CSci 1933

Spring 2022

Midterm Exam 2

(100 points)

This is a 50 minute closed book exam. *No* outside materials, calculators, phones, computers nor other electronics, can be used for this exam. Partial credit may be given on these questions, so show your effort where possible. Please read and sign the statement below.

I certify that the work on this exam represents only my own efforts and that I have neither obtained help from others nor given help to others on this exam. I have not used any outside references.

Name ____

Signature ____

ID ____________________

Lab **Section** ________

1. A Few Short Answers. (20 points)

a. (5 points) Consider your ArrayList and LinkedList implementations from Project 3. While both implementations of `remove(T item)` have *O*(n) complexity, in some cases, it appears that the LinkedList implementation is faster than the ArrayList implementation. In what situations would you expect to observe this difference in performance. *Briefly* explain.

Array:	LL:
reassign index	Loop to index
Shift over	reassign nodes

So, in a large list, LL will Perform better on removing item near beginning. Array will Perform better on removing items near end.

b. (5 points) A student in an introductory CSci course, is planning to implement a *stack* using an *array*, and is wondering whether she will need to have two variables: one that holds the index of the top of the stack and one that holds the index of the bottom of the stack. What should you tell her? Briefly explain.

No, Can only add to top of stack

So variable Pointing to bottom isn't necessary

c. (5 points) The required operations for stacks are push() and pop(). Why is it generally *not* necessary to have an additional method that allows a user to "peek" or look at the item that is at the top of the stack with*out* "popping" it?

Because without Peek you can still Pop() the item at the top and Push() it back if it isn't needed.

d. (5 points) Generic typed data (recall the <T> notation) is typically better than using Object data. Why?

Because classes and methods created with Generic Data are applicable and Usable for all Data types instead of Being Limited to Just 1

2. Give the Output. (20 points)

a. (12 points) *Next* to each `println()` in `main()` below, give the output produced when `main()` is run. Note: There are no errors in the code.

```
public interface Person {
    String getName();
    int getID();
}

public class Student implements Person {
    public Student() {}
    public Student(String name, int id) {
        this.name = name;
        studentID = id; }

    public String getName() { return "Student Name is: " + name; }

    public int getID() {
        return studentID; }

    public String toString() { return "Student " + name + " " + studentID; }  S

    protected String name;
    private int studentID;
}

public class GradStudent extends Student {
    public GradStudent(String name, int id, String degree) {
        this.name = name;
        this.id = id;
        this.degree = degree; }

    public int getID() { return id + 1000; }

    public String getDegree() { return degree; }

    public String toString() { return "Grad " + name + " " + degree; }  GS

    private int id;
    private String degree;

    public static void main(String[] args) {
        Student s = new Student("MyTA", 1);
        Student gs = new GradStudent("Smart Person", 2, "PhD");

        System.out.println("line 1: " + s.toString());  Line1: Student MyTA 1

        System.out.println("line 2: " + gs.toString());  Line2: Grad SmtPrsn PhD

        System.out.println("line 3: " + s.getID());  Line 3: 1

        System.out.println("line 4: " + gs.getID());  Line 4: 1002

        System.out.println("line 5: " + s.getName());  Ln5: Student Name is: MyTA

        System.out.println("line 6: " + gs.getName());  Ln6: Student Name is: SmtPrsn
    }
}
```

b. (8 points) Following are *four possible* additional declarations for `main()` in the code above. Identify *each* as *legal* or *illegal*.

`Person p = new Student();` illegal nota

`Object o = new Student();` illegal nota

`Person p = new Person();` Legal isa

`GradStudent grad = new Student();` Legal isa

Person
↓
Studnt
↓
gStudnt

3. Stacks and Lists. (20 points)

A linked node implementation of a stack is given below. The implementation includes `push()` and `pop()`. Write an additional public method called `getCount()` to be included in the `Stack1Gen` class that returns the number of elements in the stack. `getCount()` should *not* modify the stack contents.

```
  public class Stack1Gen <T> implements StackGen <T> {

    public Stack1Gen () {}

    public void push(T o) {
        start = new NGen <T> (o, start);
    }

    public T pop() {
        if (start == null)
          throw new RuntimeException("Tried to pop an empty stack");
        else {
          T data = start.getData();
          start = start.getNext();
          return data;
        }
    }

    private NGen <T> start = null;

}  // Stack1Gen class
```

```
    public int getCount() {  // returns the number of elements in this stack
      int counter = 0;
     while(start.getNext() != null) {
        start = start.getNext();
        counter++;

     }
  }
```

4. Lists. (20 points)

Fill in the table below with the values that will print when `List` (which uses `Node`) is run.

```
public class Node {
    public Node() {}
    public Node(int n, Node ptr) {
        data = n;
        next = ptr;
    }
    private int data;
    private Node next;

    public int getData() { return data; }
    public void setData(int n) { data = n; }
    public Node getNext() { return next; }
    public void setNext(Node ptr) { next = ptr; }
}

public class List {
    public static void addToStart(Node ls, int item) {
        ls = new Node(item, ls);
    }

    public static void addToEnd(Node ls, int item) {
        if (ls == null) {
          return;
        }
        else {
          while (ls.getNext() != null) {
              ls = ls.getNext();
          }
          ls.setNext(new Node(item, null));
        }
    }

    public static void printList(Node ls) {
        while (ls != null) {
            System.out.println(ls.getData());
            ls = ls.getNext();
        }
        System.out.println();
    }

    public static void main(String[] args) {
        Node list1 = new Node(100, null);
        list1 = new Node(50, list1);

        System.out.println("List1 to start:");
        printList(list1);

        addToEnd(list1, 200);
        System.out.println("List1 after addToEnd:");
        printList(list1);

        addToStart(list1, 25);
        System.out.println("List1 after addToStart:");
        printList(list1);
    }
}
```

List1 to start:	50 100
List1 after addToEnd:	50 100 200
List1 after addToStart:	2 50 100 200

5. Write a Method Using a 2-D Array. (20 points)

Complete the method `countHazards(char[][] a, char hazard)` below to return the number of times the character `hazard` is found in the *2-dimensional* array, `a`. For example, if the array, `a`, is:

```
a x c d e
1 x 3 4 x
a 1 b 2 3
```

and `hazard` is: 'x', `countHazards(a, 'x')` will return 3 because character 'x' is found in 3 places within array, `a`.

```
    public int countHazards(char[][] a, char hazard) {
  int counter = 0;
  for(int i = 0; i < a.Length; i++) {
      for(int J=0; J < a[i].Length; J++) {
          if(a[i][J] == hazard) {
              counter++;
          }
      }
  }
  return counter;
}
```

for office use only:

Q	P	Score
1	20	
2	20	
3	20	
4	20	
5	20	
TOTAL	**100**	