CMSC 420: Spring 2022

# Programming Assignment 0: Dual List

Handed out: Thu, Jan 27. Due: **Tue, Feb 8 (11:59pm)**.

**Overview:** This is a start-up project designed to acquaint you with the programming/testing environment and submission process we will be using this semester. This will involve only a small bit of data structure design and implementation, and the focus will be on reviewing Java programming and learning how to use our Gradescope testing environment.

**Dual List:** You will provide a Java implementation of a very simple data structure we call a `DualList`. This stores a multiset of entries (sometimes called a *bag*), each of which consists of a pair of values, which we call *keys* (e.g., (age, weight), (month, year), (country, population)). The data structure is *generic*, meaning that the types of the two keys are specified when the structure is declared. Here is the declaration:

```
class DualList<Key1 extends Comparable<Key1>, Key2 extends Comparable<Key2>>
```

The key types implement the Java interface `Comparable`, meaning that you can compare keys using `compareTo`. For example, to test $x < y$, you can do `x.compareTo(y) < 0`. This includes common object types such as `String`, `Integer`, `Float`, `Double`, and `Character`.

For example, `DualList<String, Integer>` stores string-integer pairs like `("Hello", 25)` and `DualList<Double, Character>()` stores double-character pairs like `(-23.57, 'X')`. (In the examples below, we will assume string-integer pairs.)

This *not* a map. (A *map* is a data structure storing key-value pairs, where each key is associated with a unique value.) It is just a collection of pairs. Since it is a multiset, the order of elements does not matter and duplicates are allowed. So the dual lists {`(A, 5), (Z, 3), (A, 5)`} and {`(Z, 3), (A, 5), (A, 5)`} are the same.

Efficiency is not important, and the list entries may be stored in any order you like. You may use any classes/functions from the Java libraries you like (e.g., Java `LinkedList`, `ArrayList`, `Collections.sort`).

**Operations:** Given the dual list `DualList<Key1, Key2>`, here are the operations that your program must support.

`DualList()`: Constructor, which just creates an empty dual list.

`void insert(Key1 x1, Key2 x2)`: Inserts the pair $(x_1, x_2)$ into the dual list.

For example, given {`(A, 5), (Z, 3)`}, the operation `insert(A, 3)` would result in the list {`(A, 5), (Z, 3), (A, 3)`}. (The order of entries does not matter.)

`int size()`: Returns the number of pairs in the dual list.

`ArrayList<String> listByKey1()`: This returns a Java `ArrayList` of strings. The list is to be sorted in ascending order by the first key (with ties broken by the second key). Each string has the format `"(" + key1 + ",␣" + key2 + ")"`.

For example, given {`(A, 5), (Z, 3), (A, 3)`}, this returns a 3-element `ArrayList` containing three strings: `"(A, 3)"`, `"(A, 5)"`, and `"(Z, 3)"`. If the dual list is empty, the resulting `ArrayList` is also empty.

`ArrayList<String> listByKey2():` This is identical to `listByKey1()` except that the order is by the second key, with ties broken by the first key.

For example, in the above case, the result is: `"(A, 3)"`, `"(Z, 3)"`, and `"(A, 5)"`.

`Key2 extractMin1():` If the list is nonempty, this first finds the pair with the minimum first-key value $x_1$, it removes this pair from the list, and finally returns its associated *second-key value*. As with `listByKey1()`, ties are broken by second key value. If the list is empty, this throws an `Exception` with the error message `"Attempt to extract from an empty list"`.

For example, given the dual list {`(A, 5), (Z, 3), (A, 3)`}, both `(A, 5)` and `(A, 3)` have the minimum first-key value of `A`, and so the tie is broken in favor of the latter since $3 < 5$. We then remove `(A, 3)` from the list, and return its second-key value of 3. So, the list now contains {`(A, 5), (Z, 3)`}.

`Key1 extractMin2():` This is identical to `extractMin1` but with the roles of `Key1` and `Key2` reversed. It removes the pair with the minimum second-key value and returns the associated first-key value. It throws the same exception if the list is empty.

For example, given the dual list {`(A, 5), (Z, 3)`}, `(Z, 3)` has the minimum second-key value of 3, and so we then remove `(Z, 3)` from the list, and return its first-key value of `Z`. So, the list now contains {`(A, 5)`}.

All you need to do is to implement the above functions. We will provide a program that handles the input and output. An sample of input and output is shown below.

| **Input:** | **Output:** |
|---|---|
| `insert:A:5` | `insert(A, 5): successful` |
| `insert:Z:3` | `insert(Z, 3): successful` |
| `insert:A:3` | `insert(A, 3): successful` |
| `size` | `size: 3` |
| `list-by-key1` | `list-by-key1:` |
| | `  (A, 3)` |
| | `  (A, 5)` |
| | `  (Z, 3)` |
| `list-by-key2` | `list-by-key2:` |
| | `  (A, 3)` |
| | `  (Z, 3)` |
| | `  (A, 5)` |
| `extract-min-key1` | `extract-min-key1: 3` |
| `extract-min-key2` | `extract-min-key2: Z` |
| `list-by-key1` | `list-by-key1:` |
| | `  (A, 5)` |

**What we give you:** We will provide you with skeleton code to get you started on the class Projects page (`Part0-Skeleton.zip`). This code will handle the input and output, and provide you with the template for `DualList`. All you need to do is fill in the contents of this class. Note that directory structure has been set up carefully. You should not alter it unless you know what you are doing.

**Files:** Our skeleton code provides the following files. They can be found in the directory "`cmsc420_s22`", and all must begin with the statement "`package cmsc420_s22`".

`Tester.java`**:** This contains the main Java program. It reads input commands from a file (by default `tests/test01-input.txt`) and it writes the output to a file (by default `tests/test01-output.txt`).

▷ *You should not modify this except to change the input and/or output file names.*

We will provide you with a few sample test input files along with the "expected" output results (e.g., `tests/test01-expected.txt`). Of course, you should do your own testing. To check your results, use a difference-checking program like "diff".

Note that the tester program does not generate output to the console (unless there are errors). The output is stored in the output file in the `tests` directory.

`CommandHandler.java`**:** This program provides the interface between `Tester.java` and our `DualList.java`. It invokes the functions in your `DualList` class and outputs the results. It also catches and processes any exceptions.

▷ *You should not modify this file.*

`DualList.java`**:** You provide the contents of this file. We will give you a template of its structure, and you fill in the details.

```
package cmsc420_s22; // Be sure to use this package!

import java.util.ArrayList;

public class DualList<Key1 extends Comparable<Key1>,
                      Key2 extends Comparable<Key2>> {
    public DualList() { ... }                              // constructor
    public void insert(Key1 x1, Key2 x2) { ... }          // insert new pair
    public int size() { ... }                              // number of pairs
    public Key2 extractMinKey1() throws Exception { ... } // remove Key1 min
    public Key1 extractMinKey2() throws Exception { ... } // remove Key2 min
    public ArrayList<String> listByKey1() { ... }         // Key1-sorted list
    public ArrayList<String> listByKey2() { ... }         // Key2-sorted list
}
```

▷ *Submit this file to the autograder.*

**What you give us:** All that you need to do is to fill in the implementation of the methods for the `DualList` class. Other than `DualList.java` and changing the file names in `Tester.java`, you should avoid modifying any of the directory structure or the files in the skeleton code.

**Submission Instructions:** Submissions will be made through Gradescope. There is no limit to the number of submissions you can make. The last submission will be graded. Here is what to do:

- Log into the CMSC420 page on Gradescope, select this assignment, and select "`Submit`". A window will pop up (see Fig. 1). Drag your file `DualList.java` into the window. If you generated other files, zip them up and submit them all. Select "`Upload`".

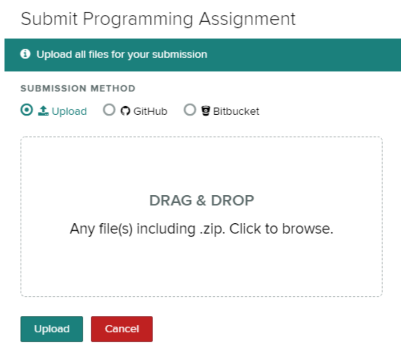
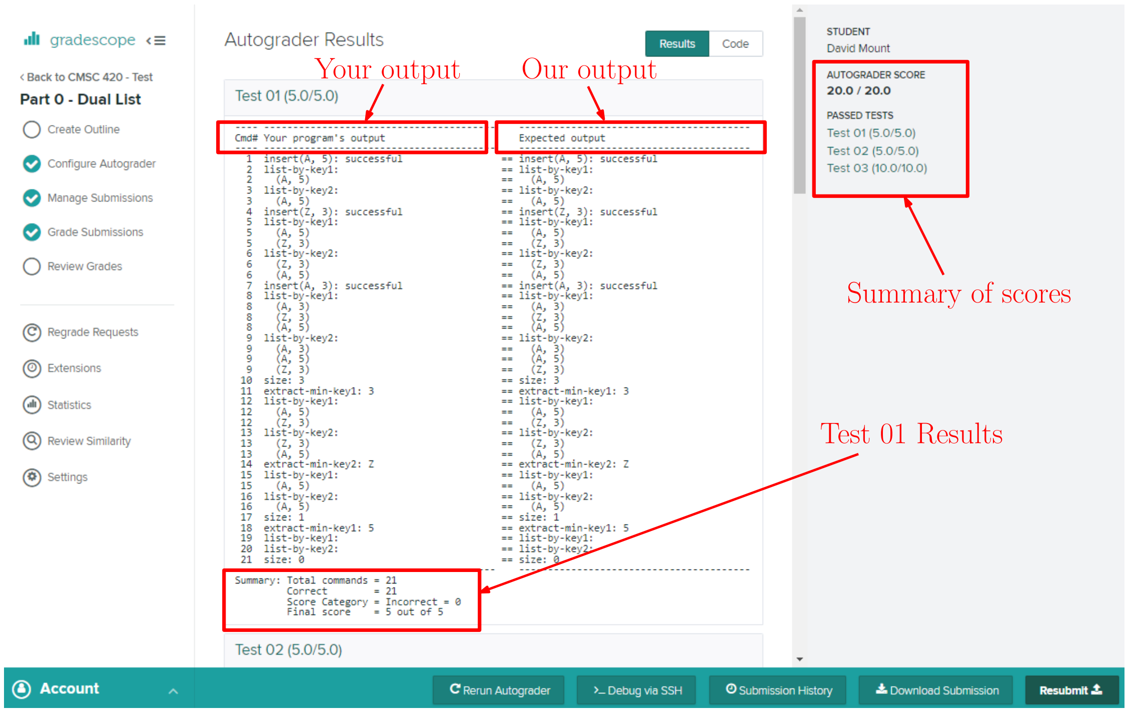

Submit Programming Assignment
Upload all files for your submission
SUBMISSION METHOD
Upload
GitHub
Bitbucket
DRAG & DROP
Any file(s) including .zip. Click to browse.
Upload
Cancel

Figure 1: Gradescope submission. Drag your file `DualList.java` into the box.

After a few minutes, Gradescope will display the results (see Fig. 2). Normally, a portion of your grade will depend on good style and efficiency, but for this initial program, only the autograder score will be used.



Test 01 (5.0/5.0)

```
---- ----------------------------------------    -----------------------------------------
Cmd# Your program's output                           Expected output
---- ----------------------------------------    -----------------------------------------
  1  insert(A, 5): successful                    == insert(A, 5): successful
  2  list-by-key1:                               == list-by-key1:
  2    (A, 5)                                    ==   (A, 5)
  3  list-by-key2:                               == list-by-key2:
  3    (A, 5)                                    ==   (A, 5)
  4  insert(Z, 3): successful                    == insert(Z, 3): successful
  5  list-by-key1:                               == list-by-key1:
  5    (A, 5)                                    ==   (A, 5)
  5    (Z, 3)                                    ==   (Z, 3)
  6  list-by-key2:                               == list-by-key2:
  6    (Z, 3)                                    ==   (Z, 3)
  6    (A, 5)                                    ==   (A, 5)
  7  insert(A, 3): successful                    == insert(A, 3): successful
  8  list-by-key1:                               == list-by-key1:
  8    (A, 3)                                    ==   (A, 3)
  8    (Z, 3)                                    ==   (Z, 3)
  8    (A, 5)                                    ==   (A, 5)
  9  list-by-key2:                               == list-by-key2:
  9    (A, 3)                                    ==   (A, 3)
  9    (A, 5)                                    ==   (A, 5)
  9    (Z, 3)                                    ==   (Z, 3)
 10  size: 3                                     == size: 3
 11  extract-min-key1: 3                         == extract-min-key1: 3
 12  list-by-key1:                               == list-by-key1:
 12    (A, 5)                                    ==   (A, 5)
 12    (Z, 3)                                    ==   (Z, 3)
 13  list-by-key2:                               == list-by-key2:
 13    (Z, 3)                                    ==   (Z, 3)
 13    (A, 5)                                    ==   (A, 5)
 14  extract-min-key2: Z                         == extract-min-key2: Z
 15  list-by-key1:                               == list-by-key1:
 15    (A, 5)                                    ==   (A, 5)
 16  list-by-key2:                               == list-by-key2:
 16    (A, 5)                                    ==   (A, 5)
 17  size: 1                                     == size: 1
 18  extract-min-key1: 5                         == extract-min-key1: 5
 19  list-by-key1:                               == list-by-key1:
 20  list-by-key2:                               == list-by-key2:
 21  size: 0                                     == size: 0
---- ----------------------------------------    -----------------------------------------
Summary: Total commands = 21
         Correct        = 21
         Score Category = Incorrect = 0
         Final score    = 5 out of 5
```

Test 02 (5.0/5.0)

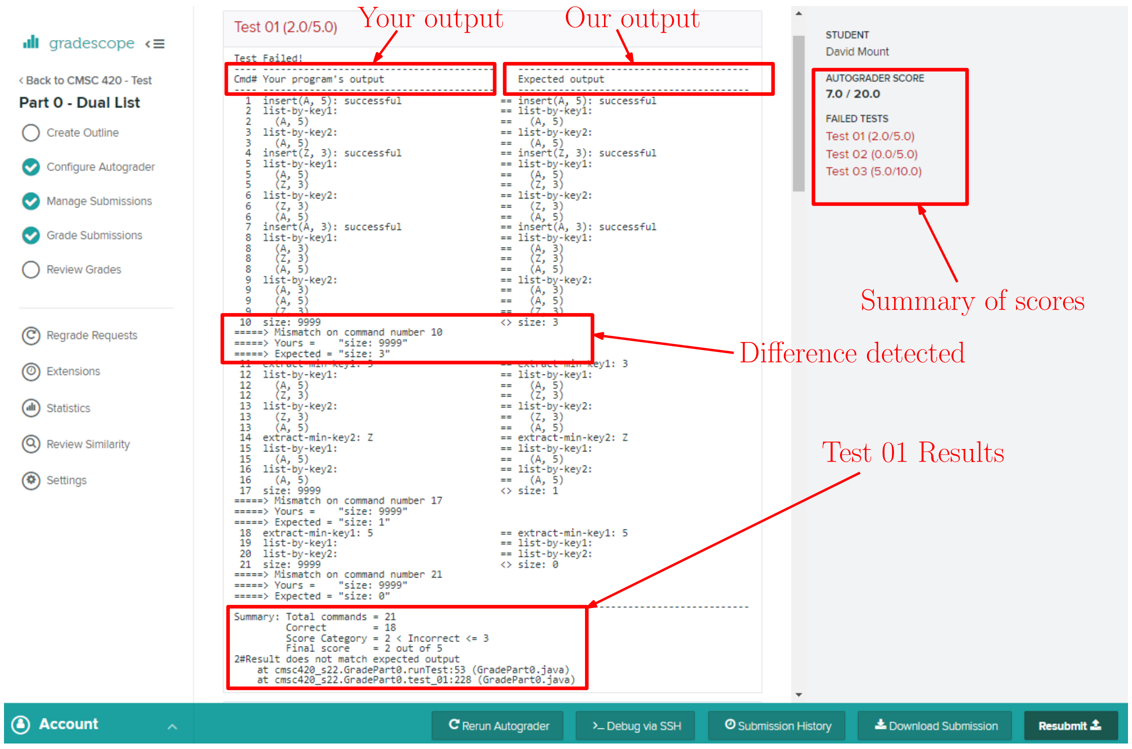
STUDENT
David Mount

AUTOGRADER SCORE
20.0 / 20.0

PASSED TESTS
Test 01 (5.0/5.0)
Test 02 (5.0/5.0)
Test 03 (10.0/10.0)

Figure 2: Gradescope autograder results (correct).

On the top-right of the page, it shows a summary of the scores of the individual tests as generated by the autograder. (If there are compilation errors, these will be displayed on this page.) The center of the window shows a line-by-line summary, with the output generated by your program on the left and the expected output on the right. If there are mismatches, these will be highlighted (see Fig. 3).



Test 01 (2.0/5.0)

```
Test Failed!
---- ----------------------------------------   ----------------------------------------
Cmd# Your program's output                      Expected output
---- ----------------------------------------   ----------------------------------------
  1  insert(A, 5): successful                == insert(A, 5): successful
  2  list-by-key1:                           == list-by-key1:
  2    (A, 5)                                ==   (A, 5)
  3  list-by-key2:                           == list-by-key2:
  3    (A, 5)                                ==   (A, 5)
  4  insert(Z, 3): successful                == insert(Z, 3): successful
  5  list-by-key1:                           == list-by-key1:
  5    (A, 5)                                ==   (A, 5)
  5    (Z, 3)                                ==   (Z, 3)
  6  list-by-key2:                           == list-by-key2:
  6    (Z, 3)                                ==   (Z, 3)
  6    (A, 5)                                ==   (A, 5)
  7  insert(A, 3): successful                == insert(A, 3): successful
  8  list-by-key1:                           == list-by-key1:
  8    (A, 3)                                ==   (A, 3)
  8    (Z, 3)                                ==   (Z, 3)
  8    (A, 5)                                ==   (A, 5)
  9  list-by-key2:                           == list-by-key2:
  9    (A, 3)                                ==   (A, 3)
  9    (A, 5)                                ==   (A, 5)
  9    (Z, 3)                                ==   (Z, 3)
 10  size: 9999                              <> size: 3
=====> Mismatch on command number 10
=====> Yours =    "size: 9999"
=====> Expected = "size: 3"
 11  extract-min-key1: 3                     == extract-min-key1: 3
 12  list-by-key1:                           == list-by-key1:
 12    (A, 5)                                ==   (A, 5)
 12    (Z, 3)                                ==   (Z, 3)
 13  list-by-key2:                           == list-by-key2:
 13    (Z, 3)                                ==   (Z, 3)
 13    (A, 5)                                ==   (A, 5)
 14  extract-min-key2: Z                     == extract-min-key2: Z
 15  list-by-key1:                           == list-by-key1:
 15    (A, 5)                                ==   (A, 5)
 16  list-by-key2:                           == list-by-key2:
 16    (A, 5)                                ==   (A, 5)
 17  size: 9999                              <> size: 1
=====> Mismatch on command number 17
=====> Yours =    "size: 9999"
=====> Expected = "size: 1"
 18  extract-min-key1: 5                     == extract-min-key1: 5
 19  list-by-key1:                           == list-by-key1:
 20  list-by-key2:                           == list-by-key2:
 21  size: 9999                              <> size: 0
=====> Mismatch on command number 21
=====> Yours =    "size: 9999"
=====> Expected = "size: 0"
----------------------------------------------------------------------------------------
Summary: Total commands = 21
         Correct        = 18
         Score Category = 2 < Incorrect <= 3
         Final score    = 2 out of 5
2#Result does not match expected output
    at cmsc420_s22.GradePart0.runTest:53 (GradePart0.java)
    at cmsc420_s22.GradePart0.test_01:228 (GradePart0.java)
```

STUDENT
David Mount

AUTOGRADER SCORE
7.0 / 20.0

FAILED TESTS
Test 01 (2.0/5.0)
Test 02 (0.0/5.0)
Test 03 (5.0/10.0)

Figure 3: Gradescope autograder results (incorrect).

The final score is based on the number of commands for which your program's output differs from ours. Note that the comparison program is very primitive. It compares line by line (without considering the possibility of inserted or deleted lines) and is sensitive to changes in case and the addition of white-space.