1. 装箱、拆箱：基本数据类型和包装类之间的转换
    - 基本数据类型转化为包装类就是装箱
    - 包装类转化为基本数据类型就是拆箱
    - 包装类就是引用类型，基本类型就是值类型
2. Java并发中：
    - CopyOnWriteArrayList适用于写少读多的并发场景
    - ReadWriteLock为读写锁，要求写与写之间互斥、读与写之间互斥、读与读之间可以并发执行。在读多写少的情况下可以提高效率。
    - ConcurrentHashMap是同步的HashMap，读和写都加锁
    - Volatile只保证多线程操作的可见性，并不能保证其原子性
3. 下面那些情况可以终止当前线程的运行?
    - 当一个优先级高的线程进入就绪状态时
        - 实际上当一个优先级高的线程进入就绪状态的时候，它只是有较高的概率能够抢到CPU的执行权，并非一定可以抢到执行权
    - 抛出一个异常时
        - 实际上，抛出异常后，线程就终止了
    - 当该线程调用sleep()方法时
        - 当线程调用sleep()或者wait()方法时，只是暂时停止了该线程的运行，并非终止线程。
    - 当创建一个新线程时
        - 创建一个新的线程时，该线程也加入到了抢占cpu执行权的队伍中，但是是否能够抢到并不确定
    - 实际上，能够结束线程的三个原因是：
        - run()方法执行完成，线程正常结束
        - 线程抛出一个未捕获的Exception或者Error
        - 直接调用该线程的Stop()方法结束线程——不建议使用，容易死锁
4. Java鲁棒性：
    - Java在编译和运行程序的时候，都要对可能出现的问题进行检查，以消除错误的产生
    - 它提供自动垃圾收集来进行内存管理，防止程序员在管理内存时容易产生的错误
    - 通过集成的面向对象的例外处理机制，在编译的时候，Java揭示可能出现但是未被处理的例外，帮助程序员正确地进行选择以防止系统的崩溃
    - Java在编译的时候可以捕获类型生命中的许多常见错误，防止动态运行时不匹配问题的出现。
5. Java中补码的形式，并非原码，第一位表示正负，1表示负，0表示正
    - 原码一个数的二进制表示
        - 3的原码是00000011，-3的原码是10000011
    - 反码：负数原码按位取反，符号位不变，正数原码本身
        - 3的反码是00000011，-3的反码是11111100

- 补码：正数是原码本身，负数反码加一
  - 3的补码是00000011，-3的补码是11111101
- 在内存中表示：因为int占4个字节，32位；byte占一个字节，8位
  - int a = 3: 00000000 00000000 00000000 00000011 （强制转型的时候，前24位0被截断）
  - byte b = 3： 00000011
  - int a = -3：11111111 11111111 11111111 11111101
  - byte b = -3：11111101

6. Java标识符：由数字、字母、下划线、美元符号组成，其首位不能是数字，且Java关键字不能作为标识符
7. 关于Java异常处理：
   - throws用在方法上，声明该方法不需要处理的异常类型，其后跟着异常类名，可以是多个异常类
   - throw用在方法内，用于抛出具体异常类的对象，后面跟的异常对象只能是一个异常类型实体
   - try块必须和catch块或者finally同在，不可以单独存在，catch和finally必有其一
   - finally块总会执行，无论是否有错误出现，但是如果try语句块或者会执行的catch语句块中是用来JVM系统退出语句，那么finally块无法执行
   - 一般豆浆关闭资源的代码放在finally里，保证资源总是能够被正确关闭
8. Java中，构造函数不能够被继承，只能够被显式或者隐式地调用。
9. 在jdk1.5的环境下，有如下4条语句：

```
Integer i01 = 59;
int i02 = 59;
Integer i03 = Integer.valuOf(59);
Integer i04 = new Integer(59);
System.out.println(i01 == i02);
System.out.println(i01 == i03);
System.out.println(i03 == i04); // false
System.out.println(i02 == i04); //i02为基本数据类型，比较的时候比较的是数值
// 因为：JVM中，一个字节以下的整型数据会在JVM启动的时候加载进入内存，除非使用new Integer()来显式地创建对象，否则都是同一个对象。
```

10. Java对象的初始化方式有：初始化块、构造器、定义变量时指定初始化值
11. byte + byte = int，低级向高级是隐式类型转换，高级向低级必须强制类型转换，byte < char < short < int < long < float < double
12. 以下代码的结果：

```
int i = 5;
int j = 10;
System.out.println(i + ~j);

// 10的原码是：00000000 00000000 00000000 00001010
// ~10的结果是：11111111 11111111 11111111 11110101 因为是负数，计算机用补码来存储
// ~10的反码是：10000000 00000000 00000000 00001010
// ~10的补码是：10000000 00000000 00000000 00001011 等于-11
// 所以结果为-6

//已知负数的补码，求负数：
// 补码 - 1 = 反码，反码按位取反 = 该负数绝对值
// 已知负数，求负数的补码：
// 1、负数原码除了符号位，按位取反（不含符号位），加1。
// 2、负数绝对值的补码（也就是原码），按位取反（含符号位），加1
```

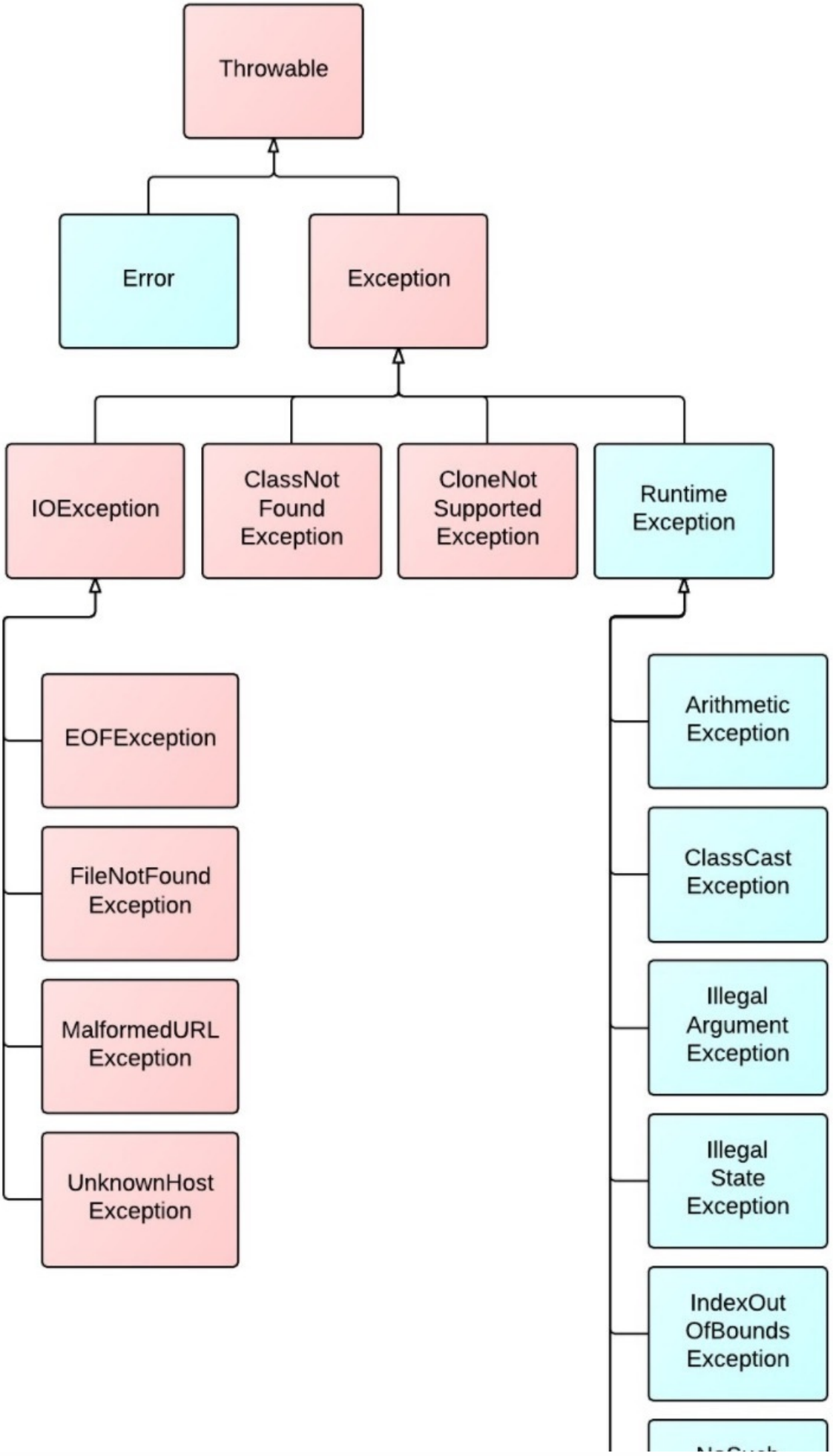
13. 关于Volatile
    - 一旦一个共享变量（类的成员变量、类的静态成员变量）被volatile修饰之后，就具备了两层语义:
      - 保证了不同线程对这个变量操作时的可见性，即一个线程修改了某个变量的值，这个新的值对于其他线程来说是立即可见的
      - 禁止进行指令重排序优化
    - volatile只提供了保证访问该变量的时候，每次都是从内存中读取最新的值，并不会使用寄存器缓存该值（每次都从内存中读取）。而对于该变量的修改，volatile本身并不提供原子性的保证
    - 由于有些时候对于volatile的操作，并不会被保存，说明不会造成阻塞
    - 多线程下，计数器必须使用锁来保护
14. Java提供了一个系统级的线程，即垃圾回收器线程，用来对每一个分配出去的内存空间进行跟踪。当JVM空闲的时候，自动回收每一块可能被回收的内存，GC完全自动，不能被强制执行。程序员最多只能通过System.gc()来建议执行垃圾回收器来回收内存，但是具体的回收时间并不可知。当对象的引用变量被赋值为null的时候，可能会被当成垃圾。对于局部变量而言，其位于栈，而栈上的垃圾回收由finalize()来实现，而非GC，因为GC适用于堆。
15. 在调用子类构造器之前，会先调用父类构造器，当子类构造器中没有使用super(参数或者无参数)来显式地指定调用父类构造器的时候，是默认调用父类的无参构造器，如果父类中含有带参构造器，却没有无参构造器，那么在子类构造器中一定要使用super(参数)来指定调用父类的带参构造器，负责会报错。
16. Servlet：
    - getParameter()获取POST/GET传递的参数值
    - getInitParameter获取Tomcat的server.xml中设置Context的初始化参数
    - getAttribute()是获取对象容器中的数据值
    - getRequestDispatcher请求转发

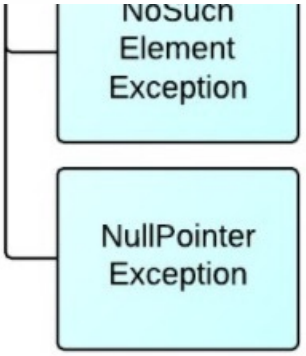
17. ArrayList和LinkedList
    - ArrayList是实现了基于动态数组的数据结构，LinkedLists是基于链表的数据结构：这里的动态数组并非是“有多少元素就申请多少空间”，而是，如果没有指定数组的大小，则默认申请大小为10的数组，当元素的个数增加，数组无法存储的时候，系统就会另外再申请一个长度为当前长度1.5倍的数组，然后，将之前的数据拷贝到新建的数组
    - 对于随机访问get和set，ArrayList觉得优先于LinkedList，因为Linkedlist需要移动指针：ArrayList是数组，所以，直接定位到相应的位置来获取元素，而LinkedList是链表，需要从前往后遍历
    - 对于新增和删除操作，add和remove，LinkedList比较占优势，因为ArrayList需要移动数据：ArrayList的新增和删除，就是数组的新增和删除，而LinkedList与链表一致
    - ArrayList的空间浪费主要体现在list列表的结尾需要预留一定的容量空间，而LinkedList的空间花费则体现在他们的每一个元素都需要消耗相当的空间：因为ArrayList空间的增长率为1.5倍，那么，最后很可能会预留一部分空间没有被用到，继而造成浪费，对于LinkedList，犹豫每一个节点都需要额外的指针，所以导致空间的浪费。
18. Java中不允许单独的方法、过程或者函数存在，必须要隶属于某一类中； Java语言中的方法属于对象的成员，而不是类的成员，不过，其中静态方法属于类的成员。
19. 接口是对一类事物的属性和行为更高层次的抽象，对修改关闭，对拓展开放，接口是对开闭原则的一种体现，所以接口的属性用public static final修饰。
20. 关于JVM内存配置参数：
    - -Xmx: 最大堆大小
    - -Xms: 初始堆大小也是最小内存值
    - -Xmn: 年轻代大小
    - -XXSurvivorRatio: 年轻代中Eden区与Survivor区的大小比值
21. JSP
    - JSP分页代码中，先取得总记录数，得到总页数，再取得所有的记录，最后显示本页的数据
22. Java的关键字包括：
    - abstract, assert, boolean, byte, break, case, catch, char, class, const, continue, default, do, double, else, extends, enum, final, finally, float, for, goto, if, implements, import, instanceof, int, interface, long, native, new, package, private, protected, public, return, short, static, strictfp, super, switch, synchronized, this, throw, throws, translent, try, void, volatile, while
    - 保留字: true, false, null
23. 关于Java文件
    - Java.exe是Java虚拟机
    - javadoc.exe用于制作java文档
    - jdb.exe是java的调试器
    - javaprof.exe是剖析工具
24. 面向对象的五大基本原则（solid）
    - 单一职责原则（SRP）：一个类，最好只做一件事情，只有一个引起它的变化。可以看做是低耦合、高内聚在面向对象原则上的引申，将职责定义为引起变化的原因，以提高内聚性来减少引起变化的原因
    - 开放封闭原则（OCP）：软件实体应该是可拓展的、而不可修改的，也就是说，对拓展开放，对修改封闭

- 里氏Liskov替换原则（LSP）：子类必须能够替换其基类。体现为对继承机制的约束规范，只有子类能够替换基类时，才能保证系统在运行期内识别子类，这是保证继承复用的基础
- 依赖倒置原则（DIP）：依赖于抽象。具体而言为高层模块不依赖于底层模块，二者都同依赖于抽象；抽象不依赖于具体，具体依赖于抽象
- 接口隔离原则（ISP）：使用多个小的专门的接口，而不是使用一个大的总接口

25. 常量对象不能修改，但是静态成员变量需要初始化，并且可以修改（例如常常利用静态成员变量统计某个函数的调用次数）。
26. 类的初始化顺序：
    - 初始化父类中的静态成员变量和静态代码块
    - 初始化子类中的静态成员变量和静态代码块
    - 初始化父类中的普通成员变量和代码块，再执行父类中的构造方法
    - 初始化子类中的普通成员变量和代码块，再执行子类中的构造方法
27. 方法的重写（override）两同、两小、一大原则：
    - 方法名相同，参数类型相同
    - 子类返回类型小于等于父类方法返回类型
    - 子类抛出异常小于等于父类方法抛出异常
    - 子类访问权限大于等于父类方法访问权限
28. Java标识符的命名规范：
    - 只能包含字母a-zA-z，数字0-9，下划线_和美元符号$
    - 首字母不能为数字
    - 关键字和保留字不能作为标识符
29. 泛型：
    - 创建泛型对象的时候，一定要指出类型变量T的具体类型。争取让编译器检查出错误，而不是留给JVM运行的时候抛出类不匹配的异常
    - JVM如何理解泛型概念——类型擦除。事实上，JVM并不知道泛型，所有的泛型在编译阶段就已经被处理成了普通类和方法。处理方法很简单，叫做类型变量T的擦除(erased)
    - JVM中没有泛型，只有普通类和方法；在编译阶段，所有泛型类的类型参数都会被Object或者它们的限定边界来替换（类型擦除）；在继承泛型类型的时候，桥方法的合成是为了避免类型变量擦除带来的多态灾难。无论我们如何定义一个泛型类型，相应的都会有一个原始类型被自动提供，原始类型的名字就是擦除类型参数的泛型类型的名字
30. 一个Java文件可以包含多个Java类，但是只能包含一个public类，并且public类的类名必须与Java文件同名。
31. 对于值传递，拷贝的值用完之后就会被释放，对原值没有任何影响，但是对于引用传递，拷贝的是对象的引用，和原值指向的是同一个地址，即操作的是同一个对象，所以操作之间会互相影响。对于String而言，操作之间互不影响，原值保持不变，而对于数组而言，拷贝的是对象的引用，值发生了改变。
32. sleep和wait的区别:
    - 这两个方法来自不同的类，Object.wait()和Thread.sleep()
    - sleep方法没有释放锁，而wait方法释放了锁，是的敏感线程可以使用同步控制块或者方法
    - wait, notify和notifyAll只能在同步控制方法或者同步控制块内使用，而sleep可以在任何地方使用
    - sleep必须捕获异常，而wait，notify，notifyAll都不需要捕获异常

33. Exceptions:





- 粉色的是受检查的异常(checked exceptions)，其必须被try-catch语句块所捕获，或者在方法签名里通过throws声明，这种异常需要在编译期得到捕获处理，Java编译器需要对其进行检查，Java虚拟机也要进行检查，以确保规则得到遵循
- 绿色的异常属于运行时异常(runtime exceptions)，需要程序员自己分析代码是否需要捕获和处理，如空指针、被0除
- 声明为Error的，是严重错误，比如系统崩溃、虚拟机错误、动态链接失败等，这些错误无法恢复或者不可能捕获，将直接导致应用程序中断，Error不需要捕捉

34. Collections