APPLICATION 1: BAG

TOPIC(S)

Bag Applications and 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:

- *Use* a/your Bag ADT in an application

INSTRUCTIONS

1. **Form groups of 3 members.** Register your groups/teams in Blackboard.
2. ***Read over the assignment and ask questions about anything that you don't understand (before you start).***
3. Select an application from the Problems section.
    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.
4. Do a Write-up (Analysis / Summary).
5. Prepare a PowerPoint presentation. Your group will present during the class following the posted submission due date.
6. Submit your work by the posted due date/time. Late submissions will incur a 25% penalty per day. Non-working projects (major or minor bugs) will incur at most a 10% penalty.
7. If you'd like my assistance, please zip your entire project folder 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).

PROBLEMS: BAG ADT APPLICATION

TITLE	INSTRUCTIONS
Select one	**1. Spell checker** Write a spell checker that: **a.** Reads in words from an external file. **b.** Tests each word against a dictionary *(an instance of a Bag<T> class)* of correctly spelled words *(if the word is found in the dictionary then it's spelled correctly*). **c.** Display the incorrectly spelled words once each (store them in a second Bag). They do not need to be displayed in any particular order. **2. Automated grocery bag filler** • Write a program that uses heuristics *(that a bagger at a grocery store might use)* to fill a shopping bag *(an instance of a Bag<T> class)* from a whole bunch of grocery items. • How you store the initial "bunch" of grocery items is up to you • Grocery items' have properties such as weight, breakability, squashability, etc. As bags are filled, items are selected so that roughly (that's why it's heuristic and not an algorithm) you place different categories of items into different bags. You may place a limit on the number, total size, and/or total weight in each bag. **3. Student choice** If you're feeling creative and have a great idea for an application of the Bag ADT (using a *Bag<T>* implementation) then make a proposal (***I need to approve it first***). **Your application must produce output (typically to the console) demonstrating its operation.** **You will not be graded on 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 out all of the words in the dictionary but you might want to 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).**

The assignment includes a zip file containing:

1. Bag interface and implementation:
   - BagInterface.java
   - ResizableArrayBag.java

   You must test your application with the supplied Bag ADT implementation. You may want to declare your reference variables as type BagInterface<> rather than ResizableArrayBag<> to simplify switching between implementation (if you do the next, optional step).

   Optional (but highly recommended): test your application with each of your Bag ADT implementations.

2. For the Spell Checker application:
   - a dictionary text file – you must demonstrate that your application works with this file
   - sample text files – you must demonstrate that your application works with all of these files

3. For the Grocery Bagger application:
   - a sample grocery list (text file) – you are not obligated to use this file, its format or contents but you must use a data file (your shopping cart).

   All data/text/input files must open from the default location (typically) the project root folder. You are not permitted to specify a path to the files in your application. You may explicitly name each data file (you do not need to find them programmatically).

4. A Word document/template for the write-up.

5. A PowerPoint document/template for the presentation.

   Unless there is a compelling reason to do otherwise, please use Word and PowerPoint for your write-up and presentation, respectively. *If you wish to use a different program, you need to get my approval in advance*. You do not need to use the provided templates, however, your documents must include all of the information in them including headers (if included), identification blocks, and footers. *Make sure you update all identifying information in the footers (typically the assignment and group numbers – the file name, date, and paging update automatically).*

GOOD PROGRAMMING PRACTICES

For all Java classes

- Name your classes meaningfully.
- Make your instance variables **private**
- Include **constructors** to initialize your instance variables.
- Derived class constructors should **leave initialization of super class instance variables** to the super classes' constructors:
  - Remember the call to the super classes constructor is: *super(<init1>, <init2>,..)*
- Include **Accessor** and **Mutator** methods for all instance variables if useful *(please ask if you're not sure what these are)*
- Include a *main()* method for testing (unless it's an abstract class or your application's main class) and test before you proceed…
- Add 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.

  Note: Javadoc comments are required for all methods. You don't need to generate the Javadoc documentation/run the Javadoc utility but, if you do, the documentation must include all classes in your project, include public members only, and go in a folder named *doc* directly under the project root folder (a sibling folder to *src*).

WRITE-UP

ANALYSIS

1. Briefly describe your application (in 'real world' terms). Include relevant decision points (e.g. how to 'clean' text in the spell checker –or– how to decide how to divide up the groceries). What issues/problems arose during the design, implementation, integration, testing phases and how did you resolve them?

2. Is the analogy of a bag of groceries represented accurately by the Bag ADT? Explain why or why not.

3. Would the Bag ADT be accurate for representing a real dictionary? Explain why or why not?

4. Would you say that the Bag ADT is useful in general? Why or why not?

5. Describe the differences (if any) between running your application using the fixed-sized array Bag ADT implementation compared to the linked-list implementation. Briefly describe the reasons for the differences (if any). If you tested your application with any of your Bag ADT implementations, include them in the comparison.

SUMMARY

1. **About your team:**
    a. How did you "divide up" the work so that each student still met the objectives for the assignment (i.e., learn, understand and apply the concepts).

    b. How did you coordinate code changes/testing?

    c. Other observations about working in a team?

2. **Where did you have trouble with this assignment? How did you move forward? What topics still confuse you?**

3. **What did you learn from this assignment? (Please be specific)**

4. **How could this assignment be improved in the future?**

SUBMITTING YOUR WORK

1. Make sure the course number, assignment number, your group number, and all of team member name(s) are in all project files you create or modify. Do not add this information to supplied files that you do not modify.

2. Your write-up and presentation file names must be consistent with the templates:
   - Required: update/insert your group number
   - Optional: update the date/version number

3. Your Java class files must be 'properly' named/match the class defined within. You may name your project folder as you wish.

4. You must include a comprehensive set of unit tests for your classes (optional but recommended for your application). I suggest that the entire team define the test 'suite'. For simplicity, include a *main()* method for each class to run your tests; create private utility methods as appropriate for the testing.

   Style requirement: For unit tests, your *main()* method must follow all public and private methods which implement the API/ADT. *private* utility methods for testing must follow the *main()* method.

5. Spell check and grammar check your work!

6. Make a **.zip file** for your project (that is the easiest compressed file for me to work with). ***.rar's, .tar's, .tar.gz's are not acceptable!***

   - Include your entire project folder (including subfolders).
   - Your PowerPoint presentation and write-up must go in the project root folder. **Do *not* attach them directly to the Blackboard submission.**
   - In **Blackboard**, **attach** your solution (.zip) file to the submission for this assignment. **Only one team member submits for the entire team**.

GROUP PRESENTATION

1. All group members are expected to participate in the presentation.

2. Make sure your group number and all of your 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 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 typically 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. I will provide HDMI, VGA, and Thunderbolt cables/connectors.

GUIDELINES ON PLAGIARISM IN COMPUTER SCIENCE

CLEAR PLAGIARISM

A clear case of plagiarism exists if a student **passes off someone else's work as his or her own**. Examples of such behavior include (but are not limited to) the following:

- A group of students each performing a separate part of an assignment. Each student in the group then hands in the cumulative effort as his or her own work.
- A student making cosmetic alterations to another's work and then handing it in as his or her own.
- A student having another person complete an assignment and then handing it in as his or her own.
- A student handing in (as his or her own) a solution to an assignment performed by someone else from a previous offering of the course.

These are all cases of indisputable plagiarism and are characterized by the submission of work, performed by another, under one's own name.

PERMITTED COLLABORATION

Collaboration and research, where they do not constitute plagiarism, should be generally encouraged. These efforts constitute honest attempts at obtaining a deeper understanding of the problem at hand. They can also serve as a validation of a student's approach to a problem. Some examples are as follows:

- A student researching existing, general public approaches (but not source code solutions) to a problem presented as an assignment.
- A group of students discussing existing, public approaches to a problem presented as an assignment.
- A group of students discussing the requirements of an assignment.
- A student discussing the merits of his or her proposed solution to an assignment with the instructor or teaching assistant.

Provided that no plagiarism is committed, these are all valid (and encouraged) activities.

THE GRAY AREA

The differences between the examples of plagiarism and encouraged collaboration above are those of originality and acknowledgment. In general, any work submitted without acknowledgment which is not the original work of the indicated author constitutes plagiarism. Some examples which illustrate this point follow. It should be noted that while some of the examples do not constitute academic misconduct, they may be of questionable academic value. It should also be noted that anyone (knowingly or through negligence) contributing to someone else's academic misconduct is also guilty of the same.

- It is acceptable to use solution proposals presented by the instructor or teaching assistant. The acknowledgment is implicit. Explicit acknowledgment is not usually required.
- It is not acceptable to use publicly available work without acknowledgment.
- It is not acceptable to use a full or partial solution obtained from or by another through any means (verbal, written, electronic, etc.), without that person's permission.
- It is not acceptable to collaborate on a single solution without acknowledgment.
- It is not acceptable to discuss solutions or partial solutions to a problem and then incorporate them without acknowledgment.
- It is acceptable to implement a standard solution to a problem without acknowledgment, but it is not acceptable to incorporate someone else's implementation without acknowledgment. Here standard solution means a commonly used data structure or algorithm.
- It is not acceptable to re-submit an assignment from another course or a previous offering of the same course without acknowledgment, regardless of authorship.
- It is not acceptable to make a solution available as an aid to others.
- You may not request solutions on any internet site (e.g. stackoverflow.com).

This set of examples helps define the bounds between encouraged behavior and misconduct, but does not constitute an exhaustive set.