GDAPS1 – Practice Exercise

Custom Stacks & Queues

# Objective

Implement and test your own custom generic Stack and Queue data structures.

# Details

Stacks and queues are used in a variety of ways in games (see below for examples). You’re going to create GameStack and GameQueue classes, based on the provided IStack and IQueue interfaces, respectively. Your implementations will use List< >’s internally to store information.

***Do NOT use C#’s built-in Stack or Queue classes!*** *You’re going to learn how they work by writing them yourself.*

In the card game Magic: the Gathering, spells being cast are always put onto a “stack” (which is actually called “The Stack”). This allows other players to respond to the spells before they actually affect the game. When all players are done casting, the spells on the stack resolve in reverse order, since stacks are First-In-Last-Out data structures.

Often in MMOs, a popular server will reach its player capacity. Players attempting to log in while the server is full are placed in a queue, and allowed in once other players log out.

# Interfaces

interface IStack<T> interface IQueue<T>

{ {

bool IsEmpty { get; } bool IsEmpty { get; }

int Count { get; } int Count { get; }

void Push(T s); void Enqueue(T s);

T Pop(); T Dequeue();

T Peek(); T Peek();

} }

# GameStack and GameQueue

Create a GameStack<T> class that implements the IStack<T> interface above. Also create a GameQueue<T> class that implements the IQueue<T> interface.

* You’ll need a List<String> field to hold the actual data in each class.
* Implement the rest of the methods required by the corresponding interface.
* Note that the *count* will always be the same as the internal List’s *count* – no need to duplicate that data
* Be sure to add and remove data from the correct place in the List.
* *Remember: Do* ***NOT*** *use C#’s built-in Stack or Queue classes for this!*

# Main Method

First, test your GameStack:

1. Create a GameStack object that can hold strings.
2. Print the name of at least 5 spells (just Strings), and also add each one to the GameStack object.
3. In a loop, pop one spell off the GameStack at a time, until the stack is empty. Each time you do this, **print out the spell**.

Then test your GameQueue:

1. Print the names of at least 5 players attempting to join a server, and add each name to your GameQueue.
2. In a loop, pull one player off the queue at a time, until the queue is empty. Each time you do this, print out a message about the player joining the server, as well as how many players are left in the queue

# Sample Run

The following spells are being put on the stack:

* Shock
* Fork
* Counterspell
* Force of Will
* Deflection

Spells resolving (in reverse order):

* Deflection
* Force of Will
* Counterspell
* Fork
* Shock

The following players are joining the queue:

* Bob
* Tom
* Sara
* Jim
* Larry

“Bob” has joined the server: 4 player(s) left in queue

“Tom” has joined the server: 3 player(s) left in queue

“Sara” has joined the server: 2 player(s) left in queue

“Jim” has joined the server: 1 player(s) left in queue

“Larry” has joined the server: 0 player(s) left in queue

# Submission

All of your work must be commented and follow this course’s coding standards. **Read through the Coding Standards document (located in MyCourses) to check over your code before you complete your program. Make sure you follow the coding standards for all code you create.**

1) Submit: Submit your program to the appropriate Assignments dropbox in MyCourses.

2) Check-off: Show your working program to the instructor or TA. If you do not finish before class ends, complete the exercise for homework and show one of us in-class on the next class period. If your program works as expected, you will be “checked off” to earn credit for the exercise.