Name:_____ (PRINT CLEARLY)

Lab Section:_____ Grader:_____

Quiz 5C – December 6

CS 2102 B19

1. Review the following HashMap code.

```
import java.util.HashMap;

public class Dates {

     private HashMap<String, Integer> dates
               = new HashMap<String, Integer>();

     public Dates() {
     }

     public Dates addDate(String dateName, int thisDate) {



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

          return this;
     }


     public int getDate(String dateName) {


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

(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 {

      private HashMap<String, LinkedList<Integer>> dates
                  = new HashMap<String, LinkedList<Integer>>();

      public Dates() {  }

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

      public LinkedList<Integer> getDate(String dateName) {

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

Scoring:

+1 Replaces all instances of `int` with `LinkedList<Integer>` 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 `int` that should be modified to a `LinkedList` OR it's unclear that certain `int`s should be modified (up to 1 point)
- No deduction if student uses primitive type `int` instead of `Integer`, but please note this correction.

+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("Batwoman");
            heroes.addLast("Spider-Man");
            heroes.addLast("She-Hulk");

            try {
                  System.out.println(heroes.get(0));
            }
            catch (IndexOutOfBoundsException e) {
                  System.out.println("Exception caught!");
            }

      }
}
```

"Batwoman". We have a list of three items and make a call to `get(0)`, which gives us the first 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) Explain how the `equals()` method is used in collision handling in hash maps.

If a hash map is using collision detection, that means that it stores values in a LinkedList so that multiple values can be stored using the same key. To then figure out which value in the LinkedList we actually want, Java uses the equals() method in the (unhashed) key object to iterate over the retrieved list and find the correct value.

Scoring:

+3 for communicating the idea that we use the `equals()` method to compare the key to the linked list values (+1) to get the value we want (+1) when more than one value is stored in the same location in the hash map (+1).

Do not deduct points for failure to mention that the key object is unhashed or that the same fields need to be used in both `equals()` and `hashCode()`.