

## Criterion C: Development

Word count: 1030

### 1. Tools used for development

#### IDE

Apache NetBeans IDE is used for development since it provides support for Java developers. Also, NetBeans' design tools for JFrame allow me to easily complete the GUI design with drag-and-drop actions, accelerating product development.

#### JCalendar

JCalendar is an external library I used for the GUI development. The JDateChooser component in this library is used to allow users to graphically pick a date.

This ensures that the user input is specified to Date objects (java.util.Date) and prevents invalid input.

```
this.endDateChooser.setMaxSelectableDate(this.manager.getProject().getEndDate());
```

Figure 1: setting maximum date for user input

JDateChooser class also provides methods to set range for selectable dates, which can prevent users from entering dates after the deadline of the project.

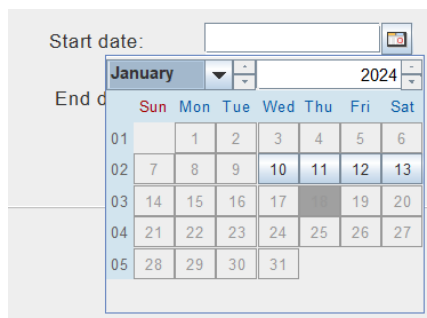


Figure 2: graphically pick a date with JDateChooser

## 2. Structure of the product

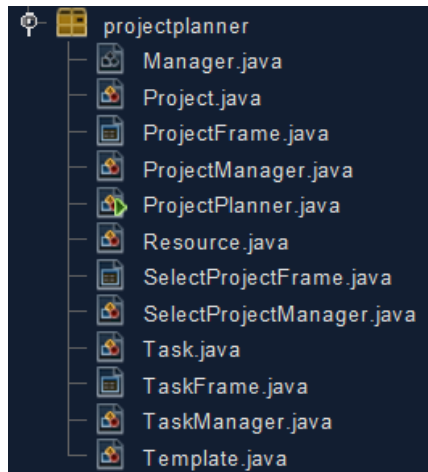


Figure 3: Structure of the program

“...Frame.java” files are GUI classes. **ProjectManager**, **SelectProjectManager**, **TaskManager** are used for data communication between GUI and backend, in which ProjectManager and TaskManager are subclasses of **Manager**. **Template** is the super class of **Project** and **Task**. Inheritance is used extensively due to the similarity between Project and Task.

## 3. GUI development

Swing is a GUI widget toolkit for Java that provides multiple common GUI components including frames, buttons, labels, and events. It is used for my product since the program only requires simple user inputs like clicking and entering.

### **JFrame**

For every frame, there is a GUI class that is responsible for handling the elements in the frame. The GUI classes are subclasses of the JFrame class.

```
public class SelectProjectFrame extends javax.swing.JFrame {
```

```
public class TaskFrame extends javax.swing.JFrame {
```

Figure 4: subclasses of JFrame

### **GUI elements**

Using NetBeans design tools, I was able to create the GUI without writing additional code.

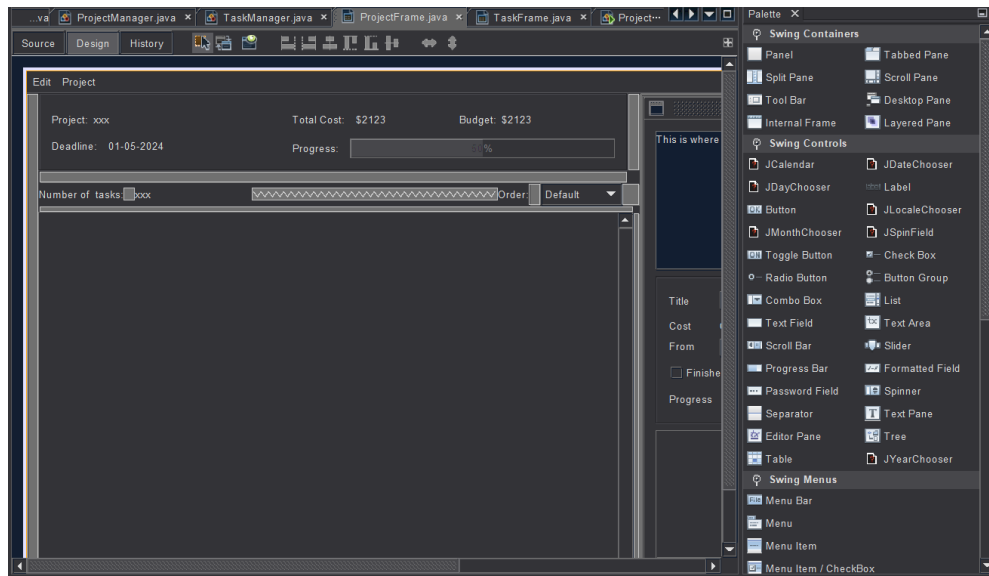


Figure 5: Creating the layout design for ProjectFrame

Through drag-and-drop actions, the code for certain components is automatically generated by NetBeans.

```
private javax.swing.JLabel swtHeaderLabel;
private javax.swing.JScrollPane swtScrollPane;
private javax.swing.JPanel swtHeaderPanel;
private javax.swing.JLabel swtName;
private javax.swing.JTextField swtNameTextField;
private javax.swing.JLabel swtNameLabel;
private javax.swing.JLabel swtNameLabel;
private javax.swing.JTextField swtNameSettingField;
private javax.swing.JLabel swtNameSettingLabel;
private javax.swing.JLabel swtName;
private javax.swing.JLabel swtName;
private javax.swing.JLabel swtNameSettingLabel;
private javax.swing.JTable swtTable;
private javax.swing.JTextField swtTable;
private javax.swing.JLabel swtTable;
private javax.swing.JLabel swtTable;
private com.toedter.calendar.JDateChooser swtJDateChooser;
private javax.swing.JLabel swtJDateChooser;
private javax.swing.JButton swtJDateChooser;
private javax.swing.JLabel swtTable;
private javax.swing.JLabel swtTable;
```

Figure 6: GUI components generated by NetBeans

NetBeans' automatic code generation is only used in the creation and initialization of the GUI elements as it provides a fast approach. Other GUI methods are manually written.

```

private void initComponents() {
    errorDialog = new javax.swing.JDialog();
    errorLabel = new javax.swing.JLabel();
    projectScrollPane = new javax.swing.JScrollPane();
    projectList = new javax.swing.JList<>();
    projectNameLabel = new javax.swing.JLabel();
    projectNameField = new javax.swing.JTextField();
    projectBudgetLabel = new javax.swing.JLabel();
    projectBudgetField = new javax.swing.JTextField();
    startDateLabel = new javax.swing.JLabel();
    startDateChooser = new com.toedter.calendar.JDateChooser();
    endDateLabel = new javax.swing.JLabel();
    endDateChooser = new com.toedter.calendar.JDateChooser();
}

```

Figure 7: Generated code for the initialization of GUI elements

#### 4. Communication between GUI and backend through Manager(s)

Manager classes allow data from the backend to be displayed in the front-end GUI, and user input to be sent to the backend for processing.

For example, when adding a new task to the project, instead of directly calling the **addTask()** method of the project, the GUI class calls **newTask()** from the Manager class.

```

private void okAddTaskButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // adding a new task
    try {
        this.manager.newTask(this.taskNameField.getText(), this.startDateChooser1.getDate(), this.endDateChooser2.getDate());
        this.addTaskFrame.dispose();
        this.updateTable();
    } catch (Exception e) {
        e.printStackTrace();
        // error message
        JOptionPane.showMessageDialog(this, "Please enter valid data.\n"
            + "1. The fields can't be left empty\n2. End date cannot be before start date", "Error", JOptionPane.WARNING_MESSAGE);
    }
}

```

Figure 8: GUI method for the 'add task' button from ProjectFrame

```

public void newTask(String name, Date startDate, Date endDate) throws Exception{
    // end date has to be after start date
    if (endDate.before(startDate)) {
        throw new Exception();
    }
    Task task = new Task(name, this.template);
    task.setStartDate(startDate);
    task.setEndDate(endDate);
    this.template.addTask(task); // add it to the project or task
}

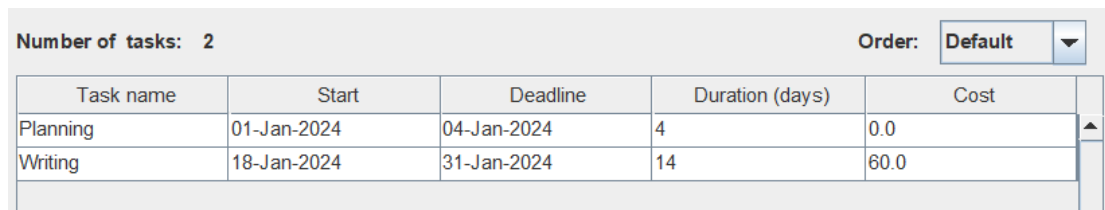
```

Figure 9: newTask() method from Manager

This implements encapsulation and improves the modularity of the program: the GUI class is only responsible for the visual presentation and shouldn't handle any algorithmic operation; also, **newTask()** validates user input.

## Task table

When a new task is added to or deleted from a project, the task table displayed on the interface needs to be updated.



The screenshot shows a GUI with a header bar. On the left, it says "Number of tasks: 2". On the right, there is a label "Order:" followed by a dropdown menu set to "Default". Below this is a table with 5 columns: "Task name", "Start", "Deadline", "Duration (days)", and "Cost". The table contains two rows: "Planning" with start "01-Jan-2024", deadline "04-Jan-2024", duration "4", and cost "0.0"; and "Writing" with start "18-Jan-2024", deadline "31-Jan-2024", duration "14", and cost "60.0". A vertical scrollbar is on the right side of the table.

Task name	Start	Deadline	Duration (days)	Cost
Planning	01-Jan-2024	04-Jan-2024	4	0.0
Writing	18-Jan-2024	31-Jan-2024	14	60.0

Figure 10: Task table in the GUI

When a table update is needed, the GUI method **updateTable()** is called by the frontend, which calls **refreshTable()** in the Manager class.

```
private void updateTable() {
    int row = this.taskTable.getSelectedRow(); // pre-store what's been selected
    this.taskNum.setText(String.valueOf(this.manager.getProject().getTaskNum())); // set the task count
    int newRow = this.manager.refreshTable(this.orderBox.getSelectedItem().toString(), row); // refreshes the table accordingly
    DefaultTableModel model = new DefaultTableModel(this.manager.getData(), this.manager.getColumns()) {
        /* prevents user from editing the cells */
        @Override
        public boolean isCellEditable(int row, int column) {
            return false;
        }
    };
    this.taskTable.setModel(model); // update
    if (newRow != -1 && this.taskTable.getRowCount() != 0) this.taskTable.setRowSelectionInterval(newRow, newRow); // this is to keep the row selected after updating
}
```

Figure 11: updateTable() method from ProjectFrame

Manager has a **taskTable**, which is an array that stores the tasks. **refreshTable()** updates the array according to the selected order and returns the new row index of the task previously selected by the user. Therefore, the method takes two arguments - an order and a row index – and returns a new row index that indicates the new position of the previously selected task.

```

// set the task table according to the order provided
// returns the current row of the previously selected task
public int refreshTable(String order, int row) {
    int newRow = -1; // the row to be selected after the update
    Task selected = null;
    if (row!=-1) { // if a row is already selected
        selected = this.taskTable[row];
    }

    /* This method is called every time new changes are made to the table, so a new task table is created every time */
    Task[] temp = new Task[this.template.getTaskNum()];
    // collect all tasks stored
    for (int i=0; i<temp.length; i++) {
        temp[i] = this.template.getTasks().get(i); // this stores the object reference of that task in the table
        if (temp[i]==selected) {
            newRow = i; // stores the new row number if this is the previous one selected
        }
    }

    /* sort and update taskTable */
    Update based on order

    return newRow;
}

```

Figure 12: refreshTable() from Manager

This method stores all tasks in **temp** through iteration, then sorts these task accordingly and updates **taskTable** (hidden in the picture, “Update based on order”). Notice how every task is compared with **selected** to keep track of the selected task in the for loop.

Manager’s **getData()** is then called to send data to frontend as a 2D-array.

```

// return a 2D array according to taskTable
public String[][] getData() {
    String data[][] = new String[this.template.getTaskNum()][5];
    for (int i=0; i<data.length; i++) {
        Task t = this.taskTable[i];
        data[i][0] = t.getTitle();
        data[i][1] = new SimpleDateFormat("dd-MM-yyyy").format(t.getStartDate()); // Turning Date objects into String with a specific format
        data[i][2] = new SimpleDateFormat("dd-MM-yyyy").format(t.getEndDate());
        data[i][3] = t.getDuration() + "";
        data[i][4] = String.valueOf(t.getCost());
    }
    return data;
}

```

Figure 13: getData() from Manager

It uses data from **taskTable** to simulate the table users see in the program.

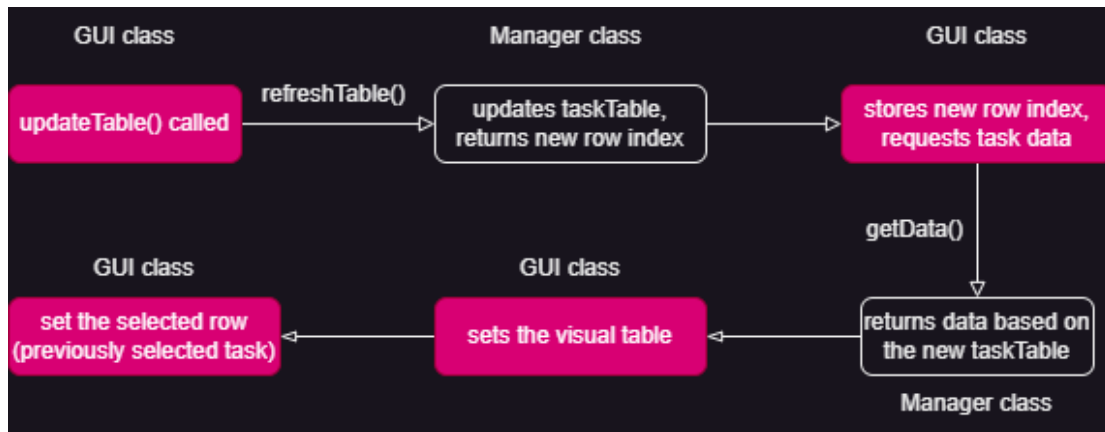


Figure 14: Diagram showing the communication process between Manager and ProjectFrame

Again, the Manager class ensures that algorithmic processing occurs in the backend and the GUI class is only used to send and display data. This is important as it hides the complexity through encapsulation. When developing, this helped me quickly identify errors. If a new sorting order is required by the client, I can simply extend the method `refreshTable()`.

## 5. Sorting

The tasks are sorted in `refreshTable()` based on the given order, so a switch statement is used to handle different requests, which also allows new sorting options to be added in the future (e.g. sort by task progress).

```

switch(order) {
    case "Default":
        /*
         By default, taskTable is not sorted,
         thus assigned the value of temp (the array of all tasks)
        */
        this.taskTable = temp;
        break;

    case "Start":
        Sorting based on start date
        break;

    case "Deadline":
        Sorting based on deadline
        break;

    case "Duration":
        Sorting based on duration
        break;

    case "Cost":
        Sorting based on cost
        break;
}
  
```

Figure 15: Switch statement used for choosing order

```

case "Cost":
    // <editor-fold desc="Sorting based on cost">
    // selection sort
    for (int i=0; i<temp.length; i++) {
        int max = i;
        for (int j=i+1; j<temp.length; j++) {
            if (temp[j].getCost()>temp[max].getCost()) {
                max = j;
            }
        }
        // swap
        Task s = temp[i];
        temp[i] = temp[max];
        temp[max] = s;
        if (temp[i]==selected) {
            newRow = i; // stores the new row number if it's the same as the one selected
        }
    }
    this.taskTable = temp;
    // </editor-fold>
break;

```

Figure 16: Sorting by cost

Selection sort is used, since the number of tasks in **temp** will more likely be small - the client wants tasks within tasks, so the task count for one task or one project shouldn't be too high – and selection sort performs well with small arrays (GeeksforGeeks, "Selection Sort").

At the end of the first for loop, an if statement checks if the current item is selected and updates the new row index if true. In doing so, the task selected by the user remains selected.

Number of tasks: 2				Order:	Duration ▼
Task name	Start	Deadline	Duration (days)	Cost	
Writing	18-Jan-2024	31-Jan-2024	14	60.0	▲
Planning	01-Jan-2024	04-Jan-2024	4	0.0	

Figure 17: Before adding a new task, 'Writing' is selected

Number of tasks: 3				Order:	Duration ▼
Task name	Start	Deadline	Duration (days)	Cost	
new task	01-Jan-2024	31-Jan-2024	31	0.0	▲
Writing	18-Jan-2024	31-Jan-2024	14	60.0	
Planning	01-Jan-2024	04-Jan-2024	4	0.0	

Figure 18: After a new task is added, the tasks are sorted, and 'Writing' remains selected



## 6. ArrayList

Since the task list behaves similarly for both **Project** and **Task**, inheritance from **Template** is implemented. The (sub)tasks of a project/task are stored in an ArrayList of **Task** objects.

```
private ArrayList<Task> tasks; // all the subtasks
```

Figure 19: ArrayList used for tasks

The dynamic data structure is resizable, meaning that the list doesn't have to be fixed-sized. This is important as the list of tasks might grow according to users' needs. Furthermore, the **size()** method is helpful when accessing the task count.

```
// adding a subtask to the project or task
public void addTask(Task task) {
    this.tasks.add(task);
}
```

Figure 20: addTask() method from Template

The recursive use of this field allows the user to create tasks within tasks.

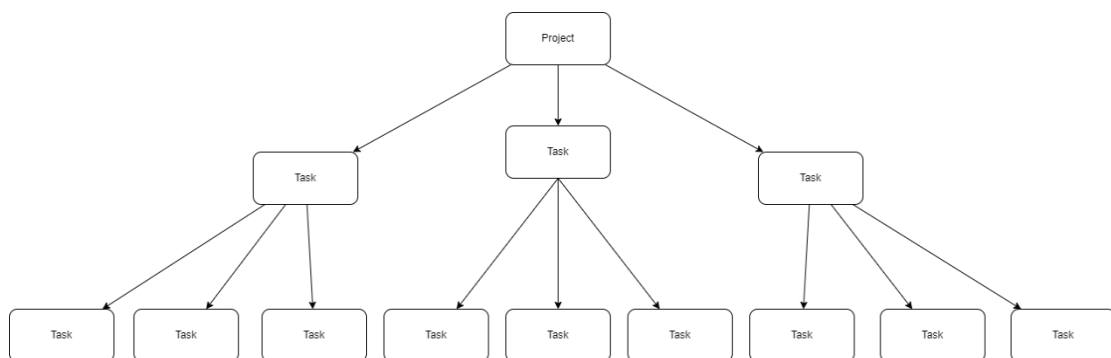


Figure 21: The hierarchy of project and tasks

## 7. Calculating progress/cost

```
// calculates the progress of the current task or project
public void calculateProgress() {
    if (!this.tasks.isEmpty()) { // if this project/task has tasks
        float totalTaskDone = 0;
        for (Task task : this.getTasks()) { // counting how many tasks are finished
            totalTaskDone = totalTaskDone + task.getProgress();
        }
        this.setProgress((float) totalTaskDone / (float) this.getTasks().size());
    } else this.progress = 0; // if no tasks are added
}
```

Figure 22: Progress calculation algorithm from Template

The method iterates through the task list and finds the sum of progress. The overall progress of a project is calculated by deviding the total progress by the total task number, which works like the average calculation (Terr, “Weighted Mean.”).

Some small modifications are made for **Task** objects through polymorphism, as it inherits **Template**.

```
@Override
public void calculateProgress() {
    if (this.isFinished()) this.setProgress(1); // if a task is finished, the progress should be 1
    else if (!this.getTasks().isEmpty()) {
        float totalTaskDone = 0;
        for (Task task : this.getTasks()) { // counting how many tasks are finished
            totalTaskDone = totalTaskDone + task.getProgress();
        }
        this.setProgress((float) totalTaskDone / (float) this.getTasks().size());
    }
    else this.setProgress(0); // if no tasks are added
}
```

Figure 23: Progress calculation algorithm from Task

If the user marks the task ‘finished’, the progress is 1. The same steps are applied to cost calculation.

```
// calculates the costs for the object (a task or a project)
public void calculateCost() {
    float costs = 0;
    for (Task task : this.tasks) { // add up the cost of each task
        costs = costs + task.getCost();
    }
    this.cost = costs;
}
```

Figure 24: Progress calculation algorithm from Template

```
@Override
public void calculateCost() {
    float costs = 0;
    for (Task task : this.getTasks()) { // add up the cost of each task
        costs = costs + task.getCost();
    }
    for (Resource r : this.getResources()) { // adding the cost of each resource
        costs = costs + r.getCost();
    }
    this.setCost(costs);
}
```

Figure 25: Overridden progress calculation algorithm in Task

**calculateCost()** is overridden because the cost of a task includes the cost of its subtasks and the cost from its resources.

## 8. Calculating duration

```
// calculates and returns the duration
public int getDuration() {
    long diff = this.endDate.getTime() - this.startDate.getTime();
    int range = (int) TimeUnit.DAYS.convert(diff, TimeUnit.MILLISECONDS); // convert milliseconds to days
    return range + 1; // the actual duration, not just days in between. e.g. from day 7 to day 8 it takes two days
}
```

Figure 26: Calculating duration using TimeUnit

To calculate the difference between two dates, using **TimeUnit** to convert milliseconds to days avoids manual calculations and extra coding (Stack Overflow, “Milliseconds to Days.”). Users can see the duration of tasks through this method.

## 9. File handling

```
public void save() throws Exception{
    String path = "data\\projects"; // where data is stored
    if (! new File(path).exists()) { // if data folder doesn't exist
        new File(path).mkdirs(); // creates the folder
    }
    File file = new File(path + "\\ " + this.getID() + ".dat"); // file name
    FileOutputStream fout = new FileOutputStream(file);
    ObjectOutputStream oout = new ObjectOutputStream(fout);
    oout.writeObject(this);
    oout.close();
}

public void delete() throws Exception{
    String path = "data\\projects\\";
    File dir = new File(path);
    for (String filename:dir.list()) {
        // delete if the file name is '(ID).dat'
        if (filename.endsWith(".dat")&&filename.startsWith(this.getID())) {
            File file = new File(path + filename);
            file.delete();
        }
    }
}
```

Figure 27: save() and delete() methods from Project

When the user decides to save, all data of a project is saved as a .dat file in data/projects (relative path). The program creates a folder if the directory does not exist.

```
private String idGenerator() { // generates the ID for the project
    Random random = new Random();
    int randomNum = random.nextInt(1000); // generates a random integer from 0-999
    String id = "p_" + randomNum;
    Instant instant = Instant.now();
    id = id + String.valueOf(instant.getEpochSecond()); // the id consists of the random number and the current epoch second
    return id;
}
```

Figure 28: generating a random ID – method from Project

The ID of a project is “p\_(random integer)(current Unix epoch second)”, generated when the project is created, which is nearly impossible to replicate (“Unix time.”). The file name is (ID).dat, hence unique for every project.



Name	Date modified	Type	Size
 p_601705281209.dat	1/17/2024 10:01 PM	DAT	2 KB
 p_2741705578873.dat	1/18/2024 12:54 PM	DAT	1 KB

Figure 29: Saved files for program data

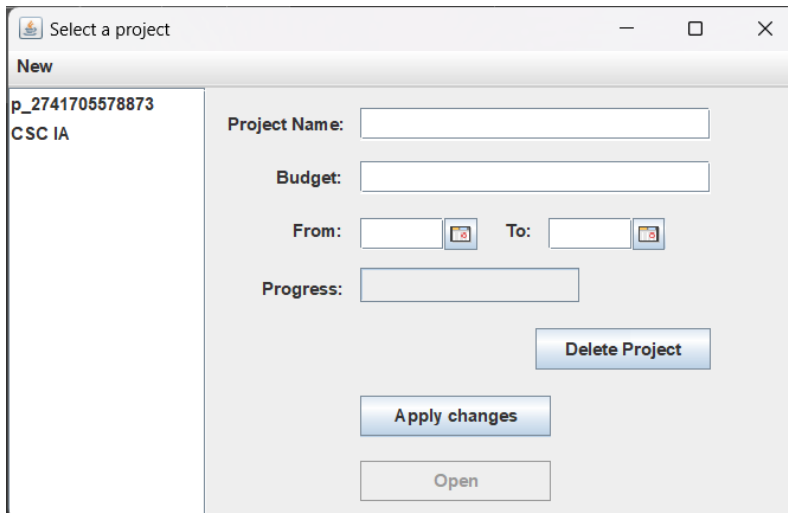


Figure 30: Loading saved files from the start menu

The program can also load saved data.

```
private void setProjectList() { // responsible for loading projects from files
    this.projects = new ArrayList();
    try {
        File dir = new File("data\\projects");
        dir.mkdirs();
        for (String file:dir.list()) {
            if (file.endsWith(".dat")) { // load files with suffix .dat
                ObjectInputStream ois = new ObjectInputStream(new FileInputStream("data\\projects\\" + file));
                Project project = (Project) ois.readObject();
                ois.close();
                this.projects.add(project);
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Figure 31: Code for loading saved files – method from SelectProjectManager

## 10. References

Apache NetBeans. [netbeans.apache.org/front/main/](https://netbeans.apache.org/front/main/).

GeeksforGeeks. "Selection Sort – Data Structure and Algorithm Tutorials." *GeeksforGeeks*, 5 Jan. 2024, [www.geeksforgeeks.org/selection-sort/](https://www.geeksforgeeks.org/selection-sort/).

Java™ Platform, Standard Ed. 8. "ArrayList" *Class ArrayList<E>*, <https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html>. Accessed 20 Jan. 2024.

"Milliseconds to Days." *Stack Overflow*, 19 Oct. 2011, [stackoverflow.com/questions/7829571/milliseconds-to-days](https://stackoverflow.com/questions/7829571/milliseconds-to-days).

Terr, David. "Weighted Mean." From *MathWorld*-A Wolfram Web Resource, created by Eric W. Weisstein. <https://mathworld.wolfram.com/WeightedMean.html>. Accessed 20 Jan. 2024.

Tödter, Kai. "JCalendar." *Toedter.Com*, 14 Sept. 2023, [toedter.com/jcalendar/](https://toedter.com/jcalendar/).

"Unix time." *Wikipedia*, 5 Jan. 2024, [https://en.wikipedia.org/wiki/Unix\\_time](https://en.wikipedia.org/wiki/Unix_time).