**课题：对大量电影数据的高效存储 | 小组队长**

**项目目的：**熟悉并熟练掌握JAVA语言与数据结构，并初步体验项目实践；**工具**：Visual Studio Code，Terminal

* 利用 Visual Studio Code 编写程序，结合终端调试，设计并实现一种高效的数据结构，用于支持数据的存储与快速搜索；基于该结构开发了电影、评分与评级模块的核心功能；
* 撰写技术报告，论证所选数据结构在设计方案中的合理性，并深入分析其在三个数据存储模块中支持关键操作的实现方式与优化路径。

1. 理解需求

需要为三个数据类（Movies、Credits、Ratings）设计高效数据结构，支持数据的存储和查询。

数据来源是CSV文件，已由LoadData类解析并传递给对应的存储类。

需要实现每个类中标注了@param和@return的函数功能。

2. 数据结构选择建议

Movies类：

需求：存储电影元数据（如标题、预算、类型等），支持按ID、标题、类型等查询。

推荐结构：

主存储：HashMap<Integer, Movie>，以tmdb\_id为键快速查找电影。

辅助索引：

按标题查询：HashMap<String, List<Movie>>（标题可能重复）。

按类型查询：HashMap<Genre, List<Movie>>（多对多关系）。

按年份查询：TreeMap<Integer, List<Movie>>（支持范围查询）。

Credits类：

需求：存储演员和剧组信息，支持按电影ID、演员/角色查询。

推荐结构：

主存储：HashMap<Integer, FilmCredits>，以tmdb\_id为键关联电影。

反向索引：

演员到电影：HashMap<Person, List<FilmCredits>>。

角色到演员：HashMap<String, List<CastCredit>>（如查询所有“导演”）。

Ratings类：

需求：存储用户评分，支持按电影ID、用户ID、评分范围查询。

推荐结构：

主存储：HashMap<Integer, List<Rating>>，以tmdbId为键关联电影。

用户索引：HashMap<Integer, List<Rating>>，以userId为键。

评分索引：TreeMap<Float, List<Rating>>（支持范围查询，如评分≥4的电影）。

3. 实现步骤

从Ratings类入手（最简单）：

实现add方法存储评分数据。

实现按电影ID、用户ID查询评分。

添加TreeMap支持评分范围查询（如subMap(4.0f, 5.0f)）。

Credits类：

使用嵌套结构存储每部电影的演员和剧组列表。

构建反向索引加速“查询某演员参与的所有电影”。

Movies类：

为每个查询需求（如按类型、年份）设计单独的索引结构。

注意处理多值字段（如genres可能包含多个类型）。

4. 优化与测试

内存效率：对于重复字符串（如语言、类型），可使用String.intern()或共享对象。

查询速度：所有操作应尽量在O(1)或O(log n)时间内完成。

测试：优先通过JUnit测试验证功能，再通过UI手动检查。

5. 报告要点

选择理由：解释为何选HashMap（快速查找）和TreeMap（范围查询）。

实际应用：举例说明如何用HashMap.get(tmdb\_id)实现电影查询。

复杂度分析：对比不同方案的查询效率（如线性搜索 vs 哈希索引）。