1.请列举出在 JDK 中几个常用的设计模式？

单例模式（Singleton pattern）用于 Runtime，Calendar 和其他的一些类中。工厂模式（Factory pattern）被用于各种不可变的类如 Boolean，像 Boolean.valueOf，观察者模式（Observer pattern）被用于 Swing 和很多的事件监听中。装饰器设计模式（Decorator design pattern）被用于多个 Java IO 类中。

2.什么是设计模式？你是否在你的代码里面使用过任何设计模式？

设计模式是世界上各种各样程序员用来解决特定设计问题的尝试和测试的方法。设计模式是代码可用性的延伸

3.Java 中什么叫单例设计模式？请用 Java 写出线程安全的单例模式

单例模式重点在于在整个系统上共享一些创建时较耗资源的对象。整个应用中只维护一个特定类实例，它被所有组件共同使用。Java.lang.Runtime 是单例模式的经典例子。从 Java5 开始你可以使用枚举（enum）来实现线程安全的单例。

4.在 Java 中，什么叫观察者设计模式（observer design pattern）？

观察者模式是基于对象的状态变化和观察者的通讯，以便他们作出相应的操作。简单的例子就是一个天气系统，当天气变化时必须在展示给公众的视图中进行反映。这个视图对象是一个主体，而不同的视图是观察者。

5.使用工厂模式最主要的好处是什么？在哪里使用？

工厂模式的最大好处是增加了创建对象时的封装层次。如果你使用工厂来创建对象，之后你可以使用更高级和更高性能的实现来替换原始的产品实现或类，这不需要在调用层做任何修改。

6.举一个用 Java 实现的装饰模式(decorator design pattern)？它是作用于对象层次还是类层次？

装饰模式增加强了单个对象的能力。Java IO 到处都使用了装饰模式，典型例子就是 Buffered 系列类如 BufferedReader 和 BufferedWriter，它们增强了 Reader 和 Writer 对象，以实现提升性能的 Buffer 层次的读取和写入。

7.在 Java 中，为什么不允许从静态方法中访问非静态变量？

Java 中不能从静态上下文访问非静态数据只是因为非静态变量是跟具体的对象实例关联的，而静态的却没有和任何实例关联。

8.设计一个 ATM 机，请说出你的设计思路？

比如设计金融系统来说，必须知道它们应该在任何情况下都能够正常工作。不管是断电还是其他情况，ATM 应该保持正确的状态（事务），想想 加锁（locking）、事务（transaction）、错误条件（error condition）、边界条件（boundary condition） 等等。尽管你不能想到具体的设计，但如果你可以指出非功能性需求，提出一些问题，想到关于边界条件，这些都会是很好的。

9.在 Java 中，什么时候用重载，什么时候用重写？

如果你看到一个类的不同实现有着不同的方式来做同一件事，那么就应该用重写（overriding），而重载（overloading）是用不同的输入做同一件事。在 Java 中，重载的方法签名不同，而重写并不是。

10.举例说明什么情况下会更倾向于使用抽象类而不是接口？

接口和抽象类都遵循"面向接口而不是实现编码"设计原则，它可以增加代码的灵活性，可以适应不断变化的需求。下面有几个点可以帮助你回答这个问题：在 Java 中，你只能继承一个类，但可以实现多个接口。所以一旦你继承了一个类，你就失去了继承其他类的机会了。

接口通常被用来表示附属描述或行为如：Runnable、Clonable、Serializable 等等，因此当你 使用抽象类来表示行为时，你的类就不能同时是 Runnable 和 Clonable(注：这里的意思是指如果把 Runnable 等实现为抽象类的情况)，因为在 Java 中你不能继承两个类，但当你使用接口时，你的类就可以同时拥有多个不同的行为。

在一些对时间要求比较高的应用中，倾向于使用抽象类，它会比接

口稍快一点。如果希望把一系列行为都规范在类继承层次内，并且可以更好地在同一个地方进行编码，那么抽象类是一个更好的选择。有时，接口和抽象类可以一起使用，接口中定义函数，而在抽象类中定义默认的实现。

11. 工厂方法模式(利用创建同一接口的不同实例)

1、普通工厂模式：建立一个工厂类，对实现了同一接口的一些类进行实例的创建；

12. 接口是什么？为什么要使用接口而不是直接使用具体类？

接口用于定义 API。它定义了类必须得遵循的规则。同时，它提供了一种抽象，因为客户端只使用接口，这样可以有多重实现，如 List 接口，你可以使用可随机访问的 ArrayList，也可以使用方便插入和删除的 LinkedList。接口中不允许写代码，以此来保证抽象，但是 Java 8 中你可以在接口声明静态的默认方法，这种方法是具体的。

13.java 中，抽象类与接口之间有什么区别？

1.一个类可以实现多个接口 ，但却只能继承最多一个抽象类。

2.抽象类可以包含具体的方法 ， 接口的所有方法都是抽象的。

3.抽象类可以声明和使用字段 ，接口则不能，但接口可以创建静态的 final 常量。

4.接口的方法都是 public 的，抽象类的方法可以是 public，protected，private 或者默认的 package；

5.抽象类可以定义构造函数，接口却不能。

14.除了单例模式，你在生产环境中还用过什么设计模式？

这需要根据你的经验来回答。一般情况下，你可以说依赖注入，工厂模式，装饰模式或者观察者模式，随意选择你使用过的一种即可。不过你要准备回答接下的基于你选择的模式的问题。

15.什么是里氏替换原则？

1、开闭原则（Open Close Principle）

开闭原则就是说对扩展开放，对修改关闭。在程序需要进行拓展的时候，不能去修改原有的代码，实现一个热插拔的效果。所以一句话概括就是：为了使程序的扩展性好，易于维护和升级。想要达到这样的效果，我们需要使用接口和抽象类，后面的具体设计中我们会提到这点。

2、里氏代换原则（Liskov Substitution Principle）

里氏代换原则(Liskov Substitution Principle LSP)面向对象设计的基本原则之一。 里氏代换原则中说，任何基类可以出现的地方，子类一定可以出现。 LSP 是继承复用的基石，只有当衍生类可以替换掉基类，软件单位的功能不受到影响时，基类才能真正被复用，而衍生类也能够在基类的基础上增加新的行为。里氏代换原则是对“开-闭”原则的补充。实现“开-闭”原则的关键步骤就是抽象化。而基类与子类的继承关系就是抽象化的具体实现，所以里氏代换原则是对实现抽象化的具体步骤的规范。—— From Baidu 百科

3、依赖倒转原则（Dependence Inversion Principle）

这个是开闭原则的基础，具体内容：真对接口编程，依赖于抽象而不依赖于具体。

4、接口隔离原则（Interface Segregation Principle）

这个原则的意思是：使用多个隔离的接口，比使用单个接口要好。还是一个降低类之间的耦合度的意思，从这儿我们看出，其实设计模式就是一个软件的设计思想，从大型软件架构出发，为了升级和维护方便。所以上文中多次出现：降低依赖，降低耦合。

5、迪米特法则（最少知道原则）（Demeter Principle）

为什么叫最少知道原则，就是说：一个实体应当尽量少的与其他实体之间发生相互作用，使得系统功能模块相对独立。

6、合成复用原则（Composite Reuse Principle）

原则是尽量使用合成/聚合的方式，而不是使用继承

16.什么情况下会违反迪米特法则？为什么会有这个问题？

迪米特法则建议“只和朋友说话，不要陌生人说话”，以此来减少类之间的耦合。

17.适配器模式是什么？什么时候使用？

适配器模式（Adapter Pattern）是作为两个不兼容的接口之间的桥梁。这种类型的设计模式属于结构型模式，它结合了两个独立接口的功能。适配器模式提供对接口的转换。如果你的客户端使用某些接口，但是你有另外一些接口，你就可以写一个适配去来连接这些接口。

18.适配器模式与装饰器模式有什么区别？

虽然适配器模式和装饰器模式的结构类似，但是每种模式的出现意图不同。适配器模式被用于桥接两个接口，而装饰模式的目的是在不修改类的情况下给类增加新的功能。

装饰者模式：动态地将责任附加到对象上，若要扩展功能，装饰者模提供了比继承更有弹性的替代方案。

通俗的解释：装饰模式就是给一个对象增加一些新的功能，而且是动态的，要求装饰对象和被装饰对象实现同一个接口，装饰对象持有被装饰对象的实例。

适配器模式：将一个类的接口，转换成客户期望的另一个接口。适配器让原本接口不兼容的类可以合作无间。

适配器模式有三种：类的适配器模式、对象的适配器模式、接口的适配器模式。

通俗的说法：适配器模式将某个类的接口转换成客户端期望的另一个接口表示，目的是消

除由于接口不匹配所造成的类的兼容性问题。

举例如下：

1、适配器模式

//file 为已定义好的文件流

FileInputStream fileInput = new FileInputStream(file);

InputStreamReader inputStreamReader = new InputStreamReader(fileInput);

以上就是适配器模式的体现，FileInputStream 是字节流，而并没有字符流读取字符的一些 api，因此通过 InputStreamReader 将其转为 Reader 子类，因此有了可以操作文本的文件方法。

2、装饰者模式

BufferedReader bufferedReader=new BufferedReader(inputStreamReader);构造了缓冲字符流，将 FileInputStream 字节流包装为 BufferedReader 过程就是装饰的过程，刚开始的字节流 FileInputStream 只有 read 一个字节的方法，包装为 inputStreamReader 后，就有了读取一个字符的功能，在包装为 BufferedReader 后，就拥有了 read 一行字符的功能。

19.适配器模式和代理模式之间有什么不同？

这个问题与前面的类似，适配器模式和代理模式的区别在于他们的意图不同。由于适配器模式和代理模式都是封装真正执行动作的类，因此结构是一致的，但是适配器模式用于接口之间的转换，而代理模式则是增加一个额外的中间层，以便支持分配、控制或智能访问。

20.什么是模板方法模式？

模板方法提供算法的框架，你可以自己去配置或定义步骤。例如，你可以将排序算法看做是一个模板。它定义了排序的步骤，但是具体的比较，可以使用 Comparable 或者其语言中类似东西，具体策略由你去配置。列出算法概要的方法就是众所周知的模板方法。

21.什么时候使用访问者模式？

访问者模式用于解决在类的继承层次上增加操作，但是不直接与之关联。这种模式采用双派发的形式来增加中间层。

22.什么时候使用组合模式？

组合模式使用树结构来展示部分与整体继承关系。它允许客户端采用统一的形式来对待单个对象和对象容器。当你想要展示对象这种部分与整体的继承关系时采用组合模式。

23.继承和组合之间有什么不同？

虽然两种都可以实现代码复用，但是组合比继承共灵活，因为组合允许你在运行时选择不同的实现。用组合实现的代码也比继承测试起来更加简单。

24.描述 Java 中的重载与重写？什么时候用重载，什么时候用重写？

重载和重写都允许你用相同的名称来实现不同的功能，但是重载是编译时活动，而重写是运行时活动。你可以在同一个类中重载方法，但是只能在子类中重写方法。重写必须要有继承。

对有经验的 Java 设计师来说，这是一个相当简单的问题。如果你看到一个类的不同实现有着不同的方式来做同一件事，那么就应该用重写（overriding），而重载（overloading）是用不同的输入做同一件事。在 Java 中，重载的方法签名不同，而重写并不是。

25.Java 中，嵌套公共静态类与顶级类有什么不同？

类的内部可以有多个嵌套公共静态类，但是一个 Java 源文件只能有一个顶级公共类，并且顶级公共类的名称与源文件名称必须一致。

26.OOP 中的组合、聚合和关联有什么区别？

如果两个对象彼此有关系，就说他们是彼此相关联的。组合和聚合是面向对象中的两种形式的关联。组合是一种比聚合更强力的关联。组合中，一个对象是另一个的拥有者，而聚合则是指一个对象使用另一个对象。如果对象 A 是由对象 B 组合的，则 A 不存在的话，B 一定不存在，但是如果 A 对象聚合了一个对象 B，则即使 A 不存在了，B 也可以单独存在。

27.给我一个符合开闭原则的设计模式的例子？

开闭原则要求你的代码对扩展开放，对修改关闭。这个意思就是说，如果你想增加一个新的功能，你可以很容易的在不改变已测试过的代码的前提下增加新的代码。有好几个设计模式是基于开闭原则的，如策略模式，如果你需要一个新的策略，只需要实现接口，增加配置，不需要改变核心逻辑。一个正在工作的例子是 Collections.sort() 方法，这就是基于策略模式，遵循开闭原则的，你不需为新的对象修改 sort() 方法，你需要做的仅仅是实现你自己的 Comparator 接口。

28.使用工厂模式最主要的好处是什么？你在哪里使用？

工厂模式的最大好处是增加了创建对象时的封装层次。如果 你使用工厂来创建对象，之后你可以使用更高级和更高性能的实现来替换原始的产品实现或类，这不需要在调用层做任何修改。可以看我的文章工厂模式得更详细的解释和和了解更多的好处。

29.工厂模式与抽象工厂模式的区别？

首先来看看这两者的定义区别：

工厂模式：定义一个用于创建对象的借口，让子类决定实例化哪一个类

抽象工厂模式：为创建一组相关或相互依赖的对象提供一个接口，而且无需指定他们的具体类

个人觉得这个区别在于产品，如果产品单一，最合适用工厂模式，但是如果有多个业务品种、业务分类时，通过抽象工厂模式产生需要的对象是一种非常好的解决方式。再通俗深化理解下：工厂模式针对的是一个产品等级结构 ，抽象工厂模式针对的是面向多个产品等级结构的。

再来看看工厂方法模式与抽象工厂模式对比：

工厂方法模式

抽象工厂模式

针对的是一个产品等级结构

针对的是面向多个产品等级结构

一个抽象产品类

多个抽象产品类

可以派生出多个具体产品类

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

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

每个具体工厂类可以创建多个具体产品类的实例

18.什么时候使用享元模式？ 享元模式通过共享对象来避免创建太多的对象。为了使用享元模式，你需要确保你的对象是不可变的，这样你才能安全的共享。JDK 中 String 池、Integer 池以及 Long 池都是很好的使用了享元模式的例子。

30.什么是设计模式？你是否在你的代码里面使用过任何设计模式？

设计模式是世界上各种各样程序员用来解决特定设计问题的尝试和测试的方法。设计模式是代码可用性的延伸。

31.你可以说出几个在 JDK 库中使用的设计模式吗？

装饰器设计模式（Decorator design pattern）被用于多个 Java IO 类中。单例模式（Singleton pattern）用于 Runtime，Calendar 和其他的一些类中。工厂模式（Factory pattern）被用于各种不可变的类如 Boolean，像 Boolean.valueOf，观察者模式（Observer pattern）被用于 Swing 和很多的事件监听中。

32.Java 中什么是单例设计模式？用 Java 写出线程安全的单例

单例对象（Singleton）是一种常用的设计模式。在 Java 应用中，单例对象能保证在一个 JVM 中，该对象只有一个实例存在。这样的模式有几个好处：

1、某些类创建比较频繁，对于一些大型的对象，这是一笔很大的系统开销。

2、省去了 new 操作符，降低了系统内存的使用频率，减轻 GC 压力。

3、有些类如交易所的核心交易引擎，控制着交易流程，如果该类可以创建多个的话，系统完全乱 了。（比如一个军队出现了多个司令员同时指挥，肯定会乱成一团），所以只有使用单例模式，才能保证核心交易服务器独立控制整个流程。

单例模式重点在于在整个系统上共享一些创建时较耗资源的对象。整个应用中只维护一个特定类实例，它被所有组件共同使用。Java.lang.Runtime 是单例模式的经典例子。你可以在我的文章 Java 单例模式的 10 个问题看到更多的问题和讨论。从 Java 5 开始你可以使用枚举（enum）来实现线程安全的单例。

33.什么是责任链设计模式？

责任链模式（Chain of Responsibility Pattern）为请求创建了一个接收者对象的链。这种模式给予请求的类型，对请求的发送者和接收者进行解耦。这种类型的设计模式属于行为型模式。在这种模式中，通常每个接收者都包含对另一个接收者的引用。如果一个对象不能处理该请求，那么它会把相同的请求传给下一个接收者，依此类推。