Hash Table Lab

"I've been doing a lot of learning from mistakes, first and foremost, and building off that."

After finishing each part of the lab, copy your entire project and work on the copy for the next part!

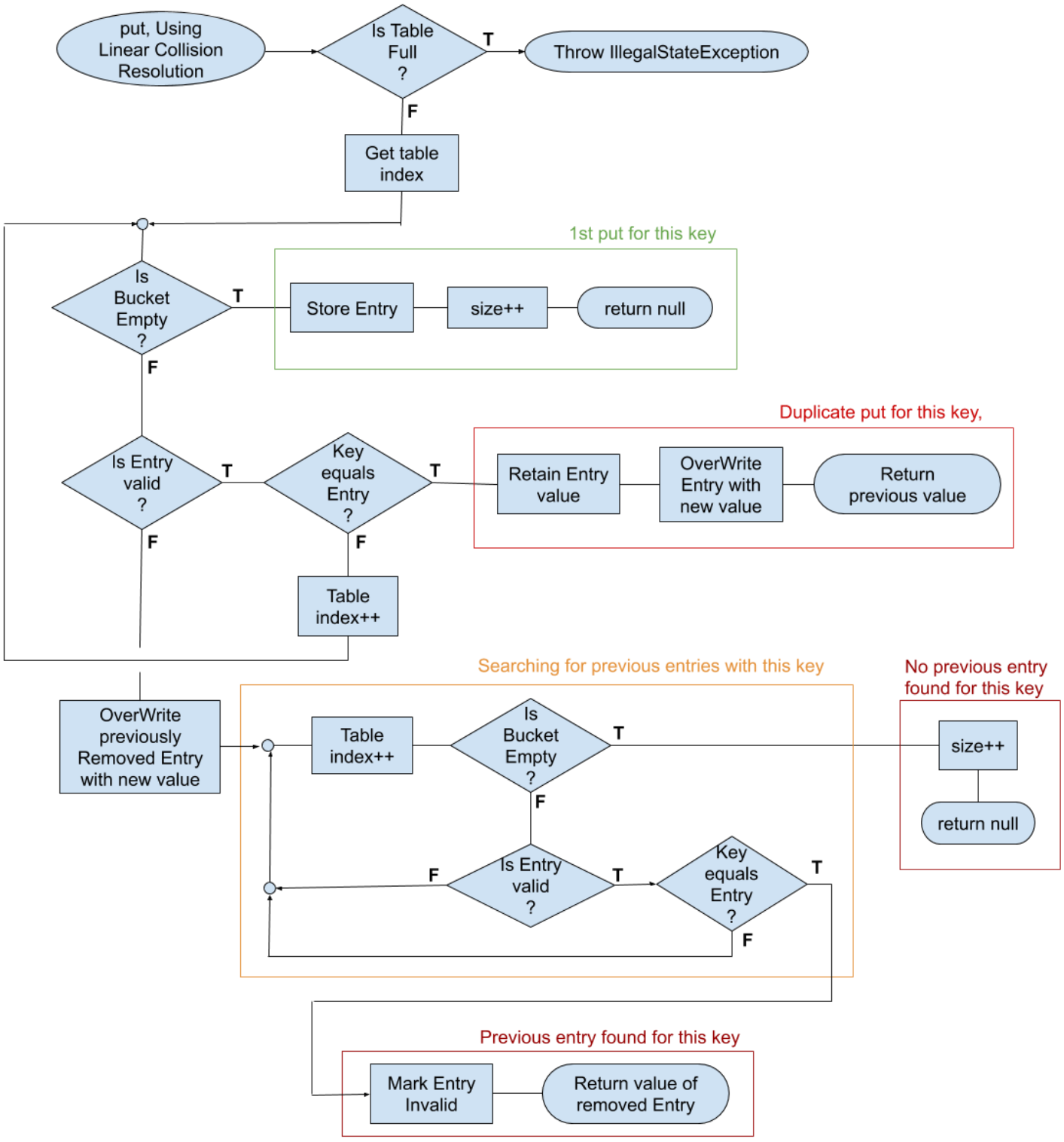
Part 2: Modify the *HashTable* class, implementing linear probing to handle collisions.

- Add a *size* field to *HashTable*.
  - Initialize to 0
  - Increment for *put*s and decrement for *remove*s.
- Add a *removed* field to *Entry* to indicate an unused bucket.
- Implement the *remove* method:
  - If the *Entry* exists, leave the *Entry* in place & mark the *removed* field as *true*.
    - Be sure to use *equals* (on the key object) to verify you've found the correct object.
  - Add linear probing when collisions occur:
    - Search until object is found or empty bucket encountered
    - Skip *removed* objects.
  - Return the previously stored value if the key is valid; otherwise, return *null* if the key was not found.
  - Decrement *size* if the *remove* succeeded.
- Modify the *get* method:
  - If the *Entry* exists, return the *value*.
    - Use *equals* (on the key object) to verify you've found the correct object.
  - If the *key* hashes to a different value, a collision occurred:
    - Use linear probing to find the object
    - Search until the object is found (verify with *equals*) or an empty bucket is encountered
    - Skip *removed* objects.
  - Return the stored value if the *key* is valid; otherwise, return *null* if the key was not found.

Continues on next page…

- Modify the *put* method (**see next page** for flowchart):
    - Check *size* to be sure space is available.
    - If the hashed location is empty, store the value, increment *size*, & return *null*
    - If a collision occurs, the key may have been used before:
        - Is the object already stored in the table?
            - Use *equals* (on the key object) to verify
            - If duplicate, overwrite the location & return the previously stored value (don't increment *size*)
        - Not at the hashed location? Use linear probing to find an empty table location:
            - Check for duplicate keys at each location
            - If an empty location is encountered:
                - Save the new object
                - Return *null*
                - Increment *size*
    - If, while searching for an unused table location, a *removed* location is encountered:
        - Save the new object in place of the *removed* Entry
        - Continue searching for a duplicate key:
            - Until an empty bucket is encountered:
                - Increment *size*
                - Return *null*
            - Or a duplicate is found:
                - Mark it *removed*
                - Return the previously stored value at the duplicate location

- Modify the *toString* method to print "dummy" for deleted locations.
- To improve readability, headers & labels have been added to the output (see the output format example).
- Verify that your modified HashTable works correctly:
    - The inputs from part 1 (no collisions), should produce the same result as before.
    - Use the small test data set to verify that you can put, ret, & remove correctly.
    - Turn in your program through the auto judge.


put, Using Linear Collision Resolution
Is Table Full ?
T
Throw IllegalStateException
F
Get table index
1st put for this key
Is Bucket Empty ?
T
Store Entry
size++
return null
F
Duplicate put for this key,
Is Entry valid ?
T
Key equals Entry ?
T
Retain Entry value
OverWrite Entry with new value
Return previous value
F
F
Table index++
Searching for previous entries with this key
No previous entry found for this key
OverWrite previously Removed Entry with new value
Table index++
Is Bucket Empty ?
T
size++
return null
F
F
Is Entry valid ?
T
Key equals Entry ?
T
F
Previous entry found for this key
Mark Entry Invalid
Return value of removed Entry

Sample Input (hash02.txt)

```
CAPACITY 17
PUT 10
92800393 LINNIE GILMAN
86770985 DUSTY CONFER
48235250 KENNITH GRASSMYER
31850991 WANETA DEWEES
25428367 DUSTY BANNON
24248685 FRANCE COELLO
23331143 JUSTIN ADKIN
68682774 MALIK TULLER
59245514 LESLEE PHIFER
24248685 ISAAC GENEY
REMOVE 2
25428367
68682774
PUT 1
54657809 MARTY ENOCHS
REMOVE 1
23331143
PUT 1
59245514 GENARO QUIDER
GET 5
24248685
54657809
59245514
23331143
31850991
STOP
```

Sample Output

```
---- ARRAY STATE AFTER PUTS ----

000
001
002
003 : 23331143 JUSTIN ADKIN
004 : 24248685 FRANCE COELLO
005 : 25428367 DUSTY BANNON
006 : 68682774 MALIK TULLER
007 : 59245514 LESLEE PHIFER
008 : 24248685 ISAAC GENEY
009
010 : 86770985 DUSTY CONFER
011 : 92800393 LINNIE GILMAN
012 : 48235250 KENNITH GRASSMYER
013 : 31850991 WANETA DEWEES
014
015
016

---- ARRAY STATE AFTER REMOVES ----

000
001
002
003 : 23331143 JUSTIN ADKIN
004 : 24248685 FRANCE COELLO
005 : dummy
006 : dummy
007 : 59245514 LESLEE PHIFER
008 : 24248685 ISAAC GENEY
009
010 : 86770985 DUSTY CONFER
011 : 92800393 LINNIE GILMAN
012 : 48235250 KENNITH GRASSMYER
013 : 31850991 WANETA DEWEES
014
015
016

---- ARRAY STATE AFTER PUT ----

000
001
002
003 : 23331143 JUSTIN ADKIN
004 : 24248685 FRANCE COELLO
005 : 54657809 MARTY ENOCHS
006 : dummy
007 : 59245514 LESLEE PHIFER
008 : 24248685 ISAAC GENEY
009
010 : 86770985 DUSTY CONFER
011 : 92800393 LINNIE GILMAN
012 : 48235250 KENNITH GRASSMYER
013 : 31850991 WANETA DEWEES
014
015
016

---- ARRAY STATE AFTER REMOVE ----

000
001
002
003 : dummy
004 : 24248685 FRANCE COELLO
005 : 54657809 MARTY ENOCHS
006 : dummy
007 : 59245514 LESLEE PHIFER
008 : 24248685 ISAAC GENEY
009
010 : 86770985 DUSTY CONFER
011 : 92800393 LINNIE GILMAN
012 : 48235250 KENNITH GRASSMYER
013 : 31850991 WANETA DEWEES
014
015
016

---- ARRAY STATE AFTER FINAL ADD ----

000
001
002
003 : dummy
004 : 24248685 FRANCE COELLO
005 : 54657809 MARTY ENOCHS
006 : 59245514 GENARO QUIDER
007 : dummy
008 : 24248685 ISAAC GENEY
009
010 : 86770985 DUSTY CONFER
011 : 92800393 LINNIE GILMAN
012 : 48235250 KENNITH GRASSMYER
013 : 31850991 WANETA DEWEES
014
015
016

---- GET TESTS ----

FRANCE COELLO
MARTY ENOCHS
GENARO QUIDER
null
WANETA DEWEES
```