1. 类的生命周期
   1. 生命周期过程：加载->链接->初始化->使用->卸载
   2. 加载
      1. 过程：
         1. 通过一个类的全限定名来获取定义此类的二进制字节流。
         2. 讲这个字节流所代表的静态存储结构转化为方法去的运行时数据结构
         3. 在内存中生产一个代表这个类的java.lang.Class对象，作为方法区这个类的各种数据的访问入口。
   3. 连接
      1. 过程：
         1. 验证：检查一些错误，例如数组下标是否越界
         2. 准备：为类变量分配内存，并初始化普通类变量为“0”值，final类变量为其指定指
         3. 解析：将常量池内符号引用替换为直接引用
   4. 初始化：调用父类类构造器及本类构造器（不是调用实例构造器！！！）
      1. 对象初始化的过程
         1. 父类static（顺序按照代码书写顺序进行）
         2. 子类static（顺序按照代码书写顺序进行）
         3. 父类非static域、构造器（顺序按照代码书写顺序进行）
         4. 子类非static域、构造器（顺序按照代码书写顺序进行）
      2. 静态语句块中只能访问到定义在静态语句块之前的变量，定义在之后的变量只能赋值但不能访问
2. 类的加载时机
   1. 主动引用：只有主动引用才会触发类的加载
      1. New实例化对象时
      2. 访问一个类的非final静态字段时
      3. 调用类的静态方法时
      4. 初始化子类时发现父类还没有被初始化，需要先出发父类的初始化过程
      5. 虚拟机启动时，用户需要指定一个要执行的主类（包含main方法），虚拟机会先初始化主类
   2. 被动引用：被动引用不会触发类的加载
      1. 父类有一个static变量，通过子类访问父类的static变量时不会触发子类的加载。对于静态字段，只有直接定义这个字段的类才会被初始化，因此通过子类来引用父类中定义的静态字段时，只会触发父类的初始化而不会触发子类的初始化。
      2. 同过数组定义引用类不会触发类的加载
      3. 访问类的static final常量时，不会触发类的加载。因为常量在编译时会存入调用类的常量池中，本质上并没有直接引用到定义常量的类，因此不会除触发类的初始化。
3. 类加载器
   1. 定义：实现“通过一个类的全限定名获取一个类的二进制字节流“这个动作的代码模块。
   2. 类与类加载器的关系：就算是来源于同一个class文件，被同一个虚拟机加载，但只要被不同的类加载器加载这就是两个不同的类
   3. 类加载器分类
      1. 启动类加载器
      2. 扩展类加载器
      3. 应用程序类加载器
      4. 自定义类加载器
   4. 双亲委派模式
      1. 工作过程：如果一个类加载器收到一个加载类的请求，它首先不会自己加载这个类，而是讲这个请求委派给上一层次的类加载器。循环这个过程。如果顶层类加载器不能加载这个类，再让第一层次的类加载器加载。
      2. 双亲委派模式存在的原因：
         1. 如果你自己写一个类名字叫Object、他用你自己的类加载器加载，会发生什么？所以通过双亲委派模式，可以防止这种混乱。因为顶层的启动类加

载器可以加载rt.jar里的Object类.

1. 运行时栈帧结构
   1. 栈帧定义：每一个方法的开始调用到执行完成的过程，都对应着一个栈帧的入栈到出栈过程。
   2. 栈帧组成：
      1. 局部变量表
         1. 以变量槽slot为最小单位
         2. 一个slot都能存放出long、double类型的基本数据类型+引用类型
         3. Long、double占两个变量槽
         4. 虚拟机对于连续的两个属于long或者double类型的变量槽都是不允许单独访问其中的一个
         5. 局部变量槽的第一个是指向方法所属对象实例的引用
      2. 操作数栈
      3. 动态连接
         1. 指向常量池中该栈帧所属方法的引用
      4. 返回地址
         1. 调用者的程序计数器的值
      5. 附加信息
2. 方法调用
   1. 定义：方法调用不等同于方法执行，方法调用阶段唯一的任务就是确定被调用方法的版本，暂时还不涉及方法内部的具体运行过程。
   2. 写在前面的：class文件的编译过程中不包含传统编译过程中的链接步骤，一切方法调用在Class文件里面存储的都是符号引用，而不是方法在实际运行时内存布局中的入口地址（直接引用）。所以有些方法调用要在执行时才能知道直接引用。
   3. 关于类生命周期过程中的解析：在类的加载阶段，会有一部分符号引用转化为直接引用，这个工作叫做解析。而解析能成立的前提是：方法在程序真正运行之前就有一个可确定的调用版本，并且这个版本在运行期不可被改变。
   4. 关于(C)的一个经典问题：静态方法和私有方法能否被重写？答案是不能的，因为静态方法和私有方法复合“编译期可知，运行期不可变的条件”，前者于类型直接关联，后者在外部不可被访问，这两种方法各自的特点决定了他们不能通过继承或者其他别的方式重写其他版本。
   5. 解析阶段能确定方法：静态方法、私有方法、实例构造器、父类方法，final方法。
   6. 调用分类
      1. 解析调用（这是静态的过程，编译期可知）
      2. 分派调用
         1. 静态分派
            1. Human man = new Man（）
            2. 静态类型：Human
            3. 实际类型：man
            4. 定义：根据静态类型决定方法版本的过程
            5. 应用：方法重载
         2. 动态分派
            1. 定义：在运行期间根据实际类型确定方法执行版本的过程
            2. 应用：重写
         3. 多分派
            1. 定义：根据多个综量判断方法执行版本
            2. 静态分派都是多分派
         4. 单分派：
            1. 定义：根据单个宗量判断方法执行版本
            2. 动态分派都是单分派
         5. 虚表：存储虚方法转变为直接引用信息的一张表