# 抽象类。

1. 抽象类不能被实例化，只有抽象类的非抽象子类可以创建对象。

2. 抽象类中的抽象方法只是声明，不包含方法体。

3. 构造方法，类方法（用 static 修饰的方法）不能声明为抽象方法。

4. 抽象类的子类必须给出抽象类中的抽象方法的具体实现，除非该子类也是抽象类。

# 接口。

1．Java接口本身没有任何实现，因为Java接口不涉及表象，而只描述public行为，所以Java接口比Java抽象类更抽象化。

2．一个类可以实现多个接口。

# 为什么多接口，少继承？

1．继承的第一个危害：继承不分青红皂白的把父类的公有和受保护的方法统统继承下来。如果你的子类没有对一些方法重写，就会对你的子类产生危害。

2．因为Java不像C++一样支持多继承，所以Java可以通过实现接口来弥补这个局限。

# HashMap和HashTable。

HashMap能接受空键值和空值，HashTable不能。HashTable多线程安全可以多线程操作一个，HashMap不能（ConcurrentModificationException）。四种遍历方式和正确的删除元素方式。

# final、finally和finalize。

final：修饰类、方法和变量。

finally：try catch finally。即使在try里return，也是return后执行finally。

finalize：作为确保某些非内存资源(如Socket、文件等)释放的一个补充。

# &和&&。

都是代表与运算，&还可以作为位运算符，&&具有短路的功能。

# Collection和Collections。

Collection是一个接口类，Collections是一个静态方法工具类。

# GC是什么? 为什么要有GC。

清除不必要的内存；提高生产率不需要管理内存问题，帮助程序的完成性避免jvm的崩溃，但是加大了程序的负担。分为新生代、老年代和永久代。

# Thread和Runnable。

生成方式的不同，类型不同，任务和线程是否绑定。

# String s = new String(xyz);创建了几个String Object?

首先在字符串常量缓冲区，找到？不创建string对象，否则创建，这样就一个string对象，遇到new运算符号了，在内存上创建string对象，并将其返回给s，又一个对象。

# private、default、protected和public。

类、包、子类和外部。

# equals和==。

比较值和地址。

# Array与ArrayList。

数组和集合，ArrayList有更多的操作。

# 八种基本类型。

byte、char、short、int、long、float、double和boolean。

# 类的初始化顺序。

（静态变量、静态初始化块）>（变量、初始化块）>构造器。

# String。

1：单独使用""引号创建的字符串都是常量,编译期就已经确定存储到String Pool中。

2：使用new String("")创建的对象会存储到heap中,是运行期新创建的。

3：使用只包含常量的字符串连接符如"aa" + "aa"创建的也是常量,编译期就能确定,已经确定存储到String Pool中。

4：使用包含变量的字符串连接符如"aa" + s1创建的对象是运行期才创建的,存储在heap中。

# 当一个线程进入一个对象的一个synchronized 方法后，其它线程是否可进入此对象的其它方法？

1：其他方法加了synchronized关键字则不能。

2：这个方法调用了wait，则其他方法都能进入。

3：其他方法是static可以。

# 简述 synchronized 和 java.util.concurrent.locks.Lock的异同。

主要相同点：Lock 能完成 synchronized 所实现的所有功能

主要不同点：Lock 有比 synchronized 更精确的线程语义和更好的性能。synchronized 会自动释放锁，而 Lock 一定要求程序员手工释放，并且必须在 finally 从句中释放。Lock 还有更强大的功能，例如，它的 tryLock 方法可以非阻塞方式去拿锁。而synchronized，多个线程访问一个synchronize修饰的方式的时，如果这个方法被占用，则线程会一直尝试获取锁，占用和浪费系统资源甚至会死锁造成程序中断。

# newInstance()方法和new关键字的异同。

使用getDeclaredConstructor()代替newInstance()。

newInstance():弱类型,效率低,只能调用无参构造

new():强类型,高效率,能调用任何public构造器

使用new关键字时你得保证import了这个驱动类才行，而使用class.formname时你只要保证引用了驱动的jar包就行了。

如果用new的话，当工程改变了数据库，你得修改.java文件和引入相应的驱动包。

而使用class.formname时可以将驱动的类配置在xml文件中。加载驱动对象时修改xml和引入jar包就行。

# 并发编程经常遇到的三个问题。

原子性问题

可见性问题

有序性问题

# 浅拷贝和深拷贝。

## 浅拷贝

Object类是类结构的根类，其中有一个方法为protected Object **clone**() throws [CloneNotSupportedException](http://tool.oschina.net/uploads/apidocs/jdk-zh/java/lang/CloneNotSupportedException.html)，这个方法就是进行的浅拷贝。有了这个浅拷贝模板，我们可以通过调用clone()方法来实现对象的浅拷贝。但是需要注意：1、Object类虽然有这个方法，但是这个方法是受保护的（被protected修饰），所以我们无法直接使用。2、使用clone方法的类必须实现Cloneable接口，否则会抛出异常CloneNotSupportedException。对于这两点，我们的解决方法是，在要使用clone方法的类中重写clone()方法，通过super.clone()调用Object类中的原clone方法。

## 深拷贝

与通过重写clone方法实现浅拷贝的基本思路一样，只需要为对象图的每一层的每一个对象都实现Cloneable接口并重写clone方法，最后在最顶层的类的重写的clone方法中调用所有的clone方法即可实现深拷贝。简单的说就是：每一层的每个对象都进行浅拷贝=深拷贝。

# 四舍五入。

BigDecimal.ROUND\_UP;远于0。

ROUND\_DOWN;趋向0。

ROUND\_CEILING;趋向正无穷。

ROUND\_FLOOR;趋向负无穷。

ROUND\_HALF\_UP;四入五入。

ROUND\_HALF\_DOWN;四舍五入。

ROUND\_HALF\_EVEN;银行家算法，舍去位后不为零进，否则看前一位，奇进偶退。

# 内部类

## 成员内部类

可以无条件访问外部类的所有成员属性和成员方法（包括private成员和静态成员）。

当成员内部类拥有和外部类同名的成员变量或者方法时，会发生隐藏现象，即默认情况下访问的是成员内部类的成员。如果要访问外部类的同名成员，需要以下面的形式进行访问：“外部类.this.成员变量”。

在外部类中如果要访问成员内部类的成员，必须先创建一个成员内部类的对象。

成员内部类是依附外部类而存在的，也就是说，如果要创建成员内部类的对象，前提是必须存在一个外部类的对象。

## 局部内部类

局部内部类是定义在一个方法或者一个作用域里面的类，它和成员内部类的区别在于局部内部类的访问仅限于方法内或者该作用域内。

## 匿名内部类

一般使用匿名内部类的方法来编写事件监听代码。同样的，匿名内部类也是不能有访问修饰符和 static 修饰符的。匿名内部类是唯一一种没有构造器的类。正因为其没有构造器，所以匿名内部类的使用范围非常有限，大部分匿名内部类用于接口回调。匿名内部类在编译的时候由系统自动起名为 Outter$1.class。一般来说，匿名内部类用于继承其他类或是实现接口，并不需要增加额外的方法，只是对继承方法的实现或是重写。

## 静态内部类：

成员内部类加了static关键字。

# 对象的比较

## Comparable接口-java.lang

实现int compareTo（T o）方法，可以通过 Collections.sort（和 Arrays.sort）进行自动排序。

## Comparator接口-java.util

单独一个类实现。(o1, o2) -> o2.age-o1.age。