Project 2: Data Structures

Introduction

Definition

Data Structure: A data structure stores and organizes data in memory so that it can be used efficiently. Data structures provide a means to manage large amounts of data efficiently. efficient data structures are a key to designing efficient algorithms. Any class is considered a data structure.

There are classes that specialize in storing a collection of user-provided data and provide management for that data. These data structures are called collections. Collections come in two flavors, those that maintain a specific order in the data and those that do not; ordered and unordered collections respectively.

An ordered collection maintains an insertion order which means upon displaying the collection's contents, it will display the data in an order relative to the order in which it was added.

For this project you will create two very common types of collections, a *stack* and a *queue*. These collections are used everywhere in programming. From compilers to operating systems, where there is a need for efficiently keeping data in order, you will find a stack or a queue.

Stack

A stack is a data structure that allows you to store data and retrieve it in a *last-in first-out* (LIFO) manner. It is similar to the way activation records are stored in the runtime stack of a program (as discussed in class). It can be used to reverse the order of data stored in another ordered collection. Mainly they are used to implement parsers, expression evaluation, and backtracking algorithms.

A pile of books, a stack of dinner plates, a Pez dispenser can all be thought of as examples of stacks. The basic operating principle is that the last item you put on the stack is the first item you can take off.

Queue

A queue is a *first-in first-out* (FIFO) collection. Queues are used for any situation where you want to efficiently maintain a First-in-first out order on some entities. These situations arise literally in every type of software development.

A good example of a queue in the physical world is the line at a grocery store. If the people in the line behave in a civilized manner, the first person in the line should be the first person that is taken care of by the cashier. In British English, such lines are referred to as "queues"--no one there like standing in a queue either..

Requirements (use Visual Studio Code)

- Implement the two classes using the descriptions on the next two pages.
  - The two collections that you build will store data of type *int*.
  - You need to include pre and post conditions comments above each method. They are provided with the class descriptions below but you should write them in your own words.
  - **You must store the values that are added to these collections in private arrays called, "store".** Each class will have one.
  - You must automatically resize the array if it gets full.
    - When resizing, simply create a new array of double the previous size.
    - Write a separate private method for resizing.
    - Resizing should only happen when adding an int and only when the array is full.
  - The stack and queue will require exception classes called, "EmptyStackException" and "EmptyQueueException" respectively. These classes are provided with this assignment.
- Create two UML Class diagrams, one for ***ArrayStack*** and the other called ***ArrayQueue***.
  - You will need to use a program that enables you to do such modeling.
  - There are free programs available, Microsoft Visio (available through school), or you can use a desktop publisher.
  - Whatever program you decide to use, You should submit two PDF files, one called **ArrayStackUML.pdf** which contains the diagram of the ArrayStack class, and the other called **ArrayQueueUML.pdf** for the diagram of the ArrayQueue.
  - **Do not copy these diagrams from the internet.**
- Implement a class called *Main* in which you will create the main method to test your two collections.
  - The main method will consist of only method calls.
  - You will write a test method for each test and call it from main.
  - Each test method should test only one method of a collection
  - Test every method of each collection.
  - Feel free to write as many helper-methods as you wish.
  - There will not be any need for a user interface in this program.
- Each test method will test the functionality of only one feature (method) of a collection and report whether it works or not.
  - The output of each test should include:
    - The name of the class under test
    - An explanation of what was tested. This output string should be stored in a variable and be the first statement in the method.
    - Whether the test passed or failed
  - Please organize this class in an orderly way.

Method Declarations for the ArrayStack and ArrayQueue Classes

Do not alter the method headers or their functionality in any way!

```
public class ArrayStack{
   /** Adds a new entry to the top of this stack.
      @param newEntry  An int to be added to the stack. */
   public void push(int newEntry);

   /** Removes and returns this stack's top entry.
      @return  The int at the top of the stack.
      @throws  EmptyStackException if the stack is empty before the operation. */
   public int pop();

   /** Retrieves this stack's top entry.
      @return  The int at the top of the stack.
      @throws  EmptyStackException if the stack is empty. */
   public int peek();

   /** Detects whether this stack is empty.
      @return  True if the stack is empty. */
   public boolean isEmpty();

   /** Removes all entries from this stack. */
   public void clear();
}
```

EmptyStackException will be provided with this assignment, **do not import it**. This exception can be thrown with the statement, “throw new EmptyStackException();” Do not change the EmptyStackException, what is provided will be all that you will.

Do not alter the method headers or their functionality in any way!
Queue:

```
public class ArrayQueue{
  /** Adds a new entry to the back of this queue.
      @param newEntry  An int to be added. */
  public void enqueue(int newEntry);

  /** Removes and returns the entry at the front of this queue.
      @return  The int at the front of the queue.
      @throws  EmptyQueueException if the queue is empty before the operation. */
  public int dequeue();

  /**  Retrieves the entry at the front of this queue.
      @return  The int at the front of the queue.
      @throws  EmptyQueueException if the queue is empty. */
  public int getFront();

  /** Detects whether this queue is empty.
      @return  True if the queue is empty, or false otherwise. */
  public boolean isEmpty();

  /** Removes all entries from this queue. */
  public void clear();|
}
```

EmptyQueueException will be provided with this assignment. An exception can be thrown with the statement, “throw new EmptyQueueException();”. Do not change the EmptyQueueException, what is provided will be all that you will need.

What To Submit

This is your first multi-file submission. You will use a zip utility to compress all of the files into a single file.

- Create a folder called, “[your cpp username]_p2”. For example, my folder would be called called, “daatanasio_p2”
- Copy into this folder your the following files that you created
    - ArrayStackUML.pdf - the UML class diagram that you created for your ArrayStack class
    - ArrayQueueUML.pdf - the UML class diagram that you created for your ArrayQueue class
    - ArrayStack.java - Your implementation of the ArrayStack class
    - ArrayQueue.java - Your implementation of the ArrayQueue class
    - Main.java - Your implementation of the the Main class that holds the main method and test-methods
    - **Do not include the exception classes, they must not be changed.**
- Once you have all of your files in this folder, zip it. All modern operating systems have the ability to compress a file pre-installed.
- This will create a file called “[your cpp username]_p2.zip”. For me, the file would be called, “daatanasio_p2.zip”.
- Check out this guide on how to zip a folder on Windows and Mac:
    - https://www.hellotech.com/guide/for/how-to-zip-a-file-mac-windows-pc
- You will be heavily graded on code quality. Please take your time when implementing your code and keep it neat.
- **Submit only the zip file before the deadline.**