MIDTERM EXAM INSTRUCTIONS

1. Midterm Exam: **25 points w/ 0 E.C. points**
2. Due Date & Time: **03-22-2021 at 09:00 AM**

WHAT TO SUBMIT

1. Take-home Exam Report, 1 PDF

HOW TO SUBMIT AND THE RULES TO FOLLOW

- Submit via iLearn, the Assignment and Exam Submission section
- Please follow the exam instructions
- Please follow the Course Policy on Student Conduct and Academic Honesty

PERFORMANCE TRACKER		
ASMT	GRADE	YOUR GRADE
ZOOM	05	
01	20	
02-PREPARATION	25	
02	75	
03	75	
MIDTERM EXAM 01	25	
TOTAL	225	

A: 90-100% **B**: 80-89% **C**: 70-79% **D**: 60-69% **F**: 0-60%
The course grader provides feedback to your assignments on iLearn.

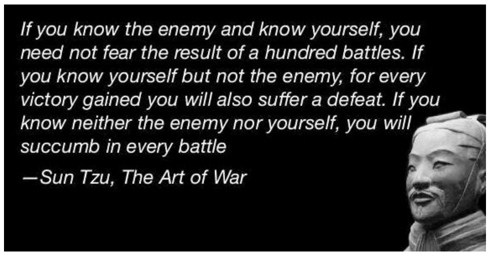
ABOUT

The goal of this take-home exam is for us to **know what we do not know**.

We are taking this exam as seriously as how we take an actual exam in class. Please,

1. Follow all the rules and the guidelines listed at the top of page 1 and page 2
2. Read each question carefully before answering

We will go through the answers to all the exam questions together in class.



If you know the enemy and know yourself, you need not fear the result of a hundred battles. If you know yourself but not the enemy, for every victory gained you will also suffer a defeat. If you know neither the enemy nor yourself, you will succumb in every battle

—Sun Tzu, The Art of War

STEP A – Take the Exam, **10 points**

1. Allocate 75 quiet minutes to take the exam on page 2 to the last page.
2. Record the date and time when you start.
3. Stop right at minute 75. Record the date and time when you stop the exam.

STEP B – Correct Your Answers, **10 points**

1. Review the related course materials and write code when necessary to find a correct answer for each question. We should be able to find all the answers using the packages, the in-class discussions, our assignments, and the other course materials.
2. At the end of each of your oringal answers, type in *italic* text and:
   - Give your orginal answer a score.
   - List all the mistakes then explain why, you think, you made the mistakes. Add the correct answer you found. Document how you found the correct answer. Document where you found the materials which support the answer.
   - If you did not make any mistakes, please document how you verified that your answer was accurate. Document how you found the correct answer. Document where you found the materials which support the answer. Outline how you could have done better.
   - Record your total score out of 100 points for all the orginal answers.

STEP C – Reflect and Retake the Exam, **5 points**

1. **Problem Solving**: Reflect if you managed the exam time efficiently and if you strategized your test-taking successfully.
2. Repeat steps A to C again if necessary. Please keep appending new contents as directed in Step B.2.
3. Think if the same topics will be tested again in our final exam, what questions we may get.

It is a good idea to do every step of this assignment thoroughly. We are creating a set of materials which we will use to review for the final exam. And this is also the best way to prepare ourselves to succeed in the second half of the semester. Thank you.

Kullathon “Mos” Sitthisarnwattanachai
921425216

Step A

- Started: 3/21 21:23 PT
- Finished: 3/21 22:38 PT
- All non-zero points questions attempted during the time above except B7 (ran out of time).

Step B

- See below each question for the self-evaluated scores, reasoning, and reflection.
- Total score: 76.7/100. See end of document for Points Breakdown.

Step C

- Upon reflection, I felt that my time management have felt short of what is expected.
- My general strategy was to skim through all of the questions first to identify which were easier to answer, and to complete them first. The next in priority were ones that I felt confident answering in entirety with large number of credits. I decided to skip questions that did not offer credits in order to complete the ones that did first.
- Even with many non-credit questions skipped, I still left one question completely unanswered. A lot of the time spent were on unimportant details, such as the diagrams. Another aspect that took up a lot of time were manual tracing. It took significant amount of effort to ensure that the output that were expected from the given question would match the given answer.
- I was unsure whether an IDE was allowed for the test, so I completed it without one. If one were permitted, it would have allowed me to complete the questions at a much faster rate. Especially, the tracing ones.

1. Section, Date and Time:
   CSC 220.02+~~03~~, Due **03-22-2021 at 09:00 ~~PM~~ AM**
2. Midterm Exam (2 exams, 0 dropped): 100 points

Full Name in CAPITAL LETTERS	SFSU ID
KULLATHON SITTHISARNWATTANACHAI	921425216

3. To prepare for this exam, please review all the related materials including WEEK 01-08 packages, slides, mock-up exam(s), reading assignments, in-class practices, sample programs posted in the File Manager, and assignments.
4. You do not need to print this exam. No paper. No handwriting. No scanning. Please type up all your answers in the answer space available in the exam. The provided exam will be in Microsoft Word format. Please submit a single PDF via iLearn.
5. All the rules of an actual exam apply to this exam such as: closed books, closed notes, and no communication with anyone except the course instructor. The course instructor will be available on Zoom or email during the exam time: zoom.ducta.net
6. Please ask all your questions, if any, during the review sessions. Thank you.

HONOR CODE:

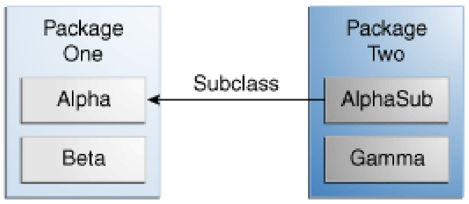
- Please follow the CS Department's policies: https://cs.sfsu.edu/student-policies

- Please follow the course's policies: http://csc340.ducta.net/00-README-StudentConduct_AcademicHonesty.pdf

PART A – 40 Points

A.1 - 10 Points
Please complete the "where the members of the Alpha class are visible" table by filling in Y or N.

Modifier	Alpha	Beta	AlphaSub	Gamma
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N



If a member of the class **Beta** is **protected**, is this member visible to **Gamma**? If yes, please explain why. If not, please explain what we should do make it possible.

- No, if a member in Beta is protected, then that member will only be accessible within Beta, its subclassess, or classes within Package One. According to the diagram provided, Gamma is neither a subclass of Beta nor does it share the share the package Beta is in. As such, a protected member of Beta cannot be accessed by Gamma.
- To make such member accessible, one could swap out the protected modifier with public.

Step B	
Original score:	*10/10*
Reasoning and corrections:	*No deductions.* • *Tested the case in an IDE and works as expected.*
Room for improvement:	• *Spending less time trying to fill the table with text boxes… consumed time that could've been spent answering other questions*

A.2 - 10 Points
- What are the 2 ways to use the keyword "**super**" in a class? - Give code examples to demonstrate your answers.

- The super keyword can be used to invoke the constructor of its parent.

  For example, let the class SFSUMember represent a member of SF State. Where it has the fields, name and age of type String and int, respectively.

  ```
  public class SFMember {
    String name;
    int age;

    public SFMember(String name, int age) {
      this.name = name;
      this.age = age;
    }
  }
  ```

  Then, let the class SFStudent represent a student of SFState, subclassing SFMember. Where it has a field grade of type char.
  SFStudent could use the sole constructor of SFMember to intialize basic information without repeating much of the same field using the super keyword like so:

  ```
  public class SFStdent {
    char grade;

    public SFStudent(String name, int age, char grade) {
      super(name, age);
      this.grade = grade;
    }
  }
  ```
- The super keyword can also be used to invoke the parents' method.

  Continuing with our example, consider SFMember to have a void method that is defined as follows:

  ```
  public void sayCheers() {
    System.out.println("Greetings, gators!");
  }
  ```

  Similarly, consider SFStudent to have a method that overrides the above method defined in SFMember:

  ```
  public void sayCheers() {
    System.out.println("GO GATORS!");
  }
  ```

  In the SFMember class, if sayCheers() were invoked, it would automatically invoke the overridden method, printing "GO GATORS!" Suppose, you needed to invoke the method in the parent, you could instead invoke it by using the super keyword: super.sayCheers();

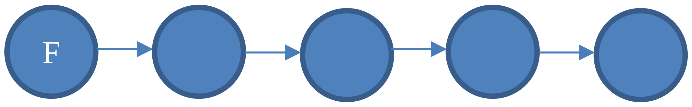
Step B

Original score:	*10/10*
Reasoning and corrections:	*No deductions.* • *These are valid use cases of the super keyword.* • *Tested the code in the IDE and the usages in the provided examples works as expected, ignoring the quotation marks.*
Room for improvement:	• *Formatting the code answers with a monospace font and correct quotations marks.*

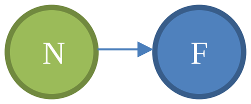
A.3 - 10 Points
When using a linked list for stack implementation, we use the first Node as the top entry of stack. Explain, in this linked list implementation, how we **add** and how we **remove** an entry from a stack. *Use linked nodes diagrams to save your time.*

add

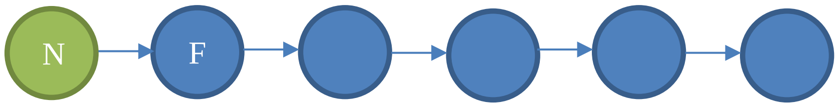
- Before adding, where each circle denotes a node, and "F" denotes the first node. Each nodes points to the next, with the last pointing to "null."



- When adding an entry, we first create a new node with the data. Then, we point that node to the first node ("F").

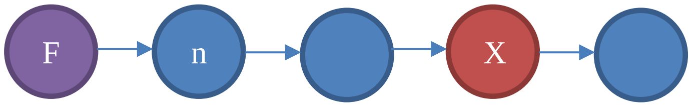


- Finally, we make the new node (green) the first node of the chain. We do this by changing the refernce to the first node with the new node we created. Note that N is now our new first node, and our old first node is now second.

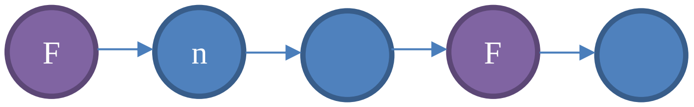


remove

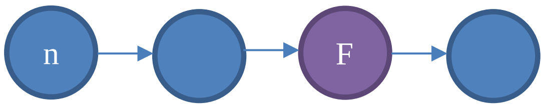
- Before removing, where each circle denotes a node, and "F" denotes the first node. Each nodes points to the next, with the last pointing to "null." Let "X" denote the node to remove, and "n" denote the node following the first node.



- Assuming the node exist, like the diagram above shows, we "remove" the targeted node by overwriting the data on it with the data of our first node. Note that at this point, we have two nodes with duplicate data.



- Then, to remove our duplicate, we replace the reference to the first node by pointing it to the following it. In so doing, the first node in the chain is now "n."



Explain one major difference between the behaviors of the LinkedBag and the Stack which we implemented.

- One major difference between the two is how we "clear" the data type.
  To clear a LinkedBag, we would replace the reference of the first node to its next node (basically, repeating the last step in remove as outlined above) until it has exhausted the chain, i.e., the first (and only) node being "null."
  To clear the Stack, we simply overwrite the top-most entry as "null." Since we are only able to access Stacks by its top-most entry, declaring the top-most as null prevents further entries in the chain to be referenced.
  Argubly, with n items in LinkedBag and Stack, it would be much "easier" to clear the items in a Stack, since it would only

require the top-most item to be overwritten, whereas LinkedBag requires each entry to be overwritten with the next until the last one point to null.

Step B

Original score: *6.7/10*

Reasoning and corrections:

Deducted 3.3 points for "remove" method explanation.

- *Answered the wrong question. The answer provided the implmentation of LinkedBag.*
- *Still unsure whether the question asks for the implementation of Stack or LinkedBag.*
- *Upon revision, it appears that the question is asking about the implementation of Stack. However, LinkedBag contains the "add" and "remove" method, while the Stack counterpart only has "push" and "pop" in contrast.*
- *The correction below will assume "remove" refers to the "pop" method.*

No deductions for "add" method.
Reference: Packages 04 and 06.

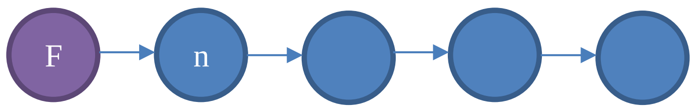
- *Regardless of wheter the question refers to LinkedBag or Stack, the explanation applies to both data types.*

No deductions for final part.
Reference: Pacakges 04 and 06.

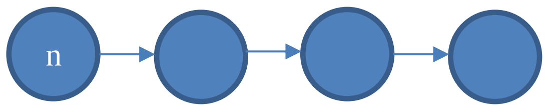
- *This was discussed extensively in the lecture. Also shown in Packages 04 and 06.*

Correction for "remove" method:
Reference: Package 06.

- *Before removing, where each circle denotes a node, and "F" denotes the first node. Each nodes points to the next, with the last pointing to "null." Let "n" denote the node following the first node.*



- *In a Stack, only the top-most element can be accessed. The first node represents the last item added to the stack. As such, that node will be removed.*
- *Assuming the first node exists (i.e., that the Stack is not empty), we replace the reference to the top-most node with the node following it, denoted "n" in our diagram.*



- *Prior to the previous step, the removed node "F" may be stored to be returned if the removal was sucessful, similar to "pop."*

Room for improvement:

- *Actually answer the correct question.*
- *Spent less time on making the diagrams. Time spent on making the diagram prevented other questions from being answered.*
- *Upon revision of the package, the diagrams used for displaying stacks and linked nodes were different. Regardless, the diagram and explanation conveys the same concept. Not that it mattered much, since it answered the wrong question…*

A.4 - 0 Points – A Practice Problem
What are stored in each activation record and Why? Which method is pushed in the Program Stack first? Which method is popped out of the Program Stack last?

Step B	
Original score:	*0/0*
Reasoning and corrections:	*Did not attempt to save time since the question did not offer credit.* • *This has not yet been discussed in class.* *Correction:* *Reference: Package 07.* • *From Pacakage 07: "The stack of activation records: Each call to a method generates an activation record that captures the state of the method's execution and that is placed into the program stack. However, these methods need not be distinct. That is, the program stack enables a run-time environment to execute recursive methods. Each invocation of any method produces an activation record that is pushed onto the program stack. The activation record of a recursive method is not special in any way."*
Room for improvement:	• *Attempt the question.*

A.5 - 10 Points - Part A of Assignment 03:

What is the ouput (what are in the Bag) when the 2 below lines are executed? **Please show the steps.**
String[] items = {"Z", "Y", "Y", "X", "S", "C", "A", "E", "M"};
testAdd(aBag, items);

- The method will iterate through each item in the array and invoking the add method of the bag with each element in the array.
    - In doing so, the bag will add the provided elements in the steps outlined in question (A)(3) under "add." In short, order of the items will be reversed since the item being added will take on the new "first" node position and pushing the rest of the chain back.
- After all the elements are added, the method invokes the display bag method, which will go through each item in the bag. As noted, the order will be reversed due to the way in which the items are added as outlined in (A)(3), so the output will begin with the element "M" and end with "Z."

What is the output (what are in the Bag) when the 3 below lines are executed? **Please show the steps.**
String[] testString = { "X", "Y", "Z" };
aBag.removeAllOccurences(testString);
displayBag(aBag);

NOTE THAT THE ORDER OF TRAVERSAL WILL BE IN THE ORDER OF THE OUTPUT FROM THE LAST STEP, NAMELY:
M, E, A, C, S, X, Y, Y, Z

- To remove the provided items from the bag, the removeAllOccurences method will iterate through the nodes in the bag, where:
    - First, it will access the item in the bag and check if the value of the node matches those in the array.
    - If it does, then it will remove the item from the bag in the steps outlined in (A)(3) under "remove." In short, this is done by replacing the item to remove with the first node and replacing the reference to the first node with the one following it.

- First remove: finds X
    - Replace X with M
    - Replace M with E
    - RESULT: E, A, C, S, M, Y, Y, Z

- Second remove: finds Y
  - Replace Y with E
  - Replace E with A
  - RESULT: A, C, S, M, E, Y, Z
- Third remove: finds another Y
  - Replace Y with A
  - Replace A with C
  - RESULT: C, S, M, E, A, Z
- Forth remove: finds Z
  - Replace Z with C
  - Replace C with S
  - RESULT: S, M, E, A, C
- **FINAL RESULT: S, M, E, A, C**

Step B	
Original score:	*10/10*
Reasoning and corrections:	*No deductions.* • *Tested the input using the provided input and a modified version of the driver class from Part (A) of ASMT 03. The expected results matches the outcome.*
Room for improvement:	• *Was not sure if using an IDE was allowed for this question. Did not use one to answer this question. If a usage of an IDE was allowed, then a lot of time would have been saved since tracing this question manually took a significant amount of time.*

PART B – 60 Points

B.1 - 5 Points
Which of the statement(s) are erroneous and why?

```
MidtermExam midtermExam = new Exam();                // A
SFSUStaff person = new Person();                    // B
StackInterface<String> s = new ArrayStack<>();       // C
```

- A and B are erroneous. Assuming:
  - MidtermExam and SFSUStaff extends Exam and Person, respectively,
  - The invoked no-arg constuctors are defined in Exam and person, repsectively.

  You cannot declare the class to be a child (SFSUStaff and StackInterface) of the class you are instantiating (Exam and Person). Although, the reverse is valid: you **can** declare the class to be the **parent** while instantiating its **child** class.

Step B	
Original score:	*5/5*
Reasoning and corrections:	*No deductions.* • *Using the assumptions stated in the original answer, the classes were defined respectively and the statements were tested. The outcomes matches the given explanation.*
Room for improvement:	• *I think the final part of the original answer could be worded better. Left as-is due to time constraints.*

B.2 - 5 Points
What is the output if any?

```
abstract class Person {

    private final String name;

    protected Person(String name) {
        this.name = name;
    }

    public String getName() {
        return this.name;
    }

    public void setName(String name) {
        this.name = name;
    }
}

class Student extends Person {

    public Student(String name) {
        super(name);
    }

    public static void main(String[] args) {
        Student stu = new Student("Mickey");
        System.out.println(stu.getName());
        stu.setName("Super Mouse");
        System.out.println(stu.getName());
    }
}
```

*Attention to the keyword “**final**”*

- Shouldn’t compile in the first place. If the field name is final, then it cannot be updated. The class defines a setter for the name regardless, which is not valid.

Step B

Original score: *1/5*

Reasoning and corrections: *Deducted 4 points for incorrect output.*
- *Implied that the code would not produce an output.*
- *The stated assumption was incorrect. The code does in fact compile.*
- *The logic of the assumption was correct, as the stated issue resulted in an expection being raised.*

Correction:
- *When tested in an IDE, the code prints “Mickey” with a newline to stdout, followed by a stack trace of the exception caused by calling setName().*

Room for improvement:
- *Correcting the answer. I did not realize that the code would run with that error.*

B.3 - 10 Points
Anything wrong with the code? If yes, how to fix?

```
class CSC220 {
    private CSC220(){}
    private CSC220(int x){}
    private CSC220(int x, int y){}
}
```

- Syntactically, no. It should compile. Convention-wise, poor variable naming and empty braces on the same line.
- Most convention requires meaningful names for the variables and recommends adding a space between the braces, even if they're empty.

And is it possible to create a sub class to this class? Why?

- No. The class provides no accessible constructors for its subclasses. All of the defined constructors are private.

Step B

Original score: *10/10*

Reasoning and corrections: *No deductions.*

- *Tested the code in the IDE and the cases matches the expected outcomes.*

Room for improvement: *n/a*

B.4 - 0 Points – A Practice Problem
What is the output if any?

```
    public static void main(String[] args) {
        int x = 12;
        System.out.println(sumOf(x) - 42);
    }

    public static int sumOf(int n) {
        int sum = 15;
        if (n == 0) {
            System.out.println("Base case: n is " + n);
            sum += 5 + n % 2;
        } else {
            sum = sumOf(n - 3) + n;
        }
        System.out.println(sum);
        return sum;
    }
```

Step B

Original score: *0/0*

Reasoning and corrections: *Did not attempt to save time since the question did not offer credit.*

- *This has not yet been discussed in class.*

Correction:
Running this code from IDE produces the following:

- *Base case: n is 0*
  20
  23
  29
  38
  50
  8

Room for improvement:	• *Attempt the question.*

B.5 - 10 Points
This program outputs 10 lines. What are they?

```
Stack<String> resume = new Stack<>();
resume.push("JavaScript");
System.out.println("Is empty: \t" + resume.isEmpty());
resume.push("Scala");
resume.push("C++");
resume.push("Dart");
resume.push("Go");

resume.pop();
System.out.println("Stack : \t" + resume);
resume.push("Python");
System.out.println("search() : \t" + resume.search("Scala"));
System.out.println("pop() : \t" + resume.pop());
System.out.println("pop() : \t" + resume.pop());
System.out.println("search() : \t" + resume.search("Dart"));
System.out.println("After pop() : \t" + resume);
System.out.println("pop() : \t" + resume.pop());
System.out.println("Is empty : \t" + resume.isEmpty());
System.out.println("Stack:  \t" + resume);
```

1. Is empty:	false

2. Stack:	[JavaScript, Scala, C++, Dart]

3. search():	4

4. pop():	Python

5. pop():	Dart

6. search():	-1

7. After pop():	[JavaScript, Scala, C++]

8. pop():	C++

9. Is empty:	false

10. Stack: [JavaScript, Scala]

Step B

Original score:	*10/10*
Reasoning and corrections:	*No deductions.* • *Tested the code in the IDE. The output matches the expected outcome, ignoring the inconsistent tab sizes between the console and this document.*
Room for improvement:	• *Was not sure if using an IDE was allowed for this question. Did not use one to answer this question. If a usage of an IDE was allowed, then a lot of time would have been saved since tracing this question manually took a significant amount of time.*

B.6 - 15 Points

Implement *findTheThird* method in linked list that searches the bag for a given *entry.* If found,

- removes the first occurrence
- leave the second occurrence intact
- then replace third occurrence with the string "Found3rd"
- remove the rest of the occurrences

Return *false* if no replacement happened. Otherwise, *true*.

`public boolean findTheThird (T entry)`

Note: You may assume that *firstNode* is a private data in list which references to first node.

-
  ```
  public boolean findTheThird(T entry) {
    int occurrence = 0;
    boolean isReplaced = false;
    Node thisNode = firstNode;

    // traverse the nodes
    while(thisNode != null) {

      if (thisNode.data.equals(entry)) {
        occurrence++;

        // special cases for 2nd and 3rd occurrences
        if (occurrence == 2) {
          continue;
        } else if (occurrence == 3) {
          thisNode.data = "Found3rd";
          isReplaced = true;
          continue;
        }

       // otherwise remove them
       thisNode.data = firstNode.data;
       firstNode = firstNode.next;
       numberOfEntries;
      }
      thisNode = thisNode.next;
    }

    return isReplaced;
  }
  ```

Step B	
Original score:	*14/15*
Reasoning and corrections:	*Deducted 1 point for not downcasting "Found3rd" during assignment.*

	• *Tested the code in the IDE. The output matches the expected outcome, ignoring smart quotes (“”) and the mentioned correction.* *Correction:* *Tested in IDE.* • *~~thisNode.data = “Found3rd”;~~ -> thisNode.data = (T) “Found3rd”;*
Room for improvement:	• *Was not sure if using an IDE was allowed for this question. Did not use one to answer this question. If a usage of an IDE was allowed, then a lot of time would have been saved since writing code in Word is terrible.*

B.7 - 15 Points
Assume that you have the following Bag object, myBag, with n String data:

BagInterface <String> myBag = new ArrayBag< >();

Write Java statements that create a newBag object which
contains non-duplicate data in myBag and marks the duplicate data.

Example:
if myBag contains data:
"A", "A", "A", "B", "B", "C", "D", " "
newBag object should contain:
“A”, “DUP.1.A”, “DUP.2.A”, “B”, “DUP.3.B”, “C”, “D”, “ “

Hint: You can use the Bag’s methods:
int getCurrentSize (); boolean isFull (); boolean isEmpty (); boolean add (T newEntry); T remove (); boolean remove (T anEntry); void clear (); int getFrequencyOf (T anEntry);
boolean contains (T anEntry); T [] toArray ();

Step B

Original score: *0/15*

Reasoning and corrections: *No points. Did not attempt question; ran out of time.*

Correction:
It’s not pretty but it works… Output tested in the IDE with the given case.

```
Object[] myBagArray = myBag.toArray();
BagInterface<String> newBag = new ArrayBag<>();
int nosOfDuplicates = 0;
for (Object item : myBagArray) {
    String entry = (String) item;
    if (myBag.getFrequencyOf(entry) > 1) {
        if (newBag.contains(entry)) {
            newBag.add(String.format("DUP.%d.%s", nosOfDuplicates, item));
        } else {
            newBag.add(entry);
        }
        nosOfDuplicates++;
    } else {
        newBag.add(entry);
    }
}
```

Room for improvement: • *Better time management on other questions may have allowed me to attempt the question.*

Points Breakdown

Question	Scored	Total
A.1	10.0	10.0
A.2	10.0	10.0
A.3	6.7	10.0
A.4	0.0	0.0
A.5	10.0	10.0
B.1	5.0	5.0
B.2	1.0	5.0
B.3	10.0	10.0
B.4	0.0	0.0
B.5	10.0	10.0
B.6	14.0	15.0
B.7	0.0	15.0
Total	**76.70**	**100.00**