LAB 1: BAG ADT

TOPIC(S)

Bag Implementations

READINGS & REVIEW

- Carrano, *Data Structures and Abstractions with Java,* Chapters 1-3 (prelude and Appendices if you need a Java review).
- Bag ADT & implementation lectures

OBJECTIVES

Be able to:

- *Implement* a Bag ADT (Abstract Data Type)
- Test your implementation
- Answer questions about the Bag ADT and your implementation

INSTRUCTIONS

1. **This is an individual assignment**.
2. ***Read the assignment and ask questions about anything that you don't understand (before you start).***
3. To use the provided Eclipse project:
    a. You must be running Eclipse 2019-03 or later. Ask if you are running an older version and aren't sure how to upgrade. I don't use IntelliJ, so I don't know what version you need to properly support this project. Note: the JUnit tests I provide require JUnit 5.4 or later (Eclipse 2019-03 provides the required JUnit version).
    b. Download and unzip the zipped (Eclipse) project to your Desktop.
    c. In Windows, rename the folder: replace *username* with your username (the first part of your WIT email address – to the left of the @WIT.edu)
    d. In Eclipse:
        i. in the File menu, click Import…
        ii. in the first dialog box, expand the General option, select Existing Projects into Workspace, then click Next >
        iii. in the next dialog box, click Browse…

iv. in the next dialog box, navigate to your desktop and select the unzipped, renamed folder, then click Select Folder

v. in the prior dialog box, make sure your project folder is checked in the list of Projects, check the box next to Copy projects into workspace – leave the others unchecked, then click Finish

vi. rename the project:

1. select the project in the Project Explorer or Package Explorer pane
2. in the File menu, click Rename…
3. replace *username* with your username (same as you did for the folder) then click OK

e. delete the unzipped project folder from your Desktop – you can keep the downloaded zip file so you can easily to go back to the provided files

f. If you don't rename the folder or the project (yes, two separate steps – see above), you will lose 20 points – do it now so you don't forget later.

4. For your application:
   a. Be sure to follow Good Programming Practices.
      Your code must comply with the Coding Standards/Guidelines posted on Blackboard.
   b. Keep in mind the Guidelines on Plagiarism (see section GUIDELINES ON PLAGIARISM IN COMPUTER SCIENCE).
5. Do a Write-up (Analysis / Summary). The write-up is located in the **admin** folder in the Eclipse project.
6. Submit your work by the posted due date/time. Late submissions will incur a 25% penalty per day. Projects that don't compile will not be graded.
7. If you'd like my assistance with your code, please (1) come to an extra help/walk-in lab session or (2) export/zip your entire project and attach it to an email message with a brief description of your question/problem. I will typically respond within a few hours. Do not expect a response after 9p (5p on the due date).
8. Post all other questions with respect to this lab on Piazza – ***I will not answer questions clarifying this lab or the material it covers by email.***

PROBLEM

ADT Implementation:

Fully develop/finish the methods for the Linked Implementation of the ADT Bag (i.e. `LinkedBag<T>`) presented in chapter 3. Use the provided source code as your starting point – incorporate the code from the book (for all methods except the cloning constructor, `difference()`, `intersection()`, `union()`). Note that the inner class `Node` is fully implemented. You must use its methods in your LinkedBag implementation.

`difference()`: `bag1.difference( bag2 )` returns a new LinkedBag containing all entries from `bag1` (`this`), minus the entries contained in `bag2`. Remember that bags can contain duplicates – if `bag1` contains A, B, B, C and `bag2` contains A, B, D, the resulting bag contains B, C. Do not modify the contents of `bag1` or `bag2`!

`intersection()`: `bag1.intersection( bag2 )` returns a new LinkedBag containing all entries that appear in both `bag1` (`this`) and `bag2`. Again, remember that bags can contain duplicates – if `bag1` contains A, B, B, C and `bag2` contains A, B, D, the resulting bag contains A, B. The contents of `bag1` and `bag2` are not changed.

`union()`: `bag1.union( bag2 )` returns a new LinkedBag containing all entries in `bag1` (`this`) and all entries in `bag2`. If `bag1` contains A, B, B, C and `bag2` contains A, B, D, the resulting bag contains A, A, B, B, B, C, D. Neither `bag1` nor `bag2` are modified.

Remember that Bags are unordered so the contents may differ in order from that described (above) but the resulting bags will hold duplicates, etc. as described.

See the "Sample algorithms for selected Bag ADT methods" document for more guidance to implement these additional methods.

Test your LinkedBag class well – complete the provided LinkedBagStudentTests class: call all methods with valid and invalid inputs/conditions to make sure your class behaves predictably/correctly. Use `main()` and the provided `testXxx()` stubs

- each `testXxx()` must run independently of the others
- do not do any console input (you may read from a file in the data folder)
- you may not change the method headers for any of the `testXxx()` methods
    - they are not permitted to take any parameters and probably don't return any results to `main()`
- you are not permitted to use any class (static) variables
- display information about what you're testing and whether the tests succeeded or failed

Do not modify LinkedBagDMRTests.java.

GOOD PROGRAMMING PRACTICES

For all classes (required except as noted):

- In your IDE, you must rename your project to "DS - 1 Bag ADT - <username>" where <username> is the portion of your WIT email address to the left of @WIT.edu – do not include the double quotes or angle brackets – you must match the spacing and single quotes precisely. Warning: You must type the project name – do not copy/paste – Word does not use the correct hyphen characters! In Eclipse, right-click the project name to select it then left-click the Refactor menu option then left-click the Rename… option. A dialog box will open - modify the project name as instructed above then left-click OK.
- Put your classes in a package named edu.wit.dcsn.comp2000.bagadt (already done)
- Use mnemonic/fully self-descriptive names for all classes and class members (methods, variables, parameters, etc.)
- Make your instance variables **private**
- Provide **constructors** to fully initialize your instance variables and, if appropriate, **mutator** and **accessor** methods for all instance variables, plus **toArray()** and **toString()** methods.
- Add brief comments to your code, not just so it's easier for other readers, but also so it's easier for you to remember your logic. You are required to provide full Javadoc comments for each class and every method (provided). If you add/modify any Javadoc comments, you must generate the Javadoc for all **public** elements – put the Javadoc documentation in the **doc** folder (in the root folder; **src** is a sibling folder) – the Javadoc for the provided code is already in the **doc** folder.
- Your unit testing must be thorough – test *all* methods; choose good, representative (not exhaustive) test cases (valid, invalid, edge, etc.) . You must implement a basic set of unit tests in LinkedBagStudentTests (as noted above).
- I provided a set of *JUnit* tests for the constructors and API methods (in LinkedBagDMRTests). Do not modify any of my test classes or test data. Your goal should be to have your code must 100% of the tests. When my tests run, they display a summary to the console. A detailed log is created in the test-logs folder – you will need to select the test-logs folder then press F5 (or right-click on the folder then left-click on Refresh) to see the new log – a new log is created each time you run the tests. Each test will time-out after a few seconds – if the tests seem to have stalled, please wait, they'll finish. The detail logs contain information about each test including the method under test, the parameter(s) provided to the method, the expected results, and the actual results. If the only tests that succeed in a section match the stub value (for instance, the `isEmpty()` stub always returns `false`), that test section is marked as failing all tests.

WRITE-UP

I provided a write-up template in the **admin** folder in the project. Rename the write-up file replacing *username* with the left portion of your @WIT.edu email address. Make sure you answer the questions in the document in the **admin** folder – and save it – before you zip the project to submit it.

SUBMITTING YOUR WORK:

1. Make sure your name is in all project files you create or modify (e.g. LinkedBag.java and LinkedBagStudentTests.java). Do not add this information to supplied files that you do not modify.

2. **Style requirement**: For stand-alone unit tests, your *main()* method must precede all private methods which exercise the API/ADT. *private* utility methods for testing must follow the *main()* method – these are typically *static* (see the additional notes in the Good Programming Practices section above). Your tests are not permitted to request input from the console.

   Style requirement: If you wish to read a file as part of your testing (or for console applications), the file must go in the data folder and your tests must open it using a relative path: "./data/foo.dat" is the correct path to access the file foo.dat in the **data** folder. Pay attention to file name capitalization – Windows' file system is case-insensitive but many other OS's are case-sensitive. Also, use the '/' path separator even on Windows – it enables the JCL to traverse paths on all OS's. Make sure you close() all resources you open!

   Style requirement: You are typically not permitted to use global variables. One acceptable use of a global/class variable is a counter which tracks the number of times the class has been instantiated to initialize an instance variable (e.g. `id`).

   Style requirement: You are not permitted to use separate, distinct variables for elements that belong in a collection; use an appropriate collection instead of multiple variables (e.g. var1, var2, var3…). It is acceptable for a unit test to have several variables of the class/interface type for testing purposes rather than a collection of them. You will likely use temporary variables to instantiate and manipulate particular instances.

   Style requirement: You must include Javadoc comments for all methods (even `private` methods) in all classes. Generate the Javadoc for the entire project (it must be in the *doc* folder 'next to' the *src* folder in the project directory tree).

3. Spell check and grammar check your work!

4. Make a **.zip file** for your project (my grading procedures expect this). ***No other compressed formats are acceptable!***

   - Include your entire project folder (root folder and subfolders, not just the contents of the root folder).
   - Your write-up must go in the admin folder. **Do *not* attach it directly to the Blackboard submission.**
   - In Blackboard, **attach** your solution file to the submission for this assignment.

Note: You are prohibited from posting or otherwise sharing this assignment or any code provided to you or implemented by you as part of this assignment. Violation of this restriction will be treated as a violation of the Wentworth Academic Honesty Policy; consequences are the same as those for plagiarism.

Reminder: you are bound by the Plagiarism & Cheating – Guidelines & Acknowledgment.

If you have any questions, ask!

Have fun! Dave