法的形参相同
Lambda表达式的本质是使用简洁的语法来创建函数式接口的实例，避免了匿名内部类的繁琐

3.方法引用与构造器引用

如果Lambda表达式花括号中只有一条语句，那么可以省略花括号，且可以使用方法引用或构造器引用

种类	示例	说明	对应的Lambda表达式
引用类方法	类名:: 类方法	函数式接口中被实现方法的全部参数传给该类方法作为参数	(a,b,...)->类名::类方法 (a,b,...)
引用特定对象的实例方法	特定对象:: 实例方法	函数式接口中被实现方法的全部参数传给该方法作为参数	(a,b,...)->特定对象:: 实例方法(a,b,...)

种类	示例	说明	对应的Lambda表达式
引用某类对象的实例方法	类名:: 实例方法	函数式接口中被实现方法的第一个参数作为调用者， 后面的参数全部传给该方法作为参数	(a,b,...)->a.实例方法 (b,...)
引用构造器	类名::new	函数式接口中被实现方法的全部参数传给该构造器作为参数	(a,b,...)->new 类名 (a,b,...)

4.Lambda表达式与匿名内部类的联系

相同点：

- Lambda表达式和匿名内部类一样，都可以直接访问“effectively final”的局部变量，以及外部类的成员变量（包括实例变量和类变量）
- Lambda表达式创建的对象与匿名内部类生成的对象一样，都可以直接调用从接口中继承的默认方法

不同点：

- 匿名内部类可以为任意接口创建实例——不管接口包含多少个抽象方法，只要全部实现即可，而Lambda表达式只能为函数式接口创建实例
- 匿名内部类可以为抽象类甚至普通类创建实例，而Lambda表达式只能为函数式接口创建实例
- 匿名内部类实现的抽象方法的方法体允许调用接口中定义的默认方法，而Lambda表达式的代码块不允许调用接口中定义的默认方法

九、枚举类

1.理解枚举类

拥有的对象是有限且固定的类是枚举类

2.枚举类入门

枚举类使用enum关键字，一个java源文件中最多只能定义一个public枚举类，且该源文件必须与该枚举类同名

- 枚举类可以实现一个或多个接口，枚举类默认继承java.lang.Enum类，而不是Object类，因此枚举类不能显式继承其他类。java.lang.Enum类实现了java.lang.Serializable和java.lang.Comparable两个接口
- 枚举类要么是抽象类，要么是final类，默认使用final修饰
- 枚举类的构造器必须使用private修饰
- 枚举类的所有实例必须在枚举类的第一行显式列出，默认使用public static final修饰，无需显示添加

switch语句的控制表达式可以是任何枚举类型，且后面的case表达式中的值可以直接使用枚举值的名字，无需添加枚举类作为限定

3.枚举类的成员变量、方法和构造器

枚举类是一种特殊的类，可以定义构造器、成员变量、方法
如果枚举类显式定义了带参数的构造器，列出枚举值时必须对应地传入参数

4.实现接口的枚举类

枚举类也可以实现一个或多个接口。枚举类实现一个或多个接口时，也需要实现该接口所包含的方法

- 如果由枚举类来实现接口里的方法，则每个枚举值在调用该方法时都有相同的行为方式，

- 如果希望不同的枚举值有不同的行为方式，那么可以由每个枚举值分别实现该方法，此时每个枚举值相当于一个匿名内部类

包含抽象方法的枚举类

- 枚举类里定义抽象方法时，不能使用abstract关键字将枚举类修饰成抽象类，系统会自动添加abstract关键字。
- 因为枚举类需要显式创建实例，而不是作为父类，所以定义每个枚举值时必须为抽象方法提供实现，否则编译时会出错

十、对象与垃圾回收

略