

More Object-Oriented Programming

What To Do:

Follow each step carefully. As you complete the lab, submit the source files (.java) problems to the autograder. After finishing, please let one of the AIs know.

You must write sufficient tests and documentation. Not writing tests will result in a zero.

In this question you will implement the `MiniStack` data structure. This is similar to the `MiniArrayList` class from the chapter, but, of course, is a stack and not an array list.

Unlike many stack implementations, however, we will use an array-backed stack. This means that, instead of using a collection of `private` and `static` `Node` classes, the stack will use an array to store its elements. When the array runs out of space, a new one is allocated and the elements are copied over.

- (a) First, design the generic `MiniStack` class. Its constructor should receive no arguments, and instantiate two instance variables: `T[] elements` and `size` to a new array and zero respectively. Remember that you cannot instantiate a generic array, so how do we do that? (Hint: watch the videos/read the book/remember from lecture.) The initial capacity of the array should be set to `INITIAL_CAPACITY`, which is a `private static final` variable declared in the class as ten. **Do not use `System.arraycopy`, even if IntelliJ prompts you to use it. You must do the copying yourself. If you use `System.arraycopy`, we will automatically deduct 20% from your grade.**
- (b) Second, design the `void add(T t)` method, which adds an element onto the top of the stack. The “top of the stack,” when using an array, is the right-most element, i.e., the element with the highest index. It might be a good idea to design a `private` helper method that resizes the underlying array when necessary. Your resize factor, i.e., by what factor you resize the underlying array, is up to you.
- (c) Third, design the `Optional<T> peek()` method, which returns (but does not remove) the top-most element of the stack. If the stack is empty, return an empty `Optional`. Otherwise, return the top-most item wrapped inside an `Optional`.
- (d) Fourth, design the `Optional<T> pop()` method, which returns *and* removes the top-most element. If the stack is empty, return an empty `Optional`. Otherwise, return the top-most item wrapped inside an `Optional`. Be sure that your `add` method still works after designing `pop`.
- (e) Fifth, design the `int size()` method, which returns the number of logical elements in the stack.
- (f) Finally, override the `public String toString()` method to return a string containing the elements of the stack from top-to-bottom, separated by commas and a space. For example, if the stack contains, from bottom-to-top, 10, 20, 30, 40, and 50, the `toString` method returns "50, 40, 30, 20, 10".