Stack, Queues, and Priority Queues: Linked List Based

10-26-2004

Opening Discussion

- What did we talk about last class? Does anyone have any code to show?
- Do you have any questions about the assignment? Remember that assignment #4 is due today and assignment #5 is due a week from today.
- Let's look at some code related to stuff from last class real quick.

Stacks as Linked Lists

- We have looked at how we can implement the Stack interface with an array, but we can also do it with a linked list.
- For a linked list stack, we only need a head pointer and all the pushes and pops go on it or pull from it.
- The main conceptual difference from an array based stack is just which "end" we are pushing to and popping from.

Queues as Linked Lists

- Queues with arrays required a bit of extra thinking to make them circular. They are actually easier with a linked list.
- We keep both a head and a tail. One is the front and the other is the back of the queue. To figure out which is which, think about which one you can easily remove from.
- We make the choice because we want O (1) operations.

The Priority Queue ADT

- An ADT that is slightly more advanced than the Stack or Queue is the Priority Queue. This ADT acts like a queue, but with the added complication that the elements have a priority.
- When elements are removed from it, it is always the highest (or lowest) priority element that is taken out next.
- We want be be able to find that element fast. Fast adds are nice too.

Sorted Linked Lists

- We can easily make a linked list data structure that is sorted by modifying the insert method so that it inserts the new node into the proper position in the list to be sorted.
- Building a sorted linked list is almost like an insertion sort. The problem is that the insert is a O(n) operation.

Using SLLs for Priority Queues

- If we build a sorted list based on priority, then it automatically works as a priority queue. Items are always removed from the front of the list and inserted where they belong in the list.
- This gives fast, O(1), removes, but the adding is O(n). We'll look at a faster alternative later in the semester.

Code

- Now we will look at code for some of these things.

Testing Code

- Testing of code is a very important topic and something that you should be doing. In some cases, just running the game does a fairly good test, but it isn't always the easiest one to debug.
- Remember that in Java you can put a main in any class. You should put a main in things like your linked list class where all you do is test and print out stuff.

Error Handling

- In the code that you are probably used to, there are two ways that a function can tell you if an error occurred.
  - Return an error code.
  - Set a flag that should be checked.
- Both of these are very easy to ignore which can lets the errors “propagate” and makes debugging much more difficult.

Enter Exceptions

- An alternate approach to error handling is the use of exceptions. As their name implies, exceptions are things used for exceptional events.
- When an error occurs, the code will "throw" an exception. When an exception is thrown control pops up the stack until code is found to handle it. If no handler is found that thread exits completely.

try, catch, and finally Blocks

- There are 4 syntactic components of dealing with exceptions. The ones you will write most are try and catch blocks.
- When you have a section of code that can have an exception thrown in it that you want to handle, you put it in a try block.
- After the try block you can have one or more catch blocks. Each one specifies a type and catches anything of that type (including subtypes).
- A finally block is like a default. In addition, it

throws and throw

- Sometimes a method can have an exception occur in it, but it doesn't know how to handle it. In this situation, the exception should be added to the throws clause of the method declaration.
- If you want to throw your own exceptions you use the throw statement. It is followed by an object that is the exception to be thrown.

Checked vs. Unchecked Exceptions

- There are two broad categories of exceptions.
  - Checked exceptions must either be caught, or they must appear in the throws clause of the method.
  - Unchecked exceptions don't have to do this and often shouldn't be handled. If an unchecked exception arises it typically signifies a major problem and the code should crash.
- Subtypes of RuntimeException are

Benefits of Exceptions

- Exceptions are nice because they can't be ignored. They tell you there is a problem immediately instead of letting the code run on until it has a serious problem or just leaving logic errors.
- The Exception class also has handy methods like printStackTrace() that can be used to help with the debugging process as well.

Code

- Let’s go look in the javadocs at some of the function calls that can throw exceptions then write some code of our own to throw them and use that code.

Minute Essay

- How do you think the linked list based queues and stacks compare to those we looked at using arrays?
- Remember that assignment #4 is due today, design #5 and quiz #4 are on Thursday.