Project 1: Bag App

Topic(s)

Bag applications

Readings & Review

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

Objectives

Be able to:

- Use multiple instances of a Bag in an application
- Test your application
- Answer questions about the Bag ADT and your application
- Present your project

Instructions

1. **This is a group assignment – you will work in self-selected groups of 2-3 members**. Register your groups/teams in Brightspace (I already did this for you).
2. ***Read the assignment and ask questions about anything that you don't understand (before you start).***
3. Select an application from the Problems section.
4. For your application:
    a. Be sure to follow Good Programming Practices.
       Your code must comply with the Coding Standards/Guidelines posted on Brightspace.
    b. Keep in mind the Guidelines on Plagiarism.
5. Do a Write-up (Analysis / Summary). The write-up/analysis is part of the individual assignment (Bag ADT) and will be submitted with that assignment.
6. Prepare a PowerPoint presentation. Your group will present during the class following the posted submission due date. Your group will submit the PP presentation file as part of the group submission – it goes in the project root folder.

7. 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.
8. If you'd like my assistance with your code, please come to an extra help/walk-in lab session. CAE also has tutors who can help you.
9. 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.***

Problems

Spell Checker (2 people)

- Write a program to use a dictionary to spell check text files
- Test each word against a 'dictionary' (an instance of a Bag) of correctly spelled words (if the word is in the dictionary then it's spelled correctly)
- Name your application `SpellChecker.java`
- I provided several text files which you must use
- Display the overall total number of words read, the number of correctly spelled words, and the number of incorrectly spelled words
- Display the incorrectly spelled words once each (store them in a second Bag). They do not need to be displayed in any particular order. Do not display a misspelled word more than once.
- Create additional classes to split up the functionality.

Grocery Bagger (2 or 3 people)

- Write a program that uses heuristics (that a bagger at a grocery store might use) to fill grocery bags (instances of a Bag) given a variety of grocery items.
- Name your application `GroceryBagger.java`
- I provided a text file of grocery items to use, including attributes for the items (size, weight, firmness, flexibility, breakability, and perishability). You can read the items, line by line using a `Scanner`, then split the line into its component attributes using `item.split( "\t" ).` `enum GroceryItemSize.java` (which I provided) includes an `interpretDescription()` method which returns the enumeration instance corresponding to the provided description – I encourage you to create additional `enum`s for some of the other grocery item attributes to make them easier to use.
- As you fill bags, select items so you place incompatible items into different bags (e.g., hard and heavy items shouldn't go in a bag with breakable items). Your heuristics may place a limit on the number of items, total size, and/or total weight in each bag.
- Create additional classes to split up the functionality (e.g., GroceryBag, GroceryItem, etc.).

Student Choice

If you're feeling creative and have a great idea for an application using multiple Bags, make a proposal (I need to approve it before you start working on it).

General Application Guidelines

All classes you create must go in the …bag.app package.

Your application must produce output (typically to the console) demonstrating its operation.

Do not prompt the user for any input – your application must run unattended.

You must access any data files using relative paths – failure to do this will likely ensure that I can't run your application on my computer with a corresponding deduction from your grade.

You will not be graded on output format, but it must be clean enough to enable the user (me) to know what it does, what is happening, whether it succeeded or failed, etc. Avoid unnecessary 'noise' (e.g., don't print all the words in the dictionary but do indicate the name of the file processed and the number of words in it; for a text file which you are spell-checking, display the name of the file, the number of words, then useful/interesting information such as a list of the misspelled words). Use white space (blank lines, tabs, etc.) to make your output easier to read.

Make sure your spelling is correct (text you explicitly display).

First Steps

- form a group
- select a problem
- high-level design ⇒ what classes?
- assign an owner (will be the `@author`) for each class
- specify APIs ⇒ verify they will provide the needed functionality ⇒ adjust APIs until they meet the needs of the application

Next Steps

- create all classes – create stubs for all API methods – note: you do not need interface(s) for your API(s) because your classes have limited scope of applicability
- implement methods – test them as you go
- 'integrate' the pieces (classes) – since they have been (thoroughly) tested by their authors, you should be tweaking the code rather than needing to do extensive debugging 8~)

Provided Files

The assignment includes a zip file containing:

- Bag interface and implementation:
  - BagInterface.java
  - ResizableArrayBag.java
  - ResizableArrayBagDemo.java
  - You must test your application with the supplied Bag ADT implementation. Declare your reference variables as type BagInterface<> rather than ResizableArrayBag<> to simplify switching implementation.
  - Do not modify any of these files.
- For the Spell Checker application:
  - a dictionary text file (american-english-JL.txt) – you must demonstrate that your application works with this file
  - sample text files (sources.txt, the-lancashire-cotton-famine.txt, wit-attendance-policy.txt) – you must demonstrate that your application works with each of these files – process them individually, not all together – make sure you display enough information to distinguish which file is being processed and which results apply to it – hard-code the file names into your program (put them in an array and process them with a loop) – do not do any prompting. Note: you must process all three sample text files in a single execution of your application.
  - delete GroceryItemSize.java and graceries.txt
- For the Grocery Bagger application:
  - a sample grocery list representing your shopping cart (text file: groceries.txt) – you are not obligated to use all the characteristics, but you must use this data file and some of the characteristics
  - GroceryItemSize.java – this is an example of how to define and use an enumeration (enum) to represent a GroceryItem attribute.
  - you may use an ArrayList, LinkedList, or array to keep track of the `GroceryBag`s but you are not permitted to store `GroceryItem`s in anything other than the GroceryBag in which it will leave the store
  - delete the other `.txt` files in the `data` folder

All data/text/input files must stay in the `data` folder. You must use relative addressing to open any data file (e.g., "`./data/groceries.txt`"). Do not specify an absolute path to the files in your application – your code needs to run on my system and my folder structure is different from yours. You may/should explicitly name each data file (you do not need to find them programmatically).

- A PowerPoint document/template for the presentation.
  - You must use PowerPoint for your presentation – you must include the .pptx file with your submission. You do not need to use the provided template; however, your document must include the same information including headers (if included), identification blocks, and footers.

Good Programming Practices/Requirements

For all classes (required except as noted):

- Rename the unzipped folder before importing into Eclipse then, in your IDE, you must rename your project to "`DS - 1 Bag App - Group ##-##`" where `##-##` is your full group number – do not include the double quotes or pound signs/hash marks – you must match the spacing and single quotes precisely – the group number is 2 digits (including leading zero-fill), a dash/hyphen/minus sign, and 2 digits (including leading zero-fill). The folder and project names must match (Eclipse doesn't require this – I do). Warning: You must type the project name – do not copy/paste – Word may 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 the `edu.wit.scds.comp2000.bag.app` package
- 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 the data fields/instance variables
- 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 (all visibilities). You may 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.
- There is a demo program to exercise ResizableArrayBag – the expected output is in the comment block at the end of the source file – I didn't give you any unit tests
- You should write your own tests for the methods in your application – they won't be graded but they can significantly reduce the time spent debugging – you can use `main()` in each class (other than the main class's `main()`) to drive the tests.

Write-up

The second section of the individual write-up contains questions about this assignment – the write-up is part of the individual assignment and must be submitted with it.

Group Presentation

1. All group members must participate in the presentation.
2. Make sure your full group number and all member names and roles are on the title slide and the group number is in the footer of all following slides.

3. You may use any theme, template, or style you wish (recommendation: keep it appropriately professional). Make sure the font size is large enough to read from the back of the room.
4. Minimally, your presentation must include all the information in the provided template.
5. Presentations should take approximately 3-5 minutes, including a demonstration of your application and Q&A.
6. ***Do not read the slides to the class***. The slides should be short, bulleted lists of talking points. Face the class when speaking and speak loudly enough to be heard in the back of the room.
7. ***Do not include code in the slides***. If you want to show selected portions of your implementation, do so using your IDE or copy the sections of code into another application for display purposes only (again, make sure you set the font to a large enough size to be readable from the back of the room).
8. ***Make sure the presenter's computer is adequately charged and has the correct versions of the PowerPoint presentation and code. Prior to coming up, open the presentation and your IDE (so you can demonstrate your application) and make sure they run properly and that the font size is large enough to be seen/read at the back of the room.***

Submitting Your Work

1. Make sure your full name (not your username or login name) is in all project files you create or modify (e.g., GroceryBagger.java, GroceryBag.java, GroceryItem.java, GroceryItemSize.java, SpellChecker.java). Do not add this information to supplied files that you do not modify. There must be a single '`@author`' for each class.

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 `main()` – 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, preferably with a try-with-resources construct!

   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 (regardless of visibility) in all classes. You may generate the Javadoc for the entire project – specify inclusion of `private` visibility (it must go 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 PowerPoint presentation must go in the `admin` folder. **Do *not* attach it directly to the Brightspace submission.**
   - In Brightspace, **attach** your solution file to the submission for this assignment.
   - Note: Only one member of each group submits for the entire group. You can submit multiple times – I check the first submission for on-time-ness – I grade the last submission.

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