Name:____ (PRINT CLEARLY)

Lab Section:________ Grader:___________________________________

Quiz 5B – December 6

CS 2102 B19

1. Review the following HashMap code.

```
import java.util.HashMap;

public class Dates {

     private HashMap<Integer, String> dates;

     public Dates() {

          this.dates = new HashMap<Integer, String>();
     }

     public Dates addDate(int thisDate, String dateName) {



          String oldDate = this.dates.put(thisDate, dateName);

          return this;
     }


     public String getDate(int thisDate) {


          return this.dates.get(thisDate);
     }
}
```

(4 points) Modify the above code to handle collisions. You do NOT need to check for null (i.e. empty) spaces in your hash map.

```
import java.util.HashMap;

public class Dates {

	HashMap<Integer, LinkedList<String>> dates;

	public Dates() {
		this.dates = new HashMap<Integer, LinkedList<String>>();
	}

	public Dates addDate(int thisDate, String dateName) {
		LinkedList<String> curDates = this.getDate(thisDate);
		curDates.add(dateName);
		String oldDate = dates.put(thisDate, curDates);
		return this;
	}


	public LinkedList<String> getDate(int thisDate) {

		return dates.get(thisDate);
	}
}
```

Scoring:

+1 Replaces all instances of `String` with `LinkedList<String>` EXCEPT in the `addDate()` parameter

- - 0.5 if `addDate()` parameter is also modified OR it's unclear that this parameter should not be modified
- - 0.5 for each unmodified `String` that should be modified to a `LinkedList` OR it's unclear that certain `String`s should be modified (up to 1 point)

+3 in `addDate()`, program gets the existing LinkedList (+1), adds the new item to it(+1), then puts the new LinkedList back in the HashMap (+1)

- Students who are writing code can use either `getDate()` or the HashMap `get()` function.
- Okay (but not required) if return type is changed to void, but then the return statement should also be removed (- 0.5 for forgetting this)
- No penalty if the student changes the return type to a LinkedList<String>, even if they don't change the type of the `oldDate` variable

2. (3 points) Review the following code. What will be printed to the console when we run the program? Explain why. Be sure to include a discussion of exceptions in your explanation.

```
public class Main {
      public static void main(String[] args) {

          private LinkedList<String> heroes = new LinkedList<String>();

           heroes.addLast("Superman");
           heroes.addLast("Black Widow");
           heroes.addLast("Green Lantern");

           try {
                System.out.println(heroes.get(1));
           }
           catch (IndexOutOfBoundsException e) {
                System.out.println("Exception caught!");
           }

      }
}
```

“Black Widow”. We have a list of three items and make a call to `get(1)`, which gives us the second item in the list. We make this call to `get()` inside a try/catch block in case we try to access a list item at an index that doesn’t exist. If this were to happen, an `IndexOutOfBoundsException` would be generated, and the catch block would run.

Scoring:

+1 Correct string printed to console.

+2 Correct explanation. (-1 if explanation does not include a discussion of exceptions, including the `IndexOutOfBoundsException` and how the try/catch block is used.)

3. (3 points) The Big-O of getting a value from a HashMap is O(1) (or nearly so). Briefly explain why.

Unlike most other data structures, we don't have to iterate through a series of values to find the one we want. Using the key, we can jump straight to the location in the hash map where the corresponding value is stored. Thus, the size of the hash map has no impact on runtime.

Scoring:

+3 Explanation indicates an understanding that we can jump straight to the location in the hash map where a value is stored (+2). Size has no impact on this operation (+1).

No deductions if students discuss collision handling, but not necessary.