面向对象的软件构造导论
课程内容总结

- 成绩构成
  - 平时考勤及作业 10 分
  - 考试 50 分
  - 实验 40 分
- 考试题型
  - 选择题
  - 判断题
  - 填空题
  - 简答题
  - 综合题

第一章-面向对象的软件构造概述

- 软件构造基本流程及目标
- 面向对象思想
- 设计模式 → 第五章 设计模式
- Java语言简介
- Java开发环境搭建及程序示例

软件构造基本流程及目标

- 传统软件开发过程模型的问题
- **测试驱动的开发**
- **面向对象与面向过程的差异**
- 了解软件构造的目标
  - 可理解性
  - 可维护性
  - 可复用性
  - 时空性能

面向对象思想

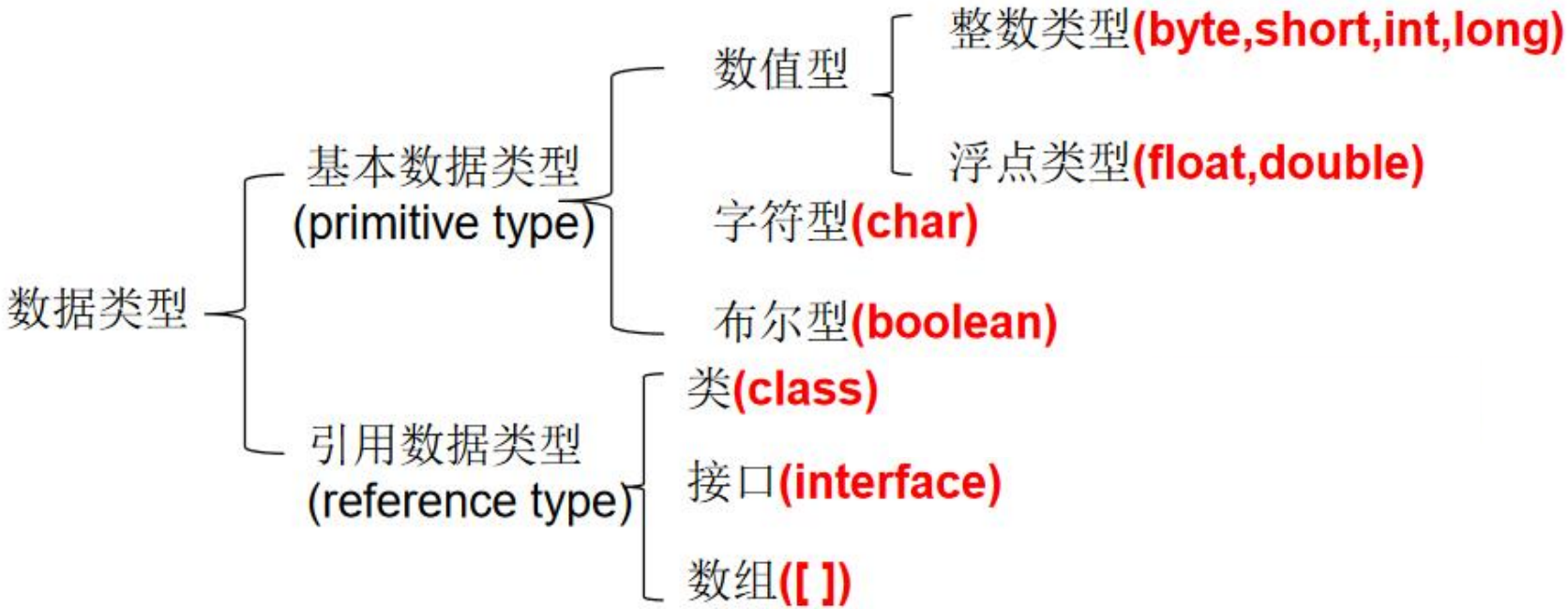
- 结构化方法 v.s. 面向对象方法
- **面向对象三大特性**
  - **封装(Encapsulation)**
    - 隐藏对象的属性和实现细节，仅对外公开访问方法;
    - 增强安全性和简化编程
  - **继承(Inheritance)**
    - 子类继承父类的特征和行为
    - 实现代码的复用
  - **多态(Polymorphism)**
    - 同一个行为具有多个不同表现形态的能力（“一个接口，多个方法”）
    - 提高了程序的扩展性和可维护性

第二章-Java语言基础

- 标识符、关键字、分隔符、注释
- Java基本数据类型、常量
- 控制流程（条件、循环、跳转语句）
- 输入与输出 → 第八章 流与输入输出
- 数组
- 异常机制 → 结合第四章
- Java虚拟机与垃圾回收

Java基本数据类型、常量

- 8种基本数据类型、表数范围
- 不同数据类型间的转换
- 常量的用法



数组

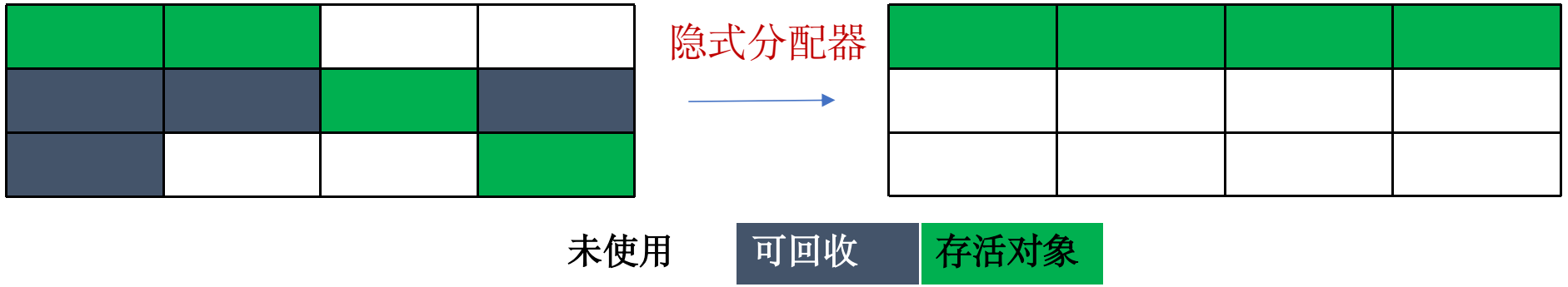
- 数组的创建与访问
- 数组的越界异常

Java虚拟机与垃圾回收

Java虚拟机的概念及其特点

- **特点：**
  - 一次编译，到处运行
  - 自动内存管理
  - 自动垃圾回收功能

Java采用隐式分配器进行垃圾回收



第三章-类和对象

- 对象与类
- 类的声明与构造
- 类的访问域
- static修饰符
- 数组

对象与类

- 类与对象之间的关系

类的声明与构造

- 类修饰符
  - public: 公共类
  - abstract：抽象类(继承)
  - final：最终类(非继承)
- 属性的封装，需要通过getter/setter对属性进行访问
- 封装的优点
- 构造方法
  - 可以有多个
  - 默认构造方法
- 掌握This关键字的用法
  - 注意和super关键字的异同

类的访问域

- 4种访问权限的作用范围
  - 类型： private、 *default*、 protected、 public

static修饰符

- 类属性的特点与好处
- 非静态方法和静态方法相互访问的问题
  - **是否能够相互访问以及背后的原因**
  - **注意**
    - 使用static声明的方法，不能访问非static的操作（属性或方法）
    - 非static声明的方法，可以访问static声明的属性或方法
  - **原因**
    - 如果一个类中的属性和方法都是非static类型的，一定要有实例化对象才可以调用
    - Static声明的属性或方法可以通过类名访问，可以在没有实例化对象的情况下调用（先于对象而存在）
- 静态块如何使用

数组

- 数组的使用（一维和多维）
  - 声明（不能指定长度）例如: int a[5]
  - 初始化（动态和静态）
  - 数组的引用
  - **注意和集合的异同**

```
int[] arr = new int[3];
arr[0] = 3;
arr[1] = 9;
arr[2] = 8;
```

```
int[] arr = new int[] {3, 9, 8};
int[] arr = {3, 9, 8};
```

第四章-接口与继承

- 继承
- 接口与抽象类
- 多态与重写
- Java多继承问题
- 超类与super关键字
- 异常的继承框架

继承

- 继承的特点
- JAVA中继承的使用，定义父类和子类，extends关键字
- 继承的优缺点

接口与抽象类

- **抽象方法与抽象类的定义**
- **抽象类的作用**
  - 抽象类是不能实例化的
  - 作用：
    - 抽象方法实际上相当于定义了“规范”
    - **只能被继承**，保证子类实现其定义的抽象方法
    - 可用于实现**多态**
- **接口的定义，使用implements 关键字**
- **接口与抽象类的对比**

多态与重写

- 多态的定义
- 三种实现方式
  - 重写
  - 抽象类与抽象方法
  - 接口
- 注意点：@override, 父类引用指向子类

Java多继承问题

- Java为什么不直接支持多继承
- Java如何实现多继承
  - 内部类
  - 接口

超类与super关键字

- Object是Java中所有类的始祖
- equals()方法比较两个对象是否相等
- **super关键字的作用**
  - 调用父类的构造方法
  - 访问父类的成员方法和变量

异常的继承框架

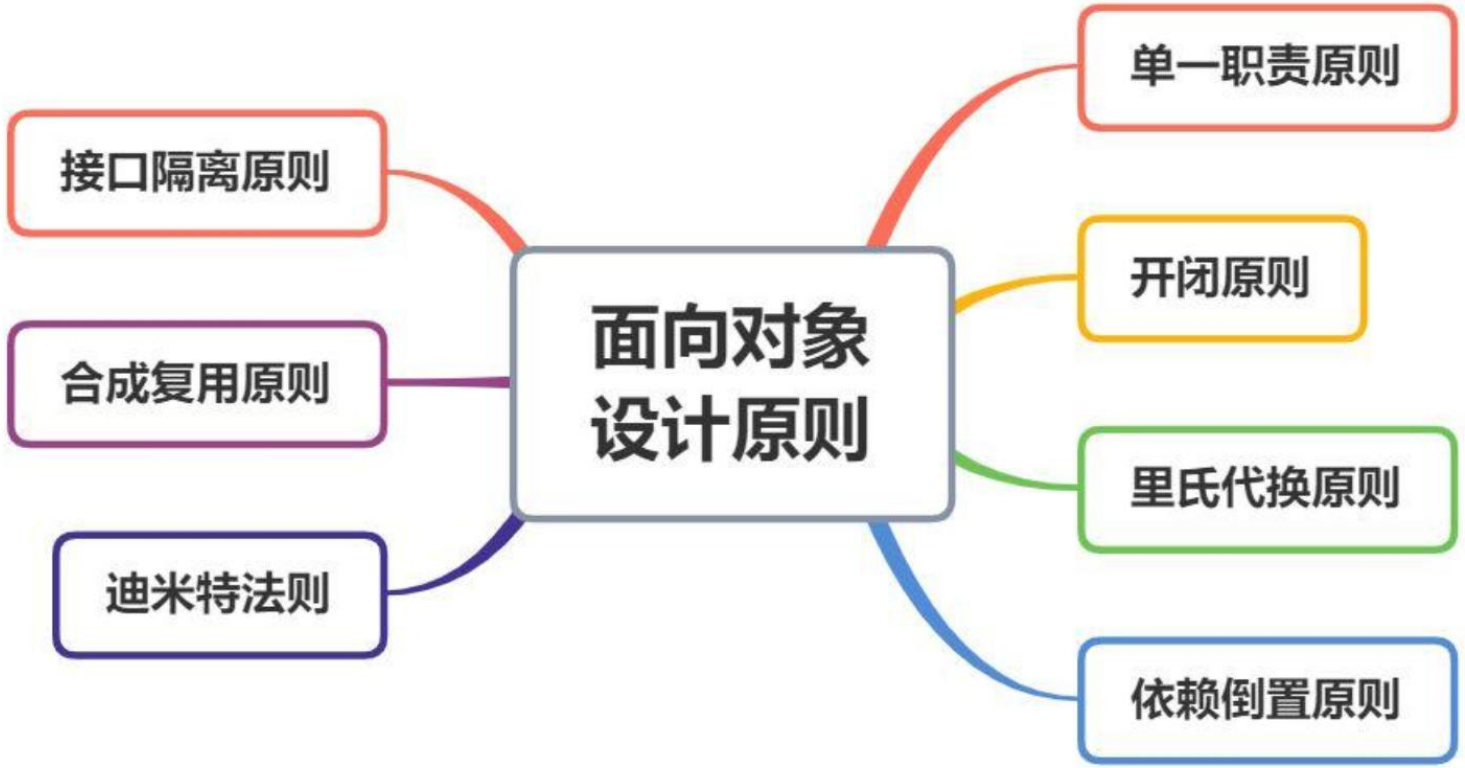
- 异常的定义
- 异常的继承框架：Throwable是异常体系的根
- 设计自定义异常

第五章-设计模式导论

- 面向对象设计原则
- 设计模式
- 单例模式
- 简单工厂模式
- 工厂模式
- 抽象工厂设计模式

面向对象设计原则

- 在设计面向对象的程序时，需遵循的常用原则有七个，包括：



设计模式

设计模式

- 设计模式的概念，作用以及分类
  - 设计模式的概念和作用
  - 根据**目的**来分（模式是用来完成什么工作）
  - 根据**范围**来分（模式主要用于类还是对象）

单例模式

- 单例模式的目的及其实现
  - 单例模式的目的
  - **单例模式的实现方法**

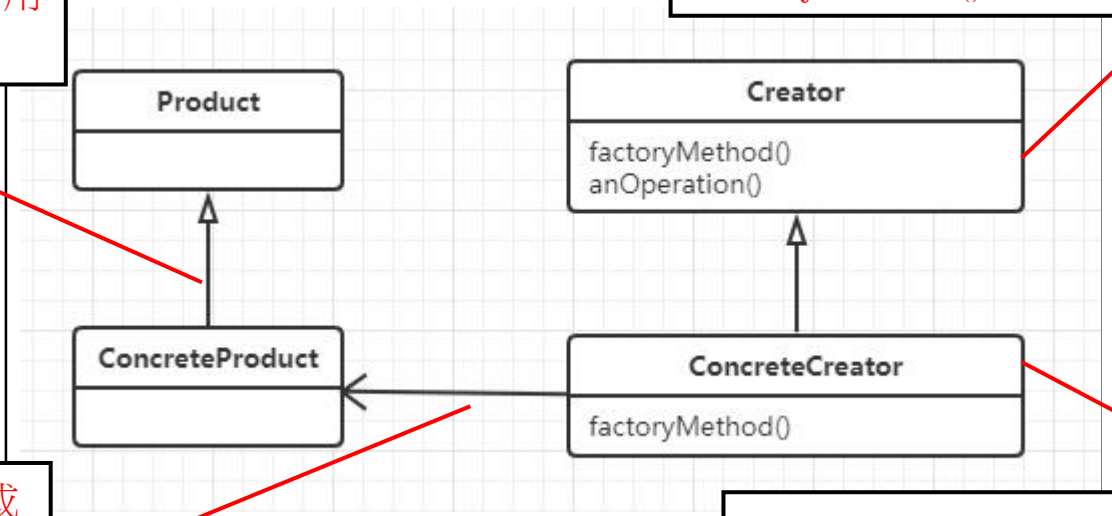
简单工厂模式与工厂模式

简单工厂模式与工厂模式

- 简单工厂模式与工厂模式的遵循的设计模式原则
- 简单工厂模式和**工厂模式的UML结构图**
- 简单工厂和工厂模式的区别（特点）

所有产品必须实现同一个接口，这样，使用这些产品的类就可以引用这个接口，而不是具体的类。

Creator类实现了所有操作产品的方法，除了抽象的工厂方法factoryMethod()。



ConcreteCreator负责创建一个或多个具体产品，只有它知道如何创建这些产品。

所有创建者的子类ConcreteCreator必须实现factoryMethod()，这个方法完成具体产品的创建。

抽象工厂设计模式

抽象工厂设计模式

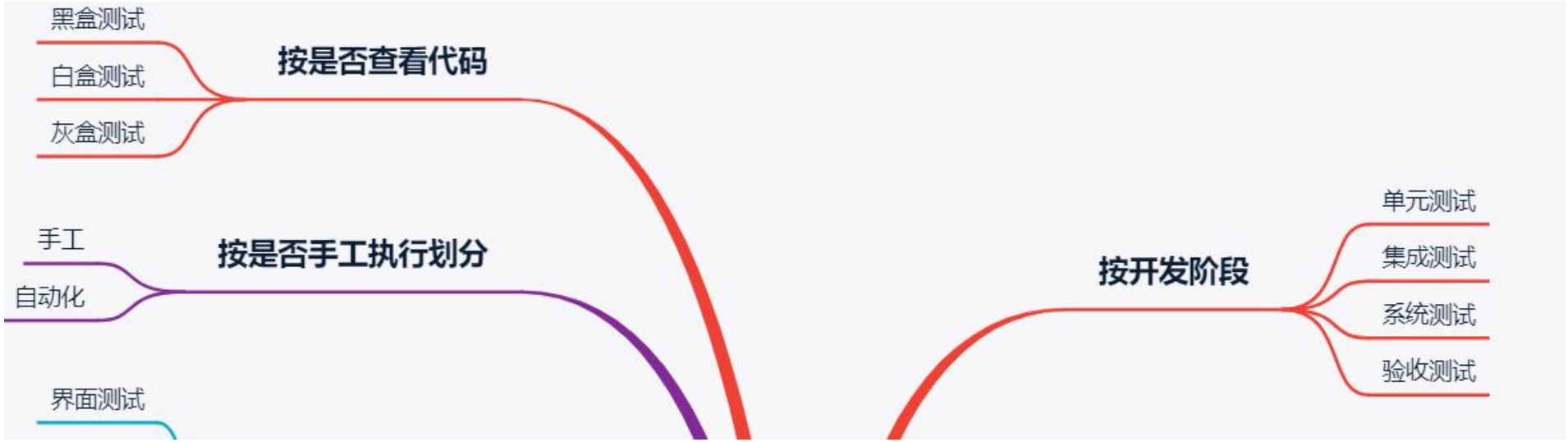
- 产品族和产品等级概念
- 抽象工厂的特点及其优缺点
- 抽象工厂模式的结构图

第六章-软件测试及代码质量保障

- 软件测试的定义和分类
- 测试用例
- 白盒测试
- 黑盒测试

软件测试的定义和分类

- 软件测试的定义
- 软件测试的目的
- 软件测试的常见分类
  - 按测试阶段
  - 是否查看源代码



测试用例

- 测试用例的定义
- 测试用例的设计原则
- 测试用例的常用设计方法

  - 测试用例（Test Case）是将软件测试的行为活动做一个科学化的组织归纳，目的是能够将软件测试的行为转化成可管理的模式；同时测试用例也是将测试具体量化的方法之一，不同类别的软件，测试用例是不同的。

  - 测试用例的设计方法主要有白盒测试法和黑盒测试法。

    - 白盒测试又称结构测试、透明盒测试、逻辑驱动测试或基于代码的测试。白盒法全面了解程序内部逻辑结构、对所有逻辑路径进行测试。

    - 黑盒测试也称功能测试，黑盒测试着眼于程序外部结构，不考虑内部逻辑结构，主要针对软件界面和软件功能进行测试。

白盒测试

- 白盒测试的定义
- 白盒测试的优势和劣势
- 逻辑覆盖方法的覆盖指标及其强度比较
- 常用白盒测试用例的设计方法
  - 语句覆盖
  - 判定覆盖
  - 条件覆盖
  - 判定条件覆盖
  - 条件组合覆盖
  - 路径覆盖

黑盒测试

- 黑盒测试的定义
- 黑盒测试的优势和劣势
- 常用黑盒测试用例的设计方法
  - **等价类划分**
  - **边界值分析**
  - **场景法**

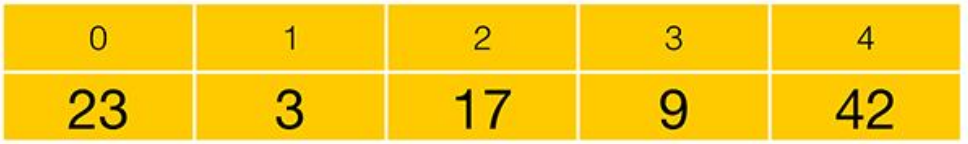
第七章-集合与策略、 迭代器模式

- 集合类概述
- 集合的继承框架
- List接口及其标准实现类ArrayList与LinkedList
- Set与Map接口
- 策略模式
- Java Iterator （迭代器模式）

集合类概述及继承框架

- 集合的定义及与数组的区别
  - 与数组的区别：
    (1) 数组长度固定，集合长度不固定
    (2) 数组可以存储基本类型和引用类型，集合只能存储引用类型
- 常用的集合类举例（List, Set, Map）
- 集合继承框架

ArrayList



LinkedList

List接口及其标准实现类ArrayList与LinkedList

- **ArrayList和LinkedList的特点及二者异同**
- ArrayList和LinkedList的元素增加，删除，访问，修改和迭代方法

Set与Map接口

Set与Map接口

- Set与Map的特点及二者异同（相同元素等）
- Set的元素添加，删除，访问以及判断元素是否存在方法
- Map的元素添加，删除，访问以及迭代方法

策略模式

策略模式

- 策略模式的目的，应用场景及其实现
- 策略模式中环境，抽象策略和具体策略三个角色的作用(画UML结构图)
- 如何让算法和对象分开来，使得算法可以独立于使用它的客户而变化?
- 策略模式的优缺点

迭代器模式

迭代器模式

- 迭代器模式的目的，应用场景及其实现
- 如何将聚合对象与其遍历行为分离开?
- 迭代器模式的效果

第八章-流与输入输出

- 流
- 输入输出流
- Java流继承框架
- 操作文件
- 对象输入/输出流与序列化
- 数据访问对象模式

流

- 流的概念
- 流的分类
  - 方向：输入流、输出流
  - 内容：字节流、字符流

输入输出流

- Java的系统流：System.in, System.out
- 读取控制台输入：代码阅读
- 文件的输入输出：代码阅读

Java流继承框架

- 四大流家族的根节点
  - **InputStream, OutputStream**
  - **Reader, Writer**
- 流需要close()方法，使用结束之后需关闭以避免耗费资源

	字节流	字符流
输入流	InputStream	Reader
输出流	OutputStream	Writer

操作文件

- Path和Files类：阅读代码
- 了解有哪些功能：阅读代码
  - 创建，复制，移动，删除，获取文件信息

对象输入输出流及序列化

- 序列化及反序列化的概念
- **对象序列化的步骤及简单实现**
  - **注意点： Serializable接口, writeObject(), readObject()**

数据访问对象模式

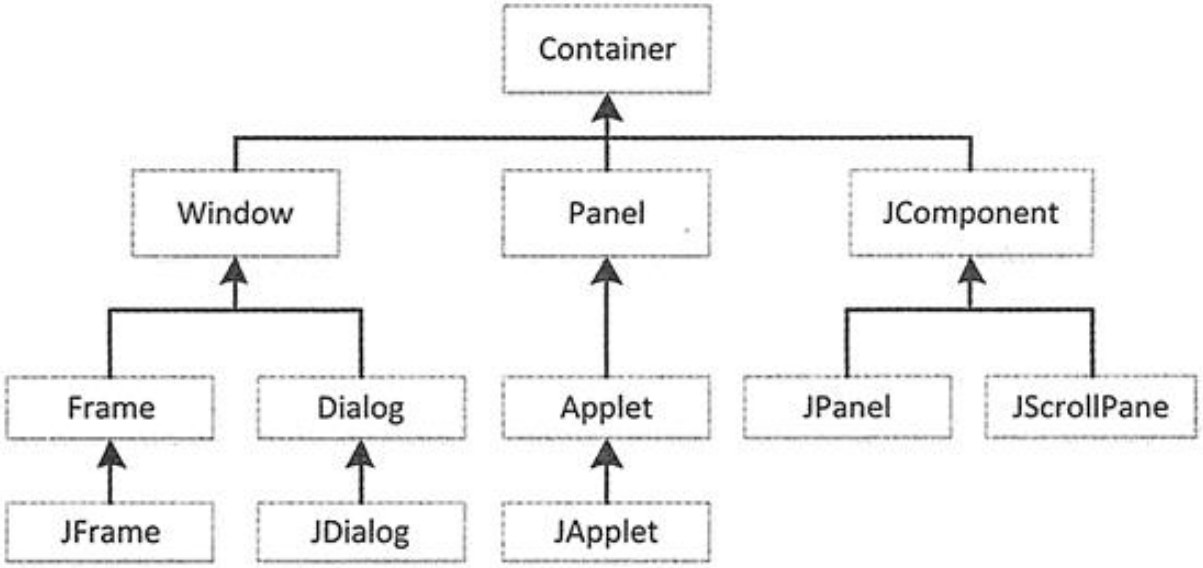
- 用处：将低级的数据操作从高级的业务服务中分离
- 参与者：
  - 数据访问对象接口
  - 数据访问对象实体类
  - 模型对象/数值对象
- 优缺点

第九章-Swing图形用户界面

- Swing框架
- Swing图形处理、绘制颜色的原理
- 事件机制
- Swing基本用户组件
- MVC模式

Swing框架

- **Swing 的两种元素：组件和容器**
- **Swing的组件继承于JComponent类**
- **容器的分类：**
  - **重量级: Jframe, Jdialog, 作为顶层容器**
  - **轻量级：作为中间容器**
- 布局管理器的作用



Swing图形处理、绘制颜色的原理

- Jframe
  - 顶层窗口, JFrame由用户的窗口系统绘制。
- 绘制2D图形：阅读代码

事件机制

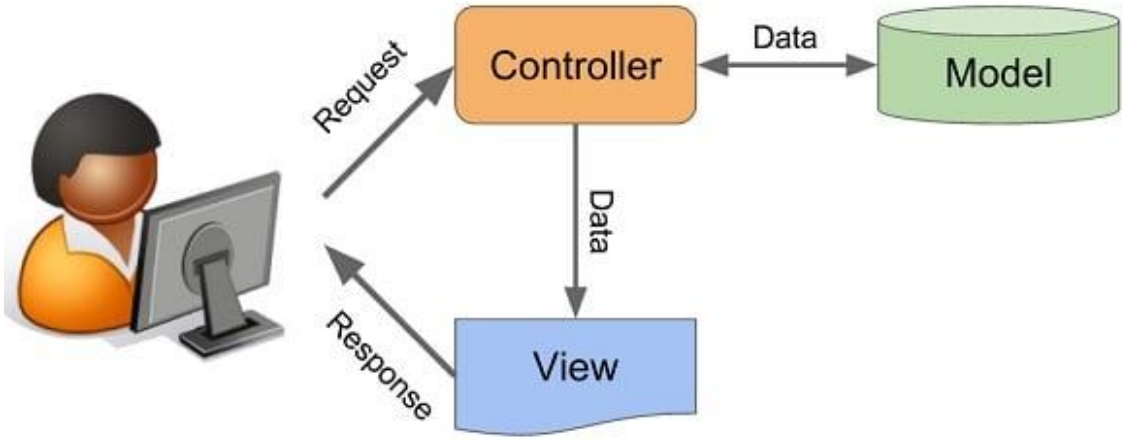
- 事件处理机制
  - 事件，事件源，事件监听器
- 能够阅读代码

Swing基本用户组件

- 了解文本输入的三种方法
  - JTextComponent类
  - 文本域，文本区，密码域
- 选择组件：阅读代码
  - 复选框 JCheckBox
  - 单选按钮 ButtonGroup,JRadioButton
  - 组合框 faceCombo
  - 菜单 JMenuBar

MVC模式

- 了解各部分的作用
  - 模型
  - 视图
  - 控制器
- 能够分析简单案例中控制器的作用



第十章-多线程

- 进程与线程
- 多线程
- Java中对线程的控制
- 同步、死锁及如何避免
- 多线程应用—生产者与消费者模式
- 任务与线程池

进程与线程

进程与线程的区别

进程（Process）	线程（Thread）
重量级	轻量级
一个应用可以包含多个进程	一个进程可以包含多个线程
多个进程间 不共享内存	一个进程的多个线程间 共享内存
进程表现为 虚拟机	线程表现为 虚拟CPU

Java中对线程的控制

- 线程的状态（6种）
  - **线程的两种建立方式（Thread子类，Runnable接口：无返回值，不可抛出异常）**
    - **为什么Runnable更常用**
  - Interrupt用法（是否立刻打断？）
    - 其作用是中断此线程（此线程不一定是当前线程，而是指调用该方法的Thread实例所代表的线程），但实际上只是给线程设置一个中断标志，线程仍会继续运行。）
  - 等待状态和计时等待状态的区别
  - 守护线程的作用
  - 区分Thread类和Object中的线程相关方法

同步、死锁及如何避免

- 为什么需要线程同步？ (原子性操作)
- 怎么实现同步
- 什么情况下会产生死锁?

多线程应用—生产者与消费者模式

- 生产者消费者设计模式的特点
- 如何简单地实现一个生产者消费者模式
  - 为什么缓冲区的判断条件是 while,而不是if
  - 理解wait方法为什么要放在同步块里
- 生产者消费者设计模式的优点

任务与线程池

- 线程的另一种建立方式（有返回值，可抛出异常）
  - 如何结合FutureTask去使用
- 为什么要引入线程池？
- 理解线程池的多任务控制方式

第十一章-泛型与反射

- 为什么需要泛型？什么是泛型？
- 泛型类、泛型方法、泛型接口
- 泛型的通配符
- 泛型的设计——模板方法模式
- 反射
- 设计安全的全局单例

为什么需要泛型？什么是泛型？

- 什么是泛型
- 使用泛型的优点
  - 不用强制转型
  - 在编译阶段检测到非法的数据类型

泛型类、泛型方法、泛型接口

- 怎么定义泛型类
- 怎么定义泛型方法
  - 可以定义在普通类里，也可以定义在泛型类里
  - **如何定义静态泛型方法**
- 怎么定义泛型接口
  - 实现类可以是非泛型类也可以是泛型类（注意区别）

泛型的通配符

- 为什么需要通配符
  - **Object是所有类型的父类，但是List<Object>并不是List<String>的父类**
- 理解三种通配符的用法
- 类型参数T与通配符的区别

泛型的设计——模板方法模式

- 理解模板方法模式
  - 什么时候定义抽象操作,什么时候定义钩子操作

反射

- 理解Class类
- 正常方式与反射方式的区别
- 如何获取Class类对象
  - **getClass()**
  - **Class.forName()**
  - **T.class (T是任意的Java类型)**
- 如何通过反射构造类的实例
  - **Class.newInstance**
  - **使用Constructor的newInstance**
  - **以上两种方法的区别**
- 如何通过反射获取和修改成员变量
- 如何通过反射获取和调用成员方法

设计安全的全局单例

- 如何通过反射破坏单例模式(饿汉式)
- 如何抵御反射破坏

第十二章-网络编程

- 网络通信的基本原理及IP地址
- Socket编程类库
- URL的使用
- 观察者模式

网络通信的基本原理及IP地址

- TCP的特点
- 理解IP地址

Socket编程类库与URL的使用

- Socket使用了TCP协议的通信机制
- 理解Socket和URL的用法

观察者模式

- 定义了对象之间一种**一对多**的依赖关系，让一个对象的改变能够影响其他对象
- 发生改变的对象称为观察目标，被通知的对象称为观察者
- 观察者模式的**UML类图**
- 理解模式的优缺点

祝同学们考试顺利！