java概述

1　java语言的主要特点：

1. **简单性**：语法简单，Java语言不使用指针，使用引用。并提供了自动垃圾回收机制，使程序员无需担忧内存管理。
2. **面向对象的程序设计**：分析解决问题的方式更接近人类的固有思维，容易理解，且软件易于创建、维护、重用。
3. **跨平台性**：一次编译到处运行
4. **安全性**：
    - 强类型语言：要求显式的变量类型声明
    - 不支持指针，杜绝了内存的非法访问
    - JVM可以自动发现数组和字符串的越界，防止堆栈溢出
    - 自动内存单元收集防止了内存泄露
    - Java还提供了异常处理机制，简化错误处理任务
    - 运行时环境还有类装载器，字节码校验器和安全管理器这三个组
5. **多线程性**：Java运行时环境本身就是多线程的。另一方面，Java 语言内置多线程控制。
6. **动态性** Java程序执行时所需要调用的类在运行时动态地加载到内存中，这使得Java程序运行所需的内存开销小。可以被用于许多嵌入式系统。

2　重载与重写

重载：多态性在单个类中表现为方法重载；一个类可以有多个名字相同、形参列表不同的方法，在使用时由传递给他们的实参来决定使用哪个方法。
重写（覆盖）：多态性在多个类中表现为继承结构中的方法重写（覆盖）；父类和其子类中具有相同的方法头，但用不同的代码实现。

3　Java语言的执行过程

Java程序运行时必须经过编译和运行两个步骤，首先编译后缀为.java的源文件，生成后缀名为.class的字节码文件，Java虚拟机将字节码文件解释执行，并将结果显示出来。

语言基础

1　运算符与表达式

如果被除数是浮点数，除数就可以为0，结果是正或负数无穷:Infinity,不是浮点数则发生异常。

2　数组

数组是相同类型的数据元素按顺序组成的有序集，元素在数组中的相对位置由下标来指定。

面向对象和类

1　面向对象编程思想

面向对象是一种程序设计方法：直接以问题域中的事物（客体）为中心来观察和分析问题。将待解问题直接用一个个相互作用、相互驱动的对象来表示，它引入类的概念实现更高一级的抽象和封装。

2　面向对象编程的主要特征

特点：

1. **封装**：实现数据隐藏，将对象的使用者和设计者分开。封装最大的好处是降低了软件系统的耦合程度。实现了代码的可重用性和可维护性
2. **继承**：可支持代码的重用和扩展。继承性是类与类之间的一种关系，通过继承，可以在无需重新编写原有类的情况下，实现代码的扩展和重用。
3. **多态**：单一的接口或方法具有不同的动作，即重载和重写。指类的某个行为具有不同的表现形式。

3 单例

定义：
指一个类有且仅有一个实例向整个系统提供

单例模式核心思想：

1. 构造方法设为私有权限，防止外界任意调用
2. 需要提供一个公有的静态方法获取创建的对象实例
3. 创建的唯一对象是被共享且只能被（2）中的静态方法直接调用，所以，需要将其定义为私有的静态对象。
4. 静态对象的初始化可以在类的加载阶段初始化，也可以在外界首次需要对象时在方法（2）中调用构造方法创建对象，之后不再创建对象。

```
public class Singleton {
    // 1. 创建私有静态实例，在类加载时就创建
    private static final Singleton INSTANCE = new Singleton();
    // 2. 私有构造方法，防止外部通过new创建实例
    private Singleton() {
        // 初始化代码
    }
    // 3. 提供公共静态方法获取实例。另一种写法是可以在get的时候才创建实例
    public static Singleton getInstance() {
        return INSTANCE;
    }
}
```

4 final

修饰	意义	形式
类	表示最终类，该类不能被继承	[修饰符] final class 类名 { 类体 }
方法	表示最终方法，即该方法不能被子类覆盖	[修饰符] final 返回类型 方法名（[形参列表]）{

修饰	意义	形式
		方法体 }
数据	即常量，包括局部变量和字段，一旦赋值就不能再修改	[修饰符] final 数据类型 变量[=值]

类的进阶设计

1　动态绑定

将方法调用同其方法体连接起来叫做“绑定”（Binding）。绑定分为静态绑定和动态绑定

1. 静态绑定：是在程序执行前的由编译器或连接程序绑定
2. 动态绑定：是在运行时绑定，JVM通过对象的类型指针绑定方法。

规则如下：

1. 被final、static、private修饰的方法执行静态绑定，与编译时类型的方法体进行绑定。
2. 其余实例方法执行动态绑定，与对象的运行时类型的方法体进行绑定。
3. 成员变量（包括静态变量和实例变量）执行静态绑定，与引用类型的成员变量绑定。

2　抽象类的特点

1. 不能创建实例，即不能new一个抽象类。
2. 可不含抽象方法，但只要有一个方法是抽象方法，这个类就要定义成抽象类
3. 若子类没有实现父类的所有抽象方法，那子类也必须为抽象类。
4. 构造方法不能都定义为私有，否则不能有子类。
5. 不能用final修饰，抽象方法必须在子类中才可实现。

3　匿名内部类

定义：
匿名类是指没有类名只有类体的内部类

4　函数式接口与lambda

如果一个接口中有且只有一个抽象的方法，那这个接口就可称为函数式接口，可以（但不强求）用注解@FunctionalInterface来表示。

Lambda表达式本身是以一种简化的语法重写了接口中的唯一方法的匿名内部类实例。

5　接口的作用

接口用于规范对象的公共行为，提供比抽象类更高级别的抽象，用来定义全局常量和公开抽象方法

6　抽象类与接口的比较

1. **语法上**，抽象类用关键字abstract class定义，并且可以定义自己的成员变量和非抽象及抽象的成员方法；接口用关键字interface定义，接口内只有公开的静态常量，所有的成员方法都是公开的抽象方法、默认方法和静态方法。
2. **使用上**，抽象类是用来被继承的，一个类只能继承一个父类，但可以实现多个接口，这样可以使用接口实现多重继承，在一定程度上弥补单继承的缺点。
3. **设计上**，抽象类作为父类，与子类之间存在“is-a”关系，即父子类本质上是一种类型；接口只能表示类支持接口的行为，具有接口的功能，因此接口和实现类之间表示的是“like-a”关系。所以在设计上，如果父子类型本质上是一种类型，那父类可设计成抽象类；如果子类型只是想额外具有一些特性，则可以将父类设计成接口，而且这些接口不宜过大，应该设计成多个专题的小接口。这也是面向对象设计的一个重要原则——接口隔离原则：一个类对另一个类的依赖性应建立在最小接口上，不应强迫调用者依赖他们不会使用到的行为，过大的接口是对接口的污染。

异常处理

1　几个常见的异常

- ArithmeticException 数学错误，如被零除
- IOException 通用的IO故障

- FileNotFoundException 企图访问一个不存在的文件
- NullPointerException 空对象引用异常
- ArrayIndexOutOfBounds 数组下标越界

2 异常的分类

1. 运行时异常（RuntimeException）：在运行时期检查异常，Java运行时系统对字节码进行解释，在程序执行时检测到许多类型的异常，这类异常可通过适当编程避免，不要求捕获这类异常，例如空指针、数组越界、除数为零等。
2. 编译时异常（受检查异常）（Checked Exception）：在编译时期就会检查异常，程序必须对可能发生的异常进行处理，否则编译不通过。

3 Java的异常处理机制有哪些

1. 捕获处理异常：Java运行时系统捕获异常并找到相应异常的捕获处理代码并运行，这一过程称为捕获处理异常，如果找不到可以捕获异常的代码，程序将终止执行。使用try-catch-finally结构。
2. 声明异常：不捕获异常，声明异常只是声明方法有可能抛出的异常，从而让该方法上层调研方法捕获异常。使用throws及throw。

4 声明异常的目的

声明异常是将问题标识出来，报告给调用者。如果方法内通过throw抛出了编译时异常，而没有捕获处理，那么必须通过throws进行声明，让调用者去处理。

5 怎样抛出一个异常

1. Java运行时环境自动抛出异常。
2. 在一定条件下，用户使用throw语句显式抛出异常。

多线程编程

1　概念

1. **程序**：程序是一段静态代码，是指令与数据的集合。通常是外存上保存的可执行的二进制文件。
2. **进程**：进程（Process）是程序的一次运行活动。它对应从代码加载、执行到结束的一个过程，进程可以申请和拥有系统一整套资源，是系统进行资源分配和调度的基本单位。
3. **线程**：线程是进程中能够独立执行的执行序列。一个进程可以产生多个线程，线程也有创建、存活到消亡的的生命周期，每个线程都有独立的运行栈和程序计数器，是CPU调度的最小单位。
4. **进程与线程关系**：一个进程中的所有线程共享相同的地址空间和这个进程所拥有的操作系统资源

2　线程互斥：

我们用`synchronized`来描述互斥对象中的互斥方法。在Java中，每个对象都有一个监视器，当一个线程进入监视器的代码块时，就获得了该对象的锁，其他需要执行该锁代码块的线程就只能等待。当一个线程执行完毕后，释放该对象的锁，其他等待该对象的线程就可以进入代码块并重复上述过程。