ENG 1D04, Lab 12: Objects, Classes, and Exceptions

This assignment should be submitted via ELM before the end of the lab session. If you are not done the lab, please finish it at a later time and submit it using ELM. Although the submissions for unmarked assignments are not graded, it is good to have a record of the work that you have done in ENG 1D04.

Background

A *stack* is a data structure that holds a finite sequence of values such that the values are accessed according to the principle of *last in first out (LIFO)*. This means the last element added to a stack is always the first element removed from the stack. The *height* of a stack is the number of elements in the stack.

The following are operators that are usually associated with a stack:

1. A constructor that builds an *empty stack* of height 0.
2. A selector named *height* that returns the height of the stack.
3. A selector named *top* that returns the top element of the stack (i.e., the last element "pushed" onto the stack).
4. A mutator named *push* that adds ("pushes") a new element to the stack and, as a result, increases its height by 1.
5. A mutator named *pop* that removes ("pops") the top element from the stack and, as a result, decreases its height by 1.

Stacks are employed extensively in computer systems (e.g., the *call stack* in an implementation of a programming language).

Overview

Your program will test a class of stack objects. Design, implement, and test the application described in the requirements below. An unfinished definition of a class named `Stack` representing stacks whose elements are of type `int` is in a file named `StackUnfinished.txt` attached to this assignment on ELM.

Requirements

1. Your project's `Form1.cs` file contains a `Stack` class after the `Form1` class. (You are free to copy and paste `StackUnifinished.txt` directly into your code.)
2. The `Stack` class contains the following public methods:
   a. `public Stack()`.
   b. `public int height()`.
   c. `public int top()`.
   d. `public void push(int x)`.
   e. `public void pop()`.

f. `public void reset()`.

g. `public string contentsToString()`.

3. The contents of the stack are stored in an array named `contents`, and the height of the stack is stored in a variable named `ht`. These two fields are marked as `private`. If the height of the stack is n, the contents of the stack is the first n elements in the array. (The other elements in the array are inaccessible and can thus be ignored.)

4. The constructor `Stack` builds an empty stack (of height 0).

5. The `top` and `pop` methods throw an exception when the stack is empty (i.e., its height is 0), and `push` throws an exception when the stack is full (i.e., its height is `MAX`).

6. The `reset` method resets the stack to the empty stack. The method does not directly access the private fields `contents` and `ht`.

7. The `contentsToString` method builds a string of the form

   `Stack contents: [`$a_0, a_2, \ldots, a_{n-1}$`]`

   where $a_0, a_2, \ldots, a_{n-1}$ is the finite sequence held by the stack (i.e., the first n elements in the array), a_0 is the first element pushed onto the stack, and a_{n-1} is the last element pushed onto the stack.

8. A graphical user interface (GUI) designed by you tests the `Stack` class. The user interface contains controls to enable the user to create an empty stack, to select its height and top element, to push elements onto and pop elements from the stack, to reset the stack to the empty stack, and to display the contents of the stack using the `contentsToString` method.

Remarks

1. It is not a requirement, but if you have time program your user interface to catch the exceptions that are thrown by the `top`, `push`, and `pop` methods using a try-catch-finally statement.

2. The `Stack` class only allows stacks of elements of type `int` to be constructed. A more sophisticated stack class would allow stacks of elements of any type to be constructed.