

# Homework M14: Hashing

**Due** May 23 at 11:59pm      **Points** 100      **Questions** 17  
**Available** until May 30 at 11:59pm      **Time Limit** None

## Instructions

Review the [Homework FAQ](#) page for details about submitting homework. In this homework, you will:

- trace hashing, HashSets, and HashMaps
- write code related to hashing



## Homework Files

Below is the driver/tester program. I strongly recommend using this to test your code before submitting. You can ignore the *test methods* at the end of the file.

[HomeworkM14Files.zip](#)  ([https://ccsf.instructure.com/courses/47904/files/7254662/download?download\\_frd=1](https://ccsf.instructure.com/courses/47904/files/7254662/download?download_frd=1))

This quiz was locked May 30 at 11:59pm.

## Attempt History

	Attempt	Time	Score
LATEST	<a href="#">Attempt 1</a>	7,010 minutes	105.13 out of 100

Score for this quiz: **105.13** out of 100


Submitted May 18 at 2:35pm

This attempt took 7,010 minutes.

## Coding Questions

Complete the Student class provided in this week's homework files. Override the equals method and the hashCode method. Take into account the student's id, first name, last name, and the variable that represents whether they have paid their fees.

1. Write both methods (equals and hashCode), including the **complete** method headers.
2. Make sure your methods satisfy the [equals and hashCode contracts](https://docs.oracle.com/javase/9/docs/api/java/lang/Object.html) [\\_ \(https://docs.oracle.com/javase/9/docs/api/java/lang/Object.html\)](https://docs.oracle.com/javase/9/docs/api/java/lang/Object.html), which are on the API linked here and also in the lecture notes.
3. You can test your methods using the driver.

Important: Use the Student class from the [homework files](#)   
([https://ccsf.instructure.com/courses/47904/files/7254662/download?download\\_frd=1](https://ccsf.instructure.com/courses/47904/files/7254662/download?download_frd=1)) , not the lecture files.

## Question 1

10 / 10 pts

Upload your revised Student.java with the equals and hashCode methods.

 [Student.java \(https://ccsf.instructure.com/files/7879016/download\)](https://ccsf.instructure.com/files/7879016/download)

Review the provided hash table class. The class is designed to store voter data in a hashtable using **separate chaining (open buckets)**.

Voters are uniquely identified by their voterID.

Implement **three** methods to complete this class:

```
public int getHashTableLocation(int voterID)
```

- takes in a voterID and returns the location in the hash table where that voter will be placed
- normally this method would be private, but I made it public so you can use it to check your output in the driver program
- this method might be very simple- that's okay!

```
public boolean addVoter(Voter voterToAdd)
```

- adds a new Voter to the hashtable and returns a status of whether or not the add was successful
- this method returns a boolean because of the extra credit; if not completing the extra credit, this will always return true

```
public Voter getVoter(int voterID)
```

- takes in a voterID and returns the Voter object associated with that id, or null if a voter for that id is not in the table

Use the driver program to test your code.

## Optional Extra Credit: 10 points

In your addVoter method, make it so the hashtable does **not** accept duplicate voters. Add a comment to your method such as "// completing extra credit."

If completing the extra credit, write only one addVoter method that accounts for duplicates, not two separate methods.

**Question 2**

**45 / 35 pts**

Upload the completed VoterHashTableSeparateChaining .java file.

↓ [VoterHashTableSeparateChaining.java](https://ccsf.instructure.com/files/7879013/download)  
(<https://ccsf.instructure.com/files/7879013/download>)

i saw your note about the extra credit, but it appears to work for all cases to me! i won't change your score, but just out of curiosity- what failed case were you thinking of?

## Multiple Choice Questions

For the next set of questions, you will evaluate a hash table that is being used to store street addresses (perhaps for a direct mailing system).

- The key is the street address.
- The value contains other information (e.g., name, zip code, age, etc.).
- Here is the hash function information:

```
h(street address) = numeric part of the address  
location = h(street address) % table size
```

### Question 3

2 / 2 pts

The table size is 5. The table uses open addressing and linear probing to resolve collisions.

0	empty/null
1	81 Manor
2	76 Orville
3	empty/null
4	empty/null

After the following statement is executed, in what position is "73 Treble"?

```
add("73 Treble");
```

**Correct!**

☒ 3

☐ 4

☐ 2

☐ 0

☐ 1

#### Question 4

2 / 2 pts

The table size is 5. The table uses open addressing and linear probing to resolve collisions.

0	empty/null
1	81 Manor

2	76 Orville
3	empty/null
4	empty/null

After the following statement is executed, in what position is "11 Mason"?

```
add("11 Mason");
```

**Correct!**

☒ 3

☐ 1

☐ 2

☐ 0

☐ 4

### Question 5

2 / 2 pts

The table size is 5. The table uses open addressing and linear probing to resolve collisions.

0	empty/null
1	81 Manor
2	76 Orville
3	empty/null
4	empty/null

After the following statement is executed, what is contained in location

1?

```
remove("81 Manor");
```

☐ 81 Manor☐ empty/null☐ 76 Orville☒ available**Correct!****Question 6****7 / 7 pts**

You have a table of size 7. The table uses open addressing and linear probing to resolve collisions.

Select what value is in each position of the table after inserting the following keys in this order.

22 Irving

49 Ocean

7 California

8 Greenwich

78 Mason

58 Judah

32 Brannan

**Correct!****Position 0**

49 Ocean

**Correct!****Position 1**

**Correct!**

22 Irving

**Position 2**

7 California

**Correct!****Position 3**

8 Greenwich

**Correct!****Position 4**

78 Mason

**Correct!****Position 5**

58 Judah

**Correct!****Position 6**

32 Brannan



Other Incorrect Match Options:

- empty/null
- available

**Question 7****7 / 7 pts**

You have a table of size 7. The table uses open addressing and linear probing to resolve collisions.

After a series of insertion, the table looks like this:

0	700 Parkway
1	15 Junipero Serra
2	85 Capistrano
3	135 Great Highway
4	940 Main Street
5	778 Kirkham



89 California

You are now trying to find each address.

How many locations do you look in before finding each address listed below? Or, if the address is not in the table, how many locations do you look in before you know the address is not in the table?

(For example, if I searched positions 0, 1, 2, 3, and 4 before finding 940 Main Street, then I had to look in 5 locations.)

Correct!

700 Parkway

1



Correct!

15 Junipero Serra

1



Correct!

85 Capistrano

2



Correct!

135 Great Highway

2



Correct!

940 Main Street

3



Correct!

778 Kirkham

5



Correct!

89 California

2



Correct!

350 Powell

7



Other Incorrect Match Options:

- 4
- 6

**Question 8****2 / 2 pts**

You have a table of size 7. The table uses open addressing and linear probing.

After a series of insertion, the table looks like this:

0	700 Parkway
1	15 Junipero Serra
2	85 Capistrano
3	135 Great Highway
4	940 Main Street
5	778 Kirkham
6	89 California

You remove the key 940 Main Street. How many locations will you now need to look in to determine that 350 Powell is not in the table?

**Correct!****Correct Answers**

7 (with margin: 0)

**Question 9****0 / 2 pts**

The table size is 5. The table uses open addressing and linear probing to resolve collisions.

0	available
1	91 Parker

2	46 Franklin
3	available
4	11 Howard

How many seeks will it take to determine that "12 Main" is not in the table?

You Answered

3

Correct Answers

5 (with margin: 0)

### Question 10

7 / 7 pts

You have a table of size 11. The table uses open addressing and linear probing to resolve collisions.

Select what value is in each position of the table after inserting the following keys in this order.

- 22 Irving
- 49 Ocean
- 7 California
- 8 Greenwich
- 78 Mason
- 58 Judah
- 32 Brannan

Correct!

Position 0

22 Irving



**Correct!****Position 1**

78 Mason

**Correct!****Position 2**

empty/null

**Correct!****Position 3**

58 Judah

**Correct!****Position 4**

empty/null

**Correct!****Position 5**

49 Ocean

**Correct!****Position 6**

empty/null

**Correct!****Position 7**

7 California

**Correct!****Position 8**

8 Greenwich

**Correct!****Position 9**

empty/null

**Correct!****Position 10**

32 Brannan



Other Incorrect Match Options:

- available

**Question 11****6.13 / 7 pts**

You have a table of size 11. The table uses open addressing and linear probing to resolve collisions.

After a series of insertion, the table looks like this:

0	available
1	89 California
2	null/empty
3	135 Great Highway
4	15 Junipero Serra
5	940 Main Street
6	null/empty
7	700 Parkway
8	85 Capistrano
9	778 Kirkham
10	available

You are now trying to find each address.

How many locations do you look in before finding each address listed below? Or, if the address is not in the table, how many locations do you look in before you know the address is not in the table?

Correct!

**700 Parkway**

1



Correct!

**15 Junipero Serra**

1



Correct!

**85 Capistrano**

1



Correct!

**135 Great Highway**

1



Correct!

**940 Main Street**

	<input type="text" value="1"/>
<b>Correct!</b>	<div>778 Kirkham</div> <div><input type="text" value="2"/></div>
<b>Correct!</b>	<div>89 California</div> <div><input type="text" value="1"/></div>
<b>Not Answered</b>	<div>350 Powell</div> <div><input type="text" value="2"/></div>

**Correct Answer**      5

Other Incorrect Match Options:

- 4
- 6
- 7
- 3

**Question 12****7 / 7 pts**

You have a table of size 7. The table uses separate chaining.

Select what position each value is in after inserting the following keys in this order.

- 22 Irving
- 49 Ocean
- 7 California
- 8 Greenwich
- 78 Mason
- 58 Judah

	32 Brannan	
Correct!	22 Irving	Position 1 ▾
Correct!	49 Ocean	Position 0 ▾
Correct!	7 California	Position 0 ▾
Correct!	8 Greenwich	Position 1 ▾
Correct!	78 Mason	Position 1 ▾
Correct!	58 Judah	Position 2 ▾
Correct!	32 Brannan	Position 4 ▾

Other Incorrect Match Options:

- Position 3
- Position 5
- Position 6

### Question 13

2 / 2 pts

What is printed?

```
Set<Integer> numberSet = new HashSet<Integer>();  
numberSet.add(1);  
numberSet.add(2);  
numberSet.add(3);
```

```
numberSet.add(1);  
System.out.println(numberSet.size());
```

**Correct!**☐ 4☒ 3☐ 5☐ none of these is correct☐ 0☐ 2☐ 1**Question 14****2 / 2 pts**

When the code completes, what is stored in the hashtable below with the key, "123"?

```
Map<String,String> stringMap = new HashMap<String, String>();  
stringMap.put("123", "key1");  
stringMap.put("123", "key2");
```

☐

none of these is correct because an exception will be thrown when the code runs

☐ key1☐ key1 -> key2



**Correct!**

- ☐ 123
- ☐ null
- ☐ key2 -> key1
- ☒ key2

**Question 15****2 / 2 pts**

What is the result of the following code?

```
Map<String,String> stringMap = new HashMap<String, String>();  
stringMap.put("123", "key1");  
System.out.println(stringMap.get("456"));
```

- ☐ false is printed
- ☐ an exception or error is thrown
- ☐ none of these is correct
- ☒ null is printed

**Correct!****Question 16****2 / 2 pts**

What values will be printed? (Note: I am **not** asking about the formatting of the output, just which values will be displayed.)

```
Map<String, Integer> numberMap = new HashMap<String, Integer>();
```

```
numberMap.put("A", 1);  
numberMap.put("B", 2);  
numberMap.put("C", 3);  
System.out.println(numberMap.keySet());
```

☐ A B C and 1 2 3

☐ nothing will print

☐ 1 2 and 3

Correct!

☒ A B and C

### Question 17

0 / 2 pts

An Employee class has instance data String name and int id.

The following equals and hashCode methods **do** satisfy the contracts for those methods.

```
public boolean equals(Object obj) {  
    if(obj instanceof Employee) {  
        Employee e = (Employee) obj;  
        return this.name.equals(e.name);  
    } else {  
        return false;  
    }  
}  
  
public int hashCode() {  
    return Objects.hash(id, name);  
}
```

You Answered

☒ True

Correct Answer

☐ False

Quiz Score: **105.13** out of 100