APPLICATION 2: STACK

TOPIC(S)

Stack Applications and Implementations

READINGS & REVIEW

- Carrano, Data Structures and Abstractions with Java, Chapters 5 & 6
- Stack ADT & implementation lectures

OBJECTIVES

Be able to:

- *Use* a Stack ADT in a **Last-In-First-Out (LIFO)** based project.

INSTRUCTIONS

1. **You will work in groups of 3 members – self-select your groups.** Contact me ASAP if you have any concerns or problems with your group assignment – I will do my best to address them.
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 or as announced.
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: STACK ADT APPLICATION

TITLE	INSTRUCTIONS
Select one	**1. Calculator** a. Read expressions from an external file. For extra credit, you may *also* create a GUI (optional). b. Using one or more Stacks, evaluate each expression and (minimally) display the expression and its result. **Detailed instructions (Project #9; page 179):** Write a program that graphically displays a working calculator for simple infix expressions that consist of: • single-digit operands • the operators: +, -, *, and / • parentheses Make the following assumptions: • unary operators (e.g. -2) are illegal • all operations, including division, are integer operations (and results are integers) • the input expression contains no embedded spaces and no illegal characters • the input expression is a syntactically correct infix expression • division by zero will not occur (consider how you can remove this restriction) If you create a GUI application, the calculator has a display and a keypad of 20 keys, arranged as follows: C | < | Q | / 7 | 8 | 9 | * 4 | 5 | 6 | - 1 | 2 | 3 | + 0 | (|) | = As the user presses keys to enter an infix expression, the corresponding characters appear in the display. The C (Clear) key erases all input entered so far; the < (Backspace) key erases the last character entered. When the user presses the = key, the expression is evaluated by your Calculator class and the result appended to the right end of the expression in the display window. The user can then press C and enter another expression. If the user presses the Q (Quit) key, the calculator ceases operation and is erased from the screen.

TITLE	INSTRUCTIONS

2. Lisp Expression Evaluator

a. Read expressions from an external file –or- create a GUI (optional).
b. Using one or more Stacks, evaluate each expression and (minimally) display the expression and its result.

Detailed instructions (Project #7; page 178):

In the language Lisp, each of the four basic arithmetic operators appears before an arbitrary number of operands, which are separated by spaces. The resulting expression is enclosed in parentheses. The operators behave as follows:

- (+ a b c …) returns the sum of all the operands, and (+) returns 0
- (- a b c …) returns a – b – c - …, and (- a) returns –a; the – (minus) operator requires at least one operand
- (* a b c …) returns the product of all of the operands, and (*) returns 1
- (/ a b c …) returns a / b / c / …, and (/ a) returns 1/a; the / (divide) operator requires at least one operand

You can form larger arithmetic expressions by combining these basic expressions using a fully parenthesized prefix notation. For example, the following is a valid Lisp expression:

(+ (- 6) (* 2 3 4) (/ (+ 3) (*) (- 2 3 1)))

This expression is evaluated successively as follows:

(+ (- 6) (* 2 3 4) (/ 3 1 -2))
(+ -6 24 -1.5)
16.5

Design and implement an algorithm that uses a stack to evaluate a legal Lisp expression composed of the four basic operators and integer values. Write a program that reads valid Lisp expressions, evaluates them and displays the results.

3. Student choice

If you're feeling creative and have a great idea for an application of the Stack ADT (using a *Stack<T>* implementation) then make a proposal (***I need to approve it first***).

Notes:
Your application must produce output (typically to the console except for GUIs) 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.' 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. Stack ADT implementation including:
    - StackInterface.java
    - VectorStack.java
    - StackDriver.java (test program for VectorStack.java in case you have problems with using the Stack)

2. For each application: a text file containing *valid expressions* to evaluate – *you must demonstrate that your application works with this file*. If you do a GUI calculator, your calculator must be able to evaluate the same expressions. In addition, two additional files are provided which test extended capabilities (multi-digit numbers) for valid expressions and invalid expressions – *your application does not need to work with either of these two files.*

3. A PowerPoint document/template for the presentation.

    Unless there is a compelling reason to do otherwise, please use PowerPoint for your presentation. *If you wish to use a different program, you need to get my approval in advance*. You do not need to use the provided template; however, your documents must include all 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).*

WRITE-UP

ANALYSIS

The write-up is included with the Stack ADT assignment and must be submitted with that assignment. The questions include:

1. In general, how useful is the Stack data structure? Can you think of other uses aside from those discussed in the applications chapter?

2. If we didn't have the Stack data structure can you think of another way to implement the LIFO (last in first out) concept?

3. Briefly describe your application (in 'real world' terms). Include relevant decision points (e.g. how to handle operator precedence and parenthesized expressions). What issues/problems arose during the design, implementation, integration, testing phases and how did you resolve them?

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

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) – be specific regarding responsibilities for implementation (each of the classes, methods, etc.), debugging, testing, write-up, presentation, etc.

    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 your project files.

2. Your presentation file name must be consistent with the template:
    a. Required: update/insert your group number
    b. 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 (not necessary 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 and any inner classes. *private* utility methods (typically *static*) for testing must follow the *main()* method.

    Style requirement: For console applications, you must read a file containing the expressions to evaluate. 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.

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). ***No other compressed formats are acceptable!***

    - Include your entire project folder (including subfolders).
    - Your PowerPoint presentation must go in the project root folder. **Do *not* attach it directly to the Blackboard submission.**
    - In **Blackboard**, **attach** your zipped solution 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. If someone is out sick or is otherwise unable to attend class the day of the presentation, be prepared to present anyway.

2. Make sure your application name, group number, and all group 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 – try 20 or 22 pt - 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 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.