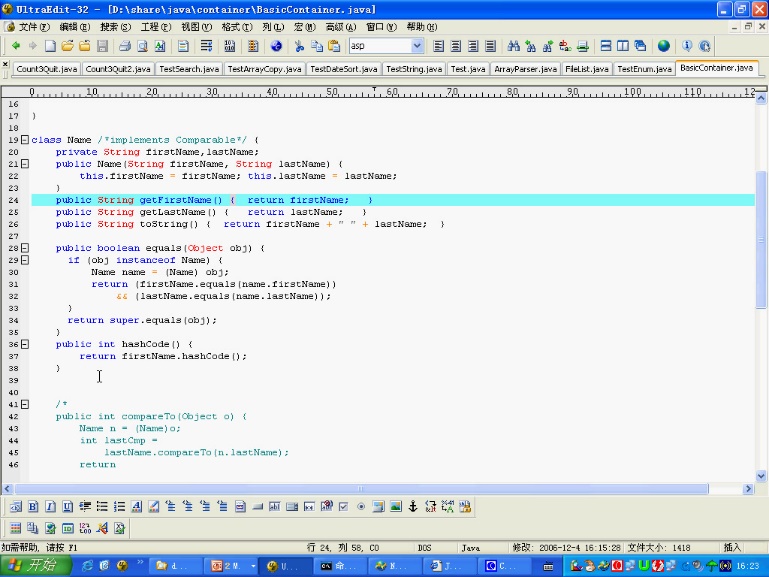
**03\_Collection\_3\_笔记**

**一、hashCode方法的实现**



* 方法定义：在Name类中实现hashCode()方法，返回一个哈希值
* 返回值类型：方法返回值为int类型，对应哈希码

**1. hashCode算法的选择 ﻿**

* 算法多样性：计算hashCode有各种不同算法，实现起来较为复杂
* 现成方案优势：优先使用已有类的hashCode实现，如String类的hashCode方法

**2. 使用现成的hashCode方法**

* 实现方式：直接调用firstName的hashCode()方法作为返回值
* 封装思想：利用String类已实现的hashCode算法，无需关心内部具体实现
* 代码示例：

**3. 保证equals的对象hashCode相等**

* 基本原则：重写equals方法时必须重写hashCode方法
* 一致性要求：两个对象如果equals返回true，它们的hashCode必须相等
* 实现保证：通过使用相同字段计算hashCode来确保一致性
* 注意事项：
  + 不能简单使用int类型成员变量作为hashCode
  + 不同对象的hashCode应尽量区分开，减少重复几率

**4. hashCode的应用场景**

* 索引使用：当对象作为键(key)时会被使用
* 实际案例：人事管理系统中，用姓名作为索引存储其他信息(年龄、性别、电话等)
* 检索流程：先通过姓名hashCode快速定位，再获取详细信息

**5. 通过hashCode提高检索效率**

* 性能优势：相比逐个比较，hashCode可以快速定位对象
* 工作原理：
  + 调用对象的hashCode方法获取哈希值
  + 根据哈希值直接定位到存储位置
  + 在该位置找到对应对象
* 系统设计：这种机制被广泛应用于Java集合框架(如HashMap)中

**二、知识小结**

|  |  |  |  |
| --- | --- | --- | --- |
| 知识点 | 核心内容 | 考试重点/易混淆点 | 难度系数 |
| hashCode方法的作用 | 用于生成对象的哈希值，便于快速查找和索引 | 必须与equals方法保持一致：两个对象equals相等时，hashCode必须相等 | ⭐⭐ |
| hashCode实现原则 | 优先使用现成实现（如String的hashCode），避免自定义复杂算法 | 自定义hashCode需保证非equals对象的哈希值尽量分散，减少冲突 | ⭐⭐⭐ |
| hashCode应用场景 | 对象作为哈希表键时（如HashMap的Key），通过哈希值快速定位数据 | 索引效率对比：直接比较字段（低效） vs 哈希值定位（高效） | ⭐⭐ |
| equals与hashCode关系 | 重写equals必须重写hashCode，确保逻辑一致性 | 常见错误：仅用部分字段（如int类型）作为哈希值导致冲突 | ⭐⭐⭐⭐ |
| 哈希冲突问题 | 即使JDK内置算法（如String.hashCode）也可能产生冲突 | 实际开发中无需过度优化，优先保证功能正确性 | ⭐⭐ |