1. **Java中==和equals方法区别是什么？**

1）对于==，如果作用于Java 8种基本数据类型的变量，则直接比较其存储的 “值”是否相等；

如果作用于引用类型的变量，则比较的是所指向的对象的地址

2）对于equals方法，注意：equals方法不能作用于基本数据类型的变量。

如果没有对equals方法进行重写，则比较的是引用类型的变量所指向的对象的地址；

诸如String、Integer、Date等类对equals方法进行了重写的话，比较的是所指向的对象的内容。

1. **抽象类和接口的区别是什么？**
2. 抽象类和普通类的主要有三点区别：

　　1）抽象方法必须为public或者protected（因为如果为private，则不能被子类继承，子类便无法实现该方法），缺省情况下默认为public。

　　2）抽象类不能用来创建对象；

　　3）如果一个类继承于一个抽象类，则子类必须实现父类的抽象方法。如果子类没有实现父类的抽象方法，则必须将子类也定义为为abstract类。

1. 接口中可以含有变量和方法。但是，接口中的变量会被隐式地指定为public static final变量；而方法会被隐式地指定为public abstract方法。并且接口中所有的方法不能有具体的实现，也就是说，接口中的方法必须都是抽象方法。
2. **区别**

1）抽象类可以提供成员方法的实现细节，而接口中只能存在public abstract 方法；

　　2）抽象类中的成员变量可以是各种类型的，而接口中的成员变量只能是public static final类型的；

　　3）接口中不能含有静态代码块以及静态方法，而抽象类可以有静态代码块和静态方法；

　　4）一个类只能继承一个抽象类，而一个类却可以实现多个接口

**补充：**抽象类是对一种事物的抽象，即对类抽象，而接口是对行为的抽象。抽象类是对整个类整体进行抽象，包括属性、行为，但是接口却是对类局部（行为）进行抽象。

1. **HashTable、HashSet和HashMap区别？**
2. 继承不同。

public class Hashtable extends Dictionary implements Map  
public class HashMap extends AbstractMap implements Map

1. Hashtable 中的方法是同步的，而HashMap中的方法在缺省情况下是非同步的。在多线程并发的环境下，可以直接使用Hashtable，但是要使用HashMap的话就要自己增加同步处理了。
2. Hashtable中，key和value都不允许出现null值。

在HashMap中，null可以作为键，这样的键只有一个；可以有一个或多个键所对应的值为null。当get()方法返回null值时，即可以表示 HashMap中没有该键，也可以表示该键所对应的值为null。因此，在HashMap中不能由get()方法来判断HashMap中是否存在某个键， 而应该用containsKey()方法来判断。

1. 两个遍历方式的内部实现上不同。Hashtable、HashMap都使用了 Iterator。而由于历史原因，Hashtable还使用了Enumeration的方式 。
2. 哈希值的使用不同，HashTable直接使用对象的hashCode。而HashMap重新计算hash值。
3. Hashtable和HashMap它们两个内部实现方式的数组的初始大小和扩容的方式。HashTable中hash数组默认大小是11，增加的方式是 old\*2+1。HashMap中hash数组的默认大小是16，而且一定是2的指数。
4. HashSet是利用HashMap来实现的，实现了Set接口；内部封装了hashmap，故也是无序的；因为实现set接口，存储的是key，value永远为PRESENT；HashSet不容许重复的对象；非线程安全；采用add(Object o)方法用来在Set中添加元素，当元素值重复时则会立即返回false，如果成功添加的话会返回true；HashSet较HashMap来说比较慢。
5. **进程与线程的区别？**

（1）调度：线程作为调度和分配的基本单位，进程作为拥有资源的基本单位

（2）并发性：不仅进程之间可以并发执行，同一个进程的多个线程之间也可并发执行

（3）拥有资源：进程是拥有资源的一个独立单位，线程不拥有系统资源，但可以访问隶属于进程的资源.

（4）系统开销：在创建或撤消进程时，由于系统都要为之分配和回收资源，导致系统的开销明显大于创建或撤消线程时的开销。

1. **堆和栈**
2. 在函数中定义的一些基本类型的变量和对象的引用变量都在函数的栈内存中分配，当超过变量的作用域后，Java 会自动释放掉为该变量分配的内存空间；
3. 堆内存用来存放由new创建的对象和数组，在堆中分配的内存，由Java虚拟机的自动垃圾回收器来管理。
4. **Java的垃圾回收机制？**

由于具有垃圾回收机制，Java中对象不再有“作用域”的概念，只有对象的引用才有作用域，垃圾回收器通常作为单独级别的线程运行，在不可预知的情况下对堆内存中已经死亡或长时间没有使用的对象进行清除和回收，即使使用了System.gc()和Runtime.getRuntime().gc() 方法也只是简单的申请垃圾回收，垃圾回收器并不一定在接到申请后就立刻回收。

回收机制有分代复制垃圾回收、标记垃圾回收和增量垃圾回收3种。

1. **Java中LinkedList和ArrayList以及Vector的区别？**
2. ArrayList是实现了基于动态数组的数据结构，LinkedList基于链表的数据结构。
3. 对于随机访问get和set，ArrayList觉得优于LinkedList，因为LinkedList要移动指针。
4. 对于新增和删除操作add和remove，LinkedList比较占优势，因为ArrayList要移动数据。
5. Vector 和ArrayList类似,但属于强同步类。如果你的程序本身是线程安全的(thread-safe,没有在多个线程之间共享同一个集合/对象),那么使用ArrayList是更好的选择。Vector和ArrayList在更多元素添加进来时会请求更大的空间。Vector每次请求其大小的双倍空间，而ArrayList每次对size增长50%。
6. **Java中sleep()和wait()方法区别？**
7. 这两个方法来自不同的类分别是Thread和Object；

（2）最主要是sleep方法没有释放锁，而wait方法释放了锁，使得其他线程可以使用同步控制块或者方法；

（3）wait，notify和notifyAll只能在同步控制方法或者同步控制块里面使用，而sleep可以在任何地方使用（使用范围）；

（4）sleep必须捕获异常，而wait，notify和notifyAll不需要捕获异常。

1. **HashMap实现原理？**

（1） HashMap是基于哈希表的Map接口的非同步实现。此实现提供所有可选的映射操作，并允许使用null值和null键。

（2）HashMap实际上是一个“链表散列”的数据结构，即数组和链表的结合体。此外，HashMap不是线程安全的。

（3）HashMap的存取：当我们往HashMap中put元素的时候，先根据key的hashCode重新计算hash值，根据hash值得到这个元素在数组中的位置（即下标），如果数组该位置上已经存放有其他元素了，那么在这个位置上的元素将以链表的形式存放，新加入的放在链头，最先加入的放在链尾。

从HashMap中get元素时，首先计算key的hashCode，找到数组中对应位置的某一元素，然后通过key的equals方法在对应位置的链表中找到需要的元素。

（4）HashMap的扩容：当HashMap中的元素个数超过数组大小\*loadFactor时，就会进行数组扩容，loadFactor的默认值为0.75，这是一个折中的取值。也就是说，默认情况下，数组大小为16，那么当HashMap中元素个数超过16\*0.75=12的时候，就把数组的大小扩展为 2\*16=32，即扩大一倍，然后重新计算每个元素在数组中的位置。

1. **String、StringBuffer和StringBuilder的区别？**

（1）可变与不可变。String类中使用字符数组保存字符串，因为有“final”修饰符，**private final char value[]，**任何对String的操作都会产生新的对象，所以string对象是不可变的；

StringBuilder与StringBuffer都继承自AbstractStringBuilder类，使用字符数组保存字符串，任何对它指向的字符串的操作都不会产生新的对象。

（2）String对象不可变，显然线程安全；StringBuffer也是线程安全的；StringBuilder是非线程安全的。如果程序不是多线程的，那么使用StringBuilder效率高于StringBuffer。

1. **Java中工厂模式？**

**工厂模式主要是为创建对象提供过渡接口，以便将创建对象的具体过程屏蔽隔离起来，达到提高灵活性的目的。主要分为三种：（1）简单工厂模式（2）工厂方法模式（3）抽象工厂模式。**

**（1）简单工厂模式**

组成：

1）**工厂类角色：这是本模式的核心，含有一定的商业逻辑和判断逻辑。在java中它往往由一个具体类实现。  
     2) 抽象产品角色：它一般是具体产品继承的父类或者实现的接口。在java中由接口或者抽象类来实现。  
      3) 具体产品角色：工厂类所创建的对象就是此角色的实例。在java中由一个具体类实现。**

**（2）工厂方法模式**

1) 抽象工厂角色： 这是工厂方法模式的核心，它与应用程序无关。是具体工厂角色必须实现的接口或者必须继承的父类。在java中它由抽象类或者接口来实现。  
       2) 具体工厂角色：它含有和具体业务逻辑有关的代码。由应用程序调用以创建对应的具体产品的对象。  
       3) 抽象产品角色：它是具体产品继承的父类或者是实现的接口。在java中一般有抽象类或者接口来实现。  
       4) 具体产品角色：具体工厂角色所创建的对象就是此角色的实例。在java中由具体的类来实现。

**（3）抽象工厂模式**

和工厂方法模式类似，只不过工厂方法模式中抽象方法是单一的，而抽象工厂模式中是多个抽象方法。在抽象工厂模式中，抽象产品 (AbstractProduct) 可能是一个或多个，从而构成一个或多个产品族(Product Family)。 在只有一个产品族的情况下，抽象工厂模式实际上退化到工厂方法模式。

**工厂方法模式：**一个抽象产品类，可以派生出多个具体产品类。     
一个抽象工厂类，可以派生出多个具体工厂类。     
每个具体工厂类只能创建一个具体产品类的实例。  
  **抽象工厂模式：**  
多个抽象产品类，每个抽象产品类可以派生出多个具体产品类。     
一个抽象工厂类，可以派生出多个具体工厂类。     
每个具体工厂类可以创建多个具体产品类的实例。     
**区别：**  
工厂方法模式只有一个抽象产品类，而抽象工厂模式有多个。     
工厂方法模式的具体工厂类只能创建一个具体产品类的实例，而抽象工厂模式可以创建多个。

1. **Java中的反射机制？**

**定义：**JAVA反射机制是在运行状态中，对于任意一个类，都能够知道这个类的所有属性和方法；对于任意一个对象，都能够调用它的任意一个方法；这种动态获取的信息以及动态调用对象的方法的功能称为java语言的反射机制。

Java反射机制主要提供了以下功能：（1） 在运行时判断任意一个对象所属的类；（2）在运行时构造任意一个类的对象；（3）在运行时判断任意一个类所具有的成员变量和方法；（4）在运行时调用任意一个对象的方法；（5）生成动态代理。

1. **Java中桥接模式？**

Bridge模式是构造型的设计模式之一。Bridge模式基于类的最小设计原则，通过使用封装，聚合以及继承等行为来让不同的类承担不同的责任。它的主要特点是把抽象（abstraction）与行为实现（implementation）分离开来，从而可以保持各部分的独立性以及应对它们的功能扩展。

通过对象组合的方式，Bridge 模式把两个角色之间的继承关系改为了耦合的关系，从而使这两者可以从容自若的各自独立的变化，这也是Bridge模式的本意。

**效果及实现要点：**1．Bridge模式使用“对象间的组合关系”解耦了抽象和实现之间固有的绑定关系，使得抽象和实现可以沿着各自的维度来变化。  
2．所谓抽象和实现沿着各自维度的变化，即“子类化”它们，得到各个子类之后，便可以任意它们，从而获得不同路上的不同汽车。  
3．Bridge模式有时候类似于多继承方案，但是多继承方案往往违背了类的单一职责原则（即一个类只有一个变化的原因），复用性比较差。Bridge模式是比多继承方案更好的解决方法。  
4．Bridge模式的应用一般在“两个非常强的变化维度”，有时候即使有两个变化的维度，但是某个方向的变化维度并不剧烈——换言之两个变化不会导致纵横交错的结果，并不一定要使用Bridge模式。

1. **Java中的观察者模式？**

**组成：**

**抽象主题角色**：把所有对观察者对象的引用保存在一个集合中，每个抽象主题角色都可以有任意数量的观察者。抽象主题提供一个接口，可以增加和删除观察者角色。一般用一个抽象类和接口来实现。

**抽象观察者角色**：为所有具体的观察者定义一个接口，在得到主题的通知时更新自己。

**具体主题角色**：在具体主题内部状态改变时，给所有登记过的观察者发出通知。具体主题角色通常用一个子类实现。

**具体观察者角色**：该角色实现抽象观察者角色所要求的更新接口，以便使本身的状态与主题的状态相协调。通常用一个子类实现。如果需要，具体观察者角色可以保存一个指向具体主题角色的引用。

观察者模式的应用场景：

1、 对一个对象状态的更新，需要其他对象同步更新，而且其他对象的数量动态可变。

2、 对象仅需要将自己的更新通知给其他对象而不需要知道其他对象的细节。

观察者模式的优点：

1、 Subject和Observer之间是松偶合的，分别可以各自独立改变。

2、 Subject在发送广播通知的时候，无须指定具体的Observer，Observer可以自己决定是否要订阅Subject的通知。

3、 遵守大部分GRASP原则和常用设计原则，高内聚、低偶合。

1. **Java中的适配器模式？**

将一个类的接口转换成客户希望的另外一个接口。Adapter模式使得原本由于接口不兼容而不能一起工作的那些类可以在一起工作。

**模式中的角色**

1. 目标接口（Target）：客户所期待的接口。目标可以是具体的或抽象的类，也可以是接口。
2. 需要适配的类（Adaptee）：需要适配的类或适配者类。
3. 适配器（Adapter）：通过包装一个需要适配的对象，把原接口转换成目标接口。

**实现方式**

（1）类的适配器模式（采用继承实现）

     （2）对象适配器（采用对象组合方式实现）

1. **JVM的实现原理？**