**匿名对象使用场景：**

　　　　1：当对方法只进行一次调用的时候，可以使用匿名对象。

　　　　2：当对象对成员进行多次调用时，不能使用匿名对象。必须给对象起名字。

　　类中怎么没有定义主函数呢？

**注意：主函数的存在，仅为该类是否需要独立运行，如果不需要，主函数是不用定义的。**

**主函数的解释**：保证所在类的独立运行，是程序的入口，被jvm调用。

**创建一个对象都在内存中做了什么事情？**

　　　　1：先将硬盘上指定位置的Person.class文件加载进内存。

　　　　2：执行main方法时，在栈内存中开辟了main方法的空间(压栈—进栈)，然后在main方法的栈区分配了一个变量p。

　　　　3：在堆内存中开辟一个实体空间，分配了一个内存首地址值。new

　　　　4：在该实体空间中进行属性的空间分配，并进行了默认初始化。

　　　　5：对空间中的属性进行显示初始化。

　　　　6：进行实体的构造代码块初始化。

　　　　7：调用该实体对应的构造函数，进行构造函数初始化。（）

　　　　8：将首地址赋值给p ，p变量就引用了该实体。(指向了该对象)

**静态的注意：静态的生命周期很长。**

**静态代码块：**就是一个有静态关键字标示的一个代码块区域。定义在类中。

**作用：可以完成类的初始化。静态代码块随着类的加载而执行，而且只执行一次（new 多个对象就只执行一次）。如果和主函数在同一类中，优先于主函数执行。**

**静态代码块、构造代码块、构造函数同时存在时的执行顺序：静态代码块 ———> 构造代码块 ———> 构造函数；**

设计模式：解决问题最行之有效的思想。是一套被反复使用、多数人知晓的、经过分类编目的、代码设计经验的总结。使用设计模式是为了可重用代码、让代码更容易被他人理解、保证代码可靠性。

**问题：super()和this()是否可以同时出现的构造函数中。**

　　两个语句只能有一个定义在第一行，所以只能出现其中一个。

　　final特点：

　　　　1：这个关键字是一个修饰符，可以修饰类，方法，变量。

　　　　2：被final修饰的类是一个最终类，不可以被继承。

　　　　3：被final修饰的方法是一个最终方法，不可以被覆盖。

　　　　4：被final修饰的变量是一个常量，只能赋值一次。

　　抽象类的细节：

**1：抽象类中是否有构造函数？有，用于给子类对象进行初始化。**

**2：抽象类中是否可以定义非抽象方法？**

　　　　　　可以。其实，抽象类和一般类没有太大的区别，都是在描述事物，只不过抽象类在描述事物时，有些功能不具体。所以抽象类和一般类在定义上，都是需要定义属性和行为的。只不过，比一般类多了一个抽象函数。而且比一般类少了一个创建对象的部分。

　　3：抽象关键字abstract和哪些不可以共存？final , private , static

**4：抽象类中可不可以不定义抽象方法？可以。抽象方法目的仅仅为了不让该类创建对象。**

**内部类：**如果A类需要直接访问B类中的成员，而B类又需要建立A类的对象。这时,为了方便设计和访问，直接将A类定义在B类中。就可以了。A类就称为内部类。内部类可以直接访问外部类中的成员。而外部类想要访问内部类，必须要建立内部类的对象。

**匿名内部类**：没有名字的内部类。就是内部类的简化形式。一般只用一次就可以用这种形式。匿名内部类其实就是一个匿名子类对象。**想要定义匿名内部类：需要前提，内部类必须继承一个类或者实现接口。**

**匿名内部类的格式**：new 父类名&接口名(){ 定义子类成员或者覆盖父类方法 }.方法。

**throw 和throws关键字的区别：**

　　　　throw用于抛出异常对象，后面跟的是异常对象；throw用在函数内。

　　　　throws用于抛出异常类，后面跟的异常类名，可以跟多个，用逗号隔开。throws用在函数上。

**异常分两种：**

　　　　1：编译时被检查的异常，只要是Exception及其子类都是编译时被检测的异常。

　　　　2：运行时异常，其中Exception有一个特殊的子类RuntimeException，以及RuntimeException的子类是运行异常，也就说这个异常是编译时不被检查的异常。

**编译时被检查的异常和运行时异常的区别**：

　　　　编译被检查的异常在函数内被抛出，函数必须要声明，否编译失败。

　　　　声明的原因：是需要调用者对该异常进行处理。

　　　　运行时异常如果在函数内被抛出，在函数上不需要声明。

　　　　不声明的原因：不需要调用者处理，运行时异常发生，已经无法再让程序继续运行，所以，不让调用处理的，直接让程序停止，由调用者对代码进行修正。

**System.exit(0)**; //退出jvm，只有这种情况finally不执行。

**当异常出现后，在子父类进行覆盖时，有了一些新的特点：**

　　　　1：当子类覆盖父类的方法时，如果父类的方法抛出了异常，那么子类的方法要么不抛出异常要么抛出父类异常或者该异常的子类，不能抛出其他异常。

　　　　2：如果父类抛出了多个异常，那么子类在覆盖时只能抛出父类的异常的子集。

**注意：**

**如果父类或者接口中的方法没有抛出过异常，那么子类是不可以抛出异常的**，**如果子类的覆盖的方法中出现了异常，只能try不能throws。**

**如果这个异常子类无法处理，已经影响了子类方法的具体运算，这时可以在子类方法中，通过throw抛出RuntimeException异常或者其子类，这样，子类的方法上是不需要throws声明的。**

　　import packa.\*;//这个仅仅是导入了packa当前目录下的所有的类。不包含子包。

　　import packa.abc.\*;//导入了packa包中的子包abc下的当前的所有类。

**等待唤醒机制**：涉及的方法：

**wait**:将同步中的线程处于冻结状态。释放了执行权，释放了资格。同时将线程对象存储到线程池中。

**notify**：唤醒线程池中某一个等待线程。

**notifyAll**:唤醒的是线程池中的所有线程。

　　注意：

　　　　1：这些方法都需要定义在同步中。

　　　　2：因为这些方法必须要标示所属的锁。

　　　　　　你要知道 A锁上的线程被wait了,那这个线程就相当于处于A锁的线程池中，只能A锁的notify唤醒。

　　　　3：这三个方法都定义在Object类中。为什么操作线程的方法定义在Object类中？

　　　　　　因为这三个方法都需要定义同步内，并标示所属的同步锁，既然被锁调用，而锁又可以是任意对象，

　　　　　　那么能被任意对象调用的方法一定定义在Object类中。

**wait和sleep区别**： 分析这两个方法：从执行权和锁上来分析：

　　　　wait：可以指定时间也可以不指定时间。不指定时间，只能由对应的notify或者notifyAll来唤醒。

　　　　sleep：必须指定时间，时间到自动从冻结状态转成运行状态(临时阻塞状态)。

**wait：线程会释放执行权，而且线程会释放锁。**

**Sleep：线程会释放执行权，但不是不释放锁。**

为了突出区别，省略了若干细节。区别有三点：

1. 1. lock不再用synchronize把同步代码包装起来；
2. 2. 阻塞需要另外一个对象condition；
3. 3. 同步和唤醒的对象是condition而不是lock，对应的方法是await和signal，而不是wait和notify。

为什么需要使用condition呢？简单一句话，lock更灵活。以前的方式只能有一个等待队列，在实际应用时可能需要多个，比如读和写。为了这个灵活性，**lock将同步互斥控制和等待队列分离开来**，互斥保证在某个时刻只有一个线程访问临界区（lock自己完成），等待队列负责保存被阻塞的线程（condition完成）。