Carleton UNIVERSITY

FINAL
EXAMINATION
FALL 2021

DURATION: _3_ HOURS **No. Of Students: 144**

Department Name & Course Number: **Systems and Computer Engineering SYSC 3110**

Course Instructor (s): Babak ESFANDIARI

AUTHORIZED MEMORADA : Open book

Students MUST count the number of pages in this examination question paper before beginning to write, and report any discrepancy to a proctor. This question paper has 4 pages + cover page = _5 pages in all.

This examination question paper may not be taken from the examination room.

In addition to this question paper, students require: an examination booklet	~~yes~~	no
Scantron Sheet	~~yes~~	no

Name: Kareem El Assad

Student Number: 101107739

Instructions:

1) Write your name and student number in the spaces provided above.
2) Attempt all questions and write your solutions below each question. You will submit the Word document on Brightspace (.doc, .docx and .pdf formats, and only those formats, will be accepted). **Read each question thoroughly before attempting to write answers**.
3) The Java API reference can be found online at https://docs.oracle.com/javase/8/docs/api/index.html
4) Please do not ask the instructor to explain a question. If you think that something is unclear or ambiguous, make a reasonable assumption (one that does not contradict the question), write it at the start of your answer, and solve the problem. If you think that you have found an error, please bring this to our attention.
5) **Marks will be deducted for "smelly" code.** It is not sufficient for code to be working correctly, it needs to follow good design and coding principles, as taught in class.

For instructor's use:

Question 1: _________	/10	Question 2: _________	/5
Question 3: _________	/5		
Total: _________	**/ 20**		

Question 1 – Design [10 marks]

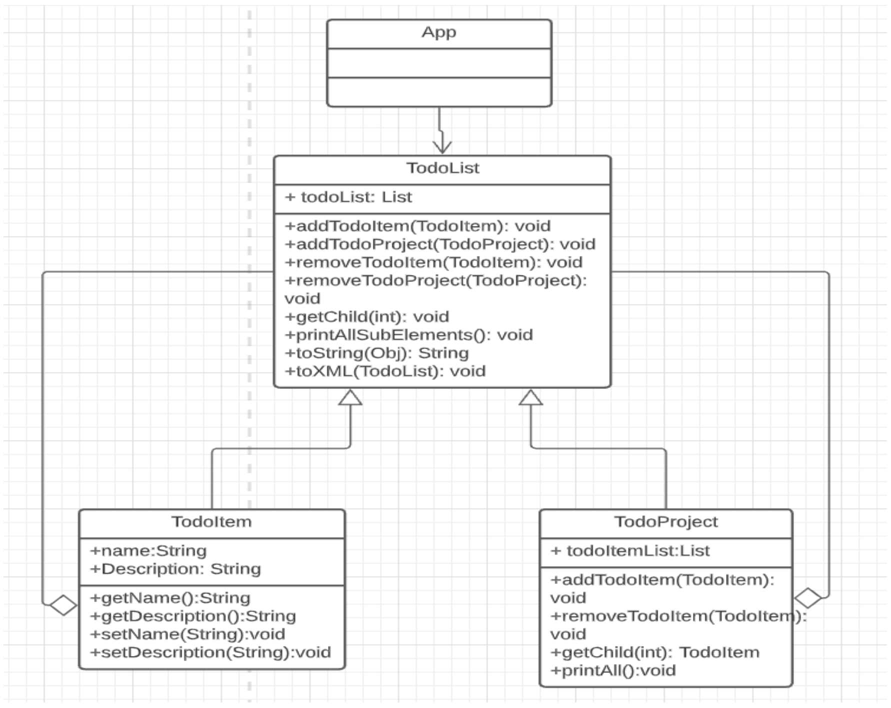
We want to come up with an application to manage and organize a to-do list. A to-do list is made up of to-do items. To-do items are either atomic, i.e. they consist of a single task with a name and description, or they are to-do projects (which do not have a name or description) containing to-do items. This means within a project one might find further projects (and some atomic to-do items), which might in turn contain more projects (and some atomic to-do items), etc.

The user should be able to create a to-do list, and populate it with to-do items. When creating an atomic to-do item, the user should specify its name and description. If the to-do item is a project, the user should be able to add further to-do items to it.

Assumptions:
The Todo-list works as follows:
Todo items have name, description.
Todo projects have Todo items in a list.
Can instantiate projects to be empty or with Todo items
The user can create Todo items and will need to specify the name and description.
Generally, there is an overarching TodoList that stores todoItems and todoProjects. todoItems have a name and description. TodoProjects contain either todoItems or are null. If they are null, the user can add todoItems to them.

Will be using the composite design pattern as the nesting of the Todo items resembles a tree.

1) **Come up with a UML class diagram representing your design. The diagram must be complete with all the variables and methods, with their type and visibility. In particular, it must include the methods and constructors that you will be asked to implement in the next few questions.**



2) **Implement all the necessary constructors.**

TodoItem.java Constructor:

```
public TodoItem(String name, String description) {
    this.name = name;
    this.description = description;
}
```

TodoProject.java Constructors:
This class is overloaded (contains more than 1 constructor). This is because java does not support optional parameters and I needed a way to instantiate an empty project.

```
public TodoProject(TodoItem item) {
    todoItemList.add(item);
}
public TodoProject() {
    //if takes nothing, contains nothing.
    this.todoItemList = null;
}
```

3) **Implement the method that adds a new to-do item to a given project.**

This is done inside TodoList.java:

```
public void addTodoItem(TodoItem todoItem){
    todoList.add(todoItem);
}
public void addTodoProject(TodoProject todoProject){
    todoList.add(todoProject);
}
```

I added two separate methods as I was treating the items as leafs and projects as composites. This could also be generalized to add any object but I wanted the functionality to be clear in the code and in the UML.

4) The user should also be able to obtain an XML representation of the to-do list. The `<to-do>` element is used for both atomic to-dos as well as the project. Additionally, the atomic to-do will contain a sub-element for the `<name>`, and one for the `<description>`. As an example, the following XML output represents a project containing another project containing an atomic task named "milestone 1", and with the description being: "text-based version":

```
<todo>
        <todo>
                <todo>
                        <name>Milestone 1</name>
                        <description>text-based version</description>
                </todo>
        </todo>
</todo>
```

Implement all the methods you need so you can obtain the XML representation (as a String) of any given to-do list. You can ignore the indentation.

```
public static void toXML(List todoList) {
    try {
        FileWriter fileWriter = new FileWriter("test.xml");
        BufferedWriter bufferedWriter = new BufferedWriter(fileWriter);
        bufferedWriter.write("<?xml version=\"1.0\" encoding=\"UTF-8\"?>");
        bufferedWriter.newLine();
        bufferedWriter.write("<todo>");
        bufferedWriter.newLine();

        for (int i = 0; i < todoList.size(); i++) {
            // No time to implement so pseudocode
            //if item at index i is null, print <todo>
            //if item at index i is not null, and contains a list, it is a
todoProject. Then iterate children and output like below

            TodoItem item = (TodoItem) todoList.get(i);
            bufferedWriter.write("\t");
            bufferedWriter.write("<todo>");
            bufferedWriter.newLine();
            bufferedWriter.write("\t\t");
            bufferedWriter.write("<name>" + item.getName().replaceAll("\\s+",
"") + "</name>");
            bufferedWriter.newLine();
            bufferedWriter.write("\t\t");
            bufferedWriter.write("<description>" +
item.getDescription().replaceAll("\\s+", "") + "</description>");
            bufferedWriter.newLine();
            bufferedWriter.write("\t");
            bufferedWriter.write("</todo>");
            bufferedWriter.newLine();
        }
        bufferedWriter.write("</todo>");
        bufferedWriter.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

5) In a main() method, create a to-do list that would output the XML example of the previous question.

Note: Added entire testing class I was using as I ran out of time

```
public static void main(String[] args) {
    TodoList todoList = new TodoList();

    System.out.println("created list");
    todoList.printAll();

    TodoItem sampleItem1 = new TodoItem("test1", "test1");
    TodoItem sampleItem2 = new TodoItem("test2", "test2");
    TodoProject todoProj1 = new TodoProject(sampleItem1);
    todoProj1.addTodoItem(sampleItem2);

    todoList.addTodoProject(todoProj1);

    System.out.println("added proj1 with 2 sub items");
    todoList.printAll();



    TodoItem item1 = new TodoItem("Kareem1", "desc1");
    TodoItem item2 = new TodoItem("Kareem2", "desc2");

    todoList.addTodoItem(item1);
    todoList.addTodoItem(item2);

    System.out.println("added item 1 and item 2 to todolist");
    todoList.printAll();

    System.out.println("removed item 1");
    todoList.removeTodoItem(item1);

    todoList.printAll();

    TodoList.toXML(todoList);
}
```

Question 2 – Refactoring to a Pattern [5 marks]

The class Bill below provides a calculation for a billable amount that only works for phone bills.

```
public class Bill {

    private int minutes;
    private double rate;
    private String type;
    private static double GST = 0.9;

    public Bill(int m, double r, String t) {
        minutes = m;
        rate = r;
        type = t;
    }

    public double getBillableAmount() {
        if (type.equals("phone")) {
            double base = 20 + minutes * rate;
            double tax = base * GST * 0.2;
            return base + tax;
        }
        else return 0;
    }
}
```

We want to introduce a new type of bill for internet bills, where the formula is as follows:

base = 30 + minutes * rate;

tax = base * GST;

total = base + tax;

We don't want the change above to make our code overly complicated and smelly. Use the Template Method pattern to come up with code that supports both phone bills and internet bills (if your code works but doesn't use the Template Method pattern you haven't answered the question!). Write below all the resulting code.

Used the template method to create an abstract class Bill and 2 sub classes that extend bill named PhoneBill and InternetBill. Total is calculated inside getBillableAmount in Bill.

```
public abstract class Bill {

    private int minutes;
    private double rate;
    private String type;
    private static double GST = 0.9;

    public final void buildBill(){
        getPhoneBillTotal();
        getInternetBillTotal();
        System.out.println("Bill is built");
    }

    public abstract double getPhoneBillTotal();

    public abstract double getInternetBillTotal();

    public double getBillableAmount() {
        double total = getInternetBillTotal() + getPhoneBillTotal();
        return total;
    }
}

public class PhoneBill extends Bill{
    double GST = 0.13;
    double minutes;
    double rate;

    public PhoneBill(int m, double r, String t) {
        minutes = m;
        rate = r;
    }

    @Override
    public double getPhoneBillTotal() {

        double base = 20 + minutes * rate;
        double tax = base * GST * 0.2;
        return base + tax;
    }
}
public class InternetBill extends Bill{
    double GST = 0.13;
    double minutes;
    double rate;

    public InternetBill(int m, double r, String t) {
        minutes = m;
        rate = r;
    }

    @Override
    public double getInternetBillTotal() {

        double base = 30 + minutes * rate;
        double tax = base * GST;
        double total = base + tax;
        return total;
    }
}
```

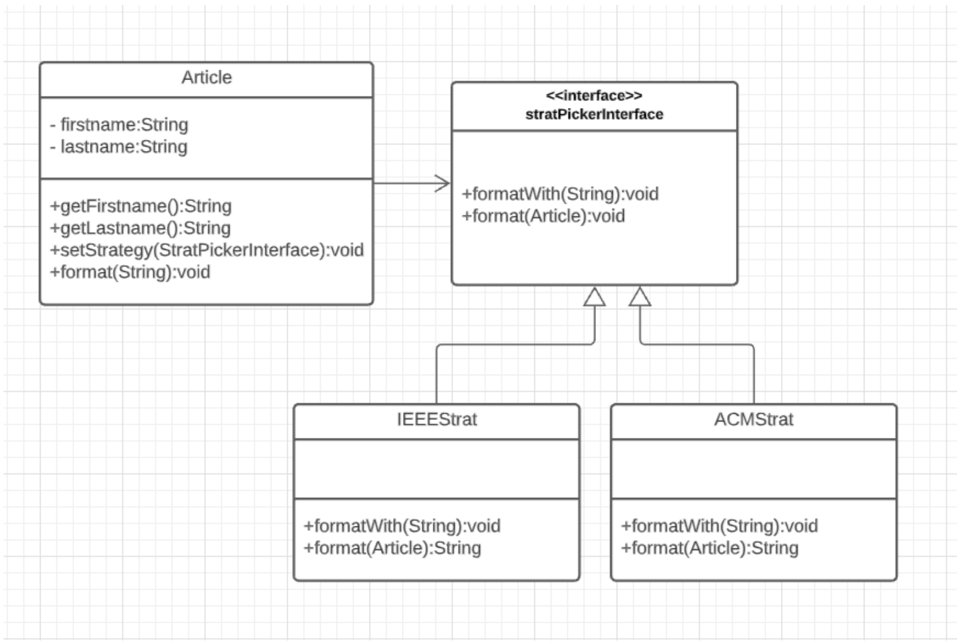
Note: Ran out of time to test it and fix smells. Formatting got ruined due to word.

Question 3 – Strategy Pattern **[5 marks]**

We want to come up (again!) with an application that helps us format an article citation using different citation formats. The article author provides the first name and last name, and selects a formatting strategy. If the "IEEE" strategy is chosen, the formatted text is the initial of the first name, followed by a period ".", a space, and the last name (e.g., "B. Esfandiari"). If "ACM" is chosen, the first name, followed by a space, followed by the last name and a period "." is obtained (e.g., "Babak Esfandiari."). Here we are focusing on the logic of this application, NOT the graphical user interface. Below is the code that could minimally take care of the logic described above:

1) As the number of fields in the article and the number of strategies increase, it would be unmaintainable to have a single method be responsible for all the different formatting strategies. Refactor the above using the Strategy Pattern, where each formatting strategy would be implemented in its own class:

1a) Provide the UML class diagram of your resulting design



1b) Provide the complete code for your design

```
public class Article {
    private String firstname, lastname;
    public Article(String firstname, String lastname) {
        this.firstname = firstname; this.lastname = lastname;
    }
    public String getFirstname() {return firstname;}
    public String getLastname() {return lastname;}
    stratPickerInterface smStrategy;

    public void setStrategy(stratPickerInterface smStrategy) {
        this.smStrategy = smStrategy;
    }

    public void format(String name) {
        smStrategy.formatWith(name);
    }
}
public interface stratPickerInterface {
    public void formatWith(String stratName);

    public void format(Article article);

}

public class IEEEStrat implements stratPickerInterface {
    public void formatWith(String stratName) {
        System.out.println("Formatting with " + stratName);
    }
    public String format(Article article) {
        String firstname = article.getFirstname();
        String lastname = article.getLastname();
        String formatted = null;
        formatted = firstname.charAt(0) + ". " + lastname;
        return formatted;
    }
}

public class ACMStrat implements stratPickerInterface {
    public void formatWith(String stratName) {
        System.out.println("Formatting with " + stratName);
    }
    public String format(Article article) {
        String firstname = article.getFirstname();
        String lastname = article.getLastname();
        String formatted = null;
        formatted = firstname + " " + lastname + ".";
        return formatted;
    }
}
```

2) Provide a couple of unit tests, one for each formatting strategy, verifying that they format your instructor's name correctly. You can use either JUnit 1.3 or 1.4 syntax.

```
import org.junit.Test;

class IEEEStratTest {

    private Article article = new Article("Kareem", "El Assad");

    @Test
    void testFormat(Article article) {
        assertEquals("K.El Assad");
    }
}
```

```
import org.junit.Test;

class ACMStratTest {

    private Article article = new Article("Kareem", "El Assad");

    @Test
    void testFormat(Article article) {
        assertEquals("Kareem El Assad.");
    }
}
```

Note: I had issues running this as my IDE thought the entire final belonged to a package (This meant that I couldn’t generate code with intellij and I couldn’t see any highlighted bugs as it was insistent on the package issue) but I hope the general idea is conveyed. I did not have the time to debug the issue and prioritized the exam.

Thank you so much! Have a great holiday season 😊