**1:**

**double** a = 90.0;

**int** b = 90;

String aa = String.*valueOf*(a);//aa=90.0

String bb = String.*valueOf*(b);//bb=90

2:

重写父类方法的参数要与父类相同，不仅仅是类型，数量还包括显示形式。

3：

Int count = 0；

for （int I = 0;i<10;i++）{

count = count ++;

}

System.out.printLn(count);//最终结果是0；

count = count ++;相当与：

public static int mockAdd(int count){

int temp = count;

count = count+1;

return temp;

}

4:

变更一个被静态导入的方法，最好的办法是在原始类中重构，而非在本类中覆盖；

5：

instanceof只能用于对象判断，不能用于基本类型的判断。其特有的规则：若左操作数是null，结果直接返回false，不在判断有操作数是什么类；

6：

java泛型是为编码服务的，在编译成字节码时，T已经是Object类型了，传递到实参时String类型，也就是说T表面是Object，实际类型是String。

7：

final修饰的基本类型和String类型，编译器认为他是稳定态的，故在编译时就直接把值编译到字节码中，避免了运行期引用，以提高代码的执行效率，故对于引用了final修饰的基本类型和String类型的类没有重新编译就会使用旧值。

所以应用系统发布时禁止实用类文件替换的方式，整体WAR包发布才最安全。

8：

判断奇偶用偶判断，不要用奇判断；因为用i%2==1来判断会导致计算的负奇数为偶数。

9：

整型池：int参数在-128和127之间，则使用integer.valueOf产生包装对象时直接从整型池中获取，不在该范围的则通过new生成包装对象。可以提高系统性能同时节约存储空间。

10：

int可以直接转变成long，之后转变为Long，但无法直接变成包装类型。

11：

设置随机数种子：

**int** Random r = **new** Random(1000);

**for** (**int** i = 0; i < 5; i++) {

System.***out***.println(r.nextInt());

}

运行多次，虽然实例不同，但都会获得相同的随机数。若非需要，不要设置随机数种子。

12：

不要覆写静态方法，隐藏用于静态方法，覆写用于非静态方法，隐藏父类和子类拥有相同名字的属性或者方法（ 方法隐藏只有一种形式，就是父类和子类存在相同的静态方法）时，父类的同名的属性或者方法形式上不见了，实际是还是存在的；调用的还是父类的属性和方法。隐藏是对于静态方法和成员变量（静态变量和实例变量）而言的。

而覆盖则指的是父类引用指向了子类对象，调用的时候会调用子类的具体方法。

13：

构造代码块：在类中没有任何前缀或后缀，并用{}括起来的代码块。编译器会把构造代码块插入到每个构造函数的最前端。构造代码块会在构造函数内首先执行。

代码块插入到没有this方法的构造函数中，而调用其他构造函数的则不插入，确保每个构造函数只执行一次构造代码块。所以不用担心重复调用代码块的问题。

14：

静态内部类和普通内部类区别：（1）：普通内部类可直接访问外部类的属性，方法，即使是private类型也可以访问，因为内部类持有一个外部类的应用，可以自由访问。而静态内部类只可以访问外部类的静态方法和静态属性。（2）：

普通内部类与外部类相互依赖，内部类实例不能脱离外部类实例。而静态内部类可以独立存在，即使外部类消亡了，静态内部类还是可以存在的。（3）：普通内部类不能声明static的方法和变量，常量（就是final static修饰的属性）还是可以的。静态内部类类似外部类，没有任何限制。

15：

List l1=new ArrayList();List l2=new ArrayList(){};

List l2=new ArrayList(){{}};其中l1，l2，l3取getClass（）都不相同。因为l2代表的是一个继承于ArrayList的匿名类的声明和赋值。l3也是一个匿名类的定义，只是多了一个初始化块而已。

16：

内部类可以继承一个与外部类无关的类，保证了内部类的独立性，基于这一点，才实现了多重继承（多个内部类继承其他的父类）。

17:

拷贝是在内存中进行，所有性能方面比直接通过new生成的对象要快很多，特别是大对象的生成上。浅拷贝存在对象属性拷贝不彻底的问题，基本类型拷贝其值，对象则拷贝地址引用，也就是说新拷贝出的对象与原有对象共享该实例对象，不受访问权限的限制。所以浅拷贝不便于直接使用。

18：

在覆写equals时建议使用getClass进行类型判断，而不要使用instanceof。使用后者很容易让子类钻空子。而且必须覆写hashCode方法。

19：string s1=”abc”;string s2 = “abc”;string s3=new string(“abc”);

String s4 = s3.intern();其中s1==s2；s1！=s3；s1==s4；原因：

Java为避免在一个系统中大量产生string对象，于是就设计一个字符串池，创建一个字符串时，首先检查池中是否有字面值相等的字符串，如果有就不在创建直接返回池中该对象的引用，如果没有则创建之，之后放的池中。但当new一个string的时候，就不会检查池，直接声明一个string对象放到池中，故两者不是同一个地址。Intern会检查当前对象在对象池中是否有字面值相同的引用对象，有则返回池中对象，没有则放置到对象池中并返回。垃圾回收不会回收字符串池。

20：stringBuffer可变字符序列，它的对象的值可以改变。StringBuilder也是，只不过前者是线程安全，后者不是。

21：string拼接方法：加号，concat方法和StringBuilder的append效率：加号（相当于调用stringBuilder的append方法以及tostring方法，每次循环都会创建一个stringBuilder对象）1438毫秒，concat（每次循环都会创建一个string）是703毫秒，append(整个方法都在做字符数组处理，加长，数组拷贝，全都基于基本的数据处理，没有创建新对象)是0毫秒。

22：原始类型数组（基本类型）不能作为asList的输入参数，否则会引起程序逻辑混乱，用之前使用对象类型。

23：asList方法产生的List对象不可更改。

24：最优列表遍历：ArrayList这种随机存取列表使用下标遍历，即：

for（int i=0;i<list.size();i++）,linkedList这种有序列表使用foreach方式遍历，如果采用下标遍历方法则会没一次get都会进行一次遍历。

25：subList产生的只是一个视图，所有的修改动作直接作用于愿列表中。subList生成字列表后，保持原列的只读表状态，一旦修改原列表，子列表就会报错。

26：两个集合的交集：list1.retainAll（list2）；

27：泛型在编译时是被擦除的；

28：class的对象都是单例的，一个实例对象值描述一个类，实例对象不区分泛型；

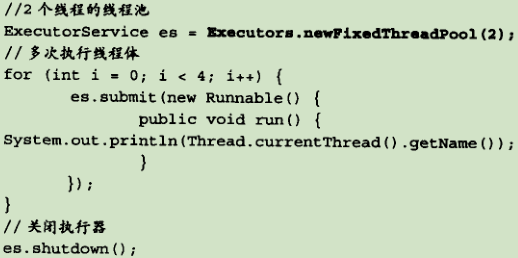
29：动态加载采用forName的方式，他只是加载类，并不执行任何代码，动态加载的原因是因为我们不知道生成的实例对象是什么类型，并不适用于数组；

30：构造函数中不要抛出异常。

31：继承thread类重写run方法但是不必重写start方法，如果确实有必要重写start方法，那么需要在start方法中加上super.start;

32：线程优先级推荐使用三个级别，分级越多，优先级相差不大则抢占cpu的机会 差别也不大，故执行顺序也不会有太大差别。

33：线程池：



输出结果：

pool-1-thread-1

pool-1-thread-2

pool-1-thread-1

pool-1-thread-2

spring整合线程池详见瑞亚游戏代码

34：java线程池最根本来讲只有两个ThreadPoolExecutor和ScheduleThreadPoolExecutor类，Executors的静态类可以直接生成不同的线程执行器，归根到底还是ThreadPoolExecutor和ScheduleThreadPoolExecutor类得分装类。

35：Lock锁是跟随对象的，synchronized锁是跟随类的。灵活强大选择Lock，快捷安全选择synchrinized

36：阻塞队列的长度是固定的。

37：CountDownLatch所有子线程完成后继续完成主线程；CyclicBarrier所有子线程处于等待状态，待都满足条件后同时执行

38：变量将可能放在小的作用范围内，目的是加快GC的回收