中心主题

- **chp6-可以工作的类**
  - 6.1 类的基础，抽象数据类型
    - 类使用方法操作内部数据
  - 6.2 良好的类接口
    - 好的抽象
      - 类接口应该展现一致的抽象层次（adt）
      - 一定要理解类所实现的抽象是什么
      - 尽量提供成对的服务（通常操作都有对应的相反的操作）
      - 把不相关的信息转移到其他类中
      - 尽量让接口可编程，而不是写到方法注释上去
      - 同时考虑内聚性和抽象性
    - 良好的封装
      - 尽可能的限制类和成员的可访问性
      - 不要公开暴露成员数据
      - 要让阅读代码比编写代码更方便
      - 警惕从语义上破坏封装性
      - 迪米特法则-最少知识法则：一个类对其他类要尽可能少的了解
  - 6.3 有关设计和实现的问题
    - 包含（has a）
      - 面向对象编程的主力技术
      - 7+-2原则：警惕有超过7个数据成员的类
    - 继承（is a）
      - 保证子类与父类有相同的接口契约
      - 子类可以通过父类的接口使用，且调用方无需关心内部差异
      - 确保只继承需要继承的部分
      - 子类中的方法名不要与父类中private的方法同名
      - 把公用的接口、数据放到继承树中尽可能高的位置
      - 只有一个子类的继承关系也值得怀疑
      - 子类覆盖父类方法，但是个空实现。这种情况也值得怀疑

- 避免让继承体系过深
- 所有数据都保持private，如果要访问提供protect方法
- 成员函数和数据成员
  - 让类中子程序的数量尽可能少
  - 减少类调用的其他类的数量
  - 类的间接调用要尽可能少
- 构造函数
  - 推荐在构造函数中初始化所有的数据成员
  - 用private构造函数实现单例
  - 对象复制优先使用深拷贝
- 6.4 创建类的原因
  - 原因
    - 对现实世界中的对象建模（长方形，三角形）
    - 为抽象的对象建模（抽象的形状类）
    - 降低复杂度（无需了解内部实现的情况下使用这个类）
    - 隔离复杂度
    - 隐藏实现细节
    - 限制变动的影响范围
    - 隐藏全局数据：通过类方法来访问，避免bad smell
    - 让参数传递更顺畅
    - 建立中心控制点：让一个类管理某类功能
    - 让代码更容易使用
    - 把相关操作放到一起
  - 应该避免的类
    - 1、 避免创建万能类
    - 2、消除无关紧要的类
    - 3、避免用动词创建的类（这种更应该是一个方法）
- 6.5超越类：包
  - 子主题 1

chp7-高质量的子程序

- 7.1 创建子程序的正当理由
  - 降低复杂度
  - 引入中间、易懂的抽象：使用方法名代替注释
  - 避免代码重复
  - 支持子类化
  - 隐藏顺序
  - 简化复杂的bool判断表达式
  - 方便改善性能，易于优化
  - 似乎过于简答而没必要写成方法的操作
    - 小巧而好的两三行代码的方法
- 7.2 在子程序层上设计（内聚性-每个方法只把一件事做好
  - 功能的内聚性：最强，一个方法只做一件事情
- 7.3 好的子程序的名字
  - 描述方法做的所有事情
  - 避免使用无意义的词
  - 合理的长度（9到15个）
  - 给函数命名时要对返回值有所描述
- 7.4 子程序可以写多长：200行
- 7.5 如何使用参数
  - 按照输入、修改、输出的顺序排列参数
  - 如果几个方法用的相似的参数，那么参数的顺序要保持一致（并不一定非得从前往后加参数）
  - 确保入参是都会被使用的
  - 把状态和出错变量放在最后
  - 不要去修改入参变量的值
  - 在接口中对参数的假定进行说明（接受范围，不该出现的值等）
  - 子程序的参数限制在7个以内
  - 考虑对参数采用输入、修改、输出的命名规则
    - 加入i m o 前缀
  - 传递参数还是传递对象（用到一个对象里的部分参数）

- 7.6 使用函数时要特殊考虑的问题
  - 子主题 1

chp8-防御式编程

- 8.1 防止非法输入造成的问题
  - 检查所有来源于外部的值是否合理
  - 检查方法的入参
  - 对错误的输入进行合理的处理
- 8.2 断言
  - 适合在开发阶段，上线前移除
  - 建立自己的断言机制
    - 不做业务处理，只处理绝不可能发生的情况（预防bug产生）
    - 避免把正常执行代码放入不一定会运行的断言中
    - 对高健壮性的复杂代码（word），要对断言的错误同样处理进行异常处理。交付前，发现并纠正所有错误是不现实的
- 8.3 错误处理技术
  - 常用的方式
    - 返回中立值（没有危害的值）
      - 空字符串、数字0等
    - 换用下一个正确数据
      - 流处理
    - 返回上一个正确的数据
      - 游戏的实时渲染
    - 换用最接近的合法值
      - 汽车倒车时速度显示为0，而不是负数
    - 警告信息打日志
    - 错误向上抛出
      - 返回一个错误码
      - 程序内部的异常机制
  - 健壮性和正确性
  - 高层次设计对处理方式的影响
- 8.4 异常

- 用异常通知程序的其他部分，发生了不可忽略的错误
- 只有真正例外的情况下才抛出异常
  - 处理罕见甚至永远不该发生的情况
- 不能用异常来推卸责任
  - 问题如果可以在局部处理，那就在局部处理掉
- 避免析构函数和构造函数抛出异常
- 在恰当的抽象层次抛出异常
  - 读取数据不应该抛出fileNotFoundException
- 在异常消息中加入错误发生是的全部信息
  - 数组越界时，要包括越界的下标相关信息
- 避免异常只捕获但是不处理
  - 至少要通过log进行日志记录
- 考虑集中的异常处理机制
  - 例。自定义的统一业务异常处理机制
- 考虑异常的替换方案

8.5 隔离程序，使之包容由错误造成的损害

- 程序进行隔离编程，在面向外部的接口中处理异常数据。内部类接收的参数可当做合法数据处理
- 数据输入后要进行判断并转化成合理的类型

8.6 辅助调试代码

- 不要自动的把产品版的限制强加在开发版之上
  - 开发期间可牺牲一些资源和速度，提供一些方便开发的工具
- 尽早引入辅助调试的代码
  - 调试助手代码（仅为了方便调试，线上不会使用）
- 采用进攻式编程方式
- 计划移除调试辅助中的代码（性能考虑）
  - 版本控制工具ant make
  - 使用内置的预处理器（c++宏定义）
- 确定在产品代码中该保留多少防御式代码
- 谨慎使用防御式编程
  - 什么东西多了都不是好事