JAVA PROGRAMING

Project Name	JAVA Programming
Project Institute	CodTech IT Solutions
Task Name	BUILD A RECOMMENDATION SYSTEM USING JAVA AND LIBRARIES LIKE APACHE MAHOUT TO SUGGEST PRODUCTS OR CONTENT BASED ON USER PREFERENCES. DELIVERABLE: A JAVA PROGRAM WITH A WORKING RECOMMENDATION ENGINE AND SAMPLE DATA
Prepared By	Janani Sri. P

Rev No	Date	Task Description	Prepared By
Rev 00	29.07.02025	Functional tests cover data loading from CSV, generating recommendations for multiple users, and evaluation using a train/test split. Non-functional tests include performance on small sample data and robustness for missing files (auto-generation of sample data).	Janani Sri. P

1. Task Name

Build a recommendation system using Java and libraries like Apache Mahout to suggest products or content based on user preferences.

JAVA PROGRAMING

2. Details (Requirements)

- Implement collaborative filtering (user-based and item-based) using Mahout's Taste APIs.
- Provide sample CSV data so the engine runs out-of-the-box.
- Console interface to request N recommendations for a given user.
- Include a README with build/run instructions using Maven.
- Provide an evaluation mode reporting Average Absolute Difference and IR stats.
- Package all files and documentation.

3. Test Scope (Description)

Functional tests cover data loading from CSV, generating recommendations for multiple users, and evaluation using a train/test split. Non-functional tests include performance on small sample data and robustness for missing files (auto-generation of sample data).

4. Tools & Versions Used

- Java 11
- Apache Maven 3.6+
- Apache Mahout (Taste) 0.9
- SLF4J Simple 2.0.13
- Apache Commons CLI 1.5.0
- Development OS: any (Windows/Linux/macOS) with JDK 11.

5. Libraries

- Mahout Core (Taste)
- Commons CLI for argument parsing,
- SLF4J Simple for logging.

6. Base Code (Full Code Used)

RecommendationApp.java

```
package com.example.recommender;

import org.apache.mahout.cf.taste.common.TasteException;

import org.apache.mahout.cf.taste.eval.DataModelBuilder;

import org.apache.mahout.cf.taste.eval.IRStatistics;

import org.apache.mahout.cf.taste.eval.RecommenderBuilder;

import org.apache.mahout.cf.taste.eval.RecommenderEvaluator;

import org.apache.mahout.cf.taste.eval.RecommenderIRStatsEvaluator;

import org.apache.mahout.cf.taste.impl.common.LongPrimitiveIterator;

import
org.apache.mahout.cf.taste.impl.eval.AverageAbsoluteDifferenceRecommenderEvaluator;

import org.apache.mahout.cf.taste.impl.eval.GenericRecommenderIRStatsEvaluator;
```

JAVA PROGRAMING

```
import org.apache.mahout.cf.taste.impl.model.file.FileDataModel;
import org.apache.mahout.cf.taste.impl.neighborhood.NearestNUserNeighborhood;
import org.apache.mahout.cf.taste.impl.recommender.GenericItemBasedRecommender;
import org.apache.mahout.cf.taste.impl.recommender.GenericUserBasedRecommender;
import org.apache.mahout.cf.taste.impl.similarity.LogLikelihoodSimilarity;
import org.apache.mahout.cf.taste.impl.similarity.PearsonCorrelationSimilarity;
import org.apache.mahout.cf.taste.model.DataModel;
import org.apache.mahout.cf.taste.neighborhood.UserNeighborhood;
import org.apache.mahout.cf.taste.recommender.RecommendedItem;
import org.apache.mahout.cf.taste.recommender.Recommender;
import org.apache.mahout.cf.taste.similarity.ItemSimilarity;
import org.apache.mahout.cf.taste.similarity.UserSimilarity;
import org.apache.commons.cli.*;

import java.io.File;
import java.util.List;
import java.util.Random;

public class RecommendationApp {

   public enum Mode { USER, ITEM, EVAL, DEMO }

   public static void main(String[] args) throws Exception {
      Options options = new Options();
      options.addOption("m", "mode", true, "Mode: user | item | eval | demo (default:
demo)");
      options.addOption("u", "user", true, "User ID for recommendations (long).");
      options.addOption("n", "num", true, "Number of recommendations to produce
(default 5).");
      options.addOption("p", "path", true, "Path to ratings.csv file (default:
data/ratings.csv)");
```

JAVA PROGRAMING

```
    CommandLineParser parser = new DefaultParser();
    CommandLine cmd = parser.parse(options, args);

    String modeStr = cmd.getOptionValue("mode", "demo").toUpperCase();
    Mode mode = Mode.valueOf(modeStr);
    int howMany = Integer.parseInt(cmd.getOptionValue("num", "5"));
    String ratingsPath = cmd.getOptionValue("path", "data/ratings.csv");

    // Create sample data if not present
    File dataFile = new File(ratingsPath);
    if (!dataFile.exists()) {
      System.out.println("No ratings data found at " + ratingsPath + ". Creating sample
data...");
      SampleData.writeSampleRatings(ratingsPath);
    }

    DataModel model = new FileDataModel(new File(ratingsPath));

    switch (mode) {
      case USER:
        long userId = Long.parseLong(cmd.getOptionValue("user", "1"));
        runUserBased(model, userId, howMany);
        break;
      case ITEM:
        long itemId = Long.parseLong(cmd.getOptionValue("user", "1"));
        runItemBased(model, itemId, howMany);
        break;
      case EVAL:
        runEvaluation(model);
```

```
            break;
        default:
            runDemo(model, howMany);
    }
  }

  private static void runUserBased(DataModel model, long userId, int howMany) throws
Exception {
    UserSimilarity similarity = new PearsonCorrelationSimilarity(model);
    UserNeighborhood neighborhood = new NearestNUserNeighborhood(3, similarity,
model);
    Recommender recommender = new GenericUserBasedRecommender(model,
neighborhood, similarity);
    List<RecommendedItem> recommendations = recommender.recommend(userId,
howMany);
    System.out.println("User-based recommendations for user " + userId + ":");
    for (RecommendedItem rec : recommendations) {
      System.out.printf("  item=%d score=%.4f%n", rec.getItemID(), rec.getValue());
    }
  }

  private static void runItemBased(DataModel model, long userId, int howMany) throws
Exception {
    ItemSimilarity similarity = new LogLikelihoodSimilarity(model);
    Recommender recommender = new GenericItemBasedRecommender(model,
similarity);
    List<RecommendedItem> recommendations = recommender.recommend(userId,
howMany);
    System.out.println("Item-based recommendations for user " + userId + ":");
    for (RecommendedItem rec : recommendations) {
      System.out.printf("  item=%d score=%.4f%n", rec.getItemID(), rec.getValue());
    }
```

```
  }

  private static void runEvaluation(DataModel model) throws Exception {
    RecommenderBuilder builder = dataModel -> {
      try {
        UserSimilarity sim = new PearsonCorrelationSimilarity(dataModel);
        UserNeighborhood nb = new NearestNUserNeighborhood(3, sim, dataModel);
        return new GenericUserBasedRecommender(dataModel, nb, sim);
      } catch (TasteException e) {
        throw new RuntimeException(e);
      }
    };
    RecommenderEvaluator evaluator = new
AverageAbsoluteDifferenceRecommenderEvaluator();
    double score = evaluator.evaluate(builder, null, model, 0.7, 1.0);
    System.out.printf("Evaluation (Average Absolute Difference): %.4f%n", score);

    RecommenderIRStatsEvaluator irEval = new
GenericRecommenderIRStatsEvaluator();
    IRStatistics stats = irEval.evaluate(builder, null, model, null, 5,
        GenericRecommenderIRStatsEvaluator.CHOOSE_THRESHOLD, 1.0);
    System.out.printf("IR Stats: Precision=%.4f Recall=%.4f%n", stats.getPrecision(),
stats.getRecall());
  }

  private static void runDemo(DataModel model, int howMany) throws Exception {
    LongPrimitiveIterator it = model.getUserIDs();
    if (!it.hasNext()) {
      System.out.println("Model has no users.");
      return;
    }
```

```
        long anyUser = it.nextLong();
        runUserBased(model, anyUser, howMany);
    }
}
```

SampleData.java

```
package com.example.recommender;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Random;

public class SampleData {

    public static void writeSampleRatings(String path) throws IOException {
        File f = new File(path);
        f.getParentFile().mkdirs();
        try (BufferedWriter bw = new BufferedWriter(new FileWriter(f))) {
            // CSV format: userID,itemID,rating  (no header)
            long[] users = {1,2,3,4,5};
            long[] items = {101,102,103,104,105,106,107,108};
            Random rnd = new Random(42);
            for (long u : users) {
                for (long i : items) {
                    // Each user rates ~60% of items with 1-5 stars
                    if (rnd.nextDouble() < 0.6) {
                        int rating = 1 + rnd.nextInt(5);
```

```
                bw.write(u + "," + i + "," + rating);
                bw.newLine();
            }
          }
        }
      }
  }
}
```

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
 <modelVersion>4.0.0</modelVersion>

 <groupId>com.example</groupId>
 <artifactId>mahout-recommender</artifactId>
 <version>1.0.0</version>
 <name>Mahout Recommender</name>
 <description>Recommendation system using Java & Apache Mahout
(Taste)</description>

 <properties>
  <maven.compiler.source>11</maven.compiler.source>
  <maven.compiler.target>11</maven.compiler.target>
 </properties>

 <dependencies>
```

JAVA PROGRAMING

```
   <!-- Apache Mahout Taste (legacy CF APIs). Version 0.9 is widely used for Taste
examples. -->
   <dependency>
    <groupId>org.apache.mahout</groupId>
    <artifactId>mahout-core</artifactId>
    <version>0.9</version>
   </dependency>
   <dependency>
    <groupId>commons-cli</groupId>
    <artifactId>commons-cli</artifactId>
    <version>1.5.0</version>
   </dependency>
   <dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-simple</artifactId>
    <version>2.0.13</version>
   </dependency>
 </dependencies>

 <build>
  <plugins>
   <plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>exec-maven-plugin</artifactId>
    <version>3.5.0</version>
    <configuration>
     <mainClass>com.example.recommender.RecommendationApp</mainClass>
    </configuration>
   </plugin>
  </plugins>
```

```
 </build>


</project>
```

7. Plugin I/P (Inputs)

Input files: data/ratings.csv (userID,itemID,rating). Command-line inputs: -m MODE (user|item|eval|demo), -u USER_ID, -n HOW_MANY, -p PATH_TO_RATINGS.

8. Alpha Stage

Iteration-1

Date & Time:
29/07/2025, 5:00 PM

Observations:
• Application starts and auto-generates sample ratings if none are found.
• User-based CF with Pearson similarity returns top-N items for a user.
• Item-based CF with log-likelihood similarity works for the same input.
• Evaluation runs with average absolute difference and IR stats on sample data.

Output (sample):
User-based recommendations for user 1:
item=106 score=3.92
item=107 score=3.51
item=103 score=3.**22**

9. Result

Thus the java code has been successfully verified and got output.

check this code on my github link: https://github.com/Janani-6382/recomendation-system-using-java-and-libraries

10. Conclusion

The prototype demonstrates collaborative filtering using Mahout Taste. It is ready for extension with domain-specific item metadata or a REST API, and for deployment behind a service layer.