Movie Recommender

Movie Recommender

Programming 3 Homework

Wali Ullah

WTW2SV

1 DESCRIPTION OF THE PROJECT

Movie Recommdner is a a DIY recommender engine for finding recommendations and information about current movies. The goal of the said project is to recommend movies based on different criteria which is further based on user ratings of those movies.

The idea is inspired by a website "twitflicks.com" which used to mine Twitter for tweets that included comments about current movies. These comments were then turned into ratings. Unfortunately, that website is no longer operational, hence I decided to revive that idea and build something similar to that.

For the project, I will be using another twitter-based source of data that's easier to parse and base recommendations off of that. The URL for the said source that provides live data is: *https://recsyswiki.com/wiki/Movietweetings*

MovieTweetings is a live, and always up-to-date dataset consisting of ratings on movies that were contained in wellstructured tweets on Twitter. While the typical datasets as Netflix, MovieLens, etc. are still popular in research, they are losing their relevancy as time goes by. The MovieTweetings dataset offers ratings on popular and contemporary movies, which can be useful for [user-centric] experiments and live demos of recommender systems. The URL for the said source that provides the dataset is:
https://github.com/sidooms/MovieTweetings

To use these ratings, I must get them, parse them, and then write a program which determines which movies should someone watch. As a user, you will also be able to make recommendations by filtering based on genre, or co-stars, or by any other criteria. This will help you further curate the recommendations for you as the program decides which genre you recommended the most, and which type of movies would suit your taste the best. Further goals include a swing-based GUI and a simple web-app for the said project.

2 External Libraries and Compilation Instructions

2.1 External Libraries

The project makes use of 2 external libraries:

1. edu.duke.FileResource
2. org.apache.commons.csv.*

The abovementioned library JARs can be found in the project folder, specifically at: *MovieRecommender/external jars*

2.2 Compilation Instructions

The project and GUI were built using IntelliJ IDEA, so it is recommended to compile/build the application JAR using the same IDE.

If you are interested in learning more about how the web-application works, here are the instructions:

1. Make a zip archive of following 3 classes*:
    a. RecommendationRunner
    b. OptimziedRatings
    c. TrueFilter (you can implement other filters by making necessary changes to the code)

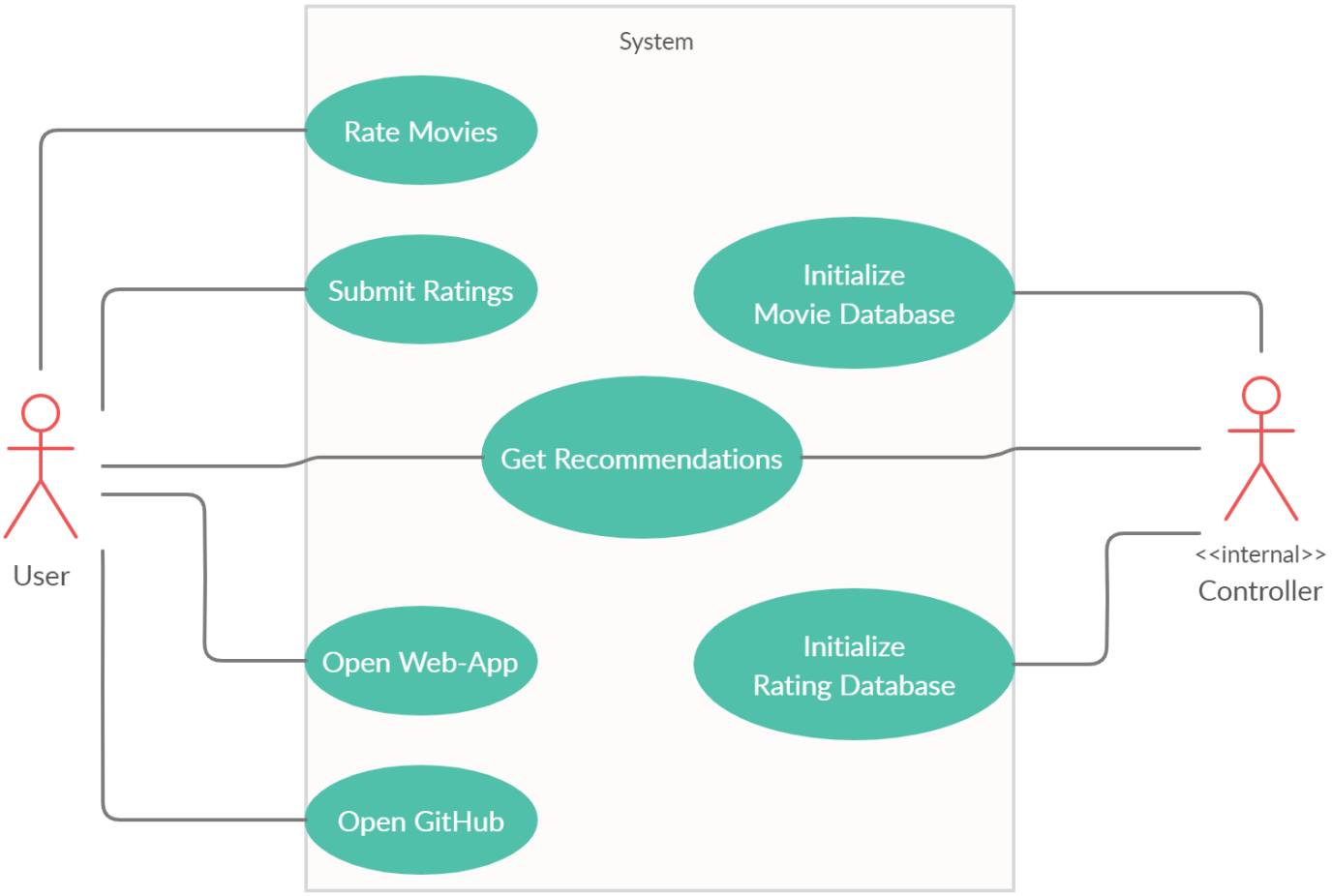
    *. Make sure you zip .class files (not .java files). The website only accepts .class files compiled with Java 10 and older versions.

2. Upload the zip archive at the following address:
    https://www.dukelearntoprogram.com//capstone/upload.php

3. Submit your zip archive and run the web application.

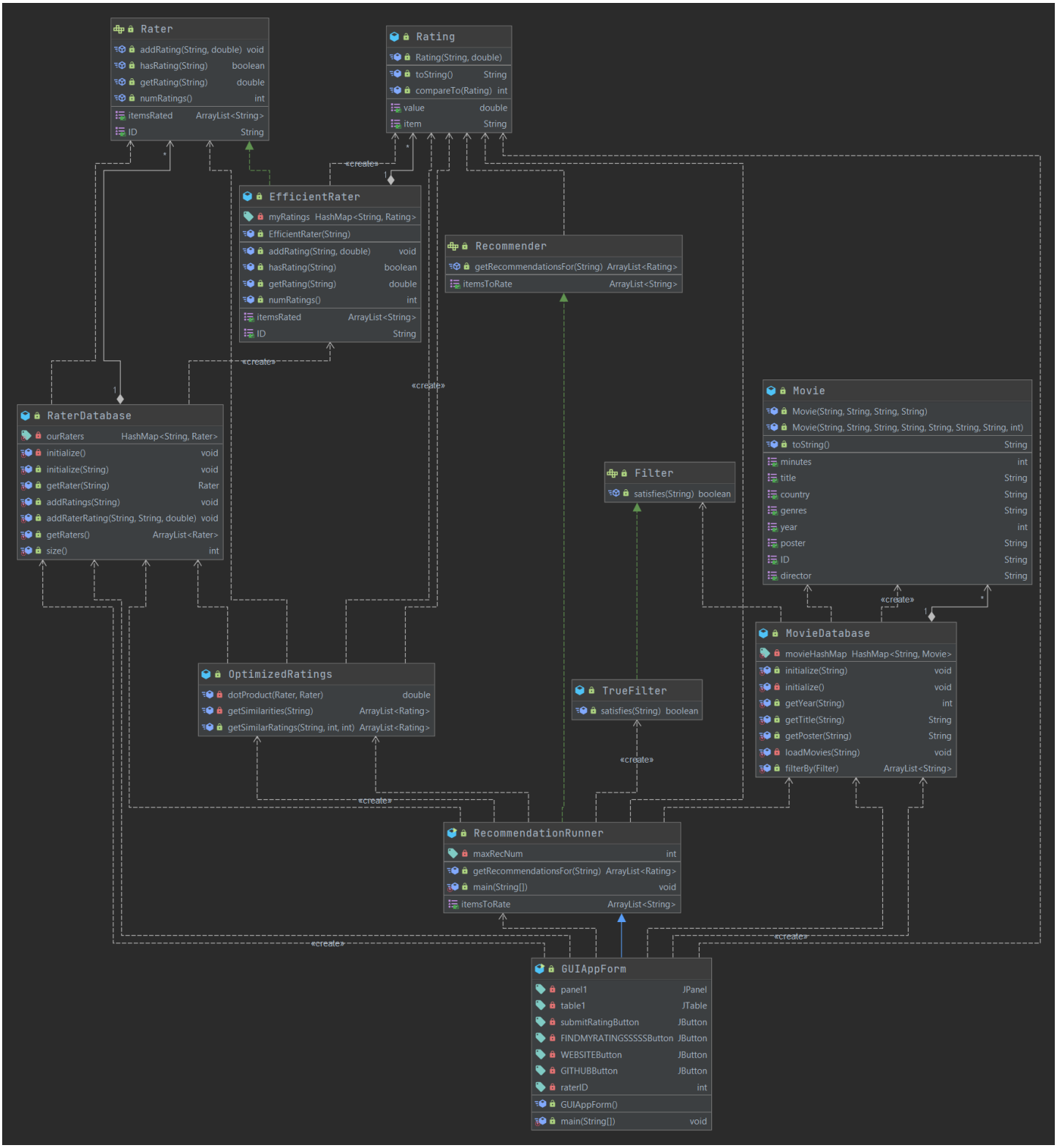
4. You can also include styling using <style> for the web application.

3 USE CASE DIAGRAM

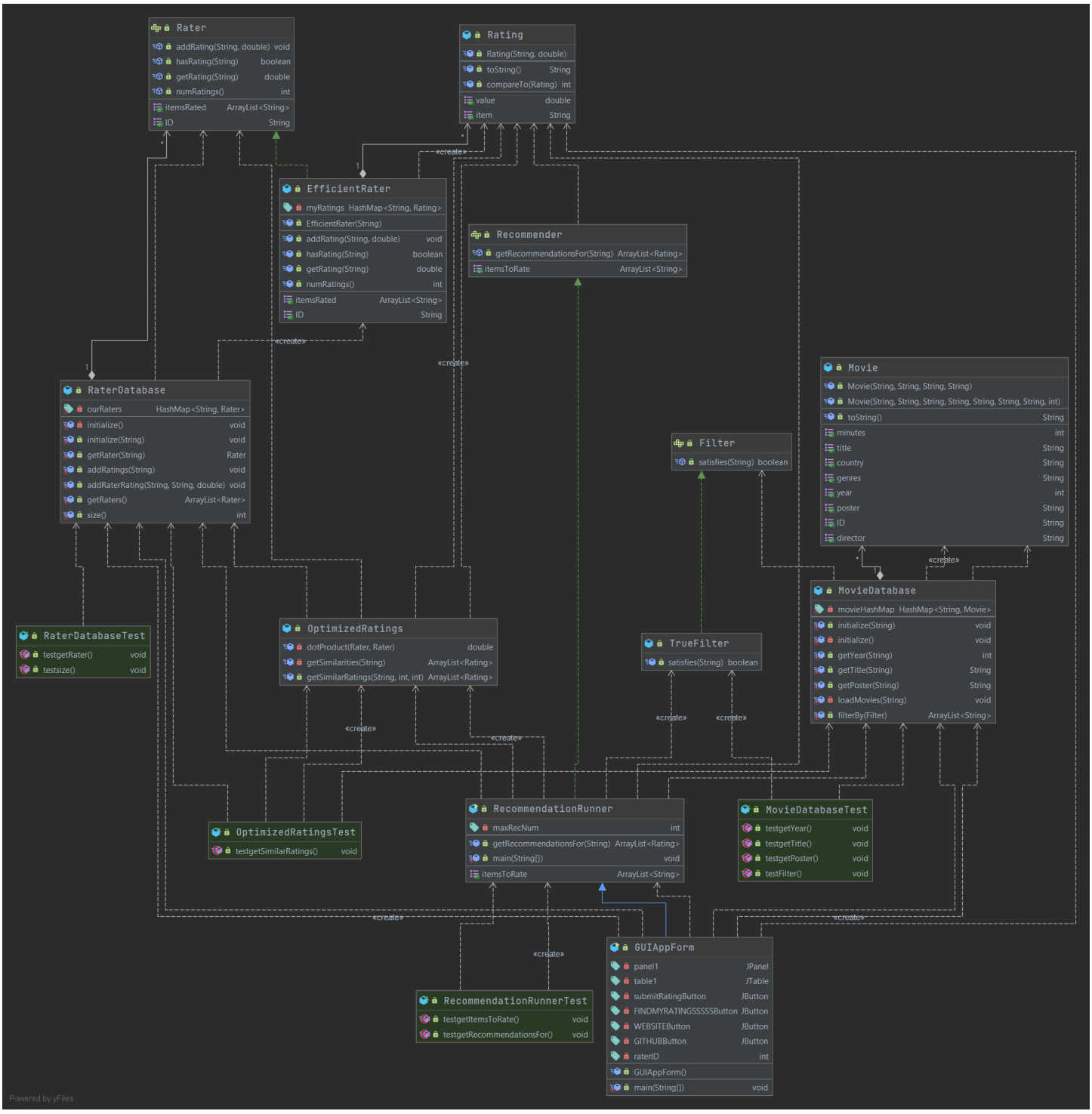


4 Structural description

4.1 Class Diagram



4.2 Class Diagram (with J-UNIT)

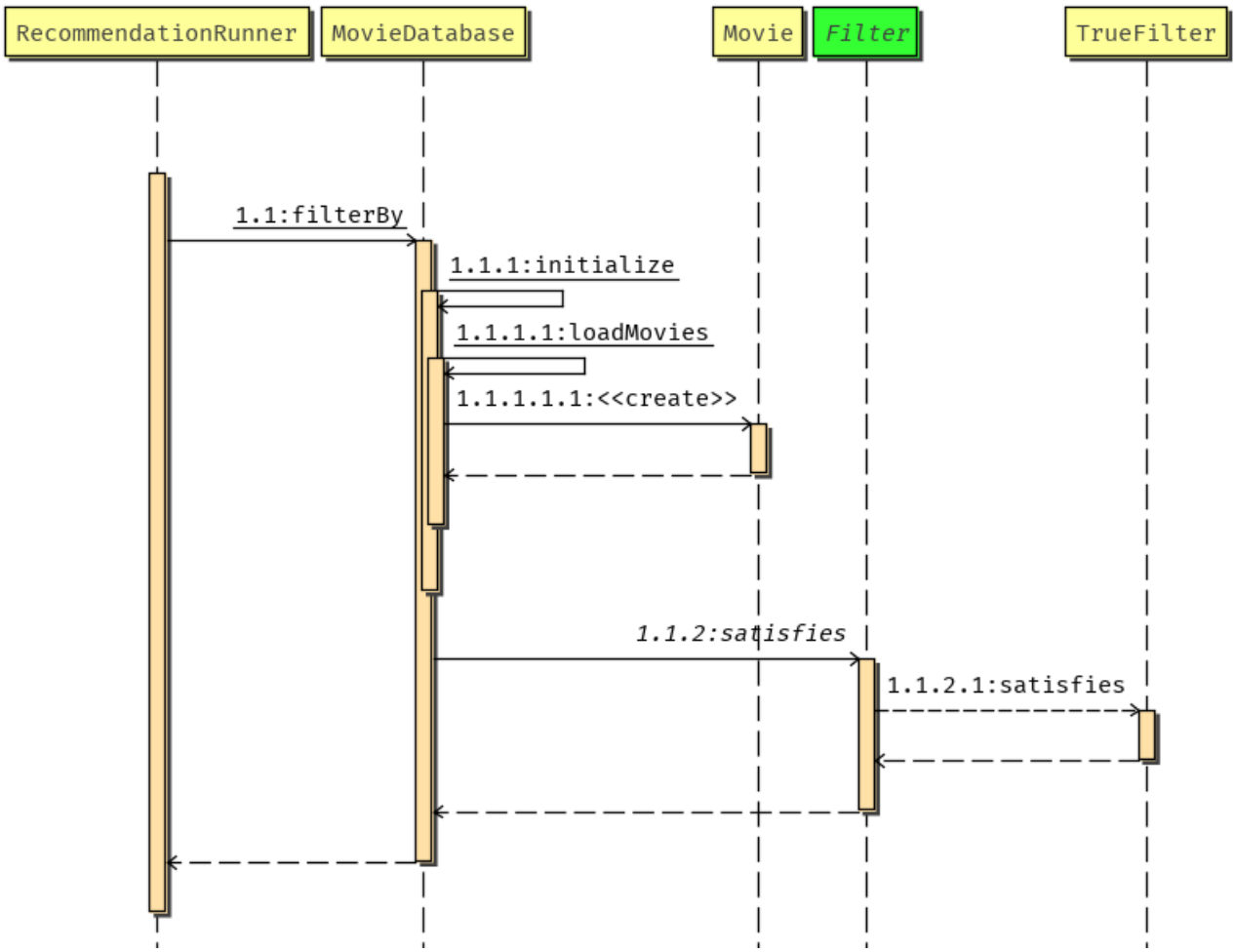


5 Behavioral description

5.1 Sequence Diagrams *(setters/getters not included)*

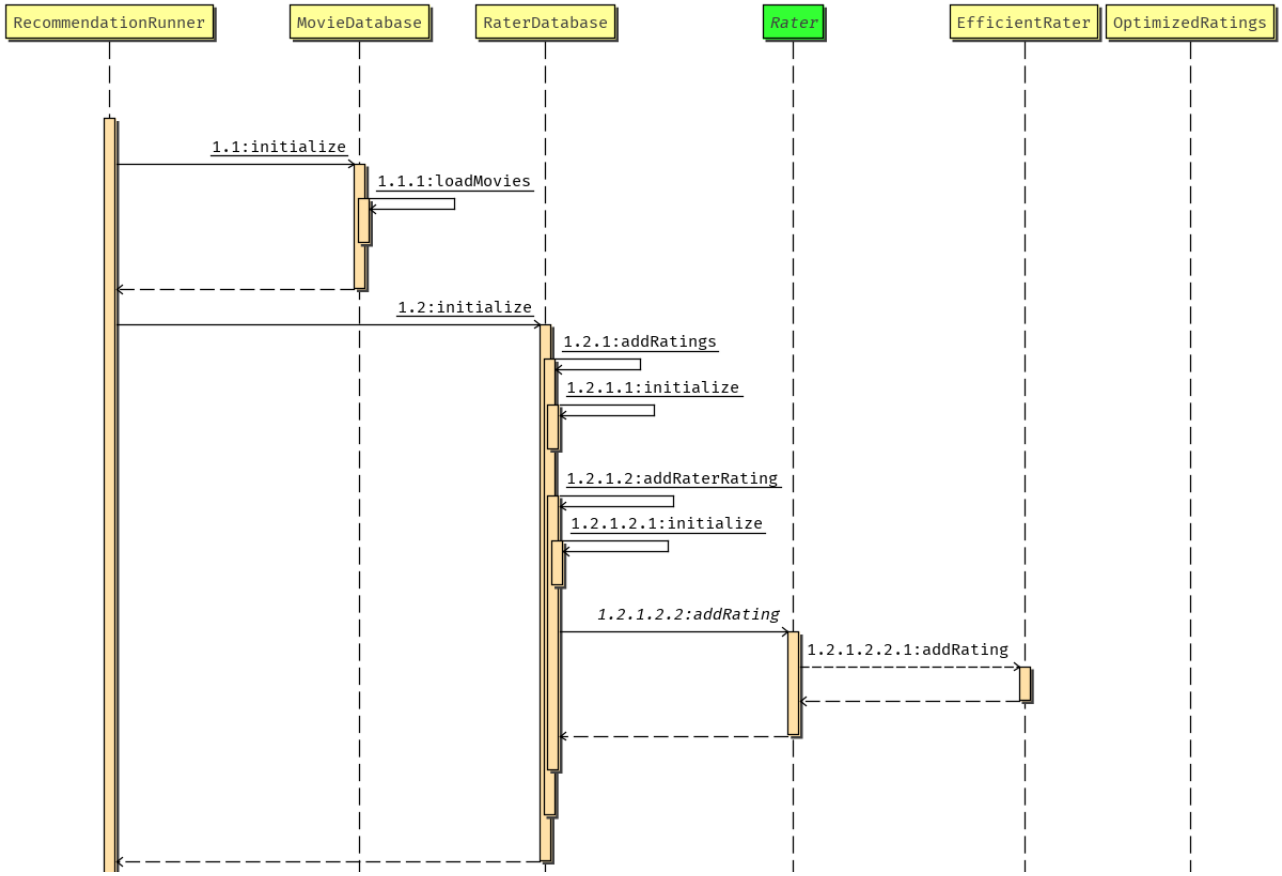
5.1.1

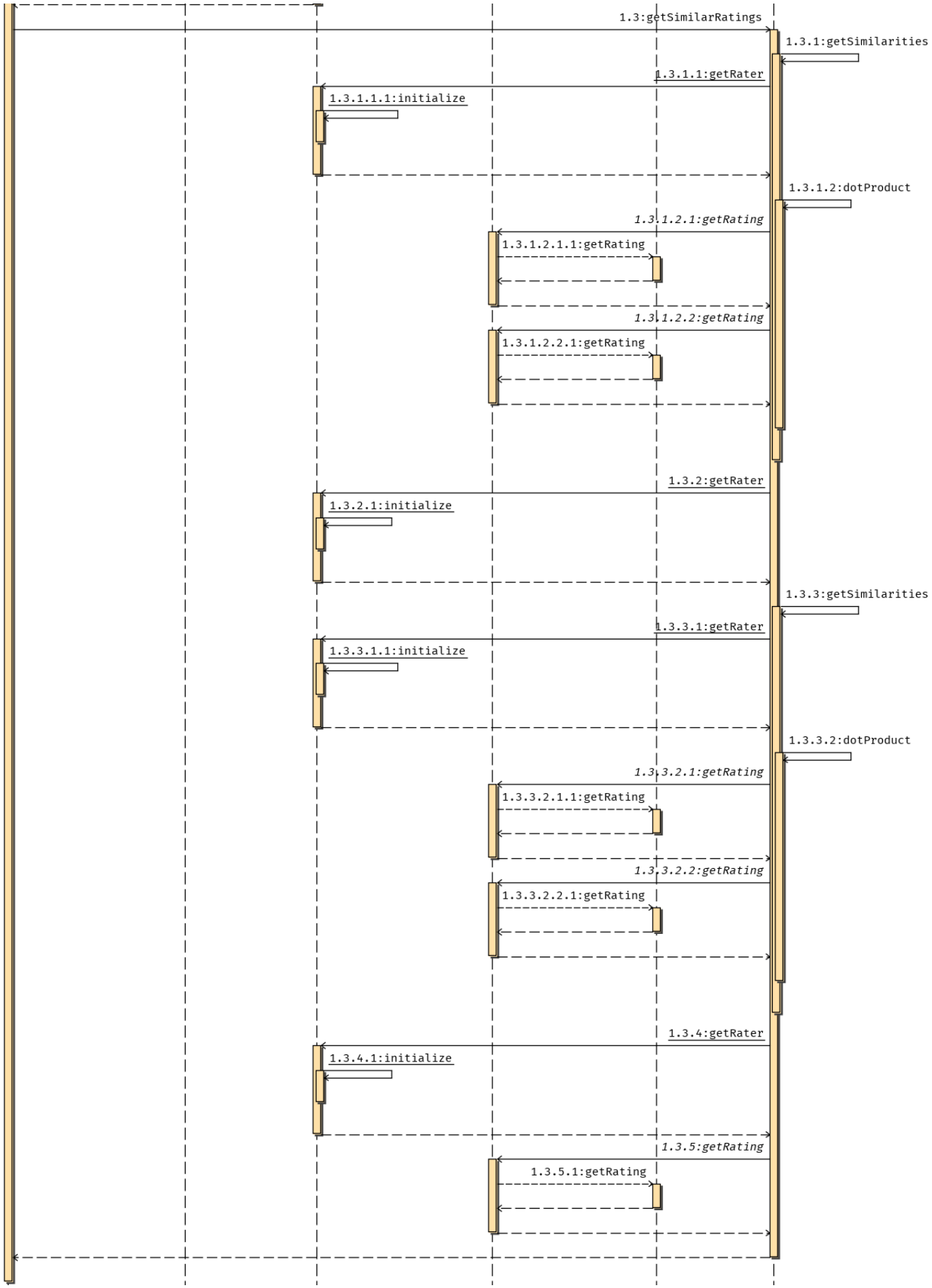
```
public ArrayList<String> getItemsToRate()
```



5.1.2

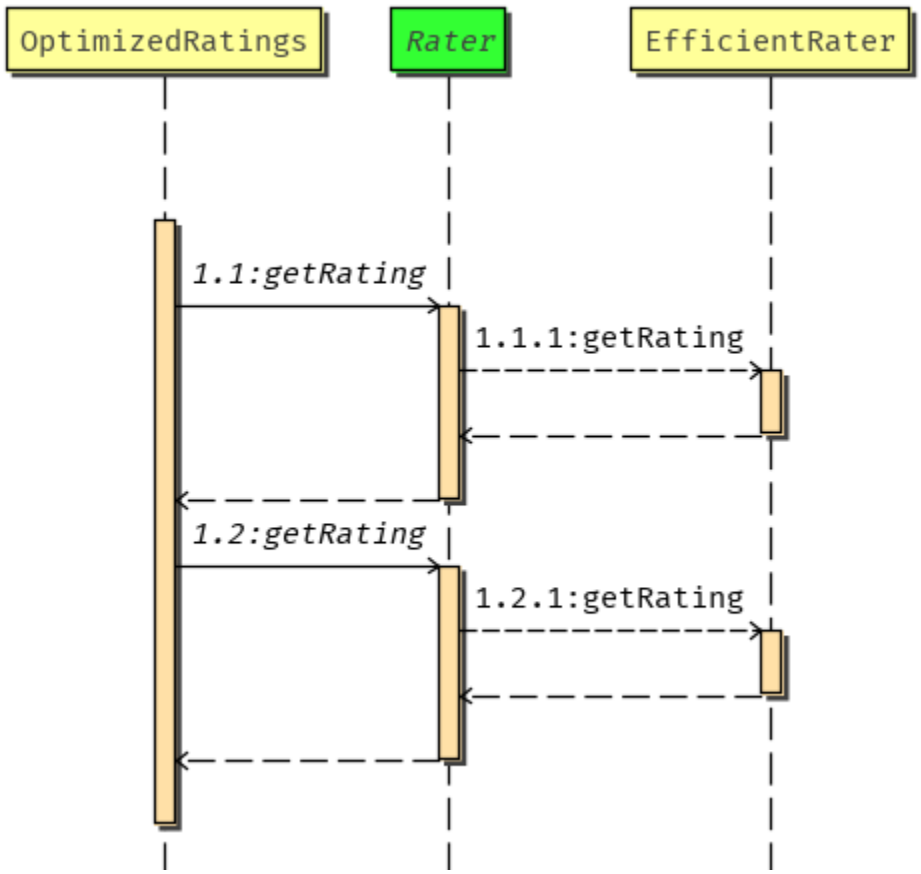
```
public ArrayList<String> getRecommendationsFor()
```




1.3:getSimilarRatings
1.3.1:getSimilarities
1.3.1.1:getRater
1.3.1.1.1:initialize
1.3.1.2:dotProduct
1.3.1.2.1:getRating
1.3.1.2.1.1:getRating
1.3.1.2.2:getRating
1.3.1.2.2.1:getRating
1.3.2:getRater
1.3.2.1:initialize
1.3.3:getSimilarities
1.3.3.1:getRater
1.3.3.1.1:initialize
1.3.3.2:dotProduct
1.3.3.2.1:getRating
1.3.3.2.1.1:getRating
1.3.3.2.2:getRating
1.3.3.2.2.1:getRating
1.3.4:getRater
1.3.4.1:initialize
1.3.5:getRating
1.3.5.1:getRating

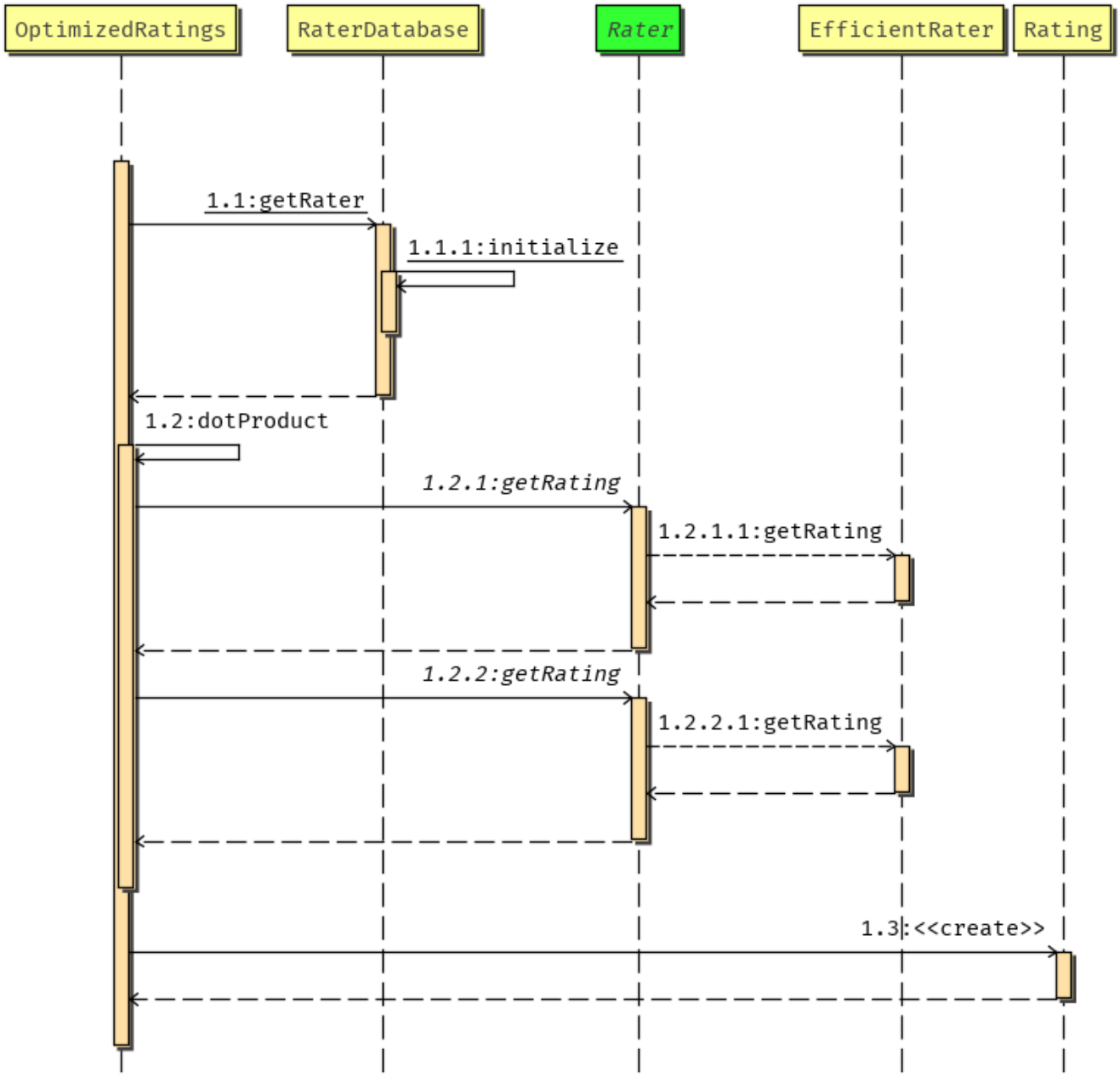
5.1.3

```
private double dotProduct(Rater me, Rater r)
```



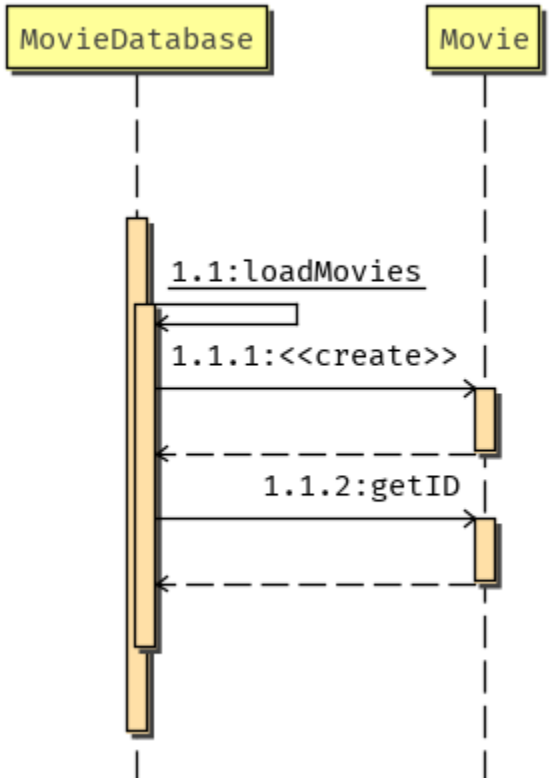
5.1.4

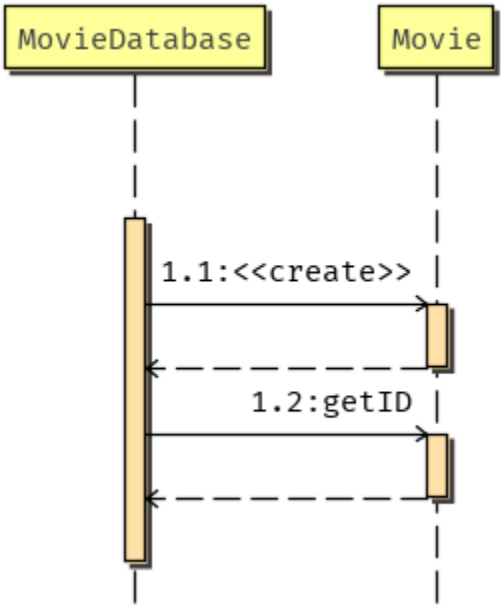
```
private ArrayList<Rating> getSimilarities(String raterId)
```



5.1.5

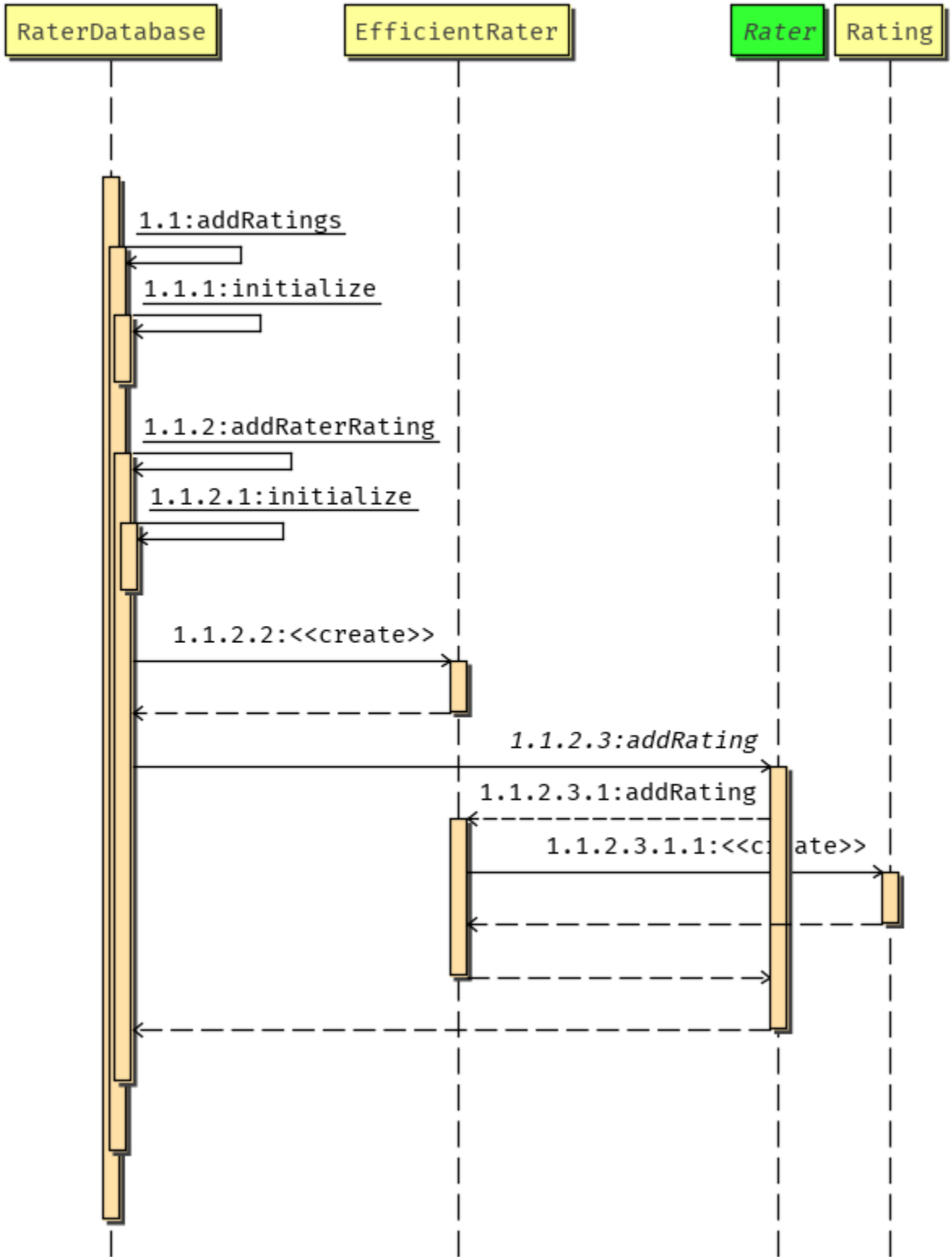
```
//... Use Case: Initialize Movie Database ...//
```





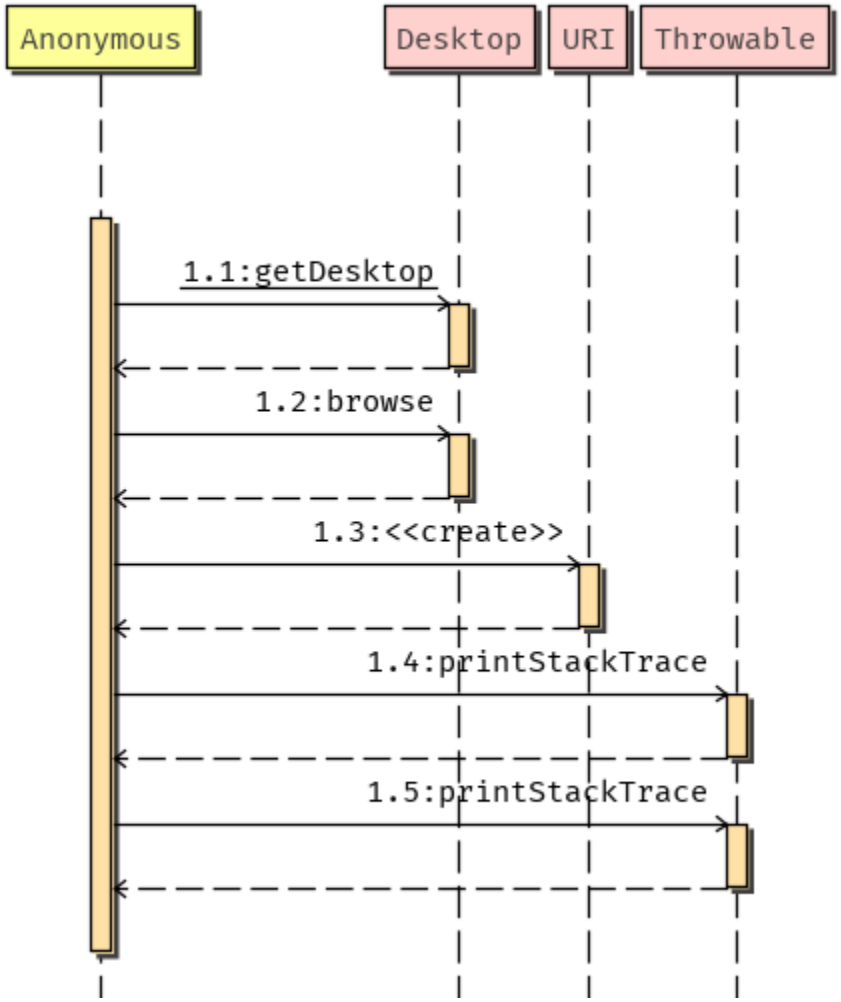
5.1.6

```
//... Use Case: Initialize Rating Database ...//
```



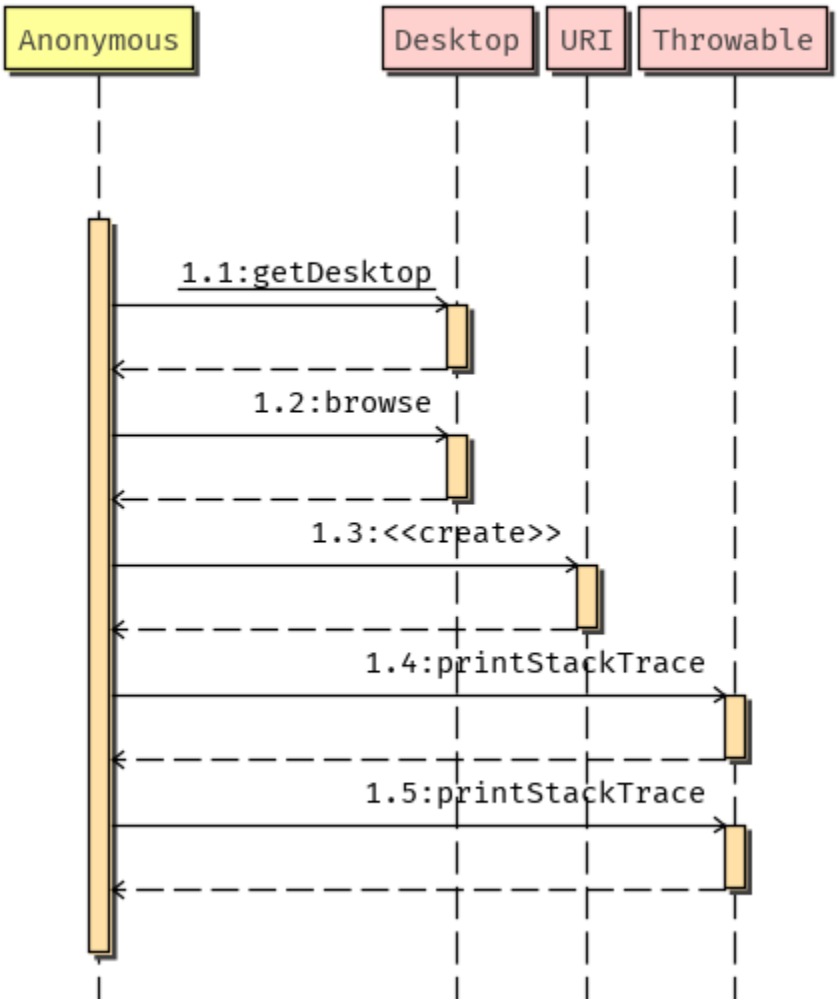
5.1.7

```
//... Use Case: Open GitHub ...//
```



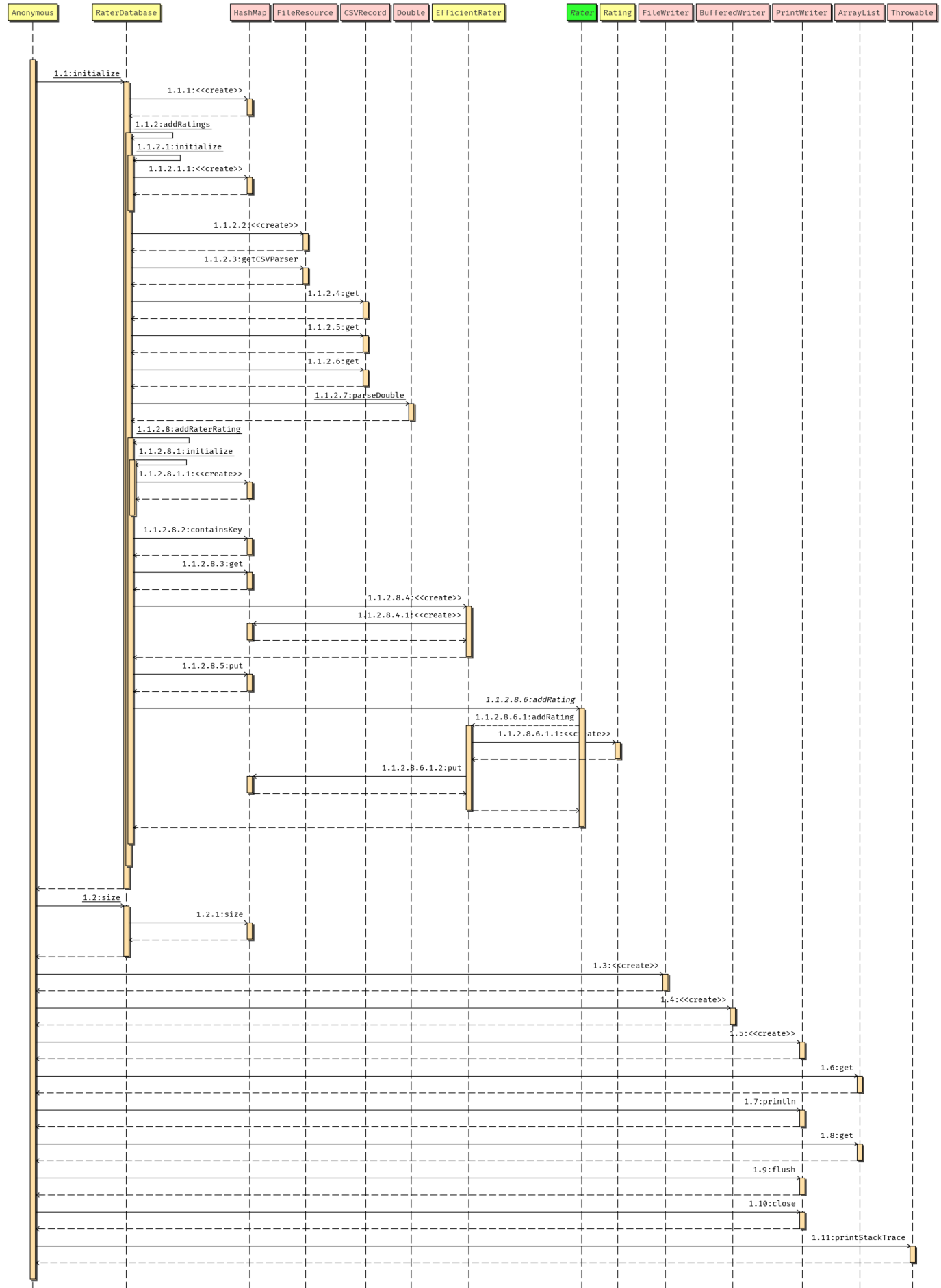
5.1.8

```
//... Use Case: Open Website ...//
```



5.1.9

//... Use Case: Submit Ratings ...//



6 UNIT TEST DESCRIPTION

6.1 RECOMMENDATION RUNNER TEST

6.1.1

```
@Test
public void testgetItemsToRate()
```

Description	Tests the method to make sure it returns 10 random movie IDs in an ArrayList.
Main success scenario	Test passes if returned value matches the expected value. It ensures that our Movie Database was loaded and parsed successfuly.
Alternate scenario	Test fails if returned value didn't match expected value. This could happen if: 1. Filepath for Movie Database file was wrong. 2. The method failed to parse the file/ file not found.

6.1.2

```
@Test
public void testgetRecommendationsFor()
```

Description	Tests the method to make sure it calculates and returns ratings in an ArrayList.
Main success scenario	Test passes if returned value matches the expected value. It ensures that our Movie Database and Rater Database was loaded and parsed successfuly. Furthermore, it also ensures that we calculated and made an ArrayList of expected ratings for movies to be recommended to the user.
Alternate scenario	Test fails if returned value didn't match expected value. This could happen if: 1. Filepath for Movie Database/ Rating Database file was wrong. 2. The method failed to parse the file/ file not found. 3. The method fails if Rater ID was not found in the Rating Database, 4. The rater didn't rate/submit any ratings.

6.2 OPTIMIZED RATINGS TEST

6.2.1

```
@Test
public void testgetSimilarRatings()
```

Description	Tests the method to make sure it returns ArrayList of similar Ratings.
Main success scenario	Test passes if returned value matches the expected value. It ensures that our method found similar ratings to the user.
Alternate scenario	Test fails if returned value didn't match expected value. This could happen if: 1. Filepath for Rater Database file was wrong. 2. The method failed to parse the file/ file not found. 3. The method fails if Rater ID was not found in the Rating Database.

6.3 Movie Database Test

6.3.1

```
@Test
public void testFilter()
```

Description	Tests the method to make sure our Filter is implemented correctly.
Main success scenario	Test passes if returned value matches the expected value. It ensures that our method returns movies curated under specified Filter.
Alternate scenario	Test fails if returned value didn't match expected value. This could happen if: 1. Filter was implemented unsuccessfully.

6.4 Rating Database Test

6.4.1

```
@Test
public void testgetRater()
```

Description	Tests the method to make sure that our Rater Database returns expected Rater ID, and Movie ID associated to that Rater.
Main success scenario	Test passes if returned value matches the expected value. It ensures that our method found the rater in the Database using the ID.
Alternate scenario	Test fails if returned value didn't match expected value. This could happen if: 1. Filepath for Rater Database file was wrong. 2. The method failed to parse the file/ file not found. 3. The method fails if Rater ID was not found in the Rating Database, 4. The rater didn't rate/submit any ratings.

6.4.2

```
@Test
public void testsize()
```

Description	Tests the method to make sure our Rater Database was parsed and stored in ArrayList successfully.
Main success scenario	Test passes if returned value matches the expected value.
Alternate scenario	Test fails if returned value didn't match expected value. This could happen if: 1. Filepath for Rater Database file was wrong. 2. The method failed to parse the file/ file not found.