This is the final examination for
CSC212: Data Structures
as taught by R. Jordan Crouser and Nicholas R. Howe in Spring 2022.

The following materials are **permitted** while taking this examination:

- a single 8.5x11 sheet of paper (double-sided) containing your own handwritten or typed notes
- blank scratch paper (provided at the end of this packet)

Honor code: no other resources are permitted during this exam.
This includes (but is not limited to): textbooks, online materials, tutors, teaching assistants, and other students.

If you encounter any issues while taking this exam,
the instructors can be reached on Slack:
`@Jordan Crouser` and `@Nicholas Howe`

NAME: SOLUTION KEY

SCORE: _____ out of 58

Question 1. Vocabulary (6 points)

Word Bank:

abstract	call signature	declare	exception	generic	inheritance
initialize	instance	interface	iterator	method	overload
override	public	recursion	static	type	void

Fill in the blank with the term or concept that matches each of the definitions below:

(a) A class, interface, or method that operates on a parameterized type (e.g. `ArrayList<T>`) is called GENERIC.

(b) A(n) ITERATOR is a specialized object used to traverse or retrieve a Collection or Stream object's elements one by one.

(c) Java allows us to OVERLOAD methods defined within a class, which means we can define multiple methods with the same name (as long as they have different argument lists).

(d) When a method in a subclass has the same name, same parameters or signature, and same return type (or sub-type) as a method in its superclass, then the method in the subclass is said to OVERRIDE the method in the superclass.

(e) In Java, space on the heap is allocated when we INITIALIZE an object.

(f) In an architecture diagram, we list the PUBLIC attributes and methods for each class, as well as indicate the relationships between the classes.

Question 2. Tracing Java Programs (6 points)

Consider the following program, which is made up of 4 files:

`MyInterface.java`:

```
1 interface MyInterface {
2   public String modify(String s);
3 }
```

`Main.java`:

```
1 class Main {
2   public static void main(String[] args) {
3     MyClass m = new MyOtherClass();
4     System.out.println(m.modify(args[0]));
5   }
6 }
```

`MyClass.java`:

```
1 class MyClass implements MyInterface {
2   public String modify(String s) {
3     return s.toUpperCase() + "!!!";
4   }
5 }
```

`MyOtherClass.java`:

```
1 class MyOtherClass extends MyClass {
2   public String modify(String s) {
3     s = super.modify(s);
4     return "***" + s + "***";
5   }
6 }
```

(a) What output is printed to the console if we run `java Main "almost done"`?

Solution: `***ALMOST DONE!!!***`

(b) What happens if we add `MyOtherClass m2 = (MyOtherClass)(new MyClass());` between lines 4 and 5 of `Main.java`?

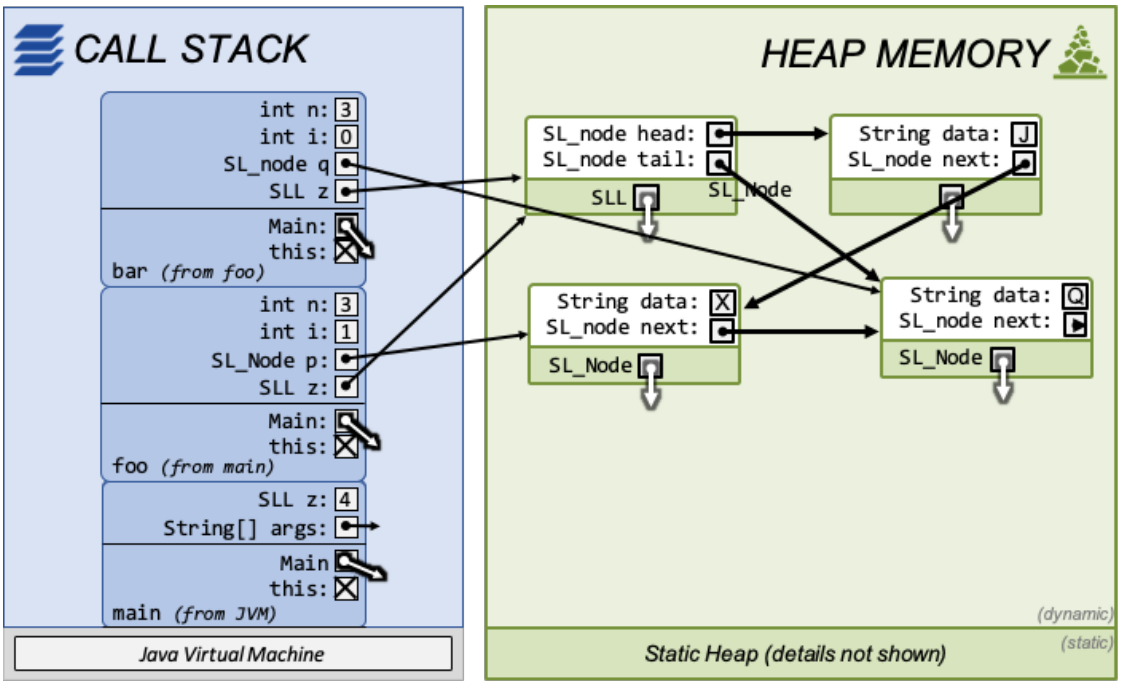
Solution: The program will still compile, but will throw a `ClassCastException` when trying to cast an instance of `MyClass` to `MyOtherClass`.

(c) What happens if we remove `extends MyClass` on line 1 of `MyOtherClass.java`?

Solution: This would cause a compile error generate by the call to `super.modify(s)` in line 3 of `MyOtherClass.java`, because the `super` keyword is only valid when called from within a subclass that has an inheritance relationship with a superclass.

Question 3. Reading Memory Diagrams (8 points)

Consider the following diagram, depicting the state of the computer's memory during the execution of a program:



(a) What is the value of `i` in the currently executing method?

Solution: 0

(b) What is the sequence of function calls that led to the situation shown?

Solution: `main(...)-->foo(...)-->bar(...)`

(c) Suppose that the current function executes `q.setData("R")` and then returns. Will this cause any changes visible to the calling function?

Solution: Yes; the calling method (foo(...) has a pointer to the same `SLL` that contains the `NodeSL` object referenced by `q` in the current context, and so changes to that data structure will be visible.

(d) What value would this expression evaluate to in the current context?
`z.getHead().getNext().getData()`

Solution: X

Question 4. (Flawed) Operations on Lists (8 points)

Consider the list method implementations shown below. Each one has at least one bug, in the form of a special case that is not handled properly.

Example:

```
1  /** Inserts the given item at the head of the SLL */
2    public void addFirst(T item) {
3      head = new NodeSL<T>(item,head);
4    }
```

What's wrong? *This method doesn't update the tail when adding to an empty list.*

For each of the following methods, describe the situation that is not properly handled.

(a)
```
1  /** Inserts the given item at the tail of the SLL */
2    public void addLast(T item) {
3        tail.setNext(new NodeSL<T>(item,null));
4        tail = tail.getNext();
5    }
```

What's wrong? Doesn't update the head when adding to an empty list.

(b)
```
1  /** Removes the given item from the head of the list
2    * @return v item removed */
3    public T removeFirst() {
4      T result = null;
5      if (head == null) {
6        throw new MissingElementException();
7      } else {
8        result = head.getData();
9        head = head.getNext();
10     }
11     return result;
12   }
```

What's wrong? Doesn't set the tail to `null` when removing from a single-element list.

(c)
```
1  /** Inserts the given item in the SLL after the here node */
2    public void addAfter(NodeSL<T> here, T item) {
3      if (here == null) {
4        // null means put at the head
5        addFirst(v);
6      } else {
7        here.setNext(new NodeSL<T>(v,here.getNext()));
8      }
9    }
```

What's wrong? Should update the tail (or call `addLast(item)`) when called on a single-element list.

(d)
```
1  /** Appends a list by transferring elements */
2    public void appendTransfer(SLL<T> suffix) {
3      tail.setNext(suffix.head);
4      tail = suffix.tail;
5      suffix.head = suffix.tail = null; // transfer
6    }
```

What's wrong? Need to set the head and tail if the original list was empty.

Question 5. Sorting (12 points)

The items below each show some of the steps of a particular sorting algorithm working on an array. The first line shows the state of the array at some point in the middle of the sort. The following lines show how that state changes as the algorithm works. Unless otherwise specificed, each row shows the next consecutive state after a swap operation.

Fill in **the next three rows** for each algorithm.

(a) SelectionSort

1	2	3	5	8	4	6	7
1	2	3	4	8	5	6	7
1	2	3	4	5	8	6	7
1	2	3	4	5	6	8	7
1	2	3	4	5	6	7	8

(b) InsertionSort

2	5	4	8	3	1	7	6
2	4	5	8	3	1	7	6
2	4	5	3	8	1	7	6
2	4	3	5	8	1	7	6
2	3	4	5	8	1	7	6
2	3	4	5	1	8	7	6

(c) MergeSort*

5	2	8	4	3	1	7	6
2	5	8	4	3	1	7	6
2	5	4	8	3	1	7	6
2	4	5	8	3	1	7	6
2	4	5	8	1	3	7	6
2	4	5	8	1	3	6	7
2	4	5	8	1	3	6	7
1	2	3	4	5	6	7	8

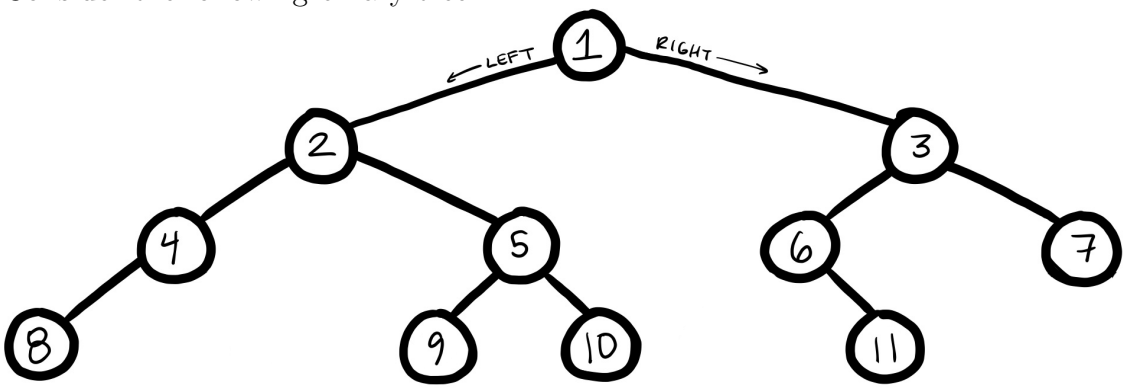
*here we show the result of each sorted merge

(d) HeapSort

5	2	4	6	3	1	7	8
5	6	4	2	3	1	7	8
6	5	4	2	3	1	7	8
6	5	7	2	3	1	4	8
7	5	6	2	3	1	4	8
7	5	6	8	3	1	4	2
7	8	6	5	3	1	4	2

Question 6. Tree Traversal (6 points)

Consider the following binary tree:



(a) In what order will the nodes be visited if we perform a **preorder** traversal?

Solution: 1,2,4,8,5,9,10,3,6,11,7

(b) In what order will the nodes be visited if we perform an **inorder** traversal?

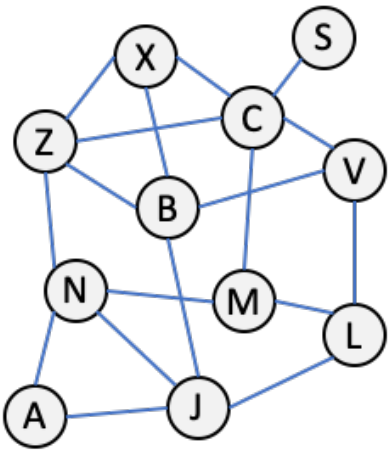
Solution: 8,4,2,9,5,10,1,6,11,3,7

(c) In what order will the nodes be visited if we perform a **postorder** traversal?

Solution: 8,4,9,10,5,2,11,6,7,3,1

Question 7. Graphs (12 points)

Consider the undirected graph shown to the right:



(a) Which node(s) could be the second to be visited (after L itself) when performing a breadth-first traversal beginning at node L?

Solution: J, M, or V

(b) Which node(s) could be the second to be visited (after L itself) when performing a depth-first traversal beginning at node L?

Solution: J, M, or V

(c) Which node(s) could be the third to be visited when performing a breadth-first traversal **beginning at node L**?

Solution: J, M, or V

(d) Which node(s) could be the third to be visited (after L itself) when performing a depth-first traversal **beginning at node L**?

Solution: J, M, V, A, N, B, or C

(e) When performing a breadth-first traversal **beginning at node A**, which node(s) could be the last visited?

Solution: S

(f) Assuming that all edges cost the same amount to traverse, what nodes lie on the least expensive path from X to M?

Solution: `X-->C-->M`

This page intentionally left blank as scratch paper.

This page intentionally left blank as scratch paper.

This page intentionally left blank as scratch paper.