Project Management System Report

By Dominykas Stasiulionis – 001044522 | Group 17

Section 1

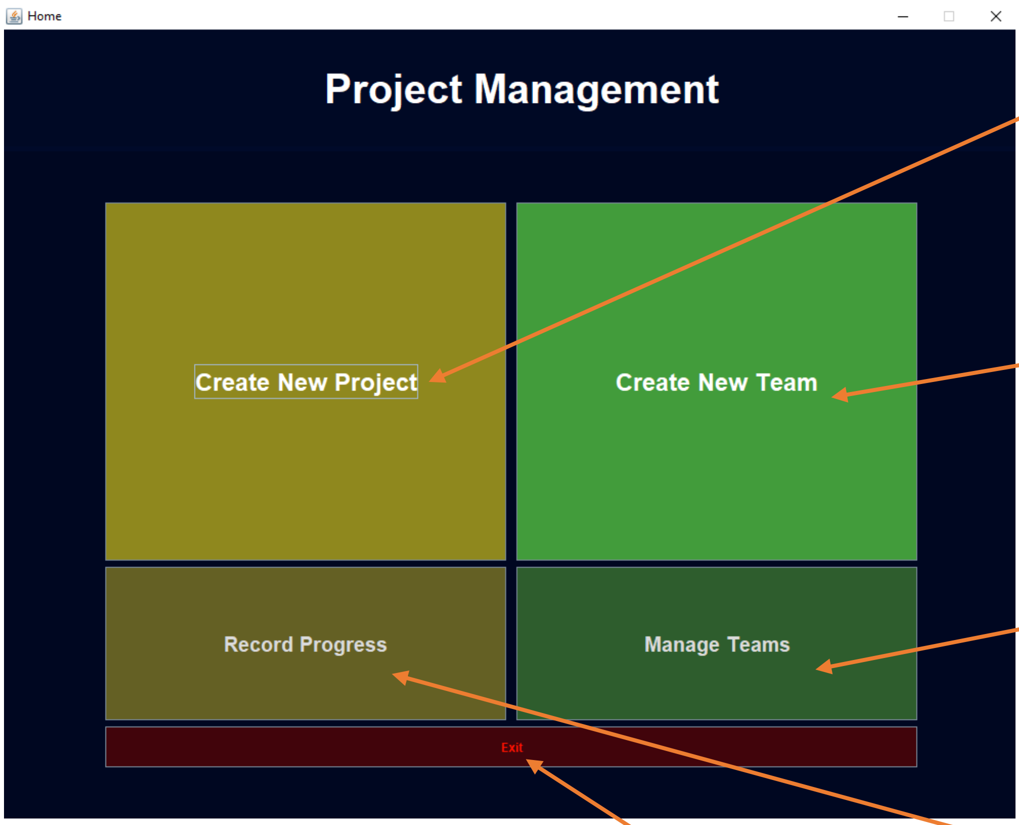
We used the Java, Kotlin and Scala programming languages to create this application. Java was always the go-to programming language throughout the production of this application. Especially when developing a new page or feature that required the use of Kotlin. Kotlin is a Java-interoperable language and also an object-oriented language that is consistent with Java. Therefore, by using our existing knowledge of Java we were able to write code in Java and use the IntelliJ IDE to translate that Java code directly into Kotlin code that served the same purpose. This aided us greatly with the development of the Kotlin critical path initially and other various features, in turn allowing us to meet the requirement. This is a huge benefit to both languages, as it is possible to also translate Kotlin code into Java code and work flexibly as well as provide seamless integration between the two languages. However, after we became more familiar with the syntax of Kotlin later on, it became apparent how much more complex and verbose Java code can be. Sometimes what feels like a small feature would require multiple expressions and statements in Java. Whereas Kotlin code seemed to be more concise despite also being an object-oriented language. On the other hand, Scala was not as welcoming to code translation as Kotlin and Java were. Although Scala is a multi-paradigm language, we were instructed to take a functional approach when coding in Scala for this project. When translating Kotlin/Java code into Scala, we had realised this technique was not going to work reliably as many errors were generated after the conversion. I believe this to be because we were not translating it into object-oriented code, therefore the output was different. Scala is a hybrid of object-oriented and functional programming, which can make type-information more difficult to understand initially. Scala, similarly, to Kotlin, is much more concise than Java. In Java many more lines of code would be required than in Scala/Kotlin in most cases to achieve the same result. But we generally found Scala harder to deal with as we were programming differently than if we were using an object-oriented language like Java and Kotlin. Furthermore, upon research of Scala, there seemed to be limited resources available for it online as this language was not very commonly used. Kotlin overtime became more clearer to me as its easier to read and interpret than some java classes would be. I hadn’t worked too much with Kotlin during the production of this application and even less with Scala. But during the times where I did work with Kotlin, I came to appreciate its conciseness of code and how much clearer it can be immediately when glanced at. I have mainly been working with object-oriented languages such as Java in the past. Java code can sometimes be unnecessarily complicated, so working with another object-oriented language such as Kotlin ultimately felt like a breath of fresh air once I had gained a better understanding of the languages syntax. I cannot say the same for Scala and I think my group members would agree. We lacked understanding of the language and how to pair with other existing elements of our work, which resulted in us struggling to implement certain parts of the application that needed to be implemented using Scala code. One of the weaknesses of Scala was how it worked with Java, unlike Kotlin, it is not consistent with Java and its libraries etc. Because of this, we found it difficult to pair the two programming languages together for a project such as this. Additionally, our inexperience with the language can also directly correlate to this negative outcome. But generally speaking, both object-oriented programming and functional programming do not reject one another. Object-oriented programming doesn't address the limitations of the other languages however it enforces/supports programming methodologies which aid in writing better programs. In turn assisting less experienced users to follow good practices that programmers who are more advanced have been following and developing without the support. In functional programming, expressing complex ideas can be done in short and more concise statements. This is especially great when working with or solving Maths related issues in programming. We had used Kotlin much more than Scala during the production of the application, therefore, we had more opportunity to learn and practice Kotlin than we did with Scala. If there were more diverse Scala implementations required aside from the Scala critical path, we would have been forced to learn Scala in more depth and that would've resulted in a better knowledge of Scala attained eventually. After very limited exposure to the Clojure programming language, my first impressions of it are that is difficult to grasp and learn. It is a functional paradigm and as mentioned in this report, I have had some trouble adjusting from object-oriented languages such as Java and Kotlin to languages that require a functional approach such as Scala. Clojure isn't any different. I think if I followed tutorials more closely and used language association, I would be able to create a better understanding of Clojure and generally how object-oriented languages translate into the functional paradigm languages.

Section 2

My main responsibilities within the group project resided in the frontend development of the application, as well as further research work being attempted by everybody, including myself, in the Scala implementation of the critical path stages as this was the part we had struggled with the most. Specifically, I took responsibility for the development of the GUI and all of its functionality and error prevention methods. As well as this, I overlooked any integration attempted by other members to the user interface and guided them in understanding how to execute integration correctly and minimise the chance of errors occurring. The GUI was created using Java, one disadvantage of this design choice is the appearance of the GUI does not provide the desired look and feel. The GUI does not look very modern and instead reminds me of older, more dated applications. It lacks the native desktop feel but I did what I could to make the GUI look presentable. I had carried out some research for inspiration and influence for my design choices. For example, I researched colour schemes and which colours I should use and which I should avoid (User-friendly colours). There was no prototyping conducted prior to the actual GUI design being coded, instead I opted for a more trial and error approach. This was mainly due to the fact I was using the "Swing UI Designer" and it used a grid layout I was unfamiliar with. It seemed to have limited the placement of objects and the layout. After working with the UI designer, I realised everything I had wanted to implement was possible but much more tedious than if I were to attempt this using another IDE such as NetBeans. In the future, if I was to do this again, using my bettered understanding of the layout system and UI designer in IntelliJ, I would at least like to create a thought-out low-fidelity sketch of the interface to guide the production of the GUI. Alternatively, I had suggested and created a draft of the GUI using the JavaFX framework with the use of two external libraries. However, the other group members did not have a great understanding of how this framework operates. Furthermore, some group members had issues with the external libraries not carrying over when pulling the project from the Git desktop app. We decided to not use JavaFX to avoid running into complications in later stages of development. During the later stages of development, we were concerned about running into deadlocks or complications if both the Kotlin and Scala critical paths were active at the same time. Therefore, I decided to temporarily disallow interchangeability between the paths for the same calculation. We ensured the user can select which critical path they would like to activate via checkboxes in the “Record Progress” window. Once selected, the other option would become disabled until the user deselected the current option they had chosen. This option was left unchanged as later in development we were not able to completely integrate our finished Scala critical path code into the GUI due to difficulties understanding the language and other complications. Any data inputted into the GUI by the user such as: team name, task, completion date, projects, etc. were stored into .txt files. This was achieved by utilising the Buffered Reader class in Java, which allowed the input to be read and stored. This feature was a fundamental in the application as the critical path code needed to be fed data for it to work. After the data is stored, with the help with another group member we had organised multiple .txt files that were assigned storage of specific data. To conform to the requirements, this part was done in Kotlin and integrated to the existing Java code. The stored data was made available for access in relevant areas of the application. For example, a user creates a team with 2 members and names it: “Team A”. Then Team A will be available for selection in the “Create New Project” page, where the user creates a new project and is able to assign it to a team. The user is able to use the drop-down combo box to select from existing teams. This was an important fundamental addition also because it meets the requirement of having the persistence in the application. Alternatively, data inputted could’ve been stored into a database which would’ve stored the data more efficiently and required less coding. Lastly, I had worked on implementing error prevention by attempting to break the program by interacting with it. I did this to outline, and account for, any possible ways the user could unintentionally crash the application. Once all the scenarios were figured out, in most cases I had added a checkbox the user would need to select to confirm they are done with editing the contents before the “Create …” button became available. The final “Create” buttons at the bottom of the page were disabled from the start until the checkboxes became ticked. For example, in the “Create New Project” page, the user is prompted to fill in text fields such as: Project Title, Tasks, and approximately how long it will take for that said task to be completed (measured in days). The contents of the error messages displayed have been tailored to each specific scenario where an error has occurred. Such as if the project title is entered but there no tasks in the JTable, then an error message will appear informing the user to enter a task and its expected completion time. Furthermore, if the user doesn’t enter a number in the task completion text field then another message would appear informing the user to only enter numbers. Overall, I think the functionality of the GUI is spot on and the general application corresponds well to the requirements set, with the exception of the Scala critical path integration to the GUI not being present. This would not be the case if we had created a better understanding of the language itself. I think this could be improved by research and practice. The leading reason as to why we were not able to integrate Scala was due to the Java hashsets using “util.Hashset”, were not compatible with the hashsets in the Scala code. Therefore, in future work, utilising a database for data storage would be more optimal and would prevent this error from occurring again. As we did not have enough relevant experience with databases, we opted to store data using .txt files instead. But now we realise the importance of using a database in a situation such as this and how much efficient it would’ve been. The GUI, although not completed as desired, still creates a pleasant atmosphere as the colours are matched well with each other and each page can be easily understood, the GUI is also friendly to newer users due to the error prevention methods. In conclusion, we had built up the fundamentals of the application first such as GUI, saving and storing data, persistence etc. We spent the majority of our time coding outside of Java, as we found Java to be the easiest to work with out of the three languages since we had a plethora of previous experience with it. We got it out the way fairly quickly. I felt some pressure as the frontend developer in for this project to create a suitable GUI as it was graded and we, as a group, would need to use it as a platform for all of our other code to run on. I took the role I enjoyed most in software development and overall, I think we had created a good application that meets most of the requirements set, with one exception being the Scala integration.

Section 3

In this section I will be showcasing what features, pages etc. I had developed for this application and briefly explain their



This button will close the current page and open the "Create New Project" page as indicated by the text in the button.

This button will close the current page and open the "Create New Team" page as indicated by the text in the button.

This button will close the current page and open the "Manage Teams" page as indicated by the text in the button.

This button will close the current page and open the "Record Progress" page as indicated by the text in the button.

This button will close the entire application. As indicated by the "Exit" text.

Figure 1. Home Page

This is the main menu/home page and it is the first page that the user will see when they open the application. It simply provides navigation to the other pages via buttons.

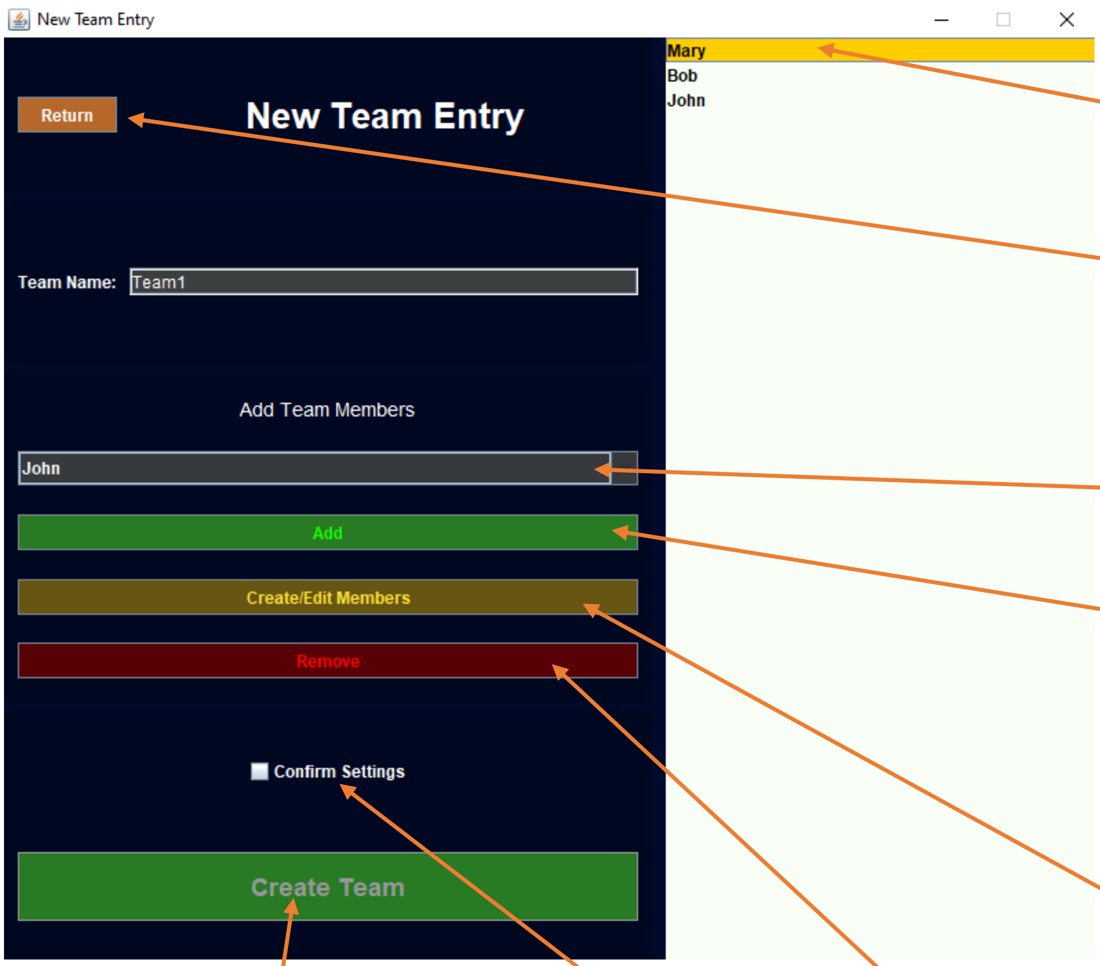


Figure 2. Create Team Page

This is a JList that stores all the members added by the user. Highlighted members are seen in yellow.

When clicked, the return button will redirect the user back to the Home page.

Combo box containing a drop-down list of all existing members.

User selects a member from the combo box they wish to add to the team and click the “Add” button. This’ll then move that member to the JList.

Opens a new page where the user can create new team members (See Figure 7).

User can remove members added to the team by selecting them in the JList and clicking the “Remove” button.

This button be disabled by default when the page opens and will be enabled after the checkbox is ticked and there are no errors. This button will extract all the info from the JList and the Team Title text field and store in the relevant .txt file.

The code in this checkbox contain error prevention. When a user selects the checkbox then it will automatically ensure all necessary fields have been filled out correctly. If they haven’t, an error message will be displayed. (See Figure 3).

In this page, the user is able to create a team and fill it with existing members. The user can also open a new window to create members. Once a team is created, the team information is stored in a .txt file.

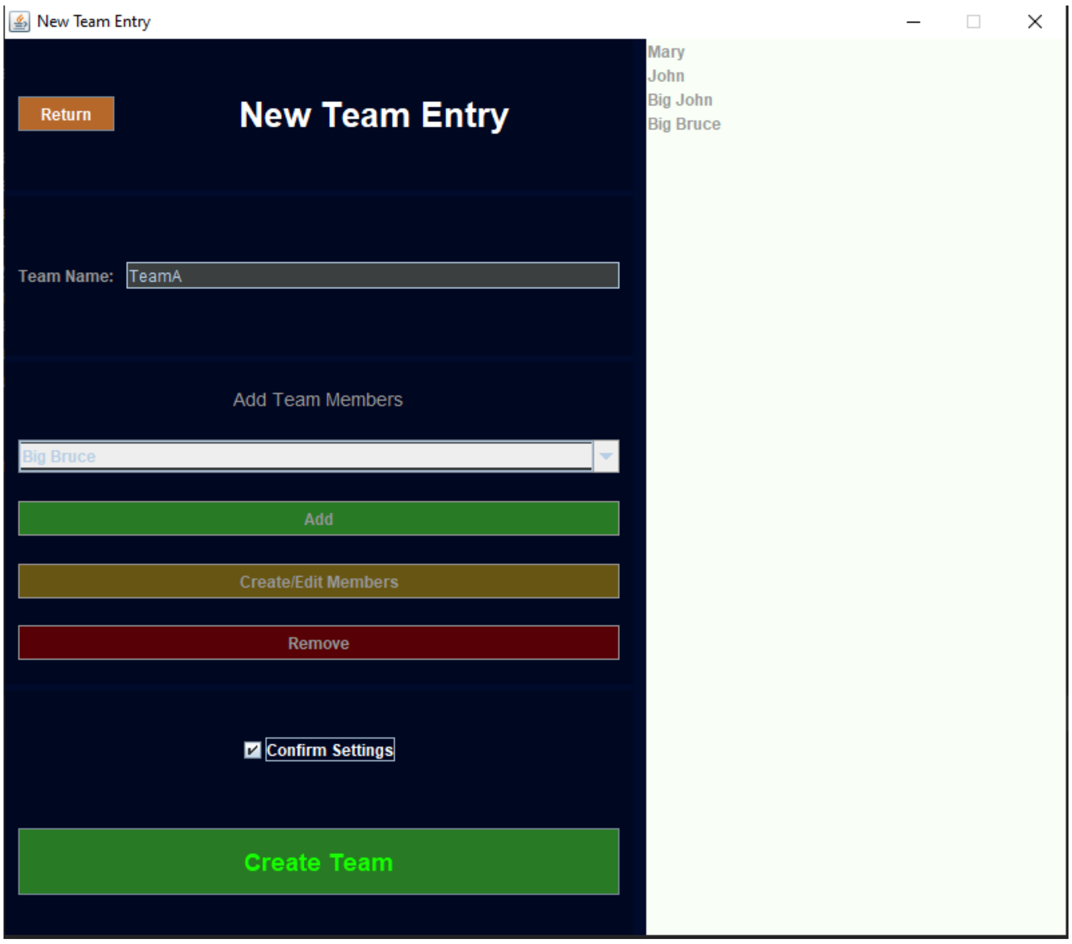
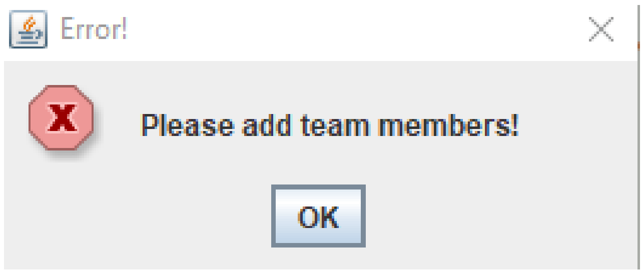


Figure 3. Create Team Page – After Checkbox Ticked

This is what the screen will look like when the “Confirm Settings” checkbox is ticked and there are no errors found. The “Create Team” button will be enabled whereas, the other buttons will be disabled to prevent the user from making any further edits unless they untick the checkbox again.

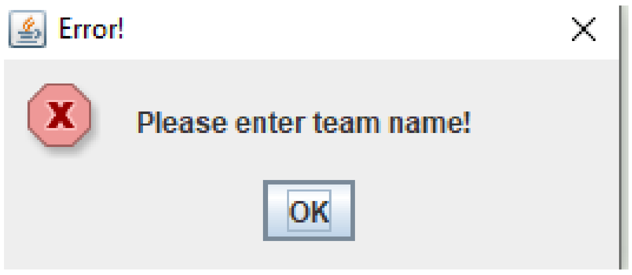


Error!

Please add team members!

Figure 4. Create Team Page – No Team Members Added

This error dialog message will appear if a user has clicked on the checkbox to confirm settings without having added any team members to the team.

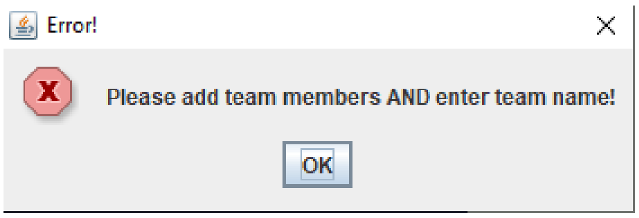


Error!

Please enter team name!

Figure 5. Create Team Page – No Team Title Error

This error dialog message will appear if a user has clicked on the checkbox to confirm settings without having entered the team title into the text field.

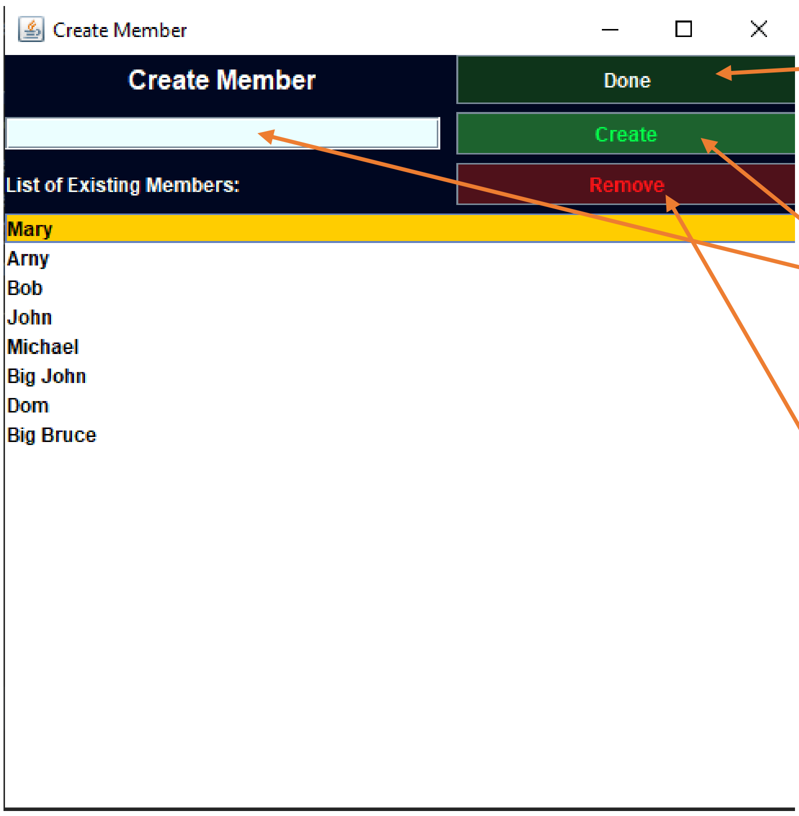


Error!

Please add team members AND enter team name!

Figure 6. Create Team Page – No Team Members Added AND No Team Title Error

This error dialog message will appear if a user has clicked on the checkbox to confirm settings without having added any team members to the team AND not having entered the team title into the text field.



Mary
Arny
Bob
John
Michael
Big John
Dom
Big Bruce

This button acts as a return button. Once clicked it will take the user back to the “Create a Team” Page and update that page with any changes made on this page.

Once the user enters a name into the text field underneath “Create Member”, they will need to click this button add that person to the list of existing members.

The user can select a member from the JList (selected members are highlighted in yellow) and click the “Remove” button to remove them from the list of existing members.

Figure 7. Create/Remove Team Members Page

In this page, the user can add new members or remove existing ones. All members added here can be selected in the “Create Team” screen.

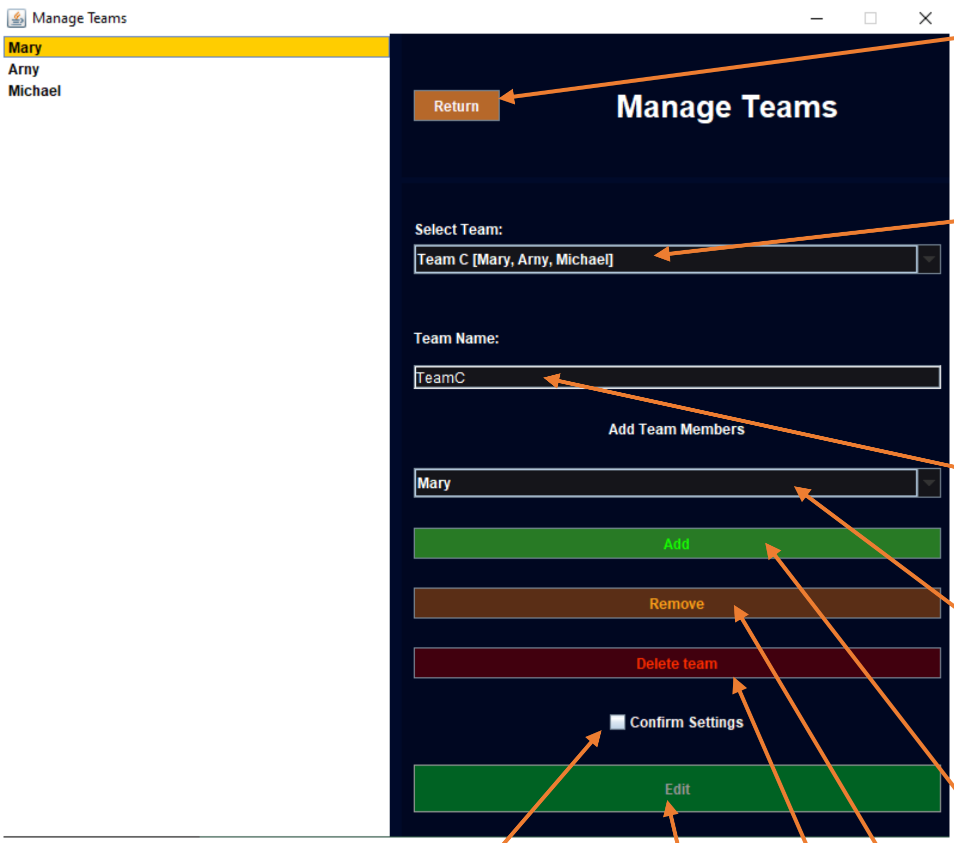


Figure 8. Manage Teams Page

When clicked, the return button will redirect the user back to the Home page.

The user can select from existing teams. Once the user selects a team, the “Team Name” and the team members JList will be auto filled in with the relevant team information.

The user is able to change the team name by entering a new team name for a team that has been already loaded into the window.

Furthermore, the user can also add existing members to that team by selecting a member from the drop-down combo box and clicking the “Add” button.

Once a member has been added, they will appear in the JList. The user can select a member and use the “Remove” button to remove them from the team. Selected members appear highlighted in yellow.

The user can also completely delete an entire time by selecting a team and clicking the “Delete Team” button.

The code in this checkbox contain error prevention. When a user selects the checkbox then it will automatically ensure all necessary fields have been filled out correctly. If they haven’t, an error message will be displayed.

This button be disabled by default when the page opens and will be enabled after the checkbox is ticked and there are no errors. This button will amend any changes made to the team information in this page to the relevant .txt files.

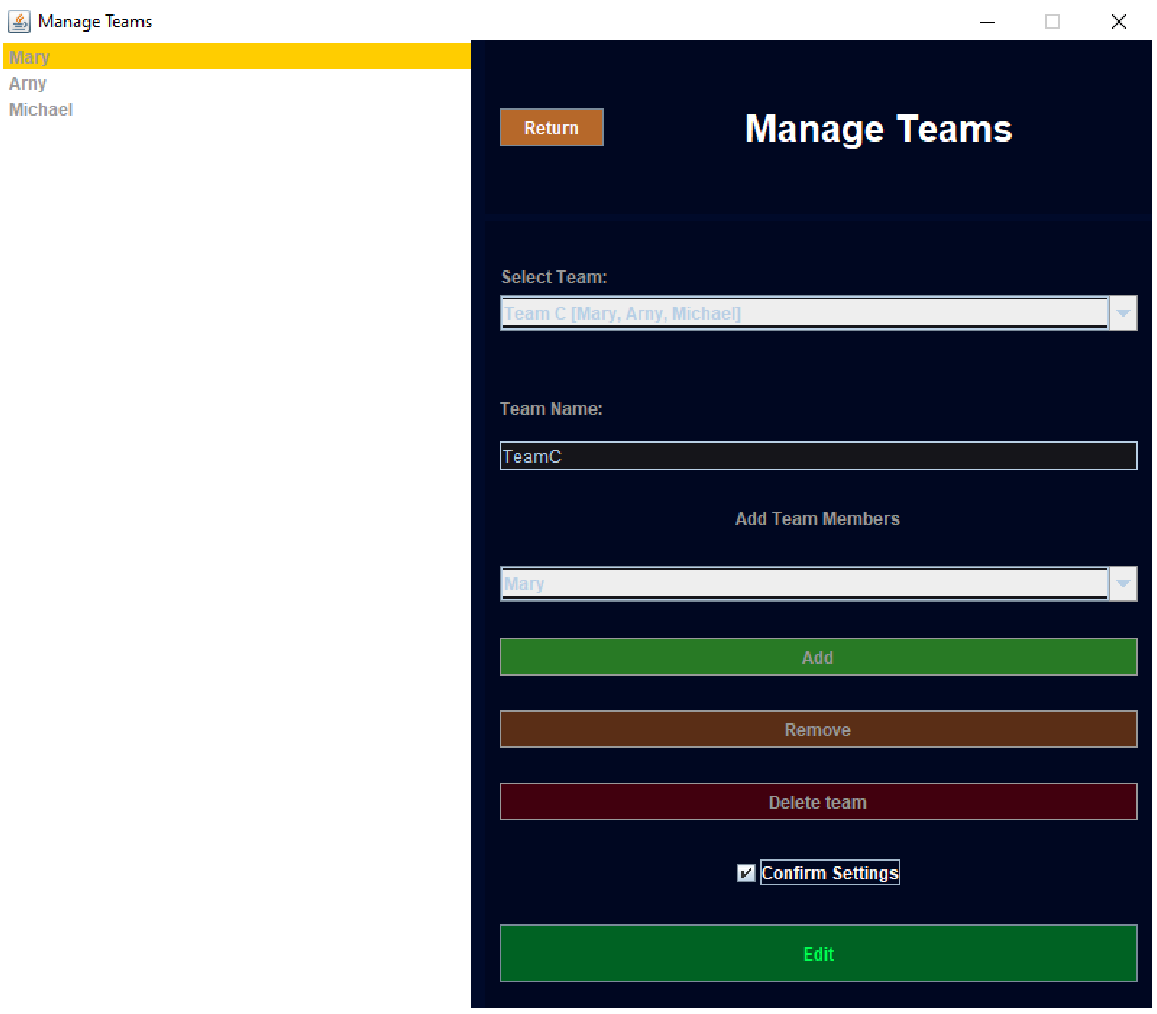
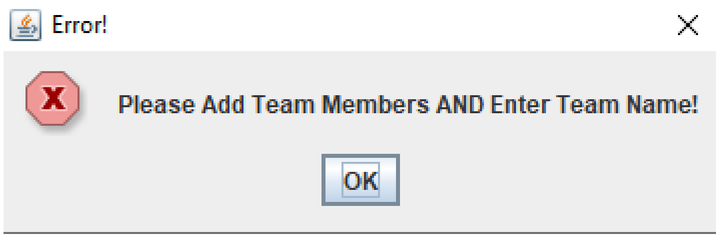


Figure 9. Manage Teams Page – After Checkbox Ticked

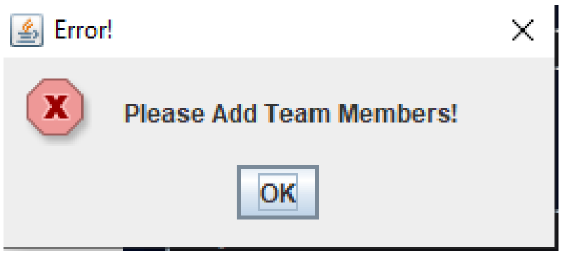
This is what the screen will look like when the “Confirm Settings” checkbox is ticked and there are no errors found. The “Edit” button will be enabled whereas, the other buttons will be disabled to prevent the user from making any further edits unless they untick the checkbox again.


Error!

Please Add Team Members AND Enter Team Name!

This error dialog message will appear if a user has clicked on the checkbox to confirm settings without having selected a team basically. As the JList containing the members and the Team Name is empty.

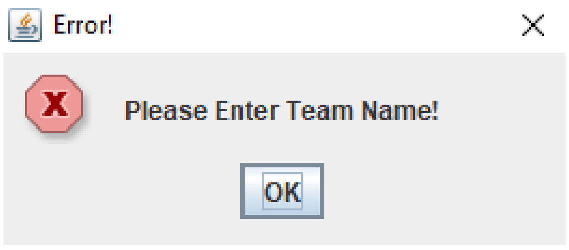
Figure 10. Manage Teams Page – No Team Selected Error


Error!

Please Add Team Members!

This error dialog message will appear if a user has clicked on the checkbox to confirm settings whilst having deleted all members in a team and not added any new ones.

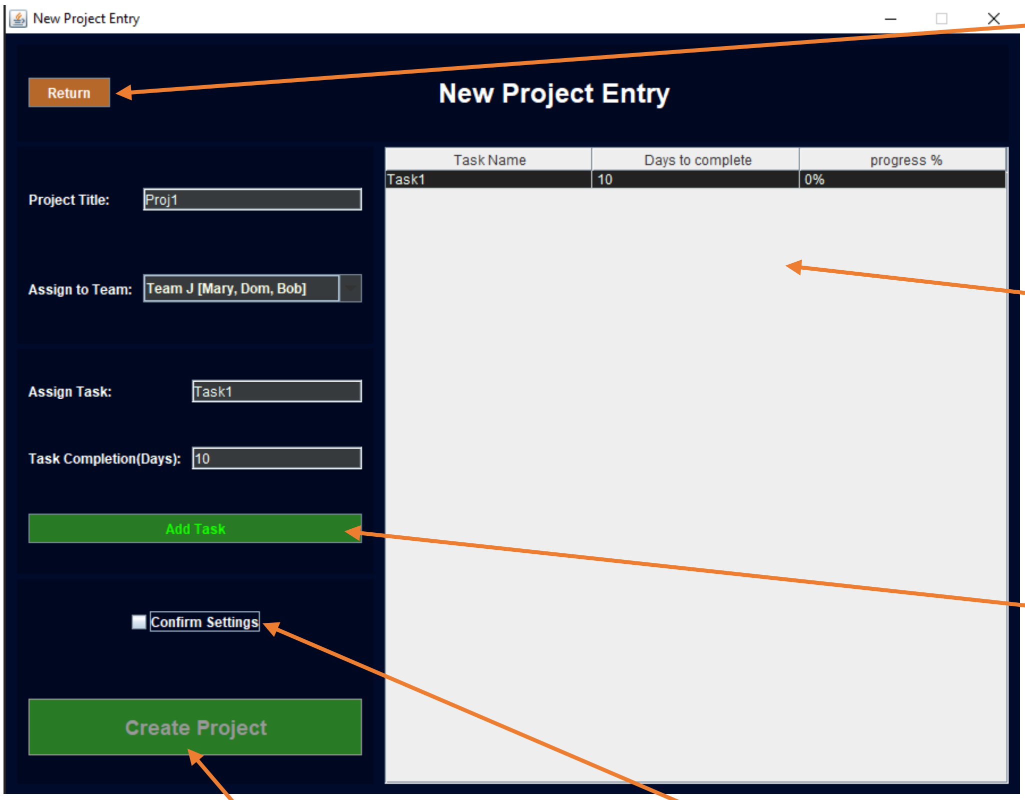
Figure 11. Manage Teams Page – No Team Members in Team


Error!

Please Enter Team Name!

This error dialog message will appear if a user has clicked on the checkbox to confirm settings whilst having entered a team name.

Figure 12. Manage Teams Page – No Team Members in Team



Task Name	Days to complete	progress %
Task1	10	0%

Figure 13. Create New Project Page.

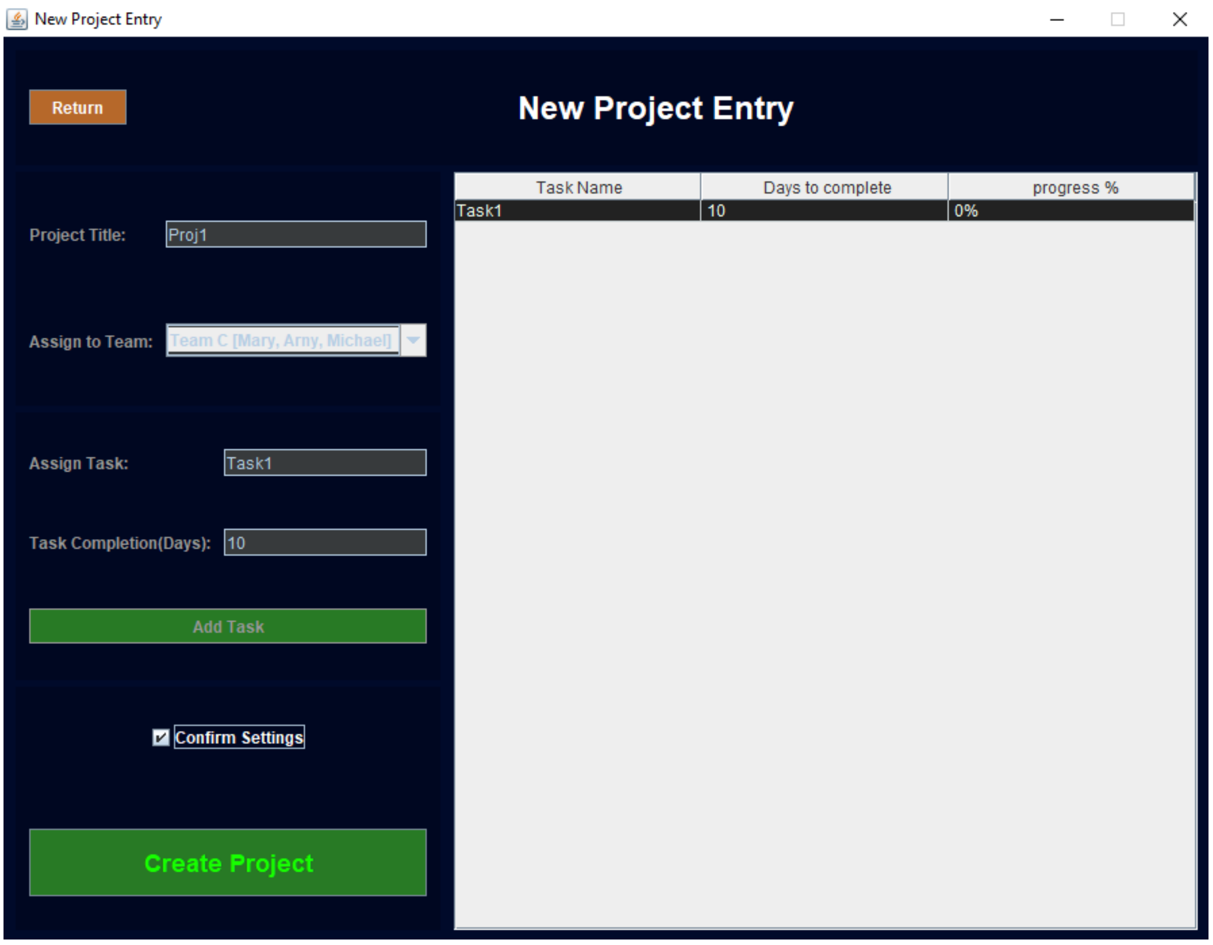
When clicked, the return button will redirect the user back to the Home page.

The JTable stores tasks that have been added by the user. The "progress" will automatically be set to 0% as the task has just been created. This can be edited in the "Record Progress" Page.

The user must input a task name and how long it will approximately take for this task to be completed (measured in days). To add this task to the project, the user must click the "Add Task" button and it will then be added to the JTable of tasks for that project.

The code in this checkbox contain error prevention. When a user selects the checkbox then it will automatically ensure all necessary fields have been filled out correctly. If they haven't, an error message will be displayed.

This button be disabled by default when the page opens and will be enabled after the checkbox is ticked and there are no errors. This button will extract the information such as the project title, the team assigned, the list of tasks in the JTable as well as their completion time and progress and store this information into the relevant .txt files.



Project Title: Proj1

Assign to Team: Team C [Mary, Arny, Michael]

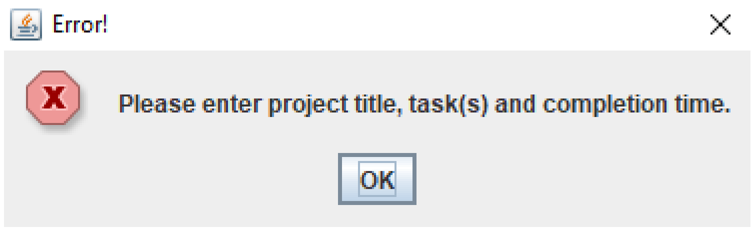
Assign Task: Task1

Task Completion(Days): 10

Task Name	Days to complete	progress %
Task1	10	0%

Figure 14. Create New Project – After Checkbox Ticked

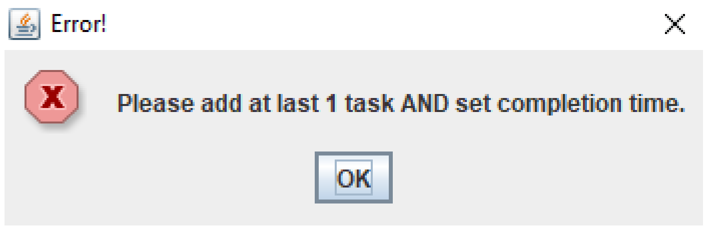
This is what the screen will look like when the “Confirm Settings” checkbox is ticked and there are no errors found. The “Create Project” button will be enabled whereas, the other buttons will be disabled to prevent the user from making any further edits unless they untick the checkbox again.


Error!

Please enter project title, task(s) and completion time.

Figure 15. Create New Project – No Project Title, No Tasks and Completion Time Entered.

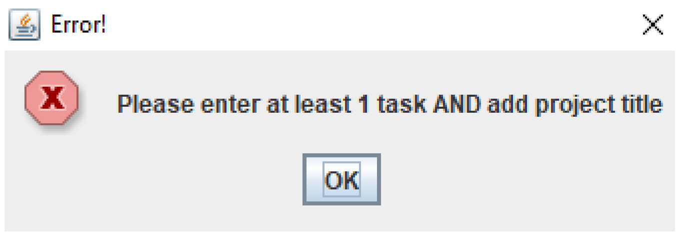
This error dialog message will appear if a user has clicked on the checkbox to confirm settings without having entered a team title or any tasks or their completion time.


Error!

Please add at last 1 task AND set completion time.

Figure 16. Create New Project – No Tasks and Completion Time Entered.

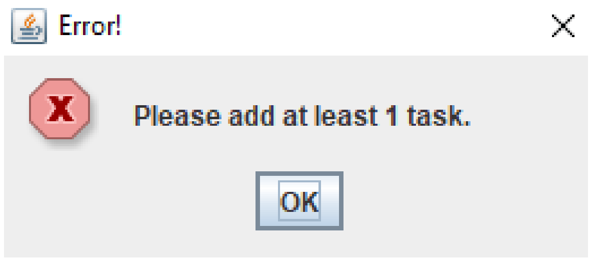
This error dialog message will appear if a user has clicked on the checkbox to confirm settings without having entered any tasks or their completion time.


Error!

Please enter at least 1 task AND add project title

Figure 17. Create New Project – No Project Title and No Tasks Entered.

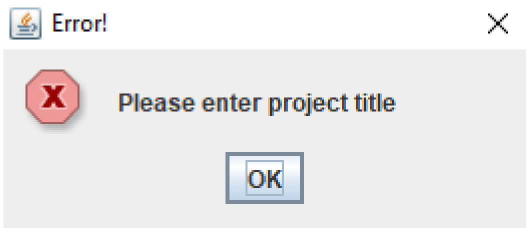
This error dialog message will appear if a user has clicked on the checkbox to confirm settings without having entered any tasks or a project title but has entered a completion time.


Error!

Please add at least 1 task.

Figure 18. Create New Project – No Tasks Added.

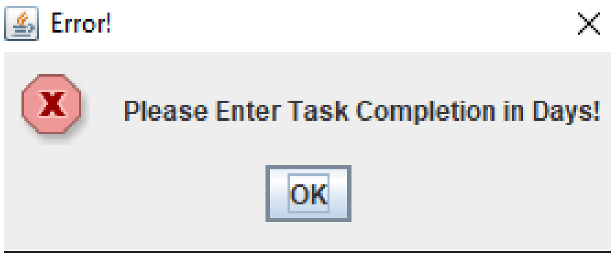
This error dialog message will appear if a user has clicked on the checkbox to confirm settings without having added any tasks, leaving the JTable empty.


Error!

Please enter project title

This error dialog message will appear if a user has clicked on the checkbox to confirm settings without having entered a project title.

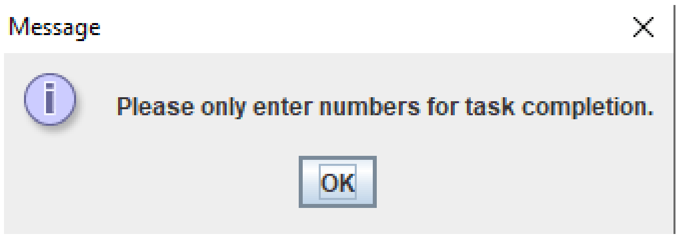
Figure 19. Create New Project – No Project Title Entered


Error!

Please Enter Task Completion in Days!

This error dialog message will appear if a user has clicked on the checkbox to confirm settings without having entered the approximate completion time of a task.

Figure 20. Create New Project – No Task Completion Time Entered


Message

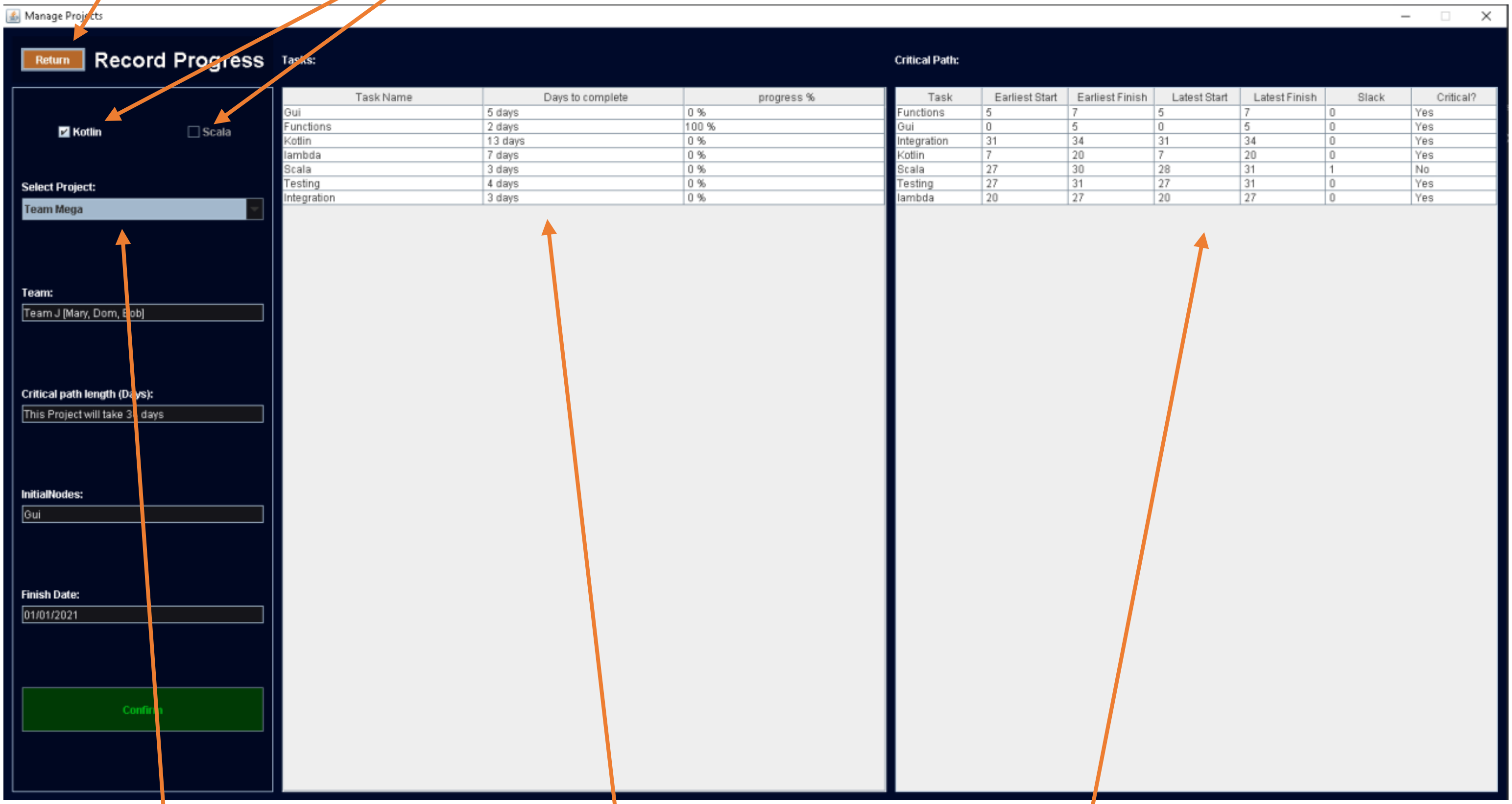
Please only enter numbers for task completion.

This error dialog message will appear if a user has clicked on the checkbox to confirm settings whilst having entered letters instead of numbers in the task completion time text field.

Figure 21. Create New Project – Letters Entered into Task Completion Text Field

When clicked, the return button will redirect the user back to the Home page.

The user must first select which critical path they want to active before any of options in the panel become available. Once one checkbox is ticked, the other will become unavailable until the currently selected check box is unticked. In this example, Kotlin is being used.



[✓ Kotlin] Scala

Select Project: Team Mega

Team: Team J [Mary, Dom, Bob]

Critical path length (Days): This Project will take 3[illegible] days

InitialNodes: Gui

Finish Date: 01/01/2021

Task Name	Days to complete	progress %
Gui	5 days	0 %
Functions	2 days	100 %
Kotlin	13 days	0 %
lambda	7 days	0 %
Scala	3 days	0 %
Testing	4 days	0 %
Integration	3 days	0 %

Task	Earliest Start	Earliest Finish	Latest Start	Latest Finish	Slack	Critical?
Functions	5	7	5	7	0	Yes
Gui	0	5	0	5	0	Yes
Integration	31	34	31	34	0	Yes
Kotlin	7	20	7	20	0	Yes
Scala	27	30	28	31	1	No
Testing	27	31	27	31	0	Yes
lambda	20	27	20	27	0	Yes

Figure 22. Record Progress Page

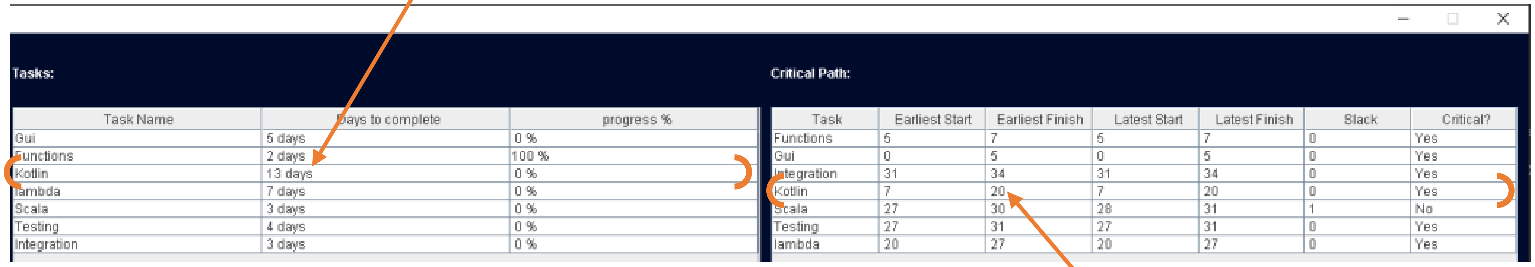
The user can select a project and the other text fields in the panel such as: team, critical path length (days), initial nodes and finish date will all be auto filled by the critical path algorithm.

When a project is selected, all of the tasks in the project will be displayed in the “Tasks” JTable. The user can make changes to the “Tasks” JTable on the left. They can change the name of the tasks, the time to complete and the progress.

If the user changes the “Days to complete” for a task, then the critical path algorithm will adjust its out accordingly to what has now been entered. By changing the progress or the task name, this will have no impact on the output of the critical path.

The contents of the critical path JTable cannot be directed modified by the user like they can be with the task JTable on the left.

Currently, the time to complete for the "Kotlin" task 13 days.



Tasks:

Task Name	Days to complete	progress %
Gui	5 days	0 %
Functions	2 days	100 %
Kotlin	13 days	0 %
lambda	7 days	0 %
Scala	3 days	0 %
Testing	4 days	0 %
Integration	3 days	0 %

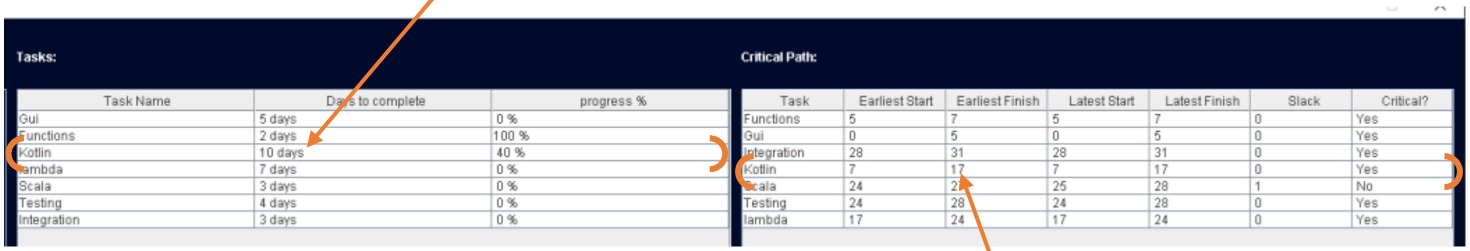
Critical Path:

Task	Earliest Start	Earliest Finish	Latest Start	Latest Finish	Slack	Critical?
Functions	5	7	5	7	0	Yes
Gui	0	5	0	5	0	Yes
Integration	31	34	31	34	0	Yes
Kotlin	7	20	7	20	0	Yes
Scala	27	30	28	31	1	No
Testing	27	31	27	31	0	Yes
lambda	20	27	20	27	0	Yes

Figure 23. Record Progress Page – Tasks and Critical Path Output

This is the output of the critical path based on the approximation of the task taking 13 days to complete. For example, the Earliest finish is 20 days.

But if the user modifies the days needed to complete the Kotlin task from 13 days to 10, then the critical path will adjust to this change automatically and display new information in its table.



Tasks:

Task Name	Days to complete	progress %
Gui	5 days	0 %
Functions	2 days	100 %
Kotlin	10 days	40 %
lambda	7 days	0 %
Scala	3 days	0 %
Testing	4 days	0 %
Integration	3 days	0 %

Critical Path:

Task	Earliest Start	Earliest Finish	Latest Start	Latest Finish	Slack	Critical?
Functions	5	7	5	7	0	Yes
Gui	0	5	0	5	0	Yes
Integration	28	31	28	31	0	Yes
Kotlin	7	17	7	17	0	Yes
Scala	24	27	25	28	1	No
Testing	24	28	24	28	0	Yes
lambda	17	24	17	24	0	Yes

Figure 24. Record Progress Page - Tasks and Critical Path Output – With Changes Made to Kotlin Task.

The earliest finish is now no longer 20 days, it has dropped down to 17 days due to the decrease in the days needed to complete the project.

Section 4

CPHandling.kt | Language - Kotlin

```
import java.io.BufferedReader
import java.io.File

object CPHandling{
  public var TasksTogether:HashSet<Task> = CpHandle()


    fun CpHandle(): HashSet<Task> {
    val fileName1 = "CriticalPath.txt"
    val file1 = File(fileName1)

    file1.printWriter().use { out ->
    //rewrite txt file
        out.write("Task, Earliest Start , Earliest Finish , Latest 
Start, Latest Finish, Slack, Critical?")
    }
    val fileName2 = "InitialNodes.txt"
    val file2 = File(fileName2)

    file2.printWriter().use { out ->
        //rewrite txt file
        out.write("")
    }
    val fileName3 = "CriticalPathDays.txt"
    val file3 = File(fileName3)

    file3.printWriter().use { out ->
        //rewrite txt file
        out.write("")
    }
    //Read txt file
    val bufferedReader: BufferedReader = 
File("Tasks.txt").bufferedReader()
    val inputString = bufferedReader.use { it.readText() }
    val fileName :String = "Tasks.txt"
    var i :Int = -1
    File(fileName).readLines().forEach {
        i++
    }
        //Make the CriticalPath Node/Nodes
    if(i == 1){
        val befsplit = inputString.replace("%", "/")
        val split: List<String> = befsplit.split("/")
        val int1 = split[2].replace("days", "").replace(" ", 
"").toInt()
        val start = Task(split[1], int1)
        TasksTogether = hashSetOf(start)


    }
    if(i == 2){
        val befsplit = inputString.replace("%", "/")
        val split: List<String> = befsplit.split("/")
        val int1 = split[2].replace("days", "").replace(" ",
"").toInt()
        val int2 = split[5].replace("days", "").replace(" ",
"").toInt()
        val end = Task(split[4], int2)
        val start = Task(split[1], int1, end)

        TasksTogether = hashSetOf(end, start)

    }
    if(i == 3){
        val befsplit = inputString.replace("%", "/")
        val split: List<String> = befsplit.split("/")
        val int1 = split[2].replace("days", "").replace(" ",
"").toInt()
        val int2 = split[5].replace("days", "").replace(" ",
"").toInt()
        val int3 = split[8].replace("days", "").replace(" ",
"").toInt()
        val end = Task(split[7], int3)
        val A = Task(split[4], int2, end)
        val start = Task(split[1], int1, A)

        TasksTogether = hashSetOf(end, A, start)


    }
    if(i == 4){
        val befsplit = inputString.replace("%", "/")
        val split: List<String> = befsplit.split("/")
        val int1 = split[2].replace("days", "").replace(" ",
"").toInt()
        val int2 = split[5].replace("days", "").replace(" ",
"").toInt()
        val int3 = split[8].replace("days", "").replace(" ",
"").toInt()
        val int4 = split[11].replace("days", "").replace(" ",
"").toInt()
        val end = Task(split[10], int4)
        val A = Task(split[7], int3, end)
        val F = Task(split[4], int2, A)
        val start = Task(split[1], int1, F)

     TasksTogether = hashSetOf(end, A, F, start)

    }
    if(i == 5){
        val befsplit = inputString.replace("%", "/")
        val split: List<String> = befsplit.split("/")
        val int1 = split[2].replace("days", "").replace(" ",
"").toInt()
```

```
        val int2 = split[5].replace("days", "").replace(" ",
"").toInt()
        val int3 = split[8].replace("days", "").replace(" ",
"").toInt()
        val int4 = split[11].replace("days", "").replace(" ",
"").toInt()
        val int5 = split[14].replace("days", "").replace(" ",
"").toInt()
        val end = Task(split[13], int5)
        val A = Task(split[10], int4, end)
        val F = Task(split[7], int3, A)
        val Q = Task(split[4], int2, F, A)
        val start = Task(split[1], int1, Q)

       TasksTogether = hashSetOf(end, A, F, Q, start)


    }
    if(i == 6){
        val befsplit = inputString.replace("%", "/")
        val split: List<String> = befsplit.split("/")
        val int1 = split[2].replace("days", "").replace(" ",
"").toInt()
        val int2 = split[5].replace("days", "").replace(" ",
"").toInt()
        val int3 = split[8].replace("days", "").replace(" ",
"").toInt()
        val int4 = split[11].replace("days", "").replace(" ",
"").toInt()
        val int5 = split[14].replace("days", "").replace(" ",
"").toInt()
        val int6 = split[17].replace("days", "").replace(" ",
"").toInt()
        val end = Task(split[16], int6)
        val F = Task(split[13], int5, end)
        val A = Task(split[10], int4, end)
        val X = Task(split[7], int3, F, A)
        val Q = Task(split[4], int2, A, X)
        val start = Task(split[1], int1, Q)
        TasksTogether = hashSetOf(end, F, A, X, Q, start)

    }
    if(i == 7){
        val befsplit = inputString.replace("%", "/")
        val split: List<String> = befsplit.split("/")
        val int1 = split[2].replace("days", "").replace(" ",
"").toInt()
        val int2 = split[5].replace("days", "").replace(" ",
"").toInt()
        val int3 = split[8].replace("days", "").replace(" ",
"").toInt()
        val int4 = split[11].replace("days", "").replace(" ",
"").toInt()
        val int5 = split[14].replace("days", "").replace(" ",
"").toInt()
```

```
            val int6 = split[17].replace("days", "").replace(" ",
"").toInt()
            val int7 = split[20].replace("days", "").replace(" ",
"").toInt()
            val end = Task(split[19], int7)
            val Z = Task(split[16], int6, end)
            val F = Task(split[13], int5, end)
            val A = Task(split[10], int4, F, Z)
            val X = Task(split[7], int3, F, A, Z)
            val Q = Task(split[4], int2, A, X, F)
            val start = Task(split[1], int1, Q)
           TasksTogether = hashSetOf(end, Z, F, A, X, Q, start)

        }
        if(i == 8){
            val befsplit = inputString.replace("%", "/")
            val split: List<String> = befsplit.split("/")
            val int1 = split[2].replace("days", "").replace(" ",
"").toInt()
            val int2 = split[5].replace("days", "").replace(" ",
"").toInt()
            val int3 = split[8].replace("days", "").replace(" ",
"").toInt()
            val int4 = split[11].replace("days", "").replace(" ",
"").toInt()
            val int5 = split[14].replace("days", "").replace(" ",
"").toInt()
            val int6 = split[17].replace("days", "").replace(" ",
"").toInt()
            val int7 = split[20].replace("days", "").replace(" ",
"").toInt()
            val int8 = split[23].replace("days", "").replace(" ",
"").toInt()
            val end = Task(split[22], int8)
            val G = Task(split[19], int7, end)
            val Z = Task(split[16], int6, end)
            val F = Task(split[13], int5, Z, G)
            val A = Task(split[10], int4, F, Z)
            val X = Task(split[7], int3, F, A, Z, G)
            val Q = Task(split[4], int2, A, X, F, Z)
            val start = Task(split[1], int1, Q)
            TasksTogether = hashSetOf(end, G, Z, F, A, X, Q, start)

        }
        if(i == 9){
            val befsplit = inputString.replace("%", "/")
            val split: List<String> = befsplit.split("/")
            val int1 = split[2].replace("days", "").replace(" ",
"").toInt()
            val int2 = split[5].replace("days", "").replace(" ",
"").toInt()
            val int3 = split[8].replace("days", "").replace(" ",
"").toInt()
            val int4 = split[11].replace("days", "").replace(" ",
"").toInt()
```

```
        val int5 = split[14].replace("days", "").replace(" ",
"").toInt()
        val int6 = split[17].replace("days", "").replace(" ",
"").toInt()
        val int7 = split[20].replace("days", "").replace(" ",
"").toInt()
        val int8 = split[23].replace("days", "").replace(" ",
"").toInt()
        val int9 = split[26].replace("days", "").replace(" ",
"").toInt()
        val end = Task(split[25], int9)
        val H = Task(split[22], int8, end)
        val G = Task(split[19], int7, end)
        val Z = Task(split[16], int6, G, H)
        val F = Task(split[13], int5, Z, G, H)
        val A = Task(split[10], int4, F, Z, G)
        val X = Task(split[7], int3, F, A, Z)
        val Q = Task(split[4], int2, A, X, F)
        val start = Task(split[1], int1, Q)
       TasksTogether = hashSetOf(end, H, G, Z, F, A, X, Q, start)

    }
    if(i == 10){
        val befsplit = inputString.replace("%", "/")
        val split: List<String> = befsplit.split("/")
        val int1 = split[2].replace("days", "").replace(" ",
"").toInt()
        val int2 = split[5].replace("days", "").replace(" ",
"").toInt()
        val int3 = split[8].replace("days", "").replace(" ",
"").toInt()
        val int4 = split[11].replace("days", "").replace(" ",
"").toInt()
        val int5 = split[14].replace("days", "").replace(" ",
"").toInt()
        val int6 = split[17].replace("days", "").replace(" ",
"").toInt()
        val int7 = split[20].replace("days", "").replace(" ",
"").toInt()
        val int8 = split[23].replace("days", "").replace(" ",
"").toInt()
        val int9 = split[26].replace("days", "").replace(" ",
"").toInt()
        val int10 = split[29].replace("days", "").replace(" ",
"").toInt()
        val end = Task(split[28], int10)
        val D = Task(split[25], int9, end)
        val H = Task(split[22], int8, end)
        val G = Task(split[19], int7, D, H)
        val Z = Task(split[16], int6, G, H, D)
        val F = Task(split[13], int5, Z, G, H)
        val A = Task(split[10], int4, F, Z, G)
        val X = Task(split[7], int3, F, A, Z)
        val Q = Task(split[4], int2, A, X, F)
        val start = Task(split[1], int1, Q)
```

```
      TasksTogether = hashSetOf(end, D, H, G, Z, F, A, X, Q, start)


    }

    return TasksTogether
}}
```

CriticalPath.scala | Language - Scala

```
import com.sun.deploy.trace.Trace.println

import scala.collection.convert.ImplicitConversions.`collection
asJava`
import scala.collection.immutable.HashSet
import scala.jdk.CollectionConverters._



class CriticalPath{


 def calcCritPath(tasks: HashSet[Task]) = {
    var completed = HashSet[Task]()
    val remaining = tasks.map(x => x)

    while (remaining.nonEmpty) {
      var progress = false

      val it = remaining.iterator()
      while (it.hasNext()) {
        val task = it.next()
        if (completed.containsAll(task.dependencies)) {
          val criticalList = task.dependencies.map { it =>
it.criticalCost }
          val critical = if (criticalList.isEmpty) 0 else
criticalList.max
          task.criticalCost = critical + task.cost
          completed.add(task)
          it.remove()
          progress = true
        }
      }
      if (!progress) throw new Exception("Cyclic dependency,
algorithm stopped")
    }

    val maxCostCheck = tasks.map(it => it.criticalCost)
    val maxCost = if (maxCostCheck.isEmpty) 0 else maxCostCheck.max
    println(s"Critical Path lenght is (Cost): ${maxCost}")

    calculateLatest(tasks, maxCost)
    calculateEarly(tasks)
    completed
 }

 def calculateLatest(tasks: HashSet[Task], maxCost: Int): Unit =
tasks.foreach(it => it.setLatest(maxCost))

 def calculateEarly(tasks: HashSet[Task]) = tasks.foreach(it => {
   it.earlyStart = 0
   it.earlyFinish = it.cost
   it.setEarlyForDependecies()
 })

 def initials(tasks: HashSet[Task]) = {
   val dependencies = tasks.map(it => it.dependencies)
   val results = tasks.filter(x => dependencies.contains(x))
   results
 }

 case class Task(
                      var name: String,
                      var cost: Int,
                      var dependencies: HashSet[Task]
                     ) {
   var criticalCost = 0
   var earlyStart = 0
   var earlyFinish = -1
   var latestStart = 0
   var latestFinish = 0

   var dependency = dependencies.map(x => x)

   def setLatest(maxCost: Int): Unit = {
     latestStart = maxCost - criticalCost
     latestFinish = latestStart + cost
   }

   def setEarlyForDependecies(): Unit = {
     val completionTime = earlyFinish
     dependency.foreach(x => {
       if (completionTime >= x.earlyStart) {
         x.earlyStart = completionTime
         x.earlyFinish = completionTime + x.cost
       }
       x.setEarlyForDependecies()
     })
   }


   def isCritical() = earlyStart == latestStart

   def isDependent(t: Task): Boolean = dependency.contains(t) ||
dependency.exists { it => it.isDependent(t) }


}}
```

CriticalPath.kt | Language - Kotlin

```
import java.io.*


fun main() {
   CPHandling.CpHandle()
   calcCPath(CPHandling.TasksTogether)
   PrintRes(CPHandling.TasksTogether)
}

fun calcCPath(tasks: Collection<Task>) {
   val completed = hashSetOf<Task>()
   val remaining = tasks.toHashSet()


   while (remaining.isNotEmpty()) {
       var progress = false

       // find task to calculate
       val it = remaining.iterator()
       while (it.hasNext()) {
           val task = it.next()
           if (completed.containsAll(task.dependencies)) {
               val critical = task.dependencies.map {
it.criticalCost }.maxOrNull() ?: 0
               task.criticalCost = critical + task.cost
               completed.add(task)
               it.remove()
               progress = true
           }
       }
       if (!progress) throw RuntimeException("Cyclic dependency,
algorithm stopped!")
   }

   // get the cost
   val maxCost = tasks.map { it.criticalCost }.maxOrNull() ?: -1
  val test = "$maxCost"
   val fileName = "CriticalPathDays.txt"
   val myfile = File(fileName)
   myfile.printWriter().use { out ->
       out.write(test)

   }
   calcLatest(tasks, maxCost)
   calcLatest(tasks)
}

fun calcLatest(tasks: Collection<Task>, maxCost: Int) =
tasks.forEach { it.setLatest(maxCost) }

fun calcLatest(tasks: Collection<Task>) = initials(tasks).forEach {
   it.earlyStart = 0
   it.earlyFinish = it.cost
   it.setEarlyForDependencies()

}

fun initials(tasks: Collection<Task>): Collection<Task> {
   val dependencies = tasks.flatMap { it.dependencies }.toSet()
   return tasks.filter { it !in dependencies }.also {
       val test = ("${it.joinToString { node -> node.name }}")
       val fileName = "InitialNodes.txt"
       val myfile = File(fileName)
       val inputAsString = myfile.bufferedReader().use {
it.readText() }
       myfile.printWriter().use { out ->
           out.write(inputAsString)
           out.write(test)

       }
   }
}

fun PrintRes(tasks: Collection<Task>) {
   tasks.sortedWith { o1, o2 -> o1.name.compareTo(o2.name) }.forEach
{
       try {
           val writer = FileWriter("CriticalPath.txt", true)
           writer.write("${it.name}/ ${it.earlyStart}/
${it.earlyFinish}/ ${it.latestStart}/ ${it.latestFinish}/
${it.latestStart - it.earlyStart}/ ${if (it.isCritical()) "Yes" else
"No"}")
           writer.close()
       } catch (e: IOException) {
           e.printStackTrace()
       }

   }
}




class Task(
   var name: String,
   var cost: Int,
   vararg dependencies: Task
) {
   var criticalCost = 0

   var earlyStart = 0

   var earlyFinish = -1

   var latestStart = 0

   var latestFinish = 0
```

```
    var dependencies = hashSetOf(*dependencies)

    fun setLatest(maxCost: Int) {
        latestStart = maxCost - criticalCost
        latestFinish = latestStart + cost
    }

    fun setEarlyForDependencies() {
        val completionTime = earlyFinish
        dependencies.forEach {
            if (completionTime >= it.earlyStart) {
                it.earlyStart = completionTime
                it.earlyFinish = completionTime + it.cost
            }
            it.setEarlyForDependencies()
        }
    }

    fun isCritical() = earlyStart == latestStart

    fun isDependent(t: Task): Boolean = dependencies.contains(t) ||
dependencies.any { it.isDependent(t) }
}
```

EditMembers.kt | Language - Kotlin

```
import java.io.FileWriter
import java.io.IOException

data class CreateMembers(var name: String = "Team") {

}
class MembersCreator() {
    fun createMember(name: String): CreateMembers {
        //Create Members

        try {
            //Write members into final members.txt file
            val writer = FileWriter("members.txt", true)
            writer.write(name + "\n")
            writer.close()
        } catch (e: IOException) {
            e.printStackTrace()
        }
        //Write members to tempmembers for editing
        try {
            val writer = FileWriter("TempMembers.txt", true)
            writer.write(name + "\n")
            writer.close()
        } catch (e: IOException) {
            e.printStackTrace()
        }

        return CreateMembers(name = name)
    }
}
```

Main.kt | Language - Kotlin

```
fun main(args: Array<String>) {
   val home = Home()
   home.setBounds(1500, 1500, 1200, 900)
   home.setLocationRelativeTo(null)
   home.isResizable = false
   home.isVisible = true
}
```

Teams.kt | Language - Kotlin

```
import java.io.File
import java.io.FileWriter
import java.io.PrintWriter
import javax.swing.DefaultListModel


data class Teams(var name: String = "Team", var members: String) {

}

class CreateHandler() {
   //Create Team
   fun createTeam(name: String, members: String):Teams{
       val fileName = "Teams.txt"
       val myfile = File(fileName)
       val inputAsString = myfile.bufferedReader().use {
it.readText() }
       myfile.printWriter().use { out ->
           out.write(inputAsString)
           out.write(name + " " + members + "\n")
       }

       return Teams(name = name, members = members)
   }

}
```

ManageTeams.kt | Language - Kotlin

```
import java.util.*

data class ManageTeam(var name: String = "Team", var members:
Array<String>) {

}
data class members(var members: Array<String>){

}
data class teamname(var teamname: String){
}

class ManageHandler() {
    //send from Team string to GUI
    fun sendlist(listing: String= "Team"):members{
        var list: String = listing
        var split = list.replace("[", ",").replace("]", "").replace("
", "")
        var everything = split.split(",".toRegex()).toTypedArray()
        var test = everything[0]
        var members = split.replace(test,
"").split(",".toRegex()).toTypedArray()

        return members(members)
    }
    //GetTeamName from JTextField
    fun getteamname(teamname: String = "Team A"):teamname{
            var list: String = teamname
            var split = list.replace("[", ",").replace("]",
"").replace(" ", "")
            var everything =
split.split(",".toRegex()).toTypedArray()
            var test = everything[0]

            return teamname(test)
        }
    }
```

Projects.kt | Language - Java

```
import java.io.*
import java.text.SimpleDateFormat
import java.util.*


fun read() {
    val inputStream: InputStream =
File("CriticalPathDays.txt").inputStream()
    val inputString = inputStream.bufferedReader().use {
it.readText() }

}
//Append the txt File so its reset
fun rewrite(){
    val fileName = "Tasks.txt"
    val myfile = File(fileName)

    myfile.printWriter().use { out ->

        out.write("Task Name, Days to complete, progress %" + "\n")
    }
}
```

```
data class Projects(var name: String, var team: String, var tasks:
String) {

}
data class Tasks(var tasks: String, var days: String) {

}
data class Progress(var tasks: String){

}

class ProjectsHandler() {
   //Add project
   fun addProject(name: String, team: String, tasks: String):
Projects {
        val formatter = SimpleDateFormat("yyyy-MM-dd")
        val date = Date(System.currentTimeMillis())
        val dateformate =(formatter.format(date))
        val fileName = "projects.txt"
        val myfile = File(fileName)
        val inputAsString = myfile.bufferedReader().use {
it.readText() }
        myfile.printWriter().use { out ->
            out.write(inputAsString)
            out.write(name + ";" + team + ";" + dateformate + ";" +
tasks + "\n")
        }

        return Projects(name = name, team = team, tasks = tasks)
    }
    //Add Tasks
    fun addTasks(tasks: String, days: String): Tasks {
        val fileName = "Tasks.txt"
        val myfile = File(fileName)
        val inputAsString = myfile.bufferedReader().use {
it.readText() }
        myfile.printWriter().use { out ->
            out.write(inputAsString)
            out.write(tasks + " / " + days + " / " + "0%" + "\n")
        }
        return Tasks(tasks = tasks, days = days)
    }
    //Add progress
    fun addprogress(tasks: String): Progress {
        val fileName = "Tasks.txt"
        val myfile = File(fileName)
        val inputAsString = myfile.bufferedReader().use {
it.readText() }
        myfile.printWriter().use { out ->
            out.write(inputAsString)
            out.write(tasks + "\n")
        }
        return Progress(tasks = tasks)
    }
    fun read(args: Array<String>) {
        val inputStream: InputStream =
File("CriticalPathDays.txt").inputStream()
        val inputString = inputStream.bufferedReader().use {
it.readText() }
    }

}
```

EditMembers.java | Language - Java

```
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.*;

public class EditMembers extends JFrame {
    private JTextField txtMemberName;
    private JButton createButton;
    private JButton doneButton;
    private JPanel CreateMember;
    private JButton removeButton;
    private JList TeamMembers;
    private MembersCreator createmembers;
    private CreateMembers members;


    DefaultListModel liss = new DefaultListModel();
    File inputFile = new File("members.txt");
    File tempFile = new File("TempMembers.txt");
    public EditMembers() {
        super("Create Member");
        this.setContentPane(this.CreateMember);
        this.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        this.pack();
        createmembers = new MembersCreator();
        createButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                members =
createmembers.createMember(txtMemberName.getText());
                members.toString();
                String line = txtMemberName.getText();
                liss.addElement(line);
                TeamMembers.setModel(liss);
                txtMemberName.setText("");

            }

        });
        String filePath2 = "members.txt";
        File file = new File(filePath2);

        try {
            BufferedReader br = new BufferedReader(new
FileReader(file));
            Object[] lines = br.lines().toArray();
            for(int i = 0; i < lines.length; i++){
                String line = lines[i].toString();
                liss.addElement(line);
                TeamMembers.setModel(liss);
            }
            br.close();
        } catch (FileNotFoundException ex) {

        } catch (IOException e) {
            e.printStackTrace();
        }
        doneButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                NewTeam nt = new NewTeam();
                nt.setVisible(true);
                nt.setBounds(1500,1500, 800 ,700);
                nt.setLocationRelativeTo(null);
                nt.setResizable(false);
                dispose();
            }
        });
        removeButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                int selectedIndex = TeamMembers.getSelectedIndex();
                String selected =
TeamMembers.getSelectedValue().toString();
                if (selectedIndex != -1) {
                    liss.remove(selectedIndex);
                }

                try {
                    BufferedReader reader = new BufferedReader(new
FileReader(inputFile));
                    BufferedWriter writer = new BufferedWriter(new
FileWriter(tempFile));

                    int lineToRemove = selectedIndex + 1;
                    String currentLine;
                    int count = 0;

                    while ((currentLine = reader.readLine()) != null)
{
                        count++;
                        if (count == lineToRemove) {
                            continue;
                        }
                        writer.write(currentLine +
System.getProperty("line.separator"));
                    }
                    writer.close();
                    reader.close();
                    inputFile.delete();
                    tempFile.renameTo(inputFile);

                } catch (IOException ioException) {
                    ioException.printStackTrace();
                }


            }});


    }

    public void main(String[] args) {
        EditMembers memberc = new EditMembers();
        memberc.setBounds(1500,1500, 1200 ,2000);
        memberc.setLocationRelativeTo(null);
        memberc.setResizable(true);
        memberc.setVisible(true);
    }
}
```

Home.java | Language - Java

```
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class Home extends JFrame {
    private JButton newProjectSideButton;
    private JButton manageTeamsSideButton;
    private JButton newTeamSideButton;
    private JButton manageProjectSideButton;
    private JButton createNewProjectBigButton;
    private JButton createNewTeamBIgButton;
    private JButton manageProjectsBigButton;
    private JButton manageTeamsBigButton;
    private JPanel MainPnl;
    private JPanel TopPnl;
    private JPanel MidPnl;
    private JButton exitButton;

    Home() {
        super("Home");
        this.setContentPane(this.MainPnl);
        this.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        this.pack();

        createNewProjectBigButton.addActionListener(new
ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                NewProject project = new NewProject();
                project.setVisible(true);
                project.setBounds(1500,1500, 900 ,700);
                project.setLocationRelativeTo(null);
                project.setResizable(false);
                dispose();
            }
        });
        createNewTeamBIgButton.addActionListener(new ActionListener()
{
            @Override
            public void actionPerformed(ActionEvent e) {
                NewTeam nt = new NewTeam();
                nt.setVisible(true);
                nt.setBounds(1500,1500, 800 ,700);
                nt.setLocationRelativeTo(null);
                nt.setResizable(false);
                dispose();
            }
        });

        manageProjectsBigButton.addActionListener(new
ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                RecordProgress mp = new RecordProgress();
                mp.setVisible(true);
                mp.setBounds(1500,1500, 1700 ,900);
                mp.setLocationRelativeTo(null);
                mp.setResizable(false);
                dispose();
            }
        });
        manageTeamsBigButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                ManageTeams mt = new ManageTeams();
                mt.setVisible(true);
                mt.setBounds(1500,1500, 800 ,700);
                mt.setLocationRelativeTo(null);
                mt.setResizable(false);
                dispose();
            }
        });


        exitButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                System.exit(0);
            }
        });
    }

    public static void main(String[] args) {
        Home home = new Home();
        home.setBounds(1500, 1500, 1000, 800);
        home.setLocationRelativeTo(null);
        home.setResizable(false);
        home.setVisible(true);
    }


}
```

NewProject.java | Language - Java

```
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.io.*;
import java.util.ArrayList;

public class NewProject extends JFrame {
    private JButton createProjectButton;
    private JTextField projTitle;
    private JComboBox assigntoTeam;
    private JTextField assignTask;
    private JButton addTaskButton;
    private JButton returnButton;
    private JPanel MainPnl;
    private JTable TableTasks;
    private JTextField txtDays;
    private JCheckBox confirmSettingsCheckBox;
    private JScrollPane taskTable;
    private JLabel lblProjTitle;
    private JLabel lblAssignTeama;
    private JLabel lblAssignTask;
    private JLabel lblTaskComp;
    private Projects projects;
    private Tasks tasks;
    private ProjectsHandler assignteamname;


    NewProject() {
        super("New Project Entry");
        this.setContentPane(this.MainPnl);
        this.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        this.pack();
        ArrayList<String> Tasks = new ArrayList<String>();
        assignteamname = new ProjectsHandler();
        String filePath = "Tasks.txt";
        String filePath2 = "Teams.txt";

        try {
            //Read from TXT file and add to the ComboBox
            File file = new File(filePath2);
            BufferedReader br = new BufferedReader(new
FileReader(file));
            Object[] lines = br.lines().toArray();

            for (int i = 0; i < lines.length; i++) {
                String line = lines[i].toString();
                assigntoTeam.addItem(line);
            }
            br.close();
        } catch (FileNotFoundException ex) {

        } catch (IOException e) {
            e.printStackTrace();
        }
        ProjectsKt.rewrite();

        returnButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                Home home = new Home();
                home.setBounds(1500, 1500, 1000, 800);
                home.setLocationRelativeTo(null);
                home.setResizable(false);
                home.setVisible(true);
                dispose();
            }
        });

        addTaskButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if (assignTask.getText().equals("")) {
                    JOptionPane optionPane = new JOptionPane("Please
Enter Task Name!", JOptionPane.ERROR_MESSAGE);
                    JDialog dialog =
optionPane.createDialog("Error!");
                    dialog.setAlwaysOnTop(true);
                    dialog.setVisible(true);
                }
                if (txtDays.getText().equals("")) {
                    JOptionPane optionPane = new JOptionPane("Please
Enter Task Completion in Days!", JOptionPane.ERROR_MESSAGE);
                    JDialog dialog =
optionPane.createDialog("Error!");
                    dialog.setAlwaysOnTop(true);
                    dialog.setVisible(true);
                } else {
                    String task = assignTask.getText() + " / " +
txtDays.getText() + " days / " + "0 %";
                    Tasks.add(task);
                    tasks =
assignteamname.addTasks(assignTask.getText(), txtDays.getText());
                    tasks.toString();
                    File file = new File(filePath);

                    try {
                        BufferedReader br = new BufferedReader(new
FileReader(file));
                        String firstLine = br.readLine().trim();
                        String[] columnsName = firstLine.split(",");
                        DefaultTableModel model = (DefaultTableModel)
TableTasks.getModel();
                        model.setColumnIdentifiers(columnsName);
                        model.setRowCount(0);
                        Object[] tableLines = br.lines().toArray();

                        for (int i = 0; i < tableLines.length; i++) {
                            String line =
tableLines[i].toString().trim();
                            String[] dataRow = line.split("/");
                            model.addRow(dataRow);
                        }


                    } catch (Exception ex) {

                    }
                }
            }
        });

        createProjectButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {

                createProjectButton.setEnabled(false);
                projTitle.setEnabled(true);
                lblProjTitle.setEnabled(true);
                assigntoTeam.setEnabled(true);
                lblAssignTeama.setEnabled(true);
                lblAssignTask.setEnabled(true);
                assignTask.setEnabled(true);
                lblTaskComp.setEnabled(true);
                txtDays.setEnabled(true);
                addTaskButton.setEnabled(true);
                taskTable.setEnabled(true);
                confirmSettingsCheckBox.setSelected(false);
                txtDays.setText("");

                StringBuffer sbTableData = new StringBuffer();
                for (int row = 0; row < TableTasks.getRowCount();
row++) {
                    for (int column = 0; column <
TableTasks.getColumnCount(); column++) {
                        sbTableData.append(TableTasks.getValueAt(row,
column)).append("/");
                    }

                }
                String edit = sbTableData.toString();

                projects =
assignteamname.addProject(projTitle.getText(),
assigntoTeam.getSelectedItem().toString(), Tasks.toString());
                projects.toString();
                projTitle.setText("");
                assignTask.setText("");
                ProjectsKt.rewrite();
                File file = new File(filePath);
                try {
                    BufferedReader br = new BufferedReader(new
FileReader(file));
                    String firstLine = br.readLine().trim();
                    String[] columnsName = firstLine.split(",");
                    DefaultTableModel model = (DefaultTableModel)
TableTasks.getModel();
                    model.setColumnIdentifiers(columnsName);
                    model.setRowCount(0);
                    Object[] tableLines = br.lines().toArray();

                    for (int i = 0; i < tableLines.length; i++) {
                        String line =
tableLines[i].toString().trim();
                        String[] dataRow = line.split("/");
                        model.addRow(dataRow);
                    }

                } catch (Exception ex) {

                }
            }
        });

        createProjectButton.setEnabled(false);

        confirmSettingsCheckBox.addActionListener(new
ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if ((TableTasks != null &&
TableTasks.getModel().getRowCount() <= 0 ? true : false) &&
projTitle.getText().equals("") && txtDays.getText().equals("")) {
                    JOptionPane optionPane = new JOptionPane("Please
enter project title, task(s) and completion time.",
JOptionPane.ERROR_MESSAGE);
                    JDialog dialog =
optionPane.createDialog("Error!");
                    dialog.setAlwaysOnTop(true);
                    dialog.setVisible(true);
                    confirmSettingsCheckBox.setSelected(false);
                } else {
                    if ((TableTasks != null &&
TableTasks.getModel().getRowCount() <= 0 ? true : false) &&
txtDays.getText().equals("")) {
                        JOptionPane optionPane = new
JOptionPane("Please add at last 1 task AND set completion time.",
JOptionPane.ERROR_MESSAGE);
                        JDialog dialog =
optionPane.createDialog("Error!");
                        dialog.setAlwaysOnTop(true);
```

```
                                        dialog.setVisible(true);
                                        confirmSettingsCheckBox.setSelected(false);
                                    } else {
                                        if ((TableTasks != null &&
TableTasks.getModel().getRowCount() <= 0 ? true : false) &&
projTitle.getText().equals("")) {
                                            JOptionPane optionPane = new
JOptionPane("Please enter at least 1 task AND add project title",
JOptionPane.ERROR_MESSAGE);
                                            JDialog dialog =
optionPane.createDialog("Error!");
                                            dialog.setAlwaysOnTop(true);
                                            dialog.setVisible(true);
                                            confirmSettingsCheckBox.setSelected(false
);
                                        } else {
                                                if ((TableTasks != null &&
TableTasks.getModel().getRowCount() <= 0 ? true : false)) {
                                                    JOptionPane optionPane = new
JOptionPane("Please add at least 1 task.",
JOptionPane.ERROR_MESSAGE);
                                                    JDialog dialog =
optionPane.createDialog("Error!");
                                                    dialog.setAlwaysOnTop(true);
                                                    dialog.setVisible(true);
                                                    confirmSettingsCheckBox.setSelect
ed(false);
                                                } else {
                                                    if
((projTitle.getText().equals(""))) {
                                                        JOptionPane optionPane = new
JOptionPane("Please enter project title",
JOptionPane.ERROR_MESSAGE);
                                                        JDialog dialog =
optionPane.createDialog("Error!");
                                                        dialog.setAlwaysOnTop(true);
                                                        dialog.setVisible(true);
                                                        confirmSettingsCheckBox.setSe
lected(false);
                                                    } else {
                                                        if
(confirmSettingsCheckBox.isSelected()) {
                                                            createProjectButton.setEn
abled(true);
                                                            projTitle.setEnabled(fals
e);
                                                            lblProjTitle.setEnabled(f
alse);
                                                            assigntoTeam.setEnabled(f
alse);
                                                            lblAssignTeama.setEnabled
(false);
                                                            lblAssignTask.setEnabled(
false);
                                                            assignTask.setEnabled(fal
se);
```

```
                                           lblTaskComp.setEnabled(fa
lse);
                                           txtDays.setEnabled(false)
;
                                           addTaskButton.setEnabled(
false);
                                           taskTable.setEnabled(fals
e);
                                       } else {
                                           createProjectButton.setEn
abled(false);
                                           projTitle.setEnabled(true
);
                                           lblProjTitle.setEnabled(t
rue);
                                           assigntoTeam.setEnabled(t
rue);
                                           lblAssignTeama.setEnabled
(true);
                                           lblAssignTask.setEnabled(
true);
                                           assignTask.setEnabled(tru
e);
                                           lblTaskComp.setEnabled(tr
ue);
                                           txtDays.setEnabled(true);
                                           addTaskButton.setEnabled(
true);
                                           taskTable.setEnabled(true
);
                                       }
                                   }
                               }
                           }
                       }
                   }
               }
           });

       txtDays.addKeyListener(new KeyAdapter() {
           @Override
           public void keyReleased(KeyEvent e) {
               super.keyReleased(e);
               try {
                   long number = Long.parseLong(txtDays.getText());
               } catch (Exception IO) {
                   JOptionPane.showMessageDialog(rootPane, "Please
only enter numbers for task completion.");
                   txtDays.setText("");
               }
           }
       });
   }

   public static void main(String[] args) {
       NewProject project = new NewProject();
       project.setBounds(1500, 1500, 1200, 900);
       project.setLocationRelativeTo(null);
       project.setResizable(false);
       project.setVisible(true);
   }


   private void createUIComponents() {
       // TODO: place custom component creation code here
   }
}
```

ManageTeams.java | Language - Java

```
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.*;
import java.util.ArrayList;

public class ManageTeams extends JFrame {
    private JPanel MainPnl;
    private JButton editButton;
    private JButton returnButton;
    private JComboBox cbteams;
    private JList TeamMembers;
    private JComboBox Comboboxmembers;
    private JButton addButton;
    private JButton removeButton;
    private JButton deleteTeamButton;
    private JTextField txtTeamTitle;
    private JPanel SettingsPnl;
    private JPanel TitlePnl;
    private JCheckBox confirmSettingsCheckBox;
    private JLabel lblSelectTeam;
    private JLabel lblAddMembers;
    private JLabel lblTeamName;
    private Teams teams;
    private CreateHandler createteam;
    private members manager;
    private ManageHandler managing;
    private teamname gettingTN;


    ManageTeams() {

        super("Manage Teams");
        this.setContentPane(this.MainPnl);
        this.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        this.pack();
        createteam = new CreateHandler();
        managing = new ManageHandler();
        //Liss created to show members in the JList
        DefaultListModel liss = new DefaultListModel();
        File inputFile = new File("Teams.txt");
        File tempFile = new File("TempTeams.txt");
        try {
            //Read from TXT file and add to the ComboBox
            String filePath1 = "Teams.txt";
            File file = new File(filePath1);
            BufferedReader br = new BufferedReader(new
FileReader(file));
            Object[] lines = br.lines().toArray();

            for (int i = 0; i < lines.length; i++) {
                String line = lines[i].toString();
                cbteams.addItem(line);
            }
            br.close();
        } catch (FileNotFoundException ex) {

        } catch (IOException e) {
            e.printStackTrace();
        }
        String filePath2 = "members.txt";
        File file = new File(filePath2);

        try {
            //Read from TXT file and add to the ComboBox
            BufferedReader br = new BufferedReader(new
FileReader(file));
            Object[] lines = br.lines().toArray();

            for (int i = 0; i < lines.length; i++) {
                String line = lines[i].toString();
                Comboboxmembers.addItem(line);
            }
            br.close();
        } catch (FileNotFoundException ex) {

        } catch (IOException e) {
            e.printStackTrace();
        }

        returnButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                Home home = new Home();
                home.setBounds(1500, 1500, 1000, 800);
                home.setLocationRelativeTo(null);
                home.setResizable(false);
                home.setVisible(true);
                dispose();
            }
        });
        addButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String member = 
 Comboboxmembers.getSelectedItem().toString();
                liss.addElement(member);
                TeamMembers.setModel(liss);
            }
        });
        removeButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                int selectedIndex = TeamMembers.getSelectedIndex();
                if (selectedIndex != -1) {
                    liss.remove(selectedIndex);
                }
            }
        });
        editButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {

                editButton.setEnabled(false);
                lblSelectTeam.setEnabled(true);
                cbteams.setEnabled(true);
                txtTeamTitle.setEnabled(true);
                lblAddMembers.setEnabled(true);
                Comboboxmembers.setEnabled(true);
                addButton.setEnabled(true);
                removeButton.setEnabled(true);
                deleteTeamButton.setEnabled(true);
                confirmSettingsCheckBox.setSelected(false);

                String member = liss.toString().replaceFirst(",",
 "").replaceFirst(" ", "");
                teams = createteam.createTeam(txtTeamTitle.getText(),
 member);
                teams.toString();
                String abc = txtTeamTitle.getText() + member;
                cbteams.addItem(abc);
                liss.removeAllElements();
                TeamMembers.setModel(liss);
                txtTeamTitle.setText("");
                try {
                    BufferedReader reader = new BufferedReader(new
 FileReader(inputFile));
                    BufferedWriter writer = new BufferedWriter(new
 FileWriter(tempFile));

                    int lineToRemove = cbteams.getSelectedIndex() +
 1;

                    String currLine;
                    int count = 0;

                    while ((currLine = reader.readLine()) != null) {
                        count++;
                        if (count == lineToRemove) {
                            continue;
                        }
                        writer.write(currLine +
 System.getProperty("line.separator"));
                    }
                    int index = cbteams.getSelectedIndex();
                    cbteams.removeItemAt(index);
                    liss.removeAllElements();
                    TeamMembers.setModel(liss);
                    txtTeamTitle.setText("");
                    writer.close();
                    reader.close();
                    inputFile.delete();
                    tempFile.renameTo(inputFile);
                } catch (IOException ioException) {
                    ioException.printStackTrace();

                }
            }

        });


        deleteTeamButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try {
                    BufferedReader reader = new BufferedReader(new
 FileReader(inputFile));
                    BufferedWriter writer = new BufferedWriter(new
 FileWriter(tempFile));

                    int lineToRemove = cbteams.getSelectedIndex() +
 1;

                    String currLine;
                    int count = 0;

                    while ((currLine = reader.readLine()) != null) {
                        count++;
                        if (count == lineToRemove) {
                            continue;
                        }
                        writer.write(currLine +
 System.getProperty("line.separator"));
                    }
                    int index = cbteams.getSelectedIndex();
                    cbteams.removeItemAt(index);
                    teams =
 createteam.createTeam(txtTeamTitle.getText(), liss.toString());
                    teams.toString();
                    liss.removeAllElements();
                    TeamMembers.setModel(liss);
                    txtTeamTitle.setText("");
                    writer.close();
                    reader.close();
                    inputFile.delete();
                    tempFile.renameTo(inputFile);
                } catch (IOException ioException) {
                    ioException.printStackTrace();
                }

            }
        });
        cbteams.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try {
                    liss.removeAllElements();
                    TeamMembers.setModel(liss);
                    ArrayList<String> al = new ArrayList<String>();
                    manager =
managing.sendlist(cbteams.getSelectedItem().toString());
                    String test =
manager.toString().replace("members(members=[", "").replace("])",
"");
                    String[] members = test.split(",");
                    gettingTN =
managing.getteamname(cbteams.getSelectedItem().toString());
                    txtTeamTitle.setText(gettingTN.toString().replace
("teamname(teamname=", "").replace(")", ""));
                    for (String str : members) {
                        liss.addElement(str);
                        TeamMembers.setModel(liss);
                    }
                } catch (Exception exception) {

                }
            }
        });

        editButton.setEnabled(false);


        confirmSettingsCheckBox.addActionListener(new
ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {

                if (txtTeamTitle.getText().equals("") &&
(liss.isEmpty())) {
                    JOptionPane optionPane = new JOptionPane("Please
Add Team Members AND Enter Team Name!", JOptionPane.ERROR_MESSAGE);
                    JDialog dialog =
optionPane.createDialog("Error!");
                    dialog.setAlwaysOnTop(true);
                    dialog.setVisible(true);
                    confirmSettingsCheckBox.setSelected(false);
                } else {
                    if (txtTeamTitle.getText().equals("")) {
                        JOptionPane optionPane = new
JOptionPane("Please Enter Team Name!", JOptionPane.ERROR_MESSAGE);
```

```
                       JDialog dialog =
optionPane.createDialog("Error!");
                       dialog.setAlwaysOnTop(true);
                       dialog.setVisible(true);
                       confirmSettingsCheckBox.setSelected(false);
                   } else {
                       if (liss.isEmpty()) {
                           JOptionPane optionPane = new
JOptionPane("Please Add Team Members!", JOptionPane.ERROR_MESSAGE);
                           JDialog dialog =
optionPane.createDialog("Error!");
                           dialog.setAlwaysOnTop(true);
                           dialog.setVisible(true);
                           confirmSettingsCheckBox.setSelected(false
);
                       } else {
                           if (confirmSettingsCheckBox.isSelected())
{
                               editButton.setEnabled(true);
                               lblSelectTeam.setEnabled(false);
                               cbteams.setEnabled(false);
                               lblTeamName.setEnabled(false);
                               txtTeamTitle.setEnabled(false);
                               lblAddMembers.setEnabled(false);
                               Comboboxmembers.setEnabled(false);
                               addButton.setEnabled(false);
                               removeButton.setEnabled(false);
                               deleteTeamButton.setEnabled(false);
                               TeamMembers.setEnabled(false);
                           } else {
                               editButton.setEnabled(false);
                               lblSelectTeam.setEnabled(true);
                               cbteams.setEnabled(true);
                               lblTeamName.setEnabled(true);
                               txtTeamTitle.setEnabled(true);
                               lblAddMembers.setEnabled(true);
                               Comboboxmembers.setEnabled(true);
                               addButton.setEnabled(true);
                               removeButton.setEnabled(true);
                               deleteTeamButton.setEnabled(true);
                               TeamMembers.setEnabled(true);
                           }
                       }
                   }
               }
           }
       });
   }

   public static void main(String[] args) {
       ManageTeams mt = new ManageTeams();
       mt.setBounds(1500, 1500, 1200, 900);
       mt.setLocationRelativeTo(null);
       mt.setResizable(false);
       mt.setVisible(true);
```

```
    }
```

NewTeam.java | Language - Java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.*;

public class NewTeam extends JFrame {
   private JTextField txtTeamTitle;
   private JTextField txtAddMemberName;
   private JButton addButton;
   private JButton createTeamButton;
   private JButton returnButton;
   private JPanel MainPnl;
   private JPanel titlePnl;
   private JPanel bottomPnl;
   private JPanel infoPnl;
   private JPanel membersPnl;
   private JButton removeButton;
   private JComboBox Comboboxmembers;
   private JButton editMemberButton;
   private JList TeamMembers;
   private JCheckBox confirmSettingsCheckBox;
   private JLabel lblAddMembers;
   private JLabel lblTeamName;
   private JButton confirmNameButton;
   private Teams teams;
   private CreateHandler createteam;

   NewTeam() {
       super("New Team Entry");
       this.setContentPane(this.MainPnl);
       this.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
       this.pack();
       createteam = new CreateHandler();

       try {
           try {
               //File reader method
               FileReader file = new FileReader("members.txt");
               BufferedReader reader = new BufferedReader(file);
               String text = "";
               String line = reader.readLine();
               while (line != null) {
                   Comboboxmembers.addItem(line);
                   line = reader.readLine();

               }
               reader.close();
           } catch (Exception e) {
               JOptionPane.showMessageDialog(null, e);
           }
       } catch (Exception e) {//Catch exception if any
```

```
            System.err.println("Error: " + e.getMessage());
        }

        DefaultListModel liss = new DefaultListModel();


        returnButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                Home home = new Home();
                home.setBounds(1500, 1500, 1000, 800);
                home.setLocationRelativeTo(null);
                home.setResizable(false);
                home.setVisible(true);
                dispose();
            }
        });

        createTeamButton.setEnabled(false

        );
        createTeamButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {

                teams = createteam.createTeam(txtTeamTitle.getText(),
liss.toString());
                teams.toString();
                liss.removeAllElements();
                TeamMembers.setModel(liss);
                txtTeamTitle.setText("");

                createTeamButton.setEnabled(false);
                removeButton.setEnabled(true);
                editMemberButton.setEnabled(true);
                addButton.setEnabled(true);
                Comboboxmembers.setEnabled(true);
                lblAddMembers.setEnabled(true);
                txtTeamTitle.setEnabled(true);
                lblTeamName.setEnabled(true);
                TeamMembers.setEnabled(true);
                confirmSettingsCheckBox.setSelected(false);

            }
        });
        addButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {

                String member =
Comboboxmembers.getSelectedItem().toString();
                liss.addElement(member);
                TeamMembers.setModel(liss);
            }
        });
        editMemberButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                EditMembers nt = new EditMembers();
                nt.setVisible(true);
                nt.setBounds(500, 500, 500, 500);
                nt.setLocationRelativeTo(null);
                nt.setResizable(true);
                dispose();
            }
        });
        removeButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                int selectedIndex = TeamMembers.getSelectedIndex();
                if (selectedIndex != -1) {
                    liss.remove(selectedIndex);
                }
            }
        });

        confirmSettingsCheckBox.addActionListener(new
ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {

                if (txtTeamTitle.getText().equals("") &&
(liss.isEmpty())) {
                    JOptionPane optionPane = new JOptionPane("Please
add team members AND enter team name!", JOptionPane.ERROR_MESSAGE);
                    JDialog dialog =
optionPane.createDialog("Error!");
                    dialog.setAlwaysOnTop(true);
                    dialog.setVisible(true);
                    confirmSettingsCheckBox.setSelected(false);
                } else {
                    if (txtTeamTitle.getText().equals("")) {
                        JOptionPane optionPane = new
JOptionPane("Please enter team name!", JOptionPane.ERROR_MESSAGE);
                        JDialog dialog =
optionPane.createDialog("Error!");
                        dialog.setAlwaysOnTop(true);
                        dialog.setVisible(true);
                        confirmSettingsCheckBox.setSelected(false);
                    } else {
                        if (liss.isEmpty()) {
                            JOptionPane optionPane = new
JOptionPane("Please add team members!", JOptionPane.ERROR_MESSAGE);
                            JDialog dialog =
optionPane.createDialog("Error!");
                            dialog.setAlwaysOnTop(true);
                            dialog.setVisible(true);
```

```
                                confirmSettingsCheckBox.setSelected(false
);
                            } else {
                                if (confirmSettingsCheckBox.isSelected())
{
                                    createTeamButton.setEnabled(true);
                                    removeButton.setEnabled(false);
                                    editMemberButton.setEnabled(false);
                                    addButton.setEnabled(false);
                                    Comboboxmembers.setEnabled(false);
                                    lblAddMembers.setEnabled(false);
                                    txtTeamTitle.setEnabled(false);
                                    lblTeamName.setEnabled(false);
                                    TeamMembers.setEnabled(false);
                                } else {
                                    createTeamButton.setEnabled(false);
                                    removeButton.setEnabled(true);
                                    editMemberButton.setEnabled(true);
                                    addButton.setEnabled(true);
                                    Comboboxmembers.setEnabled(true);
                                    lblAddMembers.setEnabled(true);
                                    txtTeamTitle.setEnabled(true);
                                    lblTeamName.setEnabled(true);
                                    TeamMembers.setEnabled(true);

                                }
                            }
                        }
                    }
                }
            });
    }

    public static void main(String[] args) {
        NewTeam nt = new NewTeam();
        nt.setBounds(1500, 1500, 1200, 900);
        nt.setLocationRelativeTo(null);
        nt.setResizable(false);
        nt.setVisible(true);
    }
```

RecordProgress. java | Language - Java

```
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.*;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.Scanner;


public class RecordProgress extends JFrame {
    private JButton confirmButton;
   private JPanel MainPnl;
   private JButton returnButton;
   private JComboBox JcbProjects;
   private JTable TableTasks;
   private JTextField txtTeam;
   private JTable JTableCriticalPath;
   private JTextField txtDays;
   private JTextField txtInitialNodes;
   private JTextField txtFinDate;
   private JCheckBox kotlinCheckBox;
   private JCheckBox scalaCheckBox;
   private JLabel lblSelectProj;
   private JLabel lblTeam;
   private JLabel lblCriticalPath;
   private JLabel lblInitialNodes;
   private JLabel lblFinishDate;
   private JScrollPane tasksTable;
   private JScrollPane criticalTable;
   private JTextField txt;
   private Progress progress;
   private ProjectsHandler assignteamname;
   private Projects projects;
   private ProjectsKt projectsKt;
   private CriticalPathKt CriticalPath;
   private CPHandling CPHandler;


   RecordProgress() {
       super("Manage Projects");
       this.setContentPane(this.MainPnl);
       this.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
       this.pack();
       assignteamname = new ProjectsHandler();
       String filePath = "Tasks.txt";
       String filePath2 = "projects.txt";
       File inputFile = new File("projects.txt");
       File tempFile = new File("TempProjects.txt");

       try {
           File file = new File(filePath2);
           BufferedReader br = new BufferedReader(new
FileReader(file));
           Object[] lines = br.lines().toArray();

           for (int i = 0; i < lines.length; i++) {
               String line = lines[i].toString();
               String[] splitter = line.split(";");
               String write = splitter[0];
               JcbProjects.addItem(write);
           }
           br.close();
       } catch (FileNotFoundException ex) {

       } catch (IOException e) {
           e.printStackTrace();
       }
       ProjectsKt.rewrite();

       returnButton.addActionListener(new ActionListener() {
           @Override
           public void actionPerformed(ActionEvent e) {
               Home home = new Home();
               home.setBounds(1500, 1500, 1000, 800);
               home.setLocationRelativeTo(null);
               home.setResizable(false);
               home.setVisible(true);
               dispose();
           }
       });
       JcbProjects.addActionListener(new ActionListener() {
           @Override
           public void actionPerformed(ActionEvent e) {
               if (kotlinCheckBox.isSelected()){
                   ProjectsKt.rewrite();
                   try {
                       String word =
JcbProjects.getSelectedItem().toString();
                       File file = new File(filePath2);
                       BufferedReader br = new BufferedReader(new
FileReader(file));
                       Scanner input = new Scanner(new
File(filePath2));
                       Integer Dayadd = 1;


                       while (input.hasNext()) {
                           String line = input.nextLine();


                           if (line.contains(word)) {
                               String[] splitter = line.split(";");
                               String write = splitter[3];
                               String dates = splitter[2];
                               String[] date = dates.split("-");
                               Calendar cal =
Calendar.getInstance();
                               cal.set(Calendar.DATE,
Integer.parseInt(date[2]));
                               cal.set(Calendar.YEAR,Integer.parseIn
t(date[0]));
                               cal.set(Calendar.MONTH,Integer.parseI
nt(date[1])-1);
                               Date d = cal.getTime();
                               File myObj = new
File("CriticalPathDays.txt");
                               Scanner myReader = new
Scanner(myObj);
                               while (myReader.hasNextLine()) {
                                   String data =
myReader.nextLine();
```

```
                                        txtDays.setText("This Project
will take " + data + " days");
                                        Dayadd = Integer.parseInt(data);

                                    }
                                    SimpleDateFormat sdf = new
SimpleDateFormat("dd/MM/yyyy");
                                    cal.add(Calendar.DATE, Dayadd); //
Adding 5 days
                                    String output =
sdf.format(cal.getTime());
                                    txtFinDate.setText(output);
                                    String team = splitter[1];
                                    txtTeam.setText(team);
                                    String replace1 = write.replace("[",
" ").replace("]", " ");
                                    String[] replaced =
replace1.split(",");
                                    for (int a = 0; a < replaced.length;
a++) {
                                        String put = replaced[a];
                                        progress =
assignteamname.addprogress(put);
                                        progress.toString();
                                    }
                                    br.close();
                                }
                            }
                            input.close();
                        } catch (FileNotFoundException ex) {

                        } catch (IOException ioException) {
                            ioException.printStackTrace();
                        }


                        try {
                            File file = new File("Tasks.txt");
                            BufferedReader br = new BufferedReader(new
FileReader(file));
                            String firstLine = br.readLine().trim();
                            String[] columnsName = firstLine.split(",");
                            DefaultTableModel model = (DefaultTableModel)
TableTasks.getModel();
                            model.setColumnIdentifiers(columnsName);
                            model.setRowCount(0);
                            Object[] tableLines = br.lines().toArray();
                            for (int i = 0; i < tableLines.length; i++) {
                                String line =
tableLines[i].toString().trim();
                                String[] dataRow = line.split("/");
                                model.addRow(dataRow);
                            }
                            br.close();
                            CriticalPath.main();
                            File myObj = new
     File("CriticalPathDays.txt");
                            Scanner myReader = new Scanner(myObj);
                            while (myReader.hasNextLine()) {
                                String data = myReader.nextLine();
                                txtDays.setText("This Project will take "
     + data + " days");


                            }
                            myReader.close();
                        } catch (Exception ex) {

                        }
                        try {
                            File myObj = new File("InitialNodes.txt");
                            Scanner myReader = new Scanner(myObj);
                            while (myReader.hasNextLine()) {
                                String data = myReader.nextLine();
                                txtInitialNodes.setText(data);
                            }
                            myReader.close();
                        } catch (Exception ex) {

                        }
                        try {
                            File file = new File("CriticalPath.txt");
                            BufferedReader br = new BufferedReader(new
     FileReader(file));
                            String firstLine = br.readLine().trim();
                            String[] columnsName = firstLine.split(",");
                            DefaultTableModel model = (DefaultTableModel)
     JTableCriticalPath.getModel();
                            model.setColumnIdentifiers(columnsName);
                            model.setRowCount(0);
                            Object[] tableLines = br.lines().toArray();
                            for (int i = 0; i < tableLines.length; i++) {
                                String line =
     tableLines[i].toString().trim();
                                String[] dataRow = line.split("/");
                                model.addRow(dataRow);
                            }
                            br.close();
                        } catch (Exception ex) {

                        }
                    }
                    if (scalaCheckBox.isSelected()) {
                        ProjectsKt.rewrite();

                        try {
                            String word =
     JcbProjects.getSelectedItem().toString();
                            File file = new File(filePath2);
                            BufferedReader br = new BufferedReader(new
     FileReader(file));
                            Scanner input = new Scanner(new
     File(filePath2));
                            Integer Dayadd = 1;


                            while (input.hasNext()) {
                                String line = input.nextLine();


                                if (line.contains(word)) {
                                    String[] splitter = line.split(";");
                                    String write = splitter[3];
                                    String dates = splitter[2];
                                    String[] date = dates.split("-");
                                    Calendar cal =
     Calendar.getInstance();
                                    cal.set(Calendar.DATE,
     Integer.parseInt(date[2]));
                                    cal.set(Calendar.YEAR,
     Integer.parseInt(date[0]));
                                    cal.set(Calendar.MONTH,
     Integer.parseInt(date[1]) - 1);
                                    Date d = cal.getTime();
                                    File myObj = new
     File("CriticalPathDays.txt");
                                    Scanner myReader = new
     Scanner(myObj);
                                    while (myReader.hasNextLine()) {
                                        String data =
     myReader.nextLine();
                                        txtDays.setText("This Project
     will take " + data + " days");
                                        Dayadd = Integer.parseInt(data);

                                    }
                                    SimpleDateFormat sdf = new
     SimpleDateFormat("dd/MM/yyyy");
                                    cal.add(Calendar.DATE, Dayadd); //
     Adding 5 days
                                    String output =
     sdf.format(cal.getTime());
                                    txtFinDate.setText(output);
                                    String team = splitter[1];
                                    txtTeam.setText(team);
                                    String replace1 = write.replace("[",
     " ").replace("]", " ");
                                    String[] replaced =
     replace1.split(",");
                                    for (int a = 0; a < replaced.length;
     a++) {
                                        String put = replaced[a];
                                        progress =
     assignteamname.addprogress(put);
```

```
                            progress.toString();
                        }
                        br.close();
                    }
                }
                input.close();
            } catch (FileNotFoundException ex) {

            } catch (IOException ioException) {
                ioException.printStackTrace();
            }


            try {
                File file = new File("Tasks.txt");
                BufferedReader br = new BufferedReader(new
FileReader(file));
                String firstLine = br.readLine().trim();
                String[] columnsName = firstLine.split(",");
                DefaultTableModel model = (DefaultTableModel)
TableTasks.getModel();
                model.setColumnIdentifiers(columnsName);
                model.setRowCount(0);
                Object[] tableLines = br.lines().toArray();
                for (int i = 0; i < tableLines.length; i++) {
                    String line =
tableLines[i].toString().trim();
                    String[] dataRow = line.split("/");
                    model.addRow(dataRow);
                }
                br.close();
                CriticalPath.main();

                File myObj = new
File("CriticalPathDays.txt");
                Scanner myReader = new Scanner(myObj);
                while (myReader.hasNextLine()) {
                    String data = myReader.nextLine();
                    txtDays.setText("This Project will take "
+ data + " days");


                }
                myReader.close();
            } catch (Exception ex) {

            }
            try {
                File myObj = new File("InitialNodes.txt");
                Scanner myReader = new Scanner(myObj);
                while (myReader.hasNextLine()) {
                    String data = myReader.nextLine();
                    txtInitialNodes.setText(data);
                }
                myReader.close();
```

```
                    } catch (Exception ex) {

                    }
                    try {
                        File file = new File("CriticalPath.txt");
                        BufferedReader br = new BufferedReader(new
FileReader(file));
                        String firstLine = br.readLine().trim();
                        String[] columnsName = firstLine.split(",");
                        DefaultTableModel model = (DefaultTableModel)
JTableCriticalPath.getModel();
                        model.setColumnIdentifiers(columnsName);
                        model.setRowCount(0);
                        Object[] tableLines = br.lines().toArray();
                        for (int i = 0; i < tableLines.length; i++) {
                            String line =
tableLines[i].toString().trim();
                            String[] dataRow = line.split("/");
                            model.addRow(dataRow);
                        }
                        br.close();
                    } catch (Exception ex) {

                    }
                }}
        });

        confirmButton.setEnabled(false);
        kotlinCheckBox.setSelected(false);
        confirmButton.setEnabled(false);
        lblSelectProj.setEnabled(false);
        JcbProjects.setEnabled(false);
        lblTeam.setEnabled(false);
        txtTeam.setEnabled(false);
        lblCriticalPath.setEnabled(false);
        txtDays.setEnabled(false);
        lblInitialNodes.setEnabled(false);
        txtInitialNodes.setEnabled(false);
        lblFinishDate.setEnabled(false);
        txtFinDate.setEnabled(false);
        tasksTable.setEnabled(false);

        kotlinCheckBox.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if (kotlinCheckBox.isSelected()) {
                    scalaCheckBox.setSelected(false);
                    scalaCheckBox.setEnabled(false);
                    confirmButton.setEnabled(true);
                    lblSelectProj.setEnabled(true);
                    JcbProjects.setEnabled(true);
                    lblTeam.setEnabled(true);
                    txtTeam.setEnabled(true);
                    lblCriticalPath.setEnabled(true);
                    txtDays.setEnabled(true);
                    lblInitialNodes.setEnabled(true);
                    txtInitialNodes.setEnabled(true);
                    lblFinishDate.setEnabled(true);
                    txtFinDate.setEnabled(true);
                    tasksTable.setEnabled(true);

                } else {
                    if (!kotlinCheckBox.isSelected()) {
                        scalaCheckBox.setEnabled(true);
                        confirmButton.setEnabled(false);
                        lblSelectProj.setEnabled(false);
                        JcbProjects.setEnabled(false);
                        lblTeam.setEnabled(false);
                        txtTeam.setEnabled(false);
                        lblCriticalPath.setEnabled(false);
                        txtDays.setEnabled(false);
                        lblInitialNodes.setEnabled(false);
                        txtInitialNodes.setEnabled(false);
                        lblFinishDate.setEnabled(false);
                        txtFinDate.setEnabled(false);
                        tasksTable.setEnabled(false);

                    }
                }
            }
        });
        scalaCheckBox.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if (scalaCheckBox.isSelected()) {
                    kotlinCheckBox.setSelected(false);
                    kotlinCheckBox.setEnabled(false);
                    confirmButton.setEnabled(true);
                    lblSelectProj.setEnabled(true);
                    JcbProjects.setEnabled(true);
                    lblTeam.setEnabled(true);
                    txtTeam.setEnabled(true);
                    lblCriticalPath.setEnabled(true);
                    txtDays.setEnabled(true);
                    lblInitialNodes.setEnabled(true);
                    txtInitialNodes.setEnabled(true);
                    lblFinishDate.setEnabled(true);
                    txtFinDate.setEnabled(true);
                    tasksTable.setEnabled(true);

                } else {
                    if (!scalaCheckBox.isSelected()) {
                        kotlinCheckBox.setEnabled(true);
                        confirmButton.setEnabled(false);
                        lblSelectProj.setEnabled(false);
                        JcbProjects.setEnabled(false);
                        lblTeam.setEnabled(false);
                        txtTeam.setEnabled(false);
                        lblCriticalPath.setEnabled(false);
                        txtDays.setEnabled(false);
```

```
                        lblInitialNodes.setEnabled(false);
                        txtInitialNodes.setEnabled(false);
                        lblFinishDate.setEnabled(false);
                        txtFinDate.setEnabled(false);
                        tasksTable.setEnabled(false);
                    }
                }
            }
        });


        confirmButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {

                StringBuffer sbTableData = new StringBuffer();
                for (int row = 0; row < TableTasks.getRowCount();
row++) {
                    for (int column = 0; column <
TableTasks.getColumnCount(); column++) {
                        sbTableData.append(TableTasks.getValueAt(row,
column)).append("/");
                    }

                }
                try {
                    BufferedReader reader = new BufferedReader(new
FileReader(inputFile));
                    BufferedWriter writer = new BufferedWriter(new
FileWriter(tempFile));

                    int lineToRemove = JcbProjects.getSelectedIndex()
+ 1;
                    String currentLine;
                    int count = 0;

                    while ((currentLine = reader.readLine()) != null)
{
                        count++;
                        if (count == lineToRemove) {
                            continue;
                        }
                        writer.write(currentLine +
System.getProperty("line.separator"));
                    }
                    writer.close();
                    reader.close();
                    inputFile.delete();
                    tempFile.renameTo(inputFile);
                    String edit = sbTableData.toString();
                    edit = edit.replaceAll("([^/]*/[^/]*/[^/]*)/",
"$1,");
                    String TeamName =
JcbProjects.getSelectedItem().toString();
```

```
                        projects =
assignteamname.addProject(JcbProjects.getSelectedItem().toString(),
txtTeam.getText(), edit);
                        projects.toString();
                        int index = JcbProjects.getSelectedIndex();
                        JcbProjects.removeItemAt(index);
                        JcbProjects.addItem(TeamName);
                        CriticalPath.main();
                        ProjectsKt.rewrite();

                   } catch (FileNotFoundException fileNotFoundException)
{
                        fileNotFoundException.printStackTrace();
                   } catch (IOException ioException) {
                        ioException.printStackTrace();
                   }

              }
         });
    }

    public static void main(String[] args) {
         RecordProgress mp = new RecordProgress();
         mp.setBounds(1500, 1500, 1200, 900);
         mp.setLocationRelativeTo(null);
         mp.setResizable(false);
         mp.setVisible(true);
    }


}
```