Java 程序设计复习大纲

题型：
选择题（共 20 分，每空 2 分）
填空题（共 10-20 分，每空 2 分）
判断题（共 10 分，每小题 2 分）
简答题 1 题或 2 题
程序阅读题（3-5 题）
编码题（1 题或 2 题）

选择+判断题+填空：

重载+重写；

https://developer.aliyun.com/article/259000

覆盖（Override）即重写，是子类和父类之间的联系，是垂直关系；重载是同一个类的方法的关系，是水平关系。

构造方法；

构造器即构造方法，是在创建类对象的实例并为该对象分配内存时调用该代码块。它是一种用于初始化对象的特殊方法，分为无参构造器和有参构造器，当开发者没有显式定义时，会有默认无参构造器。

①Java 构造函数不得具有显式返回类型；
②它不能是抽象的(abstract)、最终的(final)、静态的(static)或同步的(synchronized)；
③构造函数名称必须与属于其类的名称相同；
④调用其他构造器用在首行 this，用父类构造器在首行用 super

抽象类和接口；

接口是一种定义了一组方法签名的集合，这些方法可以被实现该接口的任何类所实现。接口中的方法默认都是公共的抽象方法，不包含具体的实现代码。

抽象类是一个不能被实例化的类，它只能作为其他类的父类来使用。抽象类可以包含抽象方法和非抽象方法，其中抽象方法没有具体的实现，而非抽象方法有具体的实现代码。

import;package;

package 放在源文件第一行，package 包名，包名都是小写字母，每个层次用’.’分割，自定义包不能用 java 开头。
https://www.liaoxuefeng.com/wiki/1252599548343744/1260467032946976

import 类名

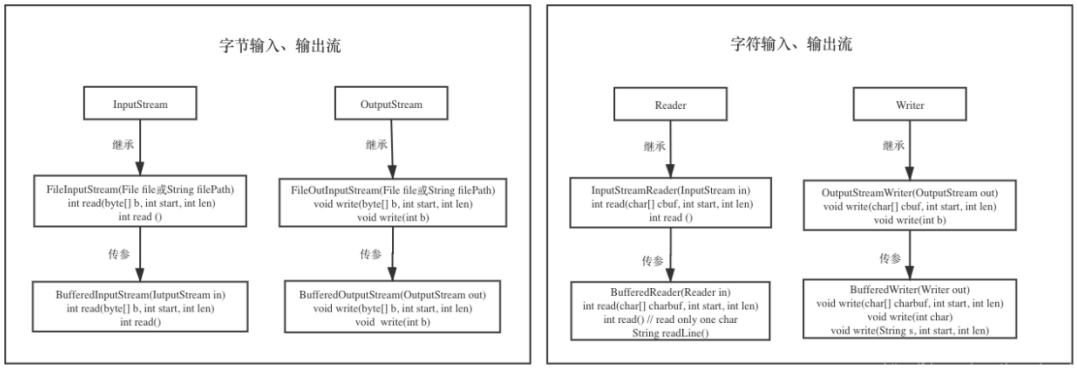
子类和父类的各种关系，例如构造函数调用方式；

当从另一个类继承时，必须在构造函数中首先调用 super（） 。 如果没有，则编译器将插入该调用。 这就是为什么在创建 Sub 对象时也调用 super 构造函数的原因。

这不会创建两个对象，只创建一个子类对象。 有父类构造函数调用的原因是，父类可能有私有字段需要由其构造函数初始化。

FileInputStream 和 FileOutputStream 等类和异常处理；

InputStream 、OutputStream 是字节流的抽象基类，派生出来的子类名称都是以其父类名作为子类名的后缀。



https://www.cnblogs.com/ShineLeBlog/p/14918032.html

通信的管道，包括它的类；

https://blog.csdn.net/qq_21484461/article/details/133094410

管道是一种特殊的流，包括输入管道流和输出管道流。

PipedInputStream inputStream = new PipedInputStream();

PipedOutputStream outputStream = new PipedOutputStream();

前者用于从一个线程读取数据，后者将数据写入另一个线程。

类包括内部类的权限；

https://zhuanlan.zhihu.com/p/97034966

https://www.cnblogs.com/yinzhenzhike/p/15588534.html

内部类包括成员内部类、局部内部类、匿名内部类、静态内部类等。

局部内部类定义在方法或作用域内，与成员内部类的区别在于访问权限不同

https://www.cnblogs.com/wuhenzhidu/p/anonymous.html

匿名内部类类似局部内部类，当局部内部类只用一次的时候可以用匿名内部类方便代替，匿名内部类在定义的同时实例化。

```
A a = new A(){//类定义}
```

https://www.cnblogs.com/nwgdk/p/8661442.html

嵌套类一般专指静态嵌套类，非静态嵌套类也叫内部类

```
Class OuterClass{
```

```
    static class StaticNestedClass{
        // do something……
    }
    Class InnerClass{
        // do something……
    }
}
OuterClass.StaticNestedClass obj = new Oute…… . Static……();
```

静态嵌套类其实就是在顶级类中封装的一个顶级类，它的行为和顶级类一样，它被封装在顶级类中其实就是为了方便打包，你甚至不需要实例化外部类。

异常处理；

https://www.runoob.com/java/java-exceptions.html

try：包裹可能会引发异常的代码块；
catch：异常发生时，执行此块代码，用于捕获和处理异常；
finally：不管是否异常，都会执行这个块的代码，用于执行清理工作。
Try 中发生 return 时，会先保存要 return 的变量值，等到执行完 finally 的语句后，再 return。

Java 的绑定方式；

https://www.cnblogs.com/jstarseven/articles/4631586.html
前期绑定和后期绑定，即静态绑定和动态绑定，java 中 final、static、private 和构造方法是前期绑定，这几个要不没法被继承要不没法覆盖，因此前期绑定;剩下的都是动态绑定。动态绑定时,比如调用 C 类对象 x.f(args)，那么编译器会列举出 C 类的所有名为 f 的方法和从 C 类的超类继承过来的 f 方法，然后根据参数类型筛选，优先看 C 类，没有的话看其父类。

static,final,abstract 等；

static 关键字可用于修饰成员变量和成员函数，想要实现对象中的共性数据的对象共享，可以将这个数据进行静态修饰，被静态修饰的成员可以直接被类名调用，静态随着类的加载而加载，而且优先于对象存在。静态方法只能访问静态成员（静态方法和静态变量），不可以访问非静态成员，这是因为静态方法加载时，优先于对象存在，所以没有办法访问对象中的成员。

静态方法中不能使用 this 和 super 关键字,因为 this 代表本类对象,super 代表父类对象，而静态内容执行时，有可能因为对象内容不存在（如对象还未创建），所以 this 和 super 无法使用。

final 关键字可用于修饰类，方法，变量（成员变量内，局部变量，静态变量），被 final 修饰的类是一个最终类，不可以被继承。被 final 修饰的方法是一个最终方法，不可以被覆盖，但是可以被继承。被 final 修饰的变量只能是一个常量，只能赋值一次。内部类被定义在类中的局部位置上时，只能访问局部被

final 修饰的局部变量。

abstract 关键字只能用于修饰类和方法，不能修饰变量。抽象方法只能定义在抽象类中，抽象方法和抽象类必须由 abstract 修饰，抽象方法只定义方法声明，不定义方法实现。抽象类不可以被实例化（创建对象），只有通过子类继承抽象类并覆盖抽象类中的所有抽象方法后，该子类才可以被实例化，否则该子类还是一个抽象类。抽象类中有构造函数用于给子类对象进行初始化，同时抽象类中可以含有非抽象方法。

注意事项：abstract 关键字不可以与 final、private、static 关键字共存，因为被 final 修饰的方法不可以被重写，意味着子类不可以重写该方法，如果 abstract 和 final 共同修饰父类中的方法，子类要实现抽象方法（abstract 的作用），而 final 又不让该方法重写，这相互矛盾。如果 private 和 abstract 共同修饰父类中的方法，private 修饰则该方法不可以被子类访问，但是 abstract 修饰需要子类去实现，两者产生矛盾。如果 static 和 abstract 共同修饰父类中的方法，static 表示是静态的方法，随着类的加载而加载，则该方法不需要在子类中去实现，这与 abstract 关键字矛盾。

面向对象 3 大特性；

封装继承多态

线程基本概念；

线程：是进程的执行单元，是进程中正在执行的子任务。

每个 Java 的应用程序运行的时候其实就是个进程，JVM 启动之后，会创建一些进行自身常规管理的线程，如**垃圾回收**和终结管理，和一个**运行 main 函数的主线程**。

Java 定义了 6 种线程状态。

NEW：还没调用 start 方法的线程

RUNNABLE：就绪（READY）+运行（RUNNING）

BLOCKED：阻塞，等待锁

WAITING：比如服务器的 accept，等待另一线程的特定操作（比如 notify）

TIMED_WAITING：具有指定等待时间的等待状态

TERMINATED：线程完成执行，终止

什么是数据流；

数据流将“基本数据类型与字符串类型”作为数据源，从而允许程序以与机器无关的方式从底层输入输出流中操作 Java 基本数据类型与字符串类型。

DataInputStream 和 DataOutputStream 提供了可以存取与机器无关的所有 Java 基础类型数据(如：int、double、String 等)的方法。
DataInputStream 和 DataOutputStream 是处理流，可以对其他节点流或处理流进行包装，增加一些更灵活、更高效的功能。只针对字节流(二进制文件)。

java 有什么数据类型，数据类型变量占多少位；

Byte: 1B -> 8bit
Short: 2B
Int: 4B
Long: 8B
Boolean: 1B
Float: 4B
Double: 8B
Char: 2B (Java 设计时支持 unicode 的 BMP 字符)

匿名内部类;

上面讲了。

不同类型的 I/O 流;

https://blog.csdn.net/qq_44715943/article/details/116501936

1. 按流的方向分为：输入流和输出流；
2. 按流的数据单位不同分为：字节流和字符流；
3. 按流的功能不同分为：节点流和处理流。

串行化机制;

https://www.cnblogs.com/xh0102/p/5759803.html

即序列化（Serialization）。

对象的寿命通常随着生成该对象的程序的终止而终止。有时候，可能需要将对象的状态保存下来，在需要时再将对象恢复。我们把对象的这种能记录自己的状态以便将来再生的能力。叫作对象的持续性(persistence)。对象通过写出描述自己状态的数值来记录自己，这个过程叫对象的串行化(Serialization)。串行化的主要任务是写出对象实例变量的数值。如果变量是另一对象的引用，则引用的对象也要串行化。这个过程是递归的，串行化可能要涉及一个复杂树结构的单行化，包括原有对象、对象的对象、对象的对象的对象等等。对象所有权的层次结构称为图表(graph)。

网络通信核心等;

Socket:

服务端:

```
import java.net.*;
import java.io.*;

public class Server {
    public static void main(String[] args) throws Exception{
        ServerSocket serverSocket = new ServerSocket(8888);
        Socket socket = serverSocket.accept();
        BufferedReader in = new BufferedReader(new
```

```
InputStreamReader(socket.getInputStream()));
        PrintWriter out = new PrintWriter(socket.getOutputStream());
        String line;
        while ((line = in.readLine()) != null) {
            System.out.println(line);
            out.println("服务器收到消息: " + line);
            out.flush();
        }
        socket.close();
    }
}
```

客户端：

```
import java.net.*;
import java.io.*;

public class Client {
    public static void main(String[] args) throws IOException {
        Socket socket = new Socket("localhost", 8888);
        BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
        PrintWriter out = new PrintWriter(socket.getOutputStream());
        out.println("Hello, Server!");
        out.flush();
        String line;
        while ((line = in.readLine()) != null) {
            System.out.println(line);
        }
        socket.close();
    }
}
```

容器；

Java 容器分为 Collection 和 Map 两大类，其下又有很多子类。

- Collection
- List
    - ArrayList
    - LinkedList
    - Vector
    - Stack

- Set

    - HashSet
    - LinkedHashSet
    - TreeSet

- Map
- HashMap
    - LinkedHashMap

- TreeMap
- ConcurrentHashMap
- Hashtable

数据容器主要分为了两类：

Collection: 存放独立元素的序列。

Map：存放 key-value 型的元素对。（这对于需要利用 key 查找 value 的程序十分的重要!）

https://blog.csdn.net/zhangqunshuai/article/details/80660974

静态和非静态；

静态方法：方法用 static 关键字修饰，静态方法与静态成员变量一样，属于类本身，在类装载的时候被装载到内存，不自动进行销毁，会一直存在于内存中，直到 JVM 关闭。使用时也是不需要实例化类，能够直接使用。**静态方法无法被重写。**

在静态方法中只能访问类中的静态成员跟静态方法，不能直接访问类中的实例变量跟实例方法，原因是静态方法在 JVM 中的加载顺序也在对象之前，直接使用实例变量跟实例方法的话，可能实例变量跟实例方法所依附的对象并没有被创建，会导致无法找到所使用的实例变量跟实例方法。
要想使用实例变量跟实例方法可以采用如下方法：在静态方法中创建实例变量和实例方法所在的对象，通过这个对象来使用实例变量跟实例方法。

https://blog.csdn.net/HaydenYu/article/details/73457278

java 编译；

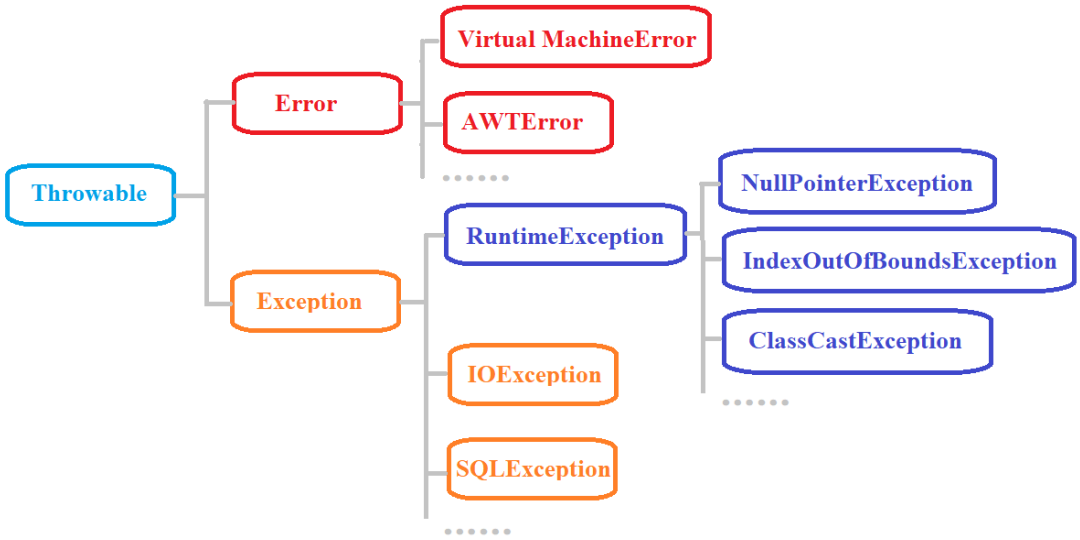
javac demo.java 能够将 java 源文件编译成.class 字节码文件
java demo 能够运行字节码文件，由 JVM 对字节码进行解释和运行

创建类实例时，编译三部曲：首先加载还没有加载解析过的类，在堆区分配对象所需的内存，包括本类和父类的所有实例变量，不包括静态变量；
然后对所有实例变量赋初始值；

最后执行实例初始化代码，先父类再子类，初始化时先执行实例代码块然后是构造方法。

简答题：

异常处理，例如异常类型，运行时异常，非运行时异常等；



包括系统错误、编译时异常（非运行时异常）、运行时异常。

运行时异常，包括空指针异常 NullPointerException、数组下标越界异常 IndexOutOfBoundException、类型转换异常 ClassCastException、数组存储异常（操作数组时类型不一致）ArrayStoreException……

非运行时异常，或者叫编译异常，可查异常。比如 IO 异常 IOException、SQL 异常、类没有找到异常 ClassNotFoundException……IDE 一般会自动爆红，开发者可以用 try{}catch(){}捕获，或者 throws 关键字抛出，如果是 main 方法，则会直接由 JVM 处理——中断运行并打印异常信息。

运行时异常直接由 JVM 进行处理，处理机制为中断运行并打印异常信息。

非运行时异常由程序设计者进行处理，处理机制为 try{}catch(){}或者 throws。

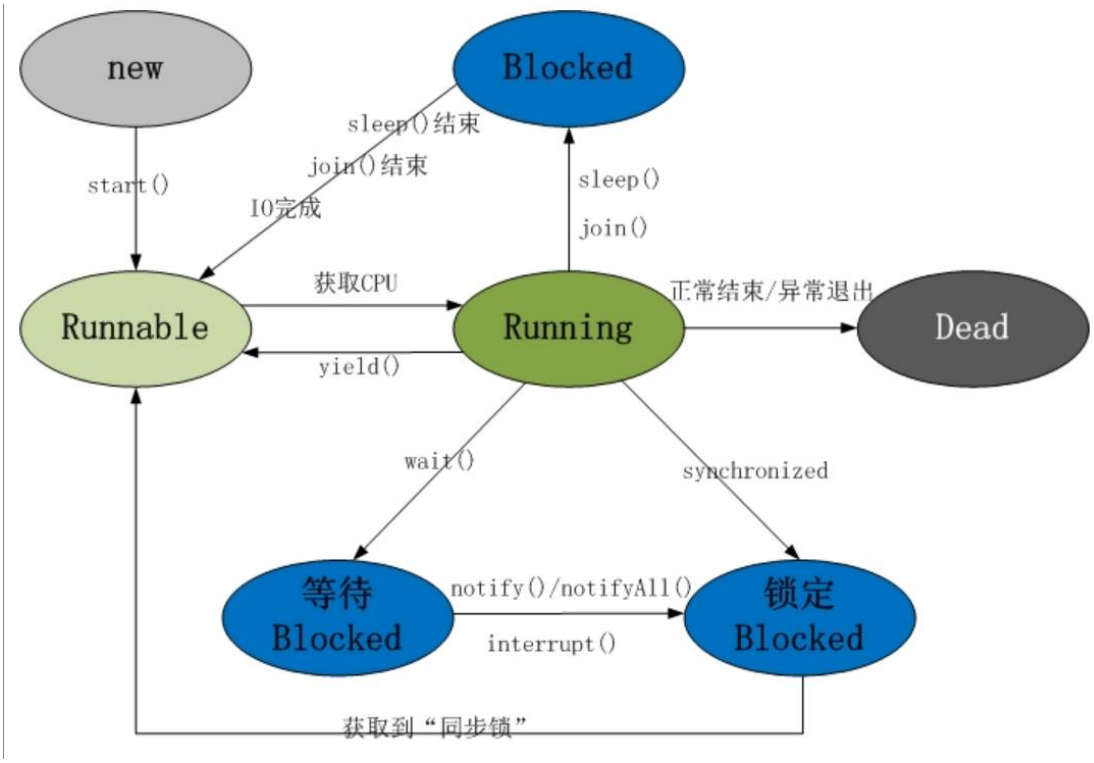
线程，例如线程的 5 个运行状态，线程互斥，线程同步；

线程：是进程的执行单元，是进程中正在执行的子任务。

划分为 6 种是根据 java，5 种是 OS。

① 新建 New；
② 就绪 Runnable；
③ 运行 Running；
④ 阻塞 Blocked；
⑤ 死亡 Dead

每个 Java 的应用程序运行的时候其实就是个进程，JVM 启动之后，会创建一些进行自身常规管理的线程，如**垃圾回收**和终结管理，和一个**运行 main 函数的主线程**。



线程互斥：当多个线程需要访问同一资源时，要求在一个时间段内只能允许一个线程来操作共享资源，操作完毕后别的线程才能读取该资源。

线程同步：如果一个线程调用了某个对象的 synchronized 方法，它在这个方法运行完之前不会被别的线程打断。

解释：

互斥解决了「多进程/线程」对临界区使用的问题，但是它没有解决「多进程/线程」协同工作的问题。所谓同步，就是「多进程/线程间」在一些关键点上可能需要互相等待与互通消息，这种相互制约的等待与互通信息称为「进程/线程」同步。

互斥：某一资源同时只允许一个访问者对其进行访问，具有性和排它性。但互斥无法限制访问者对资源的访问顺序，即访问是无序的。

「操作 A 和操作 B 不能在同一时刻执行」

同步：互斥的基础上，通过其它机制实现访问者对资源的有序访问。在大多数情况下，同步已经实现了互斥。

「操作 A 应在操作 B 之前执行」,「操作 C 必须在操作 A 和操作 B 都完成之后才能执行」

显然，同步是一种更为复杂的互斥，而互斥是一种特殊的同步。也就是说互斥是两个线程之间不可以同时运行，他们会相互排斥，必须等待一个线程运行完毕，另一个才能运行，而同步也是不能同时运行，但他是必须要按照某种次序来运行相应的线程(也是一种互斥)！

构建类的实例时，编译器的 3 个步骤等；

首先加载还没有加载解析过的类，在堆区分配对象所需的内存，包括本类和父类的所有实例变量，不包括静态变量；
然后对所有实例变量赋初始值；
最后执行实例初始化代码，先父类再子类，初始化时先执行实例代码块然后是构造方法。

编码题：

文件处理，包括文件建立，更新，复制等；

以下是一些贴士。

`File` 类有许多有用的方法来创建和获取有关文件的信息。 例如：

方法	类型	描述
`canRead()`	Boolean	测试文件是否可读
`canWrite()`	Boolean	测试文件是否可写
`createNewFile()`	Boolean	创建一个空文件
`delete()`	Boolean	删除文件
`exists()`	Boolean	测试文件是否存在
`getName()`	String	返回文件名
`getAbsolutePath()`	String	返回文件的绝对路径名
`length()`	Long	返回文件的大小（以字节为单位）
`list()`	String[]	返回目录中文件的数组
`mkdir()`	Boolean	创建目录

① Import java.io.*;

② try{创建文件等文件操作}catch(IOException e){e.printStackTree();return false}

③ 创建文件

```
File file = new File(filepath);
file.exists(); // 判断文件是否存在 true or false
```

```
file.createNewFile() // 创建目标文件 false 时失败
```

④ 写文件

```
try {
      FileWriter myWriter = new FileWriter("filename.txt");
      myWriter.write("Files in Java might be tricky, but it is fun
enough!");
      myWriter.close();
      System.out.println("Successfully wrote to the file.");
    } catch (IOException e) {
      System.out.println("An error occurred.");
      e.printStackTrace();
    }
```

⑤ 读文件

```
FileReader reader = new FileReader(file);
char[] ch = new char[100];
reader.read(ch);
for(char c:ch) {
    System.out.print(c);
}
System.out.println();
reader.close();
```

⑥ 删除文件

用 File 实例的.delete()

⑦ 文件复制/更新

总的来讲就是将文件用一个 FileReader 读取然后用 FileWriter 写到别的地方。

Reader 会获取文件的所有信息，然后你可以用 BufferedReader reader = new BufferedReader(new FileReader(file));接收。

然后用 reader 的 readLine()方法,用 while 循环逐行读取到字符串变量中，每读一行，用 FileWriter 去写到另一个文件里。

或者说，用一个 StringBuilder（字符串数组）通过 append 接收，最后对其 toString 后一次性写到另一个文件里。Filewriter 记得 flush

对于更新操作，可以修改 StringBuilder 的 toString 后的内容，用一个 String 接收，然后把这个 String 写到文件里。

※ new FileWriter(路径, 是否为追加)，所以如果要追加性写文件，可以把第二个参数设为 true。

内部类；

```java
public class Client {
    7 个用法
    public static class GroupThree{
        5 个用法
        private static int count;
        0 个用法
        private String name;
        2 个用法
        public class Student{
            2 个用法
            private int count;
            0 个用法
            private String name;
            1 个用法
            public void Output(int count){
                count++;
                this.count++; // this为Student的count
                GroupThree.count++; // 这个count是主类静态的
                GroupThree.this.count++; // 这个还是主类静态的
                System.out.println(count+" "+this.count+" "+GroupThree.count+" "+GroupThree.this.count++);
            }
        }
    }
    public static void main(String[] args){
        GroupThree g3 = new GroupThree();
        g3.count = 10;
        GroupThree.Student s1 = g3.new Student();
        s1.Output( count: 5);
    }
}
```

成员内部类：

```
public class Outer {
    public class Inner{
        // do something…
    }
}
```

可以被 public、private 等权限修饰符修饰；
Outer.this.xxx 可以访问外部类的变量；
内部类可以随意访问外部类的任何成员；
不可以定义 static 成员。

```
Outer outer=new Outer();
Outer.Inner inner=outer.new Inner();
```

局部内部类：

定义在方法或作用域内，和成员内部类的区别仅在于访问权限的不同。

```
public class Outer{
    public void test(){
            class Inner{
                    // do something…
            }
     }
}
```

局部内部类不能有访问权限修饰符，不能定义为 static 和定义 static 成员，默认包含了外部类对象的引用，也能用 Outer.this 访问外部类成员。局部内部类想要使用方法或域中的变量，该变量必须是 final 的。

匿名内部类

```
public class Outer{
    public List list=new ArrayList(){
        {
           add("test");
        }
    };
}
```

匿名内部类使用单独的块表示初始化块{}

匿名内部类想要使用方法或域中的变量,该变量必须是final修饰的,JDK1.8之后effectively final也可以

匿名内部类默认包含了外部类对象的引用

匿名内部类表示继承所依赖的类

嵌套类

是用static修饰的成员内部类。

```
public class Outer {
    public static class Inner{
           // do something…
    }
}
```

唯一不包含外部类对象引用的内部类，可以定义 static 成员，本质和一个外部类一样，只是方便管理罢了。

线程，例如线程互斥，线程同步等；

https://cloud.tencent.com/developer/article/2315701

https://blog.csdn.net/qq_44823756/article/details/120925364

```
package test.MyThread.ticketDemo;

public class RunnableThread implements Runnable{
    private int ticket = 100;
    Object obj = new Object();
    @Override
    public void run(){
        while(true){
            synchronized (obj) {
                if (ticket > 0) {
                    try {
                        Thread.sleep(100);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }

                    System.out.println(Thread.currentThread().getName()
```

```
+ "正在出售第 " + ticket + " 张票");
                    ticket--;
                }
            }
        }
    }
}
```

网络编程（包括多线程）。

Socket:

服务端：

```
import java.net.*;
import java.io.*;

public class Server {
    public static void main(String[] args) throws Exception{
        ServerSocket serverSocket = new ServerSocket(8888);
        Socket socket = serverSocket.accept();
        BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
        PrintWriter out = new PrintWriter(socket.getOutputStream());
        String line;
        while ((line = in.readLine()) != null) {
            System.out.println(line);
            out.println("服务器收到消息：" + line);
            out.flush();
        }
        socket.close();
    }
}
```

客户端：

```
import java.net.*;
import java.io.*;

public class Client {
    public static void main(String[] args) throws IOException {
        Socket socket = new Socket("localhost", 8888);
        BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
        PrintWriter out = new PrintWriter(socket.getOutputStream());
        out.println("Hello, Server!");
        out.flush();
        String line;
        while ((line = in.readLine()) != null) {
            System.out.println(line);
```

```
        }
        socket.close();
    }
}
```

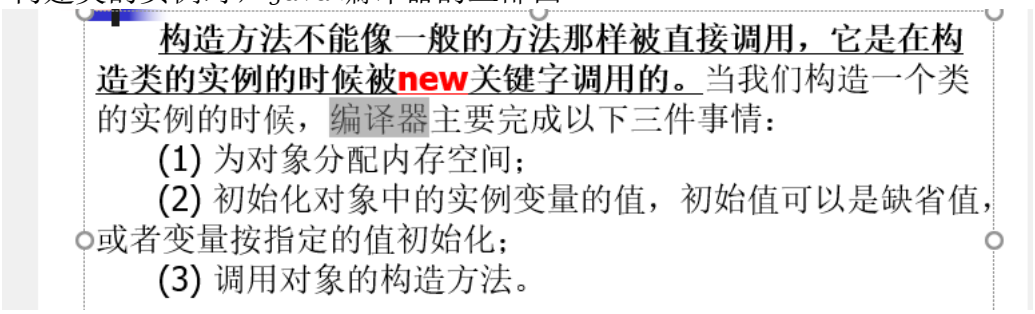
可组合，可分开

其他题型：

1、面向对象三大特征；
封装、继承、多态

2、Java 在 JDK 中的编译命令及其编译对象，运行命令及其运行对象；
编译命令：javac xx.java
编译对象：java 文件
运行命令：java xx
运行对象：class 文件

3、构建类的实例时，java 编译器的三部曲



4、Java 的源代码格式；
第一行是 package 语句，接着是 import 语句，然后定义类

5、构造器
构造器的名字必须与类名相同
构造器没有返回类型
构造器可以重载
构造器调用构造器 this 问题 一个类中，一个构造器调用另一个构造器，用关键词 this.

6、Java 的赋值，
对象间的赋值；
方法的形式参数（或叫局部变量）为对象时，在方法内对对象赋值；

7、基本的操作符，+,-,*,/,=, ==, !=等（注意 String 类的+, +=）；
(1) 二元算术运算符:运算结果的数据类型一般为两个操作数中表达范围较大的类型。e.g.整数 op 浮点数 → 浮点数；
(2) java 的==：基本数据类型比较值，对象类型比较地址。注意 Integer 的-128~127、String 的字符串常量池问题；
(3)String 类的+：当操作数中有一个是字符串时，程序运行时会对另一个操作数进行字符串转换。注意+的左结合性（运算先后顺序）。e.g.1+2+"=3"→"3=3"；"12="+1+2→"12=12"。

8、Import, package 语法

import：在 package 语句（若有）之后，分单类型导入（仅一个 public 类或接口，e.g.import java.io.File）和按需类型导入（导入某包下所有当前需使用的类，不包括子目录，可能会因同名类造成命名冲突，e.g.import java.io.*)。
java.lang 下所有 public 类自动导入；导入静态成员
e.g.import static java.lang.Math.max;

（2）package：包声明，在源文件非注释的第一行，最多一个（如不使用，将被放在无名包中)。

9、覆盖+重载

覆盖：

子类对父类方法的重写

子类方法和父类方法同名同参同返回类型

重载：

同一个类中建立多个同名方法

重载的方法同名不同参

10、 初始化（重点）

注意，static 定义的变量或方法，可以不用声明类对象，就可以直接调用，但是要注意初始化顺序；

11、 Java 的访问权限控制，关键词，权限大小，适用范围

类的访问权限控制和 import 的联系

一定要注意注意注意，从访问权限上来说

不是 public > protected > package > private

因为 protected 和 package 是继承才有区别。注意注意。

12、 抽象类、接口的定义，特点，注意事项等

13、 前期绑定+后期绑定

绑定：将一个方法调用同另一个方法主题关联起来

（1）前期绑定（静态绑定)：在程序执行之前，final、static、private、构造方法、成员变量（静态和非静态）

（2）后期绑定（动态绑定)：在运行时根据对象的类型，通过多态实现

14、 static、final 的概念，适用范围，注意事项等

（1） static 静态修饰关键字，可以修饰静态变量、类方法和静态内部类

注意：静态方法不能被重写，静态方法只能调用静态的东西

静态块在类被加载时执行且只执行一次

（2）final 可以修饰常量、类和方法

注意：被 final 修饰的类不能被继承，被修饰的方法不能重写

final 修饰的常量需要在声明时初始化或构造函数中初始化

15、 各种容器的基本概念，特点；list, set，map，队列

（List 接口存储一组不唯一，有序（插入顺序）的对象。 Set 接口存储一组唯一，无序的对象。 Map 接口存储一组键值对象，提供 key 到 value 的映射。key 无序，唯一。value 不要求有序，允许重复。）

16、 运行异常，try-catch-finally 运行原理。

（try｛//执行的代码，其中可能有异常。一旦发现异常，则立即跳到 catch 执行。否则不会执行 catch 里面的内容 ｝ catch ｛ //除非 try 里面执

行代码发生了异常，否则这里的代码不会执行 }　finally { //不管什么情况都会执行，包括 try catch 里面用了 return ,可以理解为只要执行了 try 或者 catch，就一定会执行 finally }

在 try 和 catch 中如果要 return，会先去执行 finally 中的内容再返回。）

17、 管道【查一下】

Java 提供管道功能，实现管道通信的类有两组：PipedInputStream 和 PipedOutputStream 或者是 PipedReader 和 PipedWriter。管道通信主要用于不同线程间的通信。

一个 PipedInputStream 实例对象必须和一个 PipedOutputStream 实例对象进行连接而产生一个通信管道。PipedOutputStream 向管道中写入数据，PipedIntputStream 读取 PipedOutputStream 向管道中写入的数据。一个线程的 PipedInputStream 对象能够从另外一个线程的 PipedOutputStream 对象中读取数据

18、 线程，进程（线程运行的 5 个状态）

一个进程就是一个执行中的程序，而每一个进程都有自己独立的一块内存空间、一组系统资源。在进程概念中，每一个进程的内部数据和状态都是完全独立的。Java 程序通过流控制来执行程序流，程序中单个顺序的流控制称为线程，多线程则指的是在单个程序中可以同时运行多个不同的线程，执行不同的任务。

19、 局部内部类的声明对象，调用（重点）

局部内部类不能被访问修饰符和 static 修饰，且只能访问 final 变量和形参。

1. 通过 new 一个对象来调用
   Inner in = new Inner ();
2. 使用匿名对象来调用（这个调用的只能用一次而且创建多了浪费资源）
   new Inner ().inner ();

20、 匿名内部类和嵌套类的概念和注意事项

匿名内部类：

只创建这个类的一个对象，不用为它命名。在定义类的同时，就生成该类的一个实例，并且不会在其他地方听到这个类。

用于构造对象的任何参数都要被放在超类名后面的括号内。

匿名内部类不能有构造器。

匿名内部类既可以拓展类，也可以拓展接口。同时只能且必须实现一个类或者是一个接口。

匿名内部类是局部内部类的一种。

嵌套类：

在一个类中定义另外一个类。

嵌套类的范围受其封闭类的范围限制。

嵌套类可以访问封闭类的成员，包括私有成员。

嵌套类可以被声明为 private public protected 或 package private。

内部类是非静态嵌套类。

21、 Inputstream, outputstream, fileinputstream, fileoutputst

InputStream 和 OutputStream 都是抽象类，不能实例化，因此在实际应用中都使用的是他们的子类。Java 通过系统类 System 实现标准输入输出的功能，定义了 3 个流变量，in，out 和 err。System.in 作为字节输入流类 InputStream 的对象实现标准输入。System.out 作为打印流类 PrintStream 的对象实现标准输出。

FileInputStream 和 FileOutputStream 用于进行文件的输入输出处理，其数据源和接收器都是文件。 FileInputStream 用于顺序访问本地文件，FileInputStream 重写了抽象类 InputStream 的读取数据的方法。FileOutputStream 用于向一个文本文件写数据，FileOutputStream 重写了抽象类 OutputStream 的写数据的方法。

22、 stream 概念

Stream 是 java 的 1 个类, 这个类专门用于程序和外部设备的输入输出(IO). 基本上所有流都在 java.io 这个包中.

实际上 Stream 就是数据在程序和外部设备的单向管道, 流的各种方法相当于管道上的各种按钮.

23、 互斥对象的概念和使用

1 为某个对象设置一个“互斥锁”标记。该标记保证在某一个时刻，只能有一个线程拥有该互斥锁，其它线程如果需要获得该互斥锁，必须等待当前拥有该锁的线程将其释放。该对象称为互斥对象。

2 为了配合使用对象的互斥锁，Java 语言提供了保留字 synchronized。其基本用法如下：

```
synchronized(互斥对象){
        临界代码段
}
```

24、 线程的几种状态（5 种）

创建态，就绪态，运行态，阻塞态，死亡状态

25、 Java 数据类型的几类

基本数据类型：整数类型、浮点类型、字符类型、布尔类型

引用数据类型：类、接口、数组

26、 临界资源或同步资源概念

临界资源：

在并发程序设计中，对多线程共享的资源或数据称为临界资源，而把每个线（进）程中访问临界资源的那一段代码段成为临界代码段。通过为临界代码段设置信号灯，就可以保证资源的完整性，从而安全地访问共享资源 。ppt10.27

许多物理设备都属于临界资源，如打印机等。此外，还有许多变量、数据等都可以被若干进程共享，也属于临界资源。

27、 网络通信核心

协议

28、 java 几种类型变量所占位数

byte 8 short 16 int 32 long 64 float 32 double 64 char 16

29、 向上转型 mklm

1）上转型对象不能操作子类新增的成员变量和方法。

2）上转型对象可以操作子类继承或重写的成员变量和方法。

3）如果子类重写了父类的某个方法后,当对象的上转型对象调用这个方法时一定是调用了这个重写的方法

30、 内部类

可以将一个类的定义放在里另一个类的内部，这就是内部类。广义上我们将内部类分为四种：成员内部类、静态内部类、局部（方法）内部类、匿名内部类。