New CS15 Collaboration Policy

- Starting with Tetris, can collaborate on the mini-assignment and all material covered in the mini-assignment over the course of the entire project, to the extent of the mini-assignment.
- Cannot write up your mini-assignment solution together.
- Can talk to a friend about how to design containment diagrams for Tetris.
- Cannot discuss implementation specific questions not in mini-assignment.
- Must include logins of everyone you worked with when you hand in your project.
- More details can be found on the website.

Abstract Data Types (1/2)

- To use a method, need to know its essentials: signature and return type.
- Additionally, documentation tells us purpose, error conditions, what resources (such as classes/packaged) the method needs, etc.
- Set of signatures and return types for an entire class designed to store and manage data is called an Abstract Data Type (ADT) in Java. ADTs are supported by the Interface feature of Java.
- We don’t know anything about its implementation – encapsulation.

1. This is an informal definition. ADT also has a mathematical definition.

Abstract Data Types (2/2)

- These linked list lectures show how to implement various list ADTs using linked nodes, and then use those ADTs with simple programs to demonstrate their use.
- Arraylists and LinkedLists are ADTs that adhere to Java’s list interface.
- Note: full description of an ADT is sometimes called an API (Application Program Interface).
- In “Application Program Interface” coined by former undergraduate John Croll in 1985.

Stacks

- Stack has special requirements for insertion and deletion: push and pop.
- Instead of being able to insert and delete nodes from anywhere in the list, can only add and delete nodes from top of stack.
- LIFO (Last In, First Out).
- We’ll implement a stack with a linked list and then use it in a simple demo app.
Stack Constructor
- When generic Stack is instantiated, it contains an empty MyLinkedList.
- When using a stack, you will fill in <Type> with whatever type of object your stack will hold -- enforces homogeneity.

Methods of a Stack
- Add element to top of stack
- Remove element from top of stack
- Returns whether stack has elements
- Returns number of elements in stack

Pushing an Object
- When an element is pushed, it is always added to front of list
- Let's see what this does...

Popping an Object
- When popping an element, it is always removed from top of Stack, so call removeFirst on MyLinkedList.
- removeFirst returns element removed, and Stack in turn returns it.
- Remember that the removeFirst method of MyLinkedList checks to see if list is empty.
- Let's see what this does...

isEmpty
- Stack will be empty if _list is empty.
- Returns a boolean that is true if Stack is empty and false otherwise.

size
- Size of Stack will be the number of elements that the Linked List contains.
- Size is updated whenever a Node is added to or deleted from _list during push and pop methods.
Clicker Question 1
Look over the following code:
Who’s left in the stack?

Stack myStack = new Stack();
myStack.push(thing1);
myStack.push(thing2);
myStack.pop();
myStack.push(thing3);
myStack.pop();
myStack.pop();

First Example: Execution Stacks
- Each method has an Activation Record (AR)
  - contains an execution pointer to instruction to be executed next in method
  - also contains all local variables and parameters of method
- When methods execute and call other methods, Java uses a Stack to track these calls
  - when a method calls another method, Java adds activation record of called method to stack
  - when new method is finished, its AR is removed from stack, and previous method is continued
  - method could be different or a recursively called clone

Execution Stacks

Stack Trace
- When an exception is thrown in a program, get a long list of methods and line numbers known as a stack trace
  - exception in thread "main" at ...
- A stack trace prints out all methods currently on execution stack
- If exception is thrown during execution of recursive method, prints all calls to recursive method

Bootstrapping ADT’s
- In effect, this stack ADT is implemented as a thin wrapper over a LinkedList ADT, but user has no knowledge of that
- Could also implement it with an array or ArrayList, but that would be more inefficient due to constant data movement
- We’ll use the same technique to implement a Queue
What are Queues?

- Similar to stacks, but elements are removed in different order
  - Information retrieved in the same order it was stored
  - FIFO: First In, First Out (as opposed to stacks, which are LIFO: Last In, First Out)

Examples:
- Standing in line at the checkout counter or movie theater
- Waiting list for TA hours

Enqueuing and Dequeuing

- Enqueuing: adds a node
- Dequeuing: removes a node

Our Queue

- Let’s use another `LinkedList` to help us make our `Queue`
- Contain a `MyLinkedList` within `Queue` class
  - `enqueue` will add to the end of `MyLinkedList`
  - `dequeue` will remove the first element in `MyLinkedList`

```java
public class Queue<T> {
    private MyLinkedList<T> _list;

    public Queue() {
        _list = new MyLinkedList<T>();
    }

    // Other methods added
}
```

enqueue

- Just call _list’s addLast method!
- This will add node to end of _list

```java
public void enqueue(T typeNode) {
    _list.addLast(typeNode);
}
```

dequeue

- We want first node in _list
- Use _list’s removeFirst method!
- What if _list is empty? There will be nothing to dequeue!
- Our MyLinkedList class’s removeFirst() method returns null in this case, so dequeue does as well

```java
public Type dequeue() {
    return _list.removeFirst();
}
```

isEmpty() and size()

- As with Stacks, very simple methods; just delegate to MyLinkedList

```java
public int size() {
    return _list.size();
}
```

```java
public boolean isEmpty() {
    return _list.isEmpty();
}
```

Clicker Question 2

In order from first to last, a queue contains the following: OneFish, TwoFish, RedFish, BlueFish. We remove objects by calling dequeue() and then push() them onto a stack. At the end of the process, what is the order of the stack from top to bottom?

A. OneFish, TwoFish, RedFish, BlueFish
B. OneFish, BlueFish, TwoFish, RedFish
C. BlueFish, RedFish, TwoFish, OneFish
D. It’s random every time

Exercise 1 (1/5)

- How can we use a Stack to reverse a Linked List?
- Linked List: Lorax, Grinch, Horton, Yertle

Exercise 1 (2/5)

- Solution:
  1. while Linked List is not empty, remove from Linked List and push elements onto Stack
  2. then, while Stack is not empty, pop elements from Stack and add to Linked List
Exercise 1 (3/5)

```java
while(!list.isEmpty()) {
    stack.push(list.removeLast());
}
```

Exercise 1 (4/5)

```java
while(!stack.isEmpty()) {
    _list.addLast(stack.pop());
}
```

Exercise 2 (1/2)

- Check for balanced parentheses in a given string
- Balanced: `(())`)
- Not balanced: `[]`

Exercise 2 (2/2)

- Go through every character, if it is a starting bracket, push it onto the stack
- If it is a closing bracket, pop from the stack
- The bracket you pop should be the opening bracket that corresponds to the closing bracket you are looking at
  - if it is not, return false
- If you get through every character and you haven't returned false, check if stack is empty
- If it is, the brackets are balanced!

Problem 2 Pseudocode

```java
for each character:
  if it is a starting bracket, push it onto stack
  if it is a closing bracket, pop from the stack
  if the popped character is not the matching opening bracket
    return false
  if stack is empty:
    return true
```

```java
[
(())
]
for each bracket in string:
  if it is a starting bracket:
    push onto stack
  if it is a closing bracket:
    pop from the stack
    if the popped character is not the matching opening bracket:
      return false
if stack is empty:
  return true

for each character:
  if it is a starting bracket:
    push onto stack
  if it is a closing bracket:
    pop from the stack
    if the popped character is not the matching opening bracket:
      return false
if stack is empty:
  return true

{()}

for each character:
  if it is a starting bracket:
    push onto stack
  if it is a closing bracket:
    pop from the stack
    if the popped character is not the matching opening bracket:
      return false
if stack is empty:
  return true

{()}

{()}

Announcements
• DoodleJump ontime handin is tonight 11:59pm
• DoodleJump late handin is Saturday 10pm
• Tetris is out!
  • In order to receive a grade for Tetris you must complete and sign the new collaboration form