Programming and Data Structures
Programming Assignment 10: Implementing Data Structures (Hash Table)

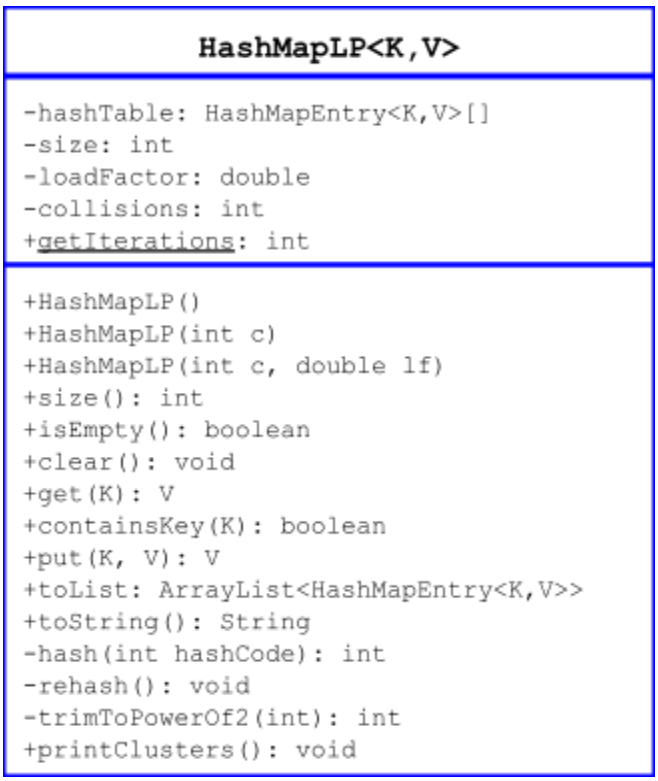
Student Learning Outcomes

Students should demonstrate the following skills:

1. Modify the implementation of the hash table class **`HashMap<K,V>`** to use linear probing to resolve the collisions
2. Test the implemented hash table class in a test program

Part 1: Implementing a hashtable that uses linear probing to resolve the collisions

Use the implementation of the class **`HashMap<K,V>`** from ALA 10 to implement a hash table that uses linear probing to resolve the collisions (as described in the UML below)

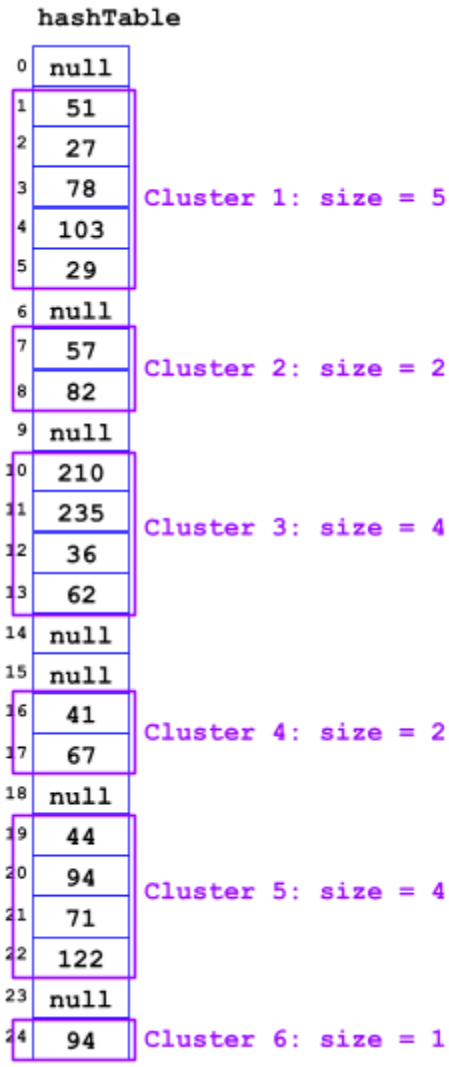


The default load factor should be 0.5.

You should modify the following methods: **`HashMap()`**, **`HashMap(int c)`**, **`clear()`**, **`get(K)`**, **`put(K,V)`**, **`toList()`**, **`toString()`**, and **`rehash()`**.

The public static variable **`getIterations`** should be used to count the number of iterations in the **`get`** method. The private variable **`collisions`** should be used to count the total number of collisions in the hashmap. Count the collisions in the **`put`** method when they occur for the first time.

`printClusters()` should print the capacity of the hashTable, the total number of collisions, the number of clusters formed in the hashtable and the size of the largest and the smallest cluster. See the example of a hash table below for the number of clusters and their size.

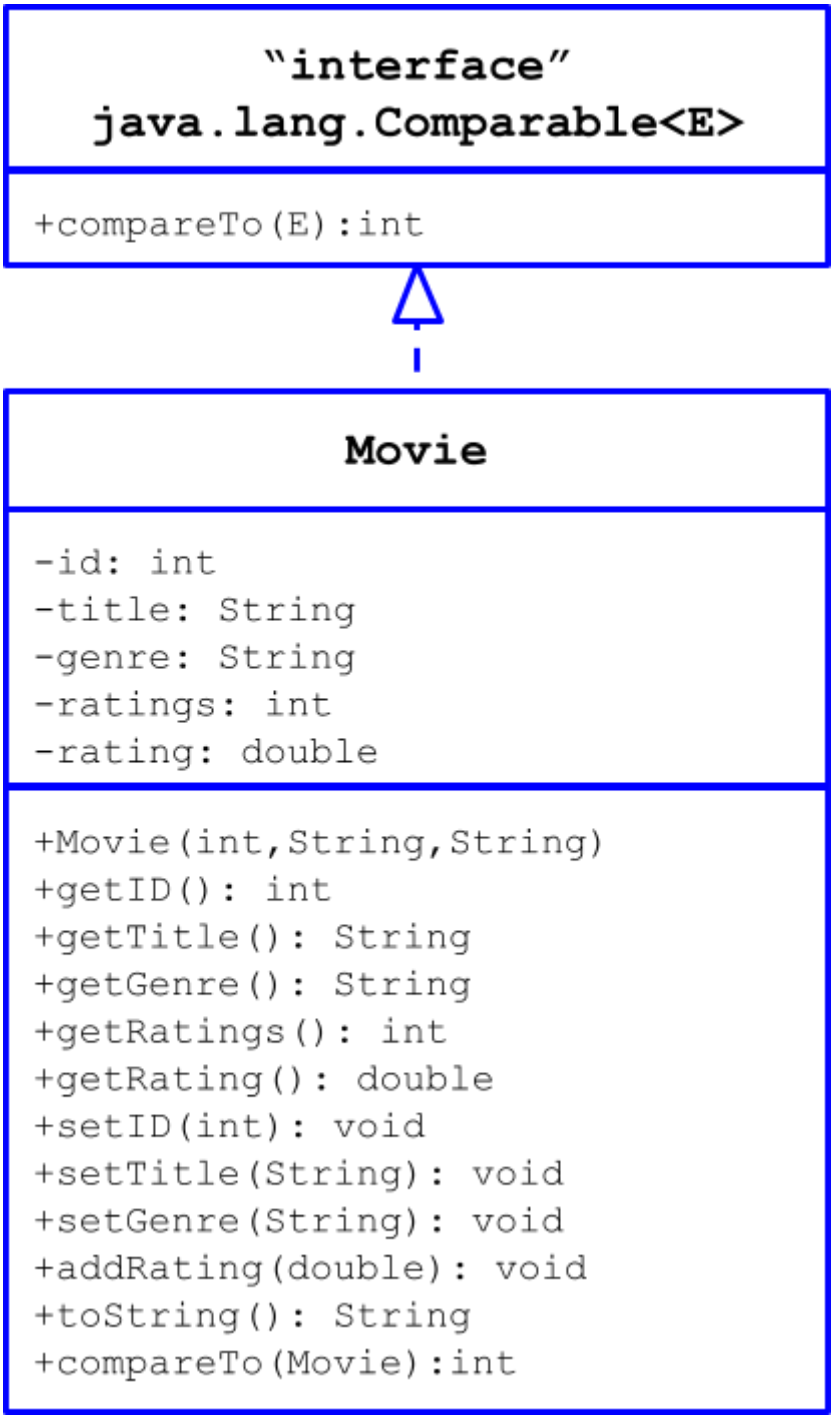



```
Hashtable capacity:           25
Number of clusters:           6
Size of the largest cluster:  5
Size of the smallest cluster: 1
```

Part 2: Create the datatype of the dataset used to test the implemented hashtable

---

Implement the class **`Movie`** as described in the UML diagram below.



`ratings` is the number of users who rated the movie

`rating` is the average of all the ratings entered by the users (out of 5)

Define the natural ordering for type **`Movie`** to order movies by the number of ratings

`addRating` should increment **`ratings`** and calculate the new average **`rating`**

Part 3: Test the implemented hashtable

Write a test program to perform the following:

1. Read the provided csv (comma separated values) file **`movies.csv`** to populate a hashmap of type **`<Integer, Movie>`** with the movie information. The initial capacity of the hash map should be 63,000 (the number of movies is 62,423). The key should be the movie id and the value is the **`Movie`** object. The file **`movies.csv`** contains a list of movies where each line contains the movie attributes, id, title, and genre separated by a comma.
2. Once you have inserted the list of **`Movie`** objects in the hashmap, read the file **`ratings.csv`** which is a list of ratings with the id of the user who rated the movie, the movie id, the rating entered for the movie, and the timestamp of the rating. You only need to read the movie id and the rating. You can ignore the information about the user id and the timestamp. Update the movie hashmap to add the ratings to each movie.
3. Lookup the following list of movie ids in the hashmap and display the result of the search (see the sample program output) including the number of iterations it takes the method get to find these movies:
    a. Movie ids: {**`1544, 2156, 31349, 3048, 4001, 356, 5672, 6287, 25738, 26`**}
4. Extract the list of movies from the hashMap using the method **`toList()`** and sort the list using the natural ordering of the class **`Movie`**
5. Display the top ten highest rated movies and the bottom ten lowest rated movies with minimum 10,000 ratings

6. Finally, invoke the method **`printClusters()`** on the hashmap and discuss the number of collisions and clusters, and the largest cluster.

Submit the files **`HashMapLP.java`**, **`Movie.java`**, and **`Test.java`** on Github

Sample output of the program

Results of the search in the hashmap

Id	Title	Number of ratings	Average rating	Iterations(get)
1544	"Lost World: Jurassic Park	15928	3.0	1
2156	"Best Man	56	3.2	1
31349	I Married a Witch (1942)	87	3.3	1
3048	Under the Rainbow (1981)	305	2.5	1
4001	Code of Silence (1985)	151	2.7	1
356	Forrest Gump (1994)	81491	4.0	1
5672	Pokemon 4 Ever (a.k.a. Pokémon 4: The Movie) (2002)	494	1.4	1
6287	Anger Management (2003)	5968	2.9	1
25738	"Last of the Mohicans	61	3.5	1
26	Othello (1995)	2549	3.6	1

Bottom Ten movies with at least 10,000 ratings

Id	Title	Number of ratings	Average rating
2701	Wild Wild West (1999)	11723	2.3
173	Judge Dredd (1995)	14758	2.6
435	Coneheads (1993)	13194	2.6
160	Congo (1995)	13853	2.6
19	Ace Ventura: When Nature Calls (1995)	21552	2.6
44	Mortal Kombat (1995)	10629	2.7
784	"Cable Guy	14457	2.7
3785	Scary Movie (2000)	11473	2.7
2054	"Honey	16092	2.7
432	City Slickers II: The Legend of Curly's Gold (1994)	11729	2.7

Top Ten movies with at least 10,000 ratings

Id	Title	Number of ratings	Average rating
318	"Shawshank Redemption	81482	4.4
858	"Godfather	52498	4.3
50	"Usual Suspects	55366	4.3
1221	"Godfather: Part II	34188	4.3
2019	Seven Samurai (Shichinin no samurai) (1954)	13367	4.3
527	Schindler's List (1993)	60411	4.2
1203	12 Angry Men (1957)	16569	4.2
904	Rear Window (1954)	20162	4.2
2959	Fight Club (1999)	58773	4.2

Hash table characteristics

```
HashTable capacity: 131072
Total number of collisions: 16997
Number of clusters: 36556
Size of the largest cluster: 14729
Size of the smallest cluster: 1
```

The following rubric is used for grading:

Objective	**How is the objective achieved?**	**Points**
Write Java programs with no errors	No compiler/runtime errors	5 pts
Document Java programs using Javadoc	Javadoc comments	5 pts
Write Java programs that run correctly	Correct Program Output	15 pts
Implement the class HashMapLP<K,V>	Class **`HashMap<K,V>`** **`HashMap()`** **`HashMap(int c)`** **`clear()`** **`get(K)`** **`put(K,V)`** **`toList()`** **`toString()`** **`rehash()`** **`printClusters`**	45 pts **`0.5`** **`0.5`** **`2`** **`10`** **`10`** **`4`** **`4`** **`4`** **`10`**
Test the HashMapLP class	Class **`Movie`** Class **`Test`**	10 pts 15 pts
Discuss the hash table characteristics	Discussion on the number of clusters and the largest cluster	5 pts
Total		100 points