**Lab 05:CustomerProjectsApp**

*In this lab, you will:*

* *Become more familiar with the idea of organising architectural code, i.e., code that realises the pattern of architecture outlined in the lectures.*
* *This will be done by looking at, and working with, the architecture of a slightly more complex application, building incrementally from previous examples and:*
  + *Looking into the levels of abstraction in the software architecture, for example*
    - *Abstract classes*
    - *Nested classes*
    - *Looking more at how interfaces are realised or ‘implemented’*

*… which will result in the following learning outcome:*

* *An appreciation of how certain architectural patters are implemented in the Java language, while further working with the CustomerProjectsApp. The scenario intended is for more realistic detail to be added to the application and, consequently, the need for code re-organisation.*

Table of Contents

[1.1 Preliminary 3](#_Toc503284279)

[1.2 Exercise 1 3](#_Toc503284280)

[1.3 Exercise 2 4](#_Toc503284281)

[1.4 Exercise 3 7](#_Toc503284282)

Figure 1: Lab5\_CustomerProjectsAppLab5Starter 3

## Preliminary

* Download the lab 5 zip file < **Lab5\_CustomerProjectsAppStarter.zip**> from GCULearn and unzip.
* Open the **Lab5\_CustomerProjectsAppStarter** project in NetBeans. The content of this project actually contains the solution to Lab 4, just renamed so that it can sit in your NetBeans workspace without problems, should you choose to have previous projects’ labs open. We will be extending this copy of the lab 4 solution in this lab.

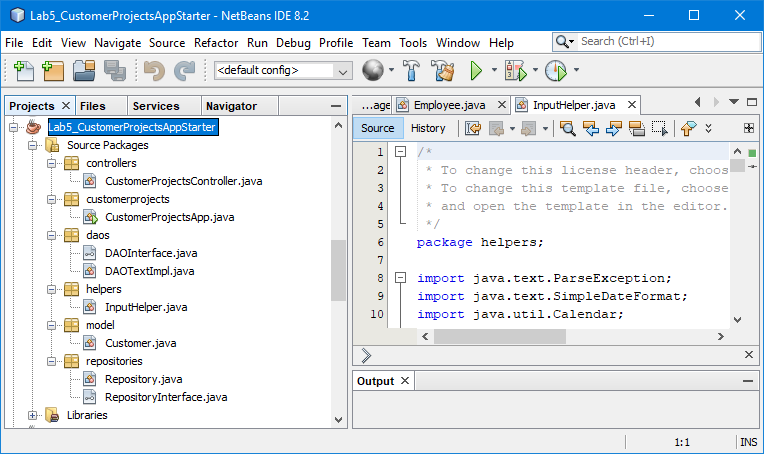


Figure 1: Lab5\_CustomerProjectsAppStarter

## Exercise 1

1. Open the **Customer** class, in the model folder, and note the attributes: **customerId**, **customerName**, and **customerProjects**. Constructor, getter and setter methods are defined as well as add, and remove project methods and the overridden method **toString()**.
2. Now open the *customers.txt* file and note the format. Ensure you understand how this might be loaded into a collection using the ideas discussed earlier.

**1,"Martin",2,"MLGProj1","MLGProj2"**

**2,"Lynn",1,"LPKProj1"**

**3,"Ciara",3,"CODProj1","CODProj2","CODProj3"**

1. Open the controller class: a constructor method requests a file name from the user and then constructs a **Repository** object.

**String fileName = inputHelper.readString("Enter filename");**

**this.repository = new Repository(fileName);**

1. The constructor of the **Repository** class creates a **Data Access Object** and executes its **load()** method to populate the **Repository** i.e. with a collection of **Customer** objects.

**DAOTextImpl dao = new DAOTextImpl();**

**this.items = dao.load(filename).getItems();**

1. The **DAOTextImpl** class realizes the **DAOInterface** by fleshing out the **load()** & **store()** methods. A **DAOObjImpl** class would flesh out the methods differently to utilise object files.

**public interface DAOInterface {**

**public Repository load(String filename);**

**public void store(String filename, Repository repository);**

**}**

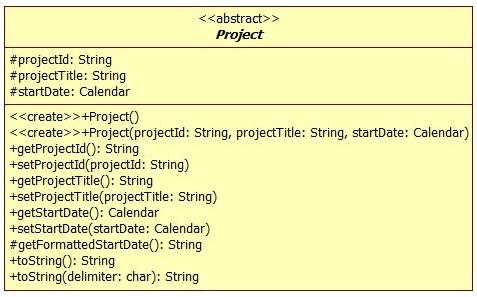
1. Ensure you understand the architecture and role and responsibilities of each class before proceeding. Complete the table for each class.

|  |  |  |  |
| --- | --- | --- | --- |
| **Class** | **Attributes** | **Methods** | **Responsibilities** |
|  |  |  |  |

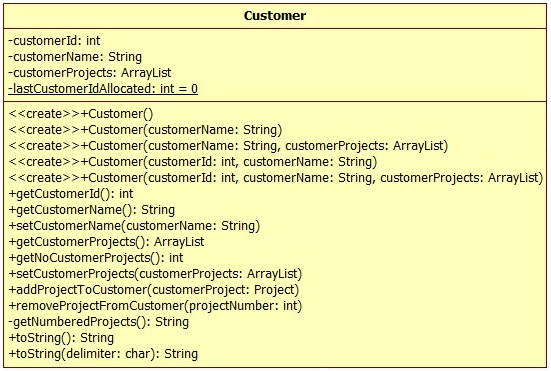
## Exercise 2

Note that at this point a customer project is simply identified by a **project id**, it is more likely that we would have further information regarding a project such as a **project title** and **start date**.

1. Add a **Project** class to the model folder as represented in the class diagram below:



1. Amend the **Customer** class to specify **customerProjects** as an **ArrayList<Project**> adjusting code as necessary.



1. Amend the **addProjectToCustomer()**method in the controller class as follows:

**System.out.println("Customer\n========\n" + requiredCustomer);**

**String newProjectId =**

**inputHelper.readString("Enter New Project Id");**

**String newProjectTitle =**

**inputHelper.readString("Enter New Project Title");**

**SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");**

**Calendar newStartDate =**

**inputHelper.readDate("Enter New Start Date", sdf);**

**Project newProject =**

**new Project(newProjectId, newProjectTitle, newStartDate);**

**requiredCustomer.addProjectToCustomer(newProject);**

Here we are asking the user to enter new project details, constructing a **Project** object and adding that object to the specified **Customer** object.

1. Now we have to consider the format of the text file before we adjust the **load()** method in the **DAOTextImpl** class:

**1,"Martin",2**

**"MLGProj1","Finance","2016-10-31"**

**"MLGProj2","Web","2017-01-03"**

**2,"Lynn",1**

**"LPKProj1","Web Update","2016-11-03"**

**3,"Ciara",3**

**"CODProj1","Accounts","2016-10-21"**

**"CODProj2","Stock Control","2016-12-03"**

**"CODProj3","Mobile App","2017-01-12"**

We can now adjust the **load()** method of the **DAOTextImpl** class, in fact, we only need to adjust the contents of the **for** loop which reads the project details from the file for each customer. Implement the necessary code for this design:

*Read line from file*

*Split line into temp array*

*Extract first value in temp array to* ***projectId***

*Extract second value in temp array to* ***projectTitle***

*Extract third value in temp array to the* ***Calendar*** *object* ***startDate*** *using a* ***SimpleDateFormat***

*Create a* ***Project*** *object with* ***projectId****,* ***projectTitle*** *and* ***startDate***

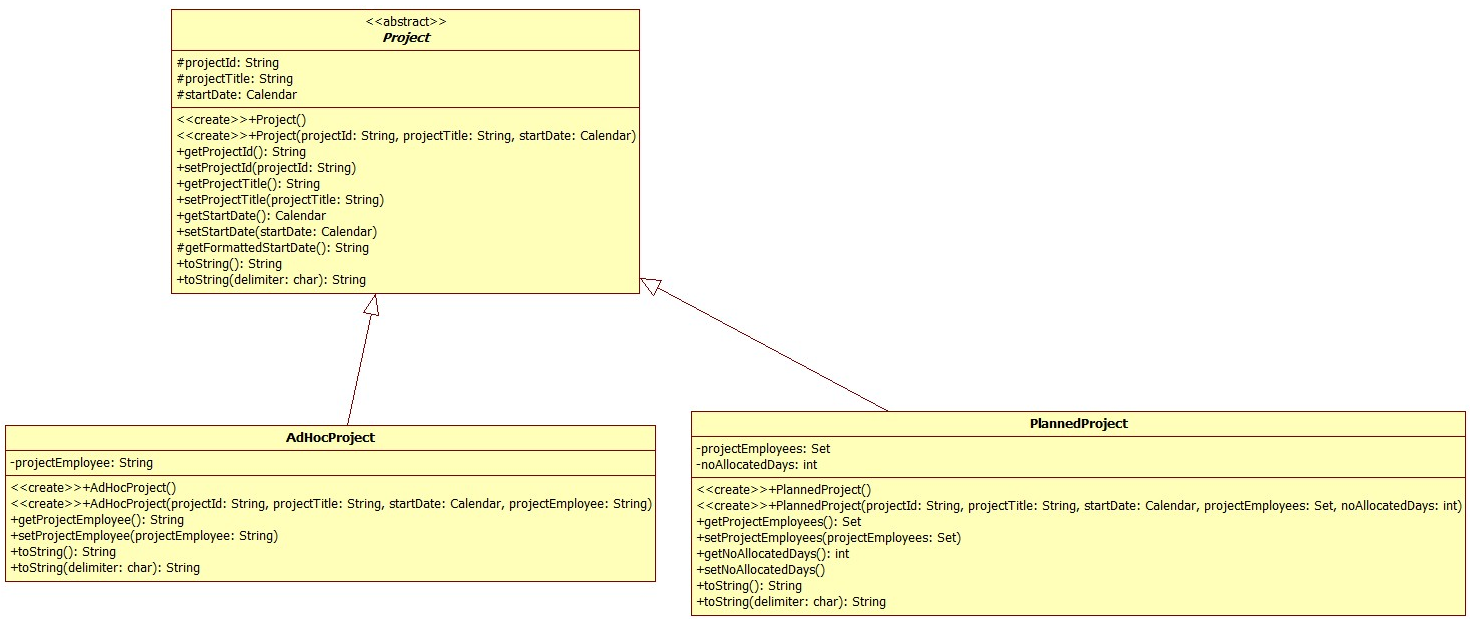
*Add the project to the customer*

1. Run the **CustomerProjectsApp** class and test that customer projects can be successfully loaded and displayed. Add and remove projects from a specified customer and store them back to a text file. Ensure you can reload the file correctly – you may need to adjust the delimited **toString()** methods of the **Customer** and **Project** classes to ensure that the output file is in the correct format.

## Exercise 3

Further clarification indicates that there are two types of projects: planned and ad hoc with the attributes specified in the class diagram below. In fact, a customer project **must** be one of these types therefore the **Project** class we have created should be **abstract** i.e. no object instance can be created and it simply specifies attributes and methods that subclasses inherit.

1. Create the **PlannedProject** and **AdHocProject** subclasses and make any adjustments necessary to the **Project** class.



|  |  |  |
| --- | --- | --- |
| **Subclass** | **Attributes** | **Methods** |
| **AdHocProject** | projectEmployee – String | AdHocProject()  AdHocProject(projectId, projectTitle, startDate,  projectEmployee)  getProjectEmployee()  setProjectEmployee(projectEmployee)  toString()  toString(delimiter) |
| **PlannedProject** | projectEmployees – Set of String  noAllocatedDays – int | PlannedProject()  PlannedProject(projectId, projectTitle, startDate,  projectEmployees, noAllocatedDays)  getProjectEmployees()  setProjectEmployees(projectEmployees)  getNoAllocatedDays()  setNoallocatedDays(noAllocatedDays)  toString()  toString(delimiter) |

1. Amend the **addProjectToCustomer()**method in the controller class to specify whether a planned or ad hoc project is being added requesting the specific information required.
2. Adjust the **load()** method of the **DAOTextImpl** class to load both types of projects. Hint: you might want to distinguish between the two types of projects using a P and an A.

**1,"Martin",2**

**"MLGProj1","Finance","2016-10-31","P","Tony","Mike",5**

**"MLGProj2","Web","2017-01-03","A","Phil"**

**2,"Lynn",1**

**"LPKProj1","Web Update","2016-11-03","A","Phil"**

**3,"Ciara",3**

**"CODProj1","Accounts","2016-10-21","A","Ant"**

**"CODProj2","Stock Control","2016-12-03","P","Ant",2**

**"CODProj3","Mobile App","2017-01-12","P","Tony","Mike","Phil",25**

1. Run the **CustomerProjectsApp** as before.

~~~~~~