```
package it.polito.bigdata.spark.exercise2;

import org.apache.spark.api.java.*;

import scala.Tuple2;

import org.apache.spark.SparkConf;

public class SparkDriver {


      public static void main(String[] args) {

            String inputPathWatched;
            String inputPathPreferences;
            String inputPathMovies;
            String outputPathUseless;
            String outputPathMisleading;

            double threshold;

            inputPathWatched=args[0];
            inputPathPreferences=args[1];
            inputPathMovies=args[2];
            threshold=Double.parseDouble(args[3]);
            outputPathUseless=args[4];
            outputPathMisleading=args[5];

            // Create a configuration object and set the name of the application
            SparkConf conf=new SparkConf().setAppName("Spark Exam 1 - Exercise
#2");

            // Create a Spark Context object
            JavaSparkContext sc = new JavaSparkContext(conf);

            // ****************************************
            // Exercise 2 - Part A
            // ****************************************

            // Read the content of movies.txt
            JavaRDD<String> moviesRDD = sc.textFile(inputPathMovies).cache();

            // Select only the genre attribute
            JavaRDD<String> genresRDD = moviesRDD.map(new SelectGenre());

            // Count the number of possible genres (=distinct genres in genresRDD)
            JavaRDD<String> distinctGenresRDD = genresRDD.distinct();
            long numGenres=distinctGenresRDD.count();

            // Count the number of liked genres for each user
            // It is like a "word count" problem where each user
            // is a "word" and the number of occurrences is given
            // by the number of "genres" associated with him/her,
            // i.e., the number of liked genres for each user
            // is equal to the number of lines of preferencesProfile
            // containing the user

            // Read the content of preferencesProfile.txt
            JavaRDD<String>
```

```
preferencesRDD=sc.textFile(inputPathPreferences).cache();

            // Generate one pair userid,1 for each line of preferencesProfile.txt
            JavaPairRDD<String,Integer> userOneRDD=preferencesRDD.mapToPair(new
UserOne());

            // Reduce by key (=userid) to compute the number of liked genres for
            // each user
            // The output RDD contains one pair (userid, num liked genres) for each
user
            JavaPairRDD<String,Integer>
usersNumLikedGenres=userOneRDD.reduceByKey(new Sum());

            // Select the user with more than 90% of the possible genes
            JavaPairRDD<String,Integer>
usersUselessProfile=usersNumLikedGenres.filter(new UserUseless(numGenres));

            // Select only the userids of the selected users
            JavaRDD<String> userIdsUseless=usersUselessProfile.keys();

            // Store the selected users in the output folder
            userIdsUseless.saveAsTextFile(outputPathUseless);


            // ****************************************
            // Exercise 2 - Part B
            // ****************************************

            // Read the content of the watched movies file
            JavaRDD<String> watchedRDD = sc.textFile(inputPathWatched);

            // Select only userid and movieid
            // Define a JavaPairRDD with movieid as key and userid as value
            JavaPairRDD<String,String> movieUserPairRDD=watchedRDD.mapToPair(new
MovieUser());

            // Read the content of the movies file
            // The content of movies is already in moviesRDD

            // Select only movieid and genre
            // Define a JavaPairRDD with movieid as key and genre as value
            JavaPairRDD<String,String> movieGenrePairRDD=moviesRDD.mapToPair(new
MovieGenre());

            // Join watched movie with movies
            // It is used to associated each user with the genres of the movies
he/she watched
            JavaPairRDD<String,Tuple2<String,String>> joinWatchedGenreRDD =
movieUserPairRDD.join(movieGenrePairRDD);

            // Associate each user with the genres of the movies he/she watched
            // Select only userid (as key) and genre (as value)
            JavaPairRDD<String,String> usersWatchedGenresRDD =
joinWatchedGenreRDD.mapToPair(new UserWatchedGenre());


            // Read the content of the preferences/liked genres
            // The content of preferencesProfile is already in preferencesRDD
```

```
            // Define a JavaPairRDD with userid as key and genre as value
            // Associate each user with the genres he/she likes
            JavaPairRDD<String,String> userLikedGenresRDD =
preferencesRDD.mapToPair(new UserLikedGenre());

            // Cogroup the lists of watched and liked genres for each user
            // There is one pair for each userid
            // The key is the userid. The value is a Tuple2 containing
            // the list of watched genres and the list of liked genres
            // the value contains the list of watched genres and the list of liked
genres
            JavaPairRDD<String,Tuple2<Iterable<String>,Iterable<String>>>
userWatchedLikedGenres = usersWatchedGenresRDD.cogroup(userLikedGenresRDD);

            // Filter the users with a misleading profile
            // For each user, the filter compares the list of watched genres and
the list of liked genres
            // The to two lists of one single user are "small".
            // Hence, the comparison can be executed in main memory
            JavaPairRDD<String,Tuple2<Iterable<String>,Iterable<String>>>
misleadingUsersListsRDD=userWatchedLikedGenres.filter(new
CheckMisleading(threshold));

            // Select only the userid of the users with a misleading profile
            JavaRDD<String> misleadingUsersRDD=misleadingUsersListsRDD.keys();

            // Store the result in the HDFS folder
            misleadingUsersRDD.saveAsTextFile(outputPathMisleading);

            // Close the Spark context
            sc.close();
      }
}
```

```
package it.polito.bigdata.spark.exercise2;

import org.apache.spark.api.java.function.Function;

@SuppressWarnings("serial")
public class SelectGenre implements Function<String, String> {

	@Override
	public String call(String line) {
		String[] fields=line.split(",");

		// fields[2] = genre
		return fields[2];
	}

}
```

```java
package it.polito.bigdata.spark.exercise2;

import org.apache.spark.api.java.function.PairFunction;

import scala.Tuple2;

@SuppressWarnings("serial")
public class UserOne implements PairFunction<String, String, Integer> {

	@Override
	public Tuple2<String, Integer> call(String line) {
		String[] fields=line.split(",");
		// fields[0]=userid

		// Return a pair (userid, 1)
		return new Tuple2<String, Integer>(fields[0], new Integer(1));
	}

}
```

```
package it.polito.bigdata.spark.exercise2;

import org.apache.spark.api.java.function.Function2;

@SuppressWarnings("serial")
public class Sum implements Function2<Integer, Integer, Integer> {

	@Override
	public Integer call(Integer value1, Integer value2) {
		return value1+value2;
	}

}
```

```
package it.polito.bigdata.spark.exercise2;

import org.apache.spark.api.java.function.Function;

import scala.Tuple2;

@SuppressWarnings("serial")
public class UserUseless implements Function<Tuple2<String, Integer>, Boolean> {

	private long numGenres;

	public UserUseless(long numGen)
	{
		this.numGenres=numGen;
	}

	@Override
	public Boolean call(Tuple2<String, Integer> userNumLikedGenres) {

		// Select the user if he/she likes more than 90% of the genres
		// The number of liked genres is in userNumLikedGenres._2();
		if ((double)userNumLikedGenres._2()>0.9*(double)numGenres)
			return true;
		else
			return false;
	}

}
```

```
package it.polito.bigdata.spark.exercise2;

import org.apache.spark.api.java.function.PairFunction;

import scala.Tuple2;

@SuppressWarnings("serial")
public class MovieUser implements PairFunction<String, String, String> {

	@Override
	public Tuple2<String, String> call(String line) throws Exception {

		String[] fields=line.split(",");
		Tuple2<String, String> movieUser=new
Tuple2<String,String>(fields[1],fields[0]);

		return movieUser;
	}

}
```

```
package it.polito.bigdata.spark.exercise2;

import org.apache.spark.api.java.function.PairFunction;

import scala.Tuple2;

@SuppressWarnings("serial")
public class MovieGenre implements PairFunction<String, String, String> {

	@Override
	public Tuple2<String, String> call(String line) {

		String[] fields=line.split(",");
		Tuple2<String, String> movieGenre=new
Tuple2<String,String>(fields[0],fields[2]);

		return movieGenre;
	}

}
```

```
package it.polito.bigdata.spark.exercise2;

import org.apache.spark.api.java.function.PairFunction;

import scala.Tuple2;

@SuppressWarnings("serial")
public class UserWatchedGenre implements PairFunction<Tuple2<String, Tuple2<String,
String>>, String, String> {

	@Override
	public Tuple2<String, String> call(Tuple2<String, Tuple2<String, String>>
userMovie) {

		// movieid - userid - genre
		Tuple2<String, String> movieGenre=
				new Tuple2<String, String>(userMovie._2()._1(),
userMovie._2()._2());

		return movieGenre;
	}

}
```

```
package it.polito.bigdata.spark.exercise2;

import org.apache.spark.api.java.function.PairFunction;

import scala.Tuple2;

@SuppressWarnings("serial")
public class UserLikedGenre implements PairFunction<String, String, String> {

	@Override
	public Tuple2<String, String> call(String line) throws Exception {

		String[] fields=line.split(",");
		Tuple2<String, String> userGenre=new
Tuple2<String,String>(fields[0],fields[1]);

		return userGenre;
	}

}
```

```
package it.polito.bigdata.spark.exercise2;

import java.util.ArrayList;

import org.apache.spark.api.java.function.Function;

import scala.Tuple2;

@SuppressWarnings("serial")
public class CheckMisleading implements Function<Tuple2<String,
Tuple2<Iterable<String>, Iterable<String>>>, Boolean> {

	double threshold;

	public CheckMisleading(double th)
	{
		this.threshold=th;
	}

	@Override
	public Boolean call(Tuple2<String, Tuple2<Iterable<String>,
Iterable<String>>> listWatchedLikedGenres) {

		// listWatchedLikedGenres
		// listWatchedLikedGenres._1() is the userid
		// listWatchedLikedGenres._2()._1() is the list of watched genres
		// listWatchedLikedGenres._2()._2() is the list of liked genres

		// Store in main memory the list of liked genres
		ArrayList<String> likedGenres=new ArrayList<String>();

		// Iterate over the liked genres
		for (String likedGenre: listWatchedLikedGenres._2()._2()) {
			likedGenres.add(likedGenre);
		}


		// Count how many of watched genres are in the liked genres
		int notLiked=0;
		// Count also the total number of watched movies
		int totalWatched=0;
		// Iterate over the watched genres
		for (String watchedGenre: listWatchedLikedGenres._2()._1()) {
			totalWatched++;
			// if the current genre is not in the liked ones
			// increment notLiked
			if (likedGenres.contains(watchedGenre)==false) {
				notLiked++;
			}
		}

		// Check if the number of watched movie with a genre that is
		// not in the liked ones is greater that threshold% of the
		// total number of watched movies
		if ((double)notLiked>threshold*(double)totalWatched)
			return true;
		else
			return false;
	}
```

```
}
```