Module 15 - Final Review and Exam

# Overview and Objectives

## OVERVIEW

In this final module we will review and have the final exam

## COURSE LEVEL OBJECTIVES (CLO)

Upon completion of this course, you should be able to:

1. Construct modern high quality software systems and reason about them.
2. Properly define software specifications and rep-invariants.
3. Leverage immutability to properly construct threat safe programs.
4. Explain object-oriented concepts such as information hiding, encapsulation, data and type abstraction, and polymorphism.
5. Properly use exception handling
6. Identify when it is appropriate to use inheritance and generics.

## MODULE LEVEL OBJECTIVES (MLO)

Upon completion of this module’s activities, you should be able to:

# Module Video (Wiley-Produced w/Dan Ramos) [3-5 minutes]

## Instructor Screencast: TITLE

# In-class Exercise (MLO 1, 2, 3) [.5 hours]

How well are you prepared for the final? This exercise should help you find out. Piazza discussions encouraged!

public class Stack {  
 private Object[] elements; private int size = 0;  
  
 public Stack() { this.elements = new Object[0]; }  
  
 public void push (Object e) {  
 if (e == null) throw new NullPointerException("Stack.push");  
 ensureCapacity(); elements[size++] = e;   
 }  
  
 public void pushAll (Object[] collection) { for (Object obj: collection) { push(obj); } }  
  
 public Object pop () {  
 if (size == 0) throw new IllegalStateException("Stack.pop");  
 Object result = elements[--size];  
 elements[size] = null;  
 return result;  
 }  
  
 @Override public String toString() {  
 String result = "size = " + size;  
 result += "; elements = [";  
 for (int i = 0; i < elements.length; i++) {  
 if (i < elements.length-1)  
 result = result + elements[i] + ", ";  
 else  
 result = result + elements[i];  
 }  
 return result + "]";  
 }  
}

1. Write a contract for push(Object e).

**SOLS**: version 1 // Effects: If the Object is null, throw NPE Expand the array if more space is needed add Object e to the top of the stack (elements) // Effects: throw NPE if attempt to add a null element, else add element to the top of the stack // Modifies: elements, size # don't go into implementation details, // Modifies: this (or modifies the stack representation) // Do we want Require (Precondition) ? In this example, no need (code already include if e == null throw … ). In general, if you can, avoid having require/precondition , you want the weakest precond (i.e., nothing)

version 2 *precond: none /* ***postcond****: iff null throw exception; ensure adequate capacity; add element to array increase size variable by 1 /* postcond (better, more general, not implementation specific): throw exception if attempt to add a null element, else add element to the top of the stack

1. What is wrong with toString()? Fix it. **SOLS**: code: just want contents up to size and not elements.length (which outputs everything in the array) fix: print "stack" , elements.length = size
2. What rep-invariant is likely broken? Fix it. This includes writing a suitable rep-invariant. SOLS:
   1. elements != null … Correct
   2. contents of elements are not null: INCORRECT (NOT a Repr Inv) size = 2 ["cat", "dog", null, "cow"]
   3. 0 <= size <= elements.length() # size != elems.length
   4. If size > 0 then elements from 0 to size should not be null
3. How would Bloch's Item 25: *Prefer Lists to Arrays* apply here? Would it make the rep-invariant simpler? **SOLS:** If we use ArrayList,
   * pop() doesn't need to set elemennts[size] = null
   * don't even need a separate variable `size` to keep track of Stack size
   * capacity() : no need
4. How would you argue that that pop() is correct (or not)? **SOLS:**
   * correct = satisfy repr and the contract
   * contract: //effects: if this (or stack) is already empty, throw ISE //else remove the top of the stack and remove it
5. What is the problem with pushAll() ? why a contract for it. What would Bloch suggest as an alternative? **SOLS:** contract: if any element is null , then raise exception, otherwise, add everything to this (or the stack)

* public void pushAll (Object[] collection) {  
   //check for null element  
   for (Object obj: collection) {if obj == null throw NPE;}  
   for (Object obj: collection) { push(obj); }  
  }
* //require: all elements are non-null

1. Override equals() (for both cases when elements is Array and ArrayList). What else do you have to do? Do that too. **SOLS**:

* if elements is array, use size (cannot use equal because ["dog", null] != ["dog"]) size = 1 size = 1 ["dog", null] == ["dog"]
* if elements is arraylist, reuse .equals() ..
* hashCode, for arrays: use Bloch's recipe, hash()\*32 + .. for arrayList use hashCode()

# FINAL EXAM – (MLO 1, 2) [~3 hours]

## Purpose

Comprehensive Exam

## Instructions

1. This is an open-book exam. This means that you can access course materials in the book/lecture notes/videos.
2. It is a violation of the honor code to communicate with any other person (except the instructor or TA) about this exam.
3. It is a violation of the honor code to discuss or share the contents of this exam in any way with any student who is currently registered for this course but who has not yet completed this exam.
4. You must type all solutions. You can use plain text format or markdown. If you use something else such as Word or LaTeX, you need to export to PDF and submit the PDF. Do Not submit any code (.java) file. if you need to change the code, put the modified code directly in your submission.
5. You need to submit on Blackboard by the deadline. If, for any reason, you have a problem submitting to BB, submit your final on Piazza in a private post. Your post should also explain your problem.

| Section | Points | Score |
| --- | --- | --- |
| Question 1 | 20 |  |
| Question 2 | 20 |  |
| Question 3 | 20 |  |
| Question 4 | 20 |  |
| Question 5 | 20 |  |
| Question 6 | 0 |  |
| Question 7 | 0 |  |
| Total | 100 |  |

## Question 1

Consider [Queue.java](https://nguyenthanhvuh.github.io/class-oo/files/Queue.java)

/\*\*  
 \* Generic Queue example  
 \* Mutable Version, without specifications  
 \* SWE 619  
 \* @author Paul Ammann  
 \*/  
  
import java.util.\*;  
  
public class Queue <E> {  
  
 private List<E> elements;  
 private int size;  
  
 public Queue() {   
 this.elements = new ArrayList<E>();  
 this.size = 0;  
 }  
  
 public void enQueue (E e) {  
 elements.add(e);  
 size++;  
 }  
  
 public List<E> enQueue\_Immutable(E e){  
 List<E> new\_elements = copy(elements);   
 new\_elements.add(e);  
 return new\_elements;  
 }  
  
  
 public E deQueue () {  
 if (size == 0) throw new IllegalStateException("Queue.deQueue");  
 E result = elements.get(0);  
 elements.remove(0);  
 size--;  
 return result;  
 }  
  
 public List<E> deQueue\_Immutable(){  
 if (size == 0) throw new IllegalStateException("Queue.deQueue");  
 List<E> new\_elements = copy(elements);  
 new\_elements.remove(0);  
 return new\_elements;  
 }  
  
 public E getFirst(){  
 if (size == 0) throw new IllegalStateException("Queue.deQueue");  
 return elements.get(0);  
 }  
  
 public boolean isEmpty() {  
 return size == 0;  
 }  
  
  
 public static void main(String[] args) {  
 // Simple exercise to enQueue/deQueue cmd line args  
 // Usage: java Queue item1 item2 item3 ...  
 Queue <String> q = new Queue <String>();  
 for (String arg : args)  
 q.enQueue(arg);  
 while (!q.isEmpty() )  
 System.out.println(q.deQueue());  
  
 }  
  
}

1. For enQueue, write (i) a partial contract and (ii) a total contract. For each part, if you need to change the code for the contract, do so and explain what you did
2. Write the rep invs for this class. Explain what they are.
3. Write a reasonable toString() implementation. Explain what you did
4. Consider a new method, deQueueAll(), which does exactly what the name suggests. Write a reasonable contract for this method and then implement it. Be sure to follow Bloch’s advice with respect to generics. Explain what you did
5. Rewrite the deQueue() method for an immutable version of this class. Explain what you did
6. Write a reasonable implementation of clone(). Explain what you did.

## Question 2

Consider Bloch’s final version of his Chooser example, namely [GenericChooser.java](https://nguyenthanhvuh.github.io/class-oo/files/GenericChooser.java).

import java.util.\*;  
import java.util.concurrent.\*;  
  
// Bloch's final version  
public class GenericChooser<T> {  
 private final List<T> choiceList;  
  
 public GenericChooser (Collection<T> choices) {  
 choiceList = new ArrayList<>(choices);  
 }  
  
 public T choose() {   
 Random rnd = ThreadLocalRandom.current();  
 return choiceList.get(rnd.nextInt(choiceList.size()));  
 }  
}

1. What would be good rep invariants for this class? Explain each.
2. Supply suitable contracts for the constructor and the choose() method and recode if necessary. The contracts should be consistent with your answer to the previous question. Explain exactly what you are doing and why.
3. Argue that the choose() method, as documented and possibly updated in your previous answers, is correct. You don’t have to be especially formal, but you do have to ask (and answer) the right questions

## Question 3

Consider [StackInClass.java](https://nguyenthanhvuh.github.io/class-oo/files/StackInClass.java). Note of the push() method is a variation on Bloch’s code.

import java.util.\*;  
  
public class StackInClass {  
 private Object[] elements; private int size = 0;  
  
 public StackInClass() { this.elements = new Object[0]; }  
  
 public void push (Object e) {  
 if (e == null) throw new NullPointerException("Stack.push");  
 ensureCapacity(); elements[size++] = e;   
 }  
  
 public void pushAll (Object[] collection) { for (Object obj: collection) { push(obj); } }  
  
 public Object pop () {  
 if (size == 0) throw new IllegalStateException("Stack.pop");  
 Object result = elements[--size];  
 // elements[size] = null;  
 return result;  
 }  
  
 @Override public String toString() {  
 String result = "size = " + size;  
 result += "; elements = [";  
 for (int i = 0; i < elements.length; i++) {  
 if (i < elements.length-1)  
 result = result + elements[i] + ", ";  
 else  
 result = result + elements[i];  
 }  
 return result + "]";  
 }  
 private void ensureCapacity() {  
 if (elements.length == size) {  
 Object oldElements[] = elements;  
 elements = new Object[2\*size + 1];  
 System.arraycopy(oldElements, 0, elements, 0, size);  
 }  
 }  
}

1. What is wrong with toString()? Fix it.
2. As written, pushAll() requires documentation that violates encapsulation. Explain why and then write a contract for pushAll().
3. Rewrite the pop() method for an immutable version of the Stack class. Keep the same instance variables. Rewrite what you did.
4. Implementing the equals() method for this class is a messy exercise, but would be much easier if the array was replaced by a list. Explain why. Note: You are not required to provide a implementation in your answer, but if you find it helpful to do so, that’s fine.

## Question 4

Consider the program below (y is the input).

1 {y ≥ 1} // precondition  
2  
3 x := 0;  
4 while(x < y)  
5 x += 2;  
6  
7 {x ≥ y} // post condition

1. Informally argue that this program satisfies the given specification (pre/post conditions).
2. Give 3 loop invariants for the while loop in this program. For each loop invariant, informally argue why it is a loop invariant.
3. Sufficiently strong loop invariants: Use a sufficiently strong loop invariant to formally prove that the program is correct with respect to given specification. This loop invariant can be one of those you computed in the previous question or something new.
   * Note: show all works for this step (e.g., obtain weakest preconditions, verification condition, and analyze the verification condition).
   * Recall that if the loop invariant is strong enough, then you will be able to do the proof. In contrast, if it is not strong enough, then you cannot do the proof.
4. Insufficiently strong loop invariants: Use another loop invariant (could be one of those you computed previously) and show that you cannot use it to prove the program.
   * Note: show all work as the previous question.

## Question 5

Note: you can reuse your answers/examples in previous questions to help you answer the following questions.

1. What does it mean that a program (or a method) is correct? Give (i) an example showing a program (or method) is correct, an (ii) an example showing a program (or method) is incorrect.
2. Explain the difference between rep invariants, loop invariants, and contract/specifications (i.e., pre/post conds). Use concrete examples to demonstrate the difference.
3. What are the benefits of using JUnit Theories comparing to standard JUnit tests. Use examples to demonstrate your understanding.
4. Explain the differences between proving and testing. In addition, if you cannot prove (e.g., using Hoare logic), then what does that mean about the program (e.g., is it wrong)?
5. Explain the Liskov Substitution Principle (LSP). Use a concrete example to demonstrate LSP. Note: use a different example than the one given in Liskov.

## Question 6

This question helps me determine the grade for group functioning. It does not affect the grade of this final.

1. Who are your group members?
2. For each group member, rate their participation in the group on the following scale:
   * A: Completely absent
   * B: Occasionally attended, but didn’t contribute reliably
   * C: Regular participant; contributed reliably

## Deliverable

* Submit a .java file for your implementation.

## Due Date

Your assignment is due by Sunday 11:59 PM, ET.