Stacks and Queues Summary

By Spencer Alves and Brandon Cox

Stack

A stack is a collection of objects that are added and removed in ”last in, first out” or “LIFO” order. That is, the first thing added is the last thing removed. For example, if you add A, B, and then C, then the first removed will be C, then B, then A. Adding an item to a stack is called “pushing” to the stack, and removing an item is called “popping” off of the stack. You can’t access anything in the stack that’s not on top without first popping off everything above it.

You can think of a stack as a stack of cards on a table. You can add to the stack of cards by putting one on top, and you can remove from the stack of cards by taking the top one off. You can put as many cards on as you want, and you can take off as many cards as there are in the pile. However, you cannot easily take out a card in the middle or at the bottom of the stack, without taking some cards off first.

In Java, the class is called `java.util.Stack`. Just like `ArrayList`, it needs to be qualified with a class, like `Stack<Integer>`. To push to the stack, call `push(E item)`. To pop off the stack, `pop()` returns the top of the stack. You can also call `peek()` to look at the top item in the stack without removing it. `empty()` tests if there are objects left in the stack, and `search(Object o)` returns how deep a given object is into the stack. Since stacks in Java are Java collections, they inherit many methods shared between all data structures that go against the purpose of a stack, including `get`, `set`, `remove`, `index`, and iteration. If you try to pop an item off an empty stack, you will receive an `EmptyStackException`.

Stacks are used whenever the “first-in, last-out” rule is required. For example, if every possibility in a tree, like a maze, needs to be explored, then a stack should be used. An important use of the stack is the runtime stack on any computer. Each time a new method is called, the old method’s memory is pushed onto the stack, and when the new method is done, the old method’s memory is popped from the stack into current memory space.

Queue

A queue is kind of like the opposite of a stack. Instead of having a “last-in, first-out” structure, it is “first-in, first-out”. Inserting A, B, and then C, will make the first removed A, then B, then C. The first item inserted is called the “head” of the queue, and the last item is the “tail”. You can think of a queue like a line at, say, an amusement park ride. The first person to get in the line is the first person to get on the line. The last person in line will have to wait until the people in front of him/her have gone.

In Java, queue is not a class; rather it is an interface, `java.util.Queue`. The most useful implementing class is `LinkedList`. LinkedList also needs to be qualified with a class. To push to the queue, use `add(E item)`. To pop the head, use `remove()`. `peek()` can also be used to look at the

head without removing it. Finally, all of the methods in `LinkedList` and other Java collections can be used. If you try to remove from an empty queue, you will receive a `NoSuchElementException`.

Queues are generally used when there is a number of things that need to be serviced in order. For example, a printer has a number of jobs that need to be completed in order. Events in event-driven interfaces, like Swing, or, in fact, most GUIs, are stored in queues so that the computer responds to events in the order the user sends them.