CS124 Group Project

Design then Code with Blue-J

Connor Goddard, Samuel Jackson, Craig Heptinstall
(clg11, slj11, crh13)

Table of Contents

Introduction

In this report, we will be documenting the design and analysis for, the implemented product, a and the testing of a program written in java language, which we will go into further detail in the next section of this report. As with any project like this type, we will require to begin by outlining the problem, giving a definition of what will be required by the completed program, and what we may need to do to achieve it. This will form the first section of the analysis for our project, and will lead up to the design stage.

By looking at the definition we create, we will then look at designing the specified java program using a number of tools. First a simple use case diagram will allow us to identify what the different operations taken by the user will need to be and what the program will allow them to do. Once this is complete, we will then move onto creating a class diagram, which will form the basis of our program on later. The class diagram will prove a great deal of worth in providing us the option of designing each and every class we think we will require and what methods will belong to which. Also, having the relationships worked out in this diagram will make it much easier when implementing in terms of the links from class to class, and for example placing 'get' and 'set' methods.
Once the class diagram is completed, we can then look at making an activity diagram which will be used to identify the workflow of actions carried along the program when users interact different ways. For instance, we will show the way in which the program should deal with the user creating an object, or removing etc. Again this will prove useful in the implementation because of the ability to look back at it and see what order different operations in the code must be performed.

The next two and final design methods we should perform within this project will be a sequence diagram followed by a package diagram. A sequence diagram is much like an activity but shows the flow of data, along with the operations of the programs inner workings. It shows different operations/ processors or objects that 'live' alongside one another and in more detail the messages that are sent between them. The key to this diagram is that it will help us plan in which order they occur. A package diagram may be provided in this report also, which shows the dependencies between the different packages we could have implemented within our java solution. For instance, sectioning different types of classes into one package and then another type into a different package will allow us to create a neater, and easier to use code plan. The package diagram, should also deal with any package imports we may have to use for the program.

Once we have completed the design and analysis for the project, and the program is complete of course, we will then be able to perform testing. For this project, we will be performing manual testing which will mean creating multiple test tables, and testing multiple actions or inputs with different pieces of the program to see where the product works and meets its requirement, and where it fails and needs improvement.
Finally, we will add on a conclusion to the end of this report, briefly outlining what we thought went well about the project, and what we had issues with, also looking at how we overcame these. This will allow us to improve for a further project in the future.

Analysis

Problem Definition

In this group assignment we have been handed, we have been asked to design, implement and test a small GUI program using java swing that will provide a workspace for users to design a number of classes, add relationships to these classes such as inheritance and cardinalities, and also use these to create on-the-fly java files containing a basic skeleton for each class. This first section of the assignment will outline the problems and tasks we will need to perform in order to get our program working correctly.

To break this down further and looking firstly at the GUI part of this assignment, as a group we will be able to split this section of work up from the data handling coding which will store the class information. We will need to draw out a user friendly and relatively attractive looking interface which will work smoothly with no errors. Contained

within the GUI work required will be the ability to draw, drag, and remove objects from the workspace in our program. The classes will need to be drawn in relevance to the user's needs, and for instance make sure the length of the text fits inside the class drawn. We need to decide how we will input the information for the classes, for instance either inputting class information straight into an already drawn class, or inputting it into a pop up dialog box then adding this to a new class when it is created.

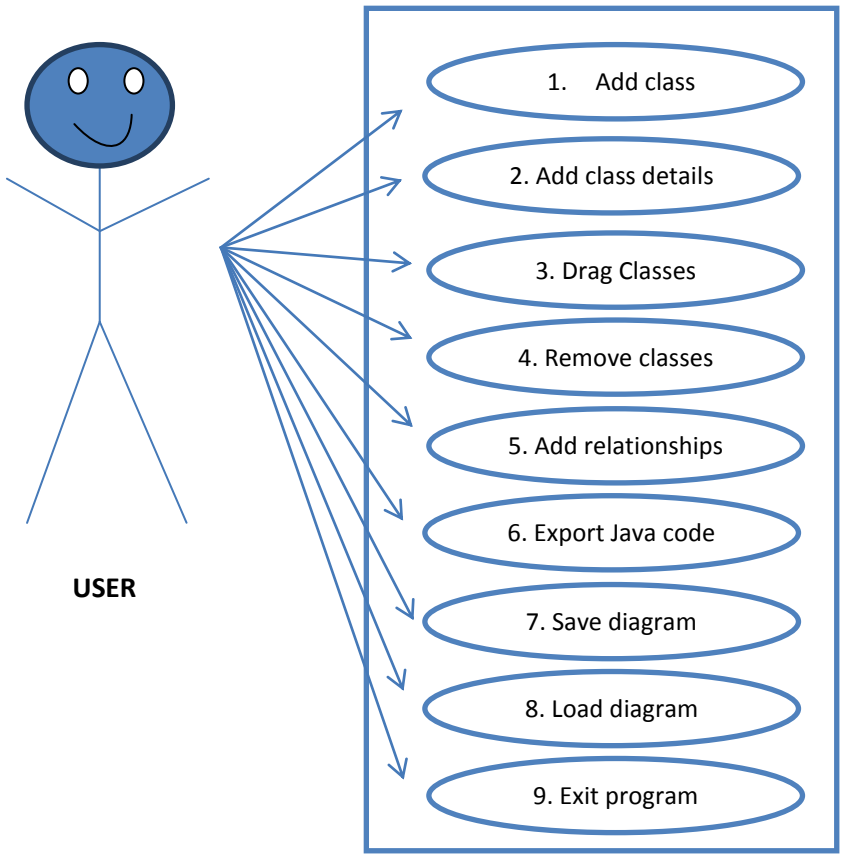
Another problem we will need to solve concerning the GUI is the requirement for the relationships between classes. Because classes will need to be linked by lines, and to avoid using diagonal lines to link them, we will need to add code that calculates the correct path to the classes using horizontal and vertical lines only. Adding the inheritance type to the end of lines is also a key part of the GUI we will need to implement. This will need to match the java skeleton classes the program will implement when the user chooses to.

This brings me to the last problem which will need to be looked at, which is the storage of data and looking at how to get our swing program to use the data inputted to add information to the class, and also to store it for when implementing the java file creation. It will need to be translated from the users input into the java files, for example knowing to add an array list to a class if the user added cardinalities of one to many. It should also be able to add the get, set and add methods to automatically written classes.

We must make sure throughout the implementation of our GUI project that we keep the form layout as simple and easy to use as possible, with the use of menu bars for optional save, load, and exit buttons. Buttons also to add new classes and input information into those classes may be required. If at the end of the project we decide we want to make our program a bit more appealing, we could add additional options and play about the colours and themes of the program.

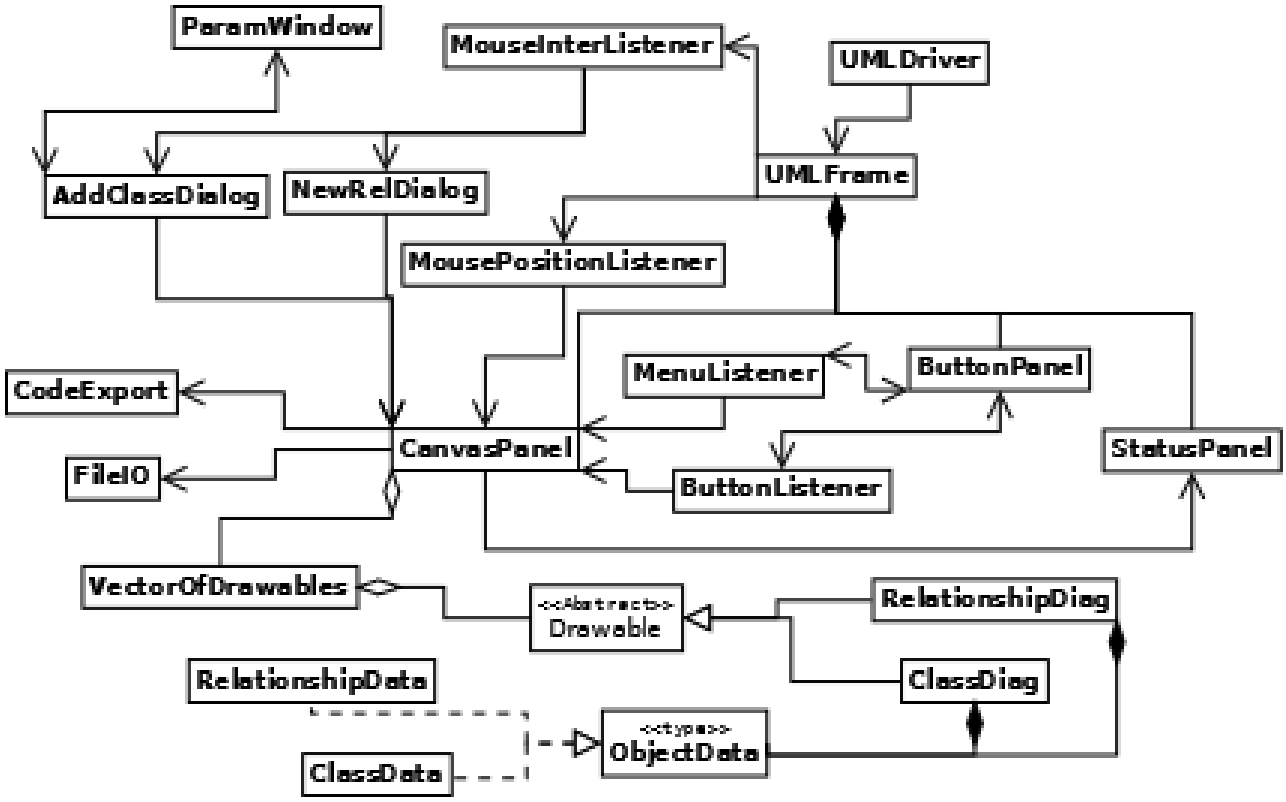
Design

Use Case Diagram



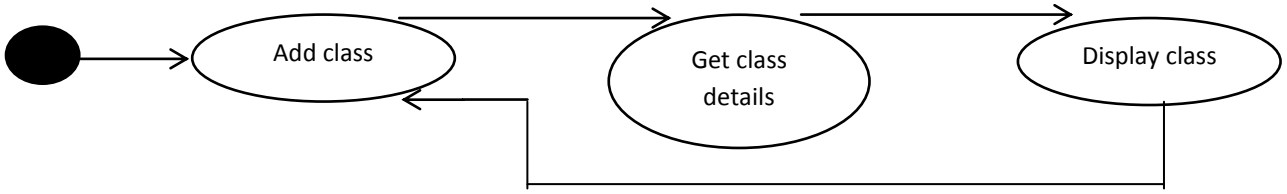
To the left we can show our use case diagram which illustrate the functions and operations the user will be able to perform using our program.

Class Diagram

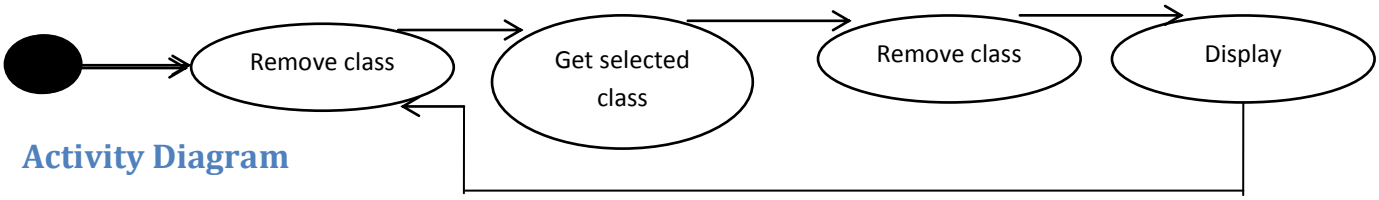


State Diagrams

State Diagram 1: Adding classes to the canvas:

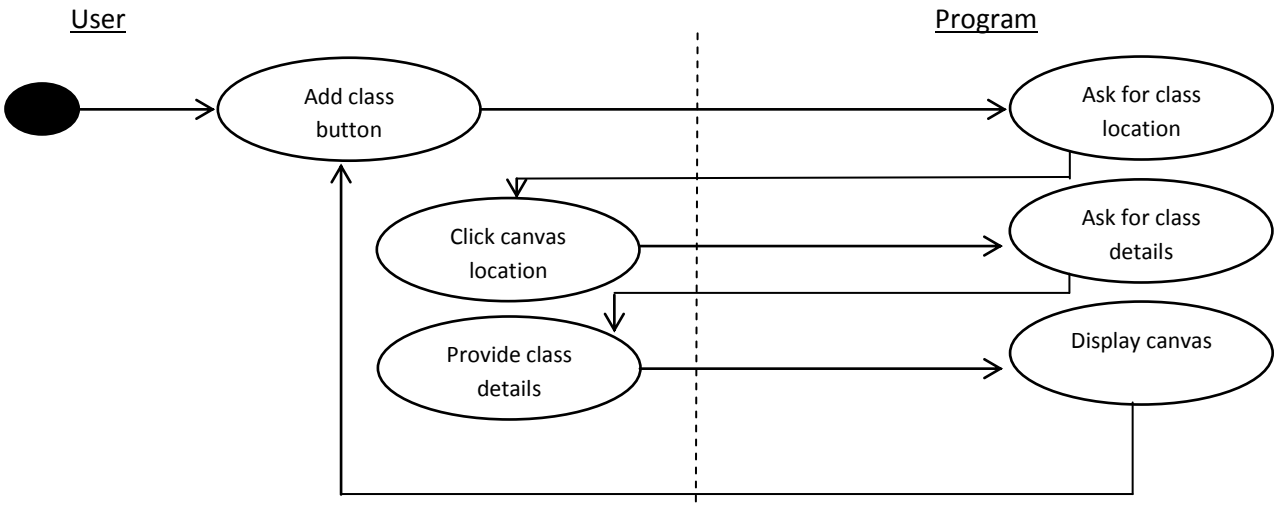


State Diagram 2: Removing

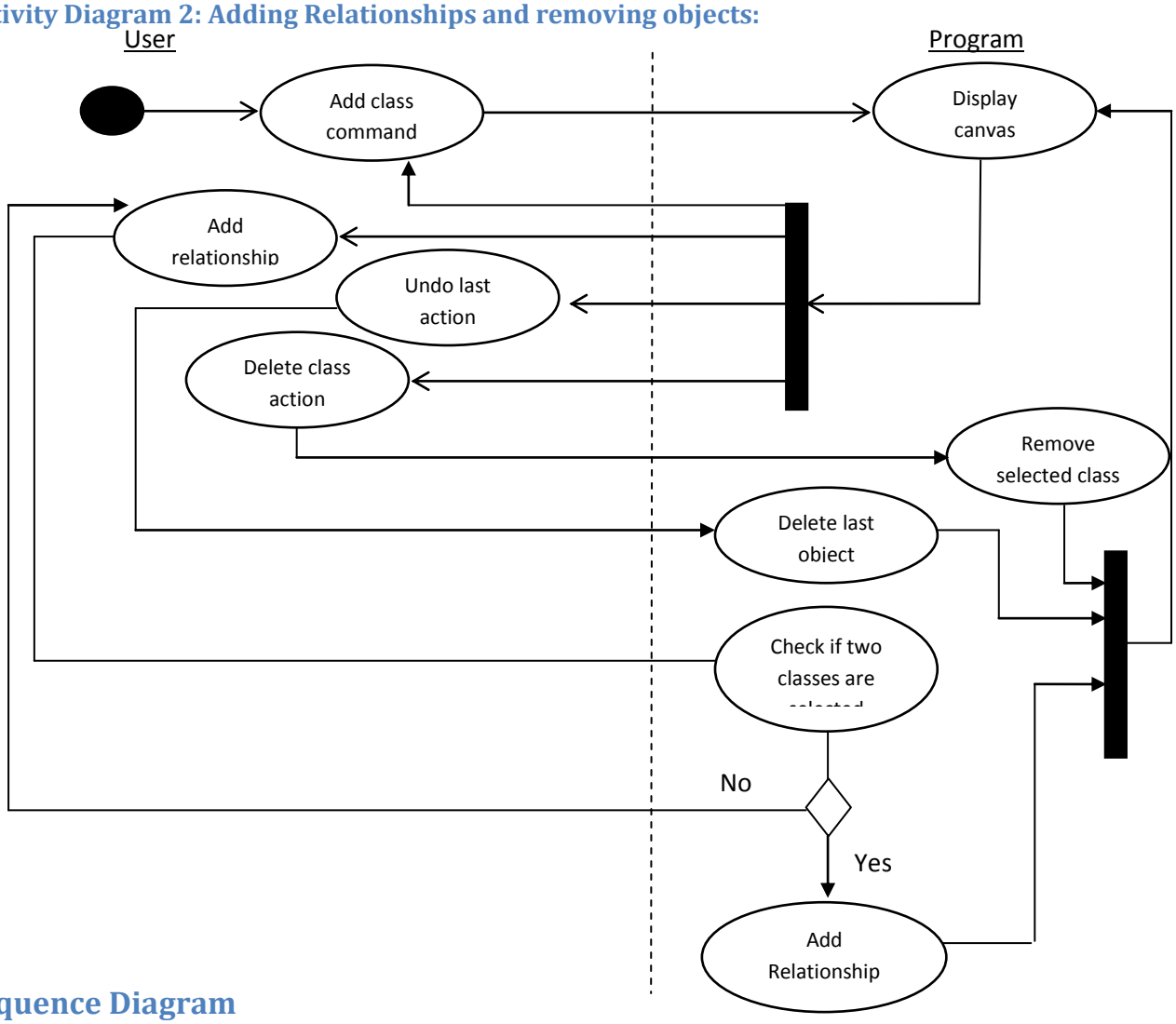


Activity Diagram

Activity Diagram 1: Adding classes:

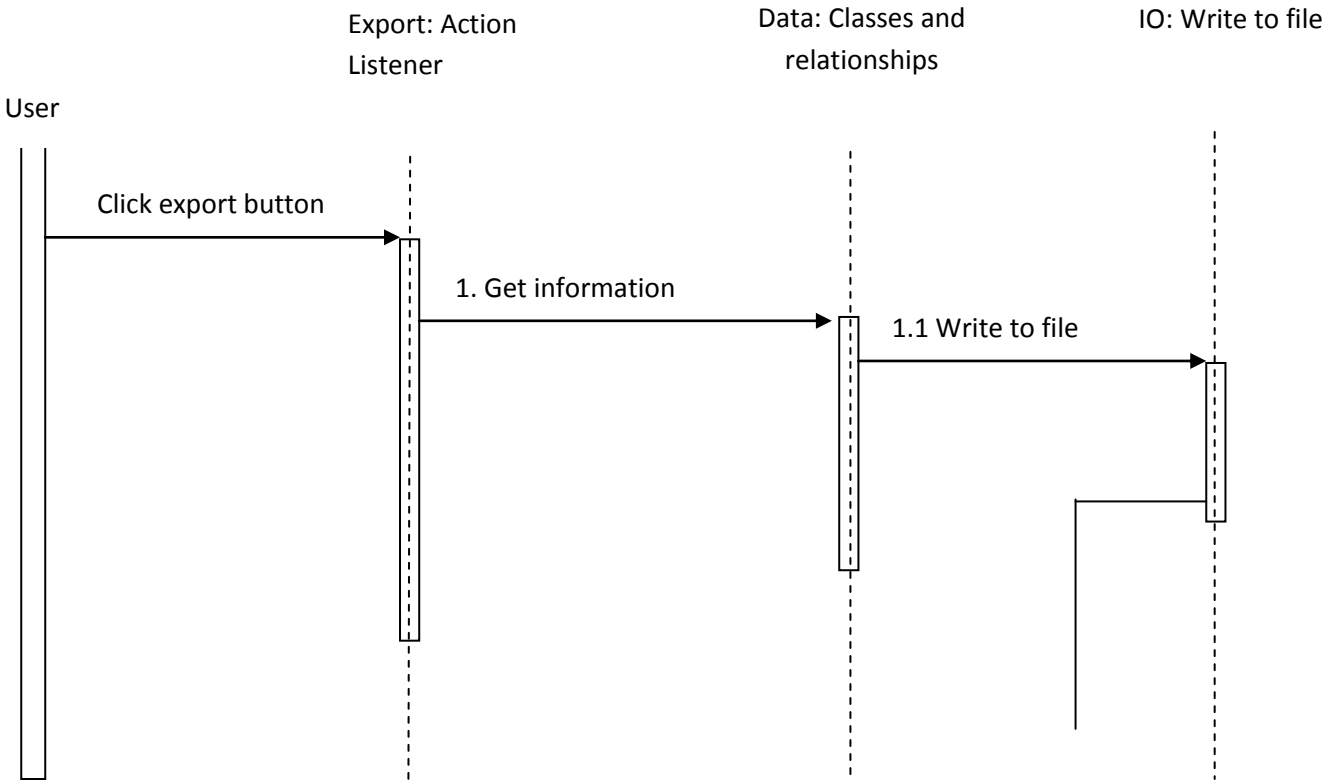


Activity Diagram 2: Adding Relationships and removing objects:

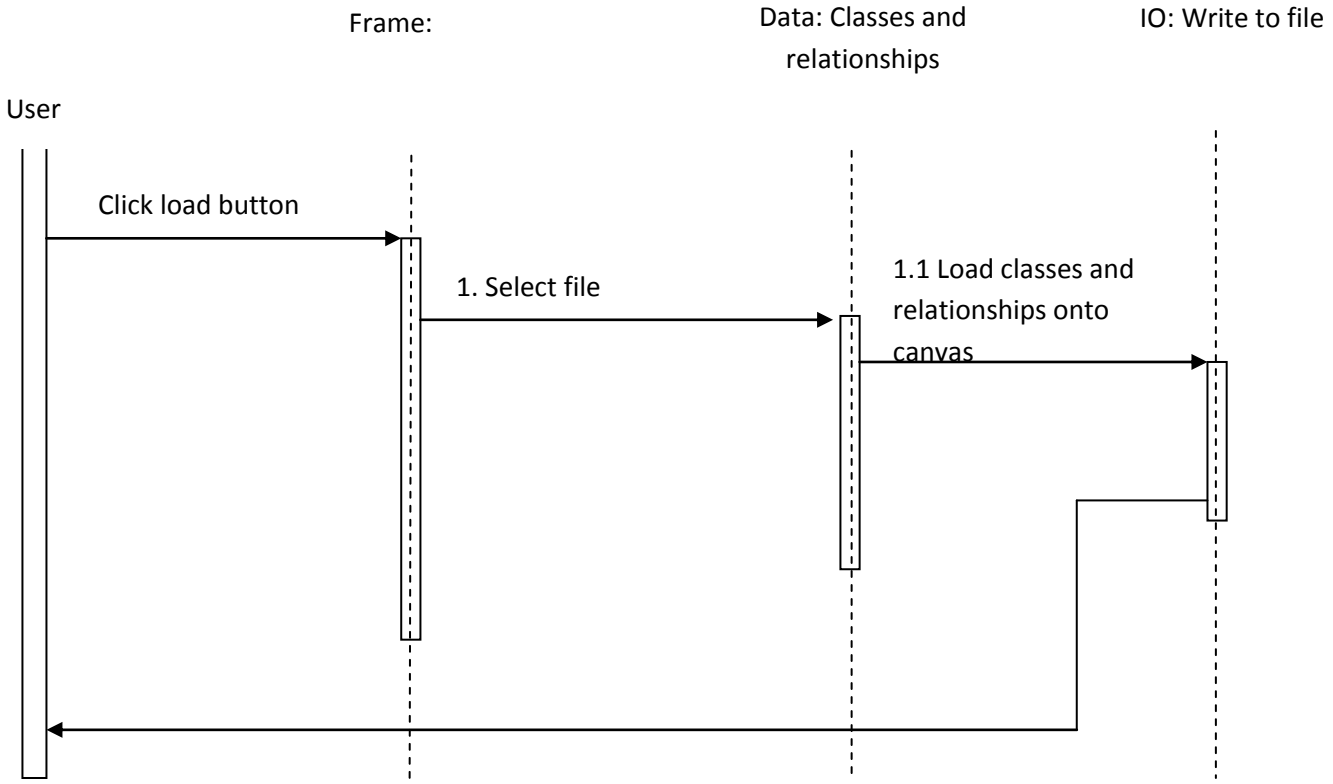


Sequence Diagram

Sequence Diagram 1: Exporting to Java Code:



Activity Diagram 2: Saving and Loading Diagrams:



UML Justification and class description

Now we have completed creating the class diagram, along with the other UML diagrams supporting our design, we will create a justification below on how the design will meet the objectives and aims of the program we will be creating. We will include short descriptions of the class diagram classes, followed by how these classes help meet the requirements of the project. The UML section of the project is one of the key sections which will allow us to create the most efficient and requirement meeting program.

Starting with the class diagram design:

UMLDriver: This class acts as the entry point for the application. It contains the main class and simply creates an instance of aUMLFrame.

UMLFrame:UMLFrames purpose is to collate all the different GUI aspects of the application. It is the main window you see when running the program and contains the three sub panels (ButtonPanel, StatusPanel and CanvasPanel). Its function is to create and manage its GUI sub components. as well as define basic details about the window.- This class solves the very first of our aims of the program, to make it graphical. By providing a window within the program, we could add to this buttons, menus and panels where required.

ButtonPanel: ButtonPanel is one of the three panels which are contained within the UMLFrame. It is responsible for creating and displaying the ButtonPanel to the user. It accepts and communicates user input with the ButtonListener and with MenuListener. – We would require buttons to allow user to select options such as when inputting classes choosing which type of variable they would prefer along with submitting the data.

StatusPanel: The StatusPanel is used to give the user some feedback independent of pop up windows. This panel is exclusively updated form the canvas panel, in response to events occurring there. Example events are when a file is saved or loaded, as well as when adding classes and relationships.

CanavasPanel: CanavasPanel is the most complex and important panel within the whole canvas. It acts as the drawing canvas for any diagram created. Its primary purpose is to organise drawing Drawable entities within it. All Listener

objects in the application affect the Canvas panel in some way. It also contains the methods for adding and removing classes and relationships. The key to drawing objects to the canvas is that it keeps track of a VectorOfDrawables object, used to store anything that can be drawn. The canvas panel also has methods to liaise with both FileIO and CodeExport.

MousePositionListener: This class is used to detect the position of the mouse in response to user actions. It is used allow classes to be dragged around the screen and implements the mouseDraggedmehthod.- Another integral part to the program, this allows users to operate the program at all, because without the mouse position listener, moving and adding classes to specific locations would not be possible.

MouseInterListener: This class is used to detect user input via the mouse relative to the state of the program. For example, it is in this class that the position of where the user clicks to add a class is recorded. It then create appropriate dialogues to take in user input and then pass that data to the CanvasPanel to turn it into a diagram.

MenuListener: This object is used to record messages send by the user from the menu bar at the top of the application and create appropriate responses in the CanvasPanel. Overall, this class works very similarly to the ButtonListener class.

NewRelDialog: The NewRelDialog class is designed to take user input about adding a relationship to the diagram. This class is responsible for its own independent GUI frame and has its own action listener. Recorded information is passed back to the CanavasPanel in order to create a relationship on the diagram.- As analysed within the problem definition, being able to add relationships between classes would be another key feature of a program like this.

AddClassDialog: The AddClassDialog class is designed to take user input about adding a new class to the diagram. Again, this class has its own independent GUI frame and action listener. It passes the information that it records about the classes back to the CanvasPanel in order to create a class in the diagram.- Another problem solved would be with this class, we required the ability to add classes to the canvas panel.

ParamWindow: The ParamWindow class is another self contained GUI class which is a child window ofAddClassDialog designed to take input from the user about what parameters they want to add to a selected method.

CodeExport: This class handles the conversion of the stored data in the diagram model in the CanvasPanel into template Java files. Its methods are called from the CanvasPanel and it is designed to do a lot of sting manipulation of the stored data to covert it from UML design representation to source code.- Exporting the code into self contained java files is another feature we would need to acquire for our program.

FileIO: The FileIO class handles converting the entire diagram currently being drawn in the CanvasPanel into a single XML file for the project. This class has features for both saving and loading the XML file to the user file system, as well as displaying simple user input dialogues to help better navigate the file system.- More of a Wow feature, this class would allow the ability to save and load user's projects adding to the programs usability.

VectorOfDrawables: The VectorOfDrawables acts very similarly to any container with the Java system, however, it has a few extra methods tacked on which are more specific to our application. Its main purpose is to store references to all the Drawable objects in our application. This class also contains methods for finding/removing the nearest class to the user's current mouse position. It also has a draw all method which loops through its internal collection and paints them to the screen.

Drawable: Drawable is an abstract class which is a generalisation of both a class diagram and a relationship diagram. It contains simple data about the objects x and y position on the screen as well as providing an abstract method for drawing the component that all sub classes must implement. It is also useful to have this abstract so that it cannot be directly instantiated anywhere in the program.

RelationshipDiag: This class is concerned with drawing a relationship between to diagrams to the screen. As such it has a reference to both the classes that it links between. It also has an extensive paint method which draws the relationship onto the canvas. This function also defines how the relationship is drawn relative to its type (e.g. Association, aggregation).

ClassDiag: The ClassDiag class is concerned with drawing the classes to the canvas. It implements a draw method which is used to output a diagram which is contains all methods and attributes that the user input as well as making

sure that it fits well with box.

ObjectData: ObjectData is an interface which allows its implementing classes to effectively take the same type as it. This means that an instance variable called data of type ObjectData can be stored in the Drawable class allowing a Drawable object to store specific data.

RelationshipDiagData: This class stores all the data associated with a Relationship. This includes the relationship type and its multiplicities. This can then be accesses from the RelationshipDiag object to output the data onto the Canvas. It is also useful when it is output into an XML file.

ClassDiagData: ClassDiagData stores all the information about a single class within the diagram. This class contains all the information about a particular class, including what its instance variables are, what its methods are, what types they are, what methods its parameters have etc.

Now we have justified the class diagram, we can look at the other UML design solutions we have implemented from state diagrams to sequence diagrams. Each of these provided very useful in terms of looking in more detail at the operations and commands of the program we will be implementing. We chose to use a wider spread of diagram types to allow us to plan our program in more detail, which will make it easier to cover the key objectives more thoroughly.

Looking at the state diagrams, used to show the operations of creating classes and relationships, provided us a simple way of designing the steps and loops of performing these methods. The first which was creating classes to be added to the canvas, illustrated the program firstly opening the ‘add class’ operation, then getting the data from user input, followed by displaying the canvas. We added a loop back from the last step of this state, as our program would need t be able to allow the addition of multiple classes. This is a similar story with the second of the state diagrams which shows the simple operations behind the addition of relationships. Although simple, the state diagrams helped illustrate how to objective of adding multiple classes and relationships to diagrams drawn in our program could be achieved.

The activity diagram followed the state diagrams in the design section, and along with going into furtherdetail from the state diagram, these allowed us to show the interaction between the user and program, meaning we could see where the user would need to have control for input of data, where the program would then deal with it. The first of the activity diagrams showed adding classes alike the in the state diagrams but using the cascading platform of this diagram gave us more of an ordered set of operations. The fist of these diagrams allowed us to see the various points in the addition of a class operation where users input would be needed, and what the different pieces of user input was required. The second more detailed activity diagram was an expansion of the adding relationship and also the removal of classes/ relationships. Again using the cascading layout of an activity diagram, it allowed us to see where user interaction would be required and what operations would need to run once the user has chose that operation. E.g. once the user has selected the ‘add relationship’ the system should then check if the user has selected two classes before adding. If not, then this loops back to wait for the user to select two classes.

Sequence diagrams were next in the design section of the project, and again helped illustrate the actions taken out by the program based on an operation the user could begin. We wanted to show the operation taken out by the program when file exporting and saving and loading a canvas of classes and relationships. The first of which involved the order in which the methods were performed from getting the information of the vectors and arrays of classes/ relationships to converting this to writable data, to writing this to file. The sequence diagram helped identify again the order in which performing tasks should be carried out by the program and user alike.

Test Plan

ID	Description	Inputs	Expected Outputs	Pass/ Fail	Comments
1	Create a Class	Enter just a class name	A class diagram with the specified input is added to the canvas.	P	
		Entering a single attribute		P	
		Entering multiple attributes		P	
		Entering a single method		P	
		Entering multiple methods		P	
		Entering methods with parameters		P	
		Create attributes with different types and access modifiers.		F	Attributes could be set to type void. Access modifiers were not correct for type “no modifier”. **Corrected**
		Create methods with different types and access modifiers.		F	Access modifiers were not correct for type “no modifier”. **Corrected**
2	Classes can be moved	Left clicking the mouse on a class and dragging it.	The class diagram moves.	F	Class only drags if you drag the top left corner of the class. **Corrected**
3	Removing a class	Click the remove class button and then click on a class	The class is removed	P	
		Removing a class also removes its joint relationships	The class an relationship are removed	P	
4	Create a relationship	Selecting each of the four relationship types in turn	All relationship types are drawn correctly	F	Composition drawn as aggregation and visa versa. Inheritance drawn as association and visa versa. **Corrected**
		Entering suitable multiplicities into the multiplicities box.	Multiplicities are drawn correctly	P	
		Entering a suitable variable name into the variable box.	Associated variables are drawn correctly	F	Associated variables are not drawn at all. **Corrected**

5	Resetting the Canvas	Clicking the reset canvas button.	Canvas is cleared of all diagram data.	P	
6	Exporting diagram to Java code	Creating a class and attempting to export	Diagram is successfully exported to code	P	
		Creating two classes and a relationship between them and exporting it.	Diagram is successfully exported with associated variables	P	
		Create a class that contains methods which define parameters	Diagram is successfully exported.	P	
7	Saving the diagram	Create a class diagram and click the save diagram button	Diagram is exported to xml format	P	
8	Loading the diagram	Click the load project button and select a valid xml file	Diagram is successfully loaded into the program	F	Application fails to load the XML outline into the application. **Corrected**
9	Undo	Adding objects to the diagram and then clicking the undo button.	Objects removed from the diagram in turn.	P	
10	Changing Colours	Using the edit menu to change background colour	Background colour gets changed to the one the user selected	P	
		Using the edit menu to change class colour	Class colour gets changed to the user selected value.	P	
		Using the edit menu to change the font colour of the diagram	Font colour gets changes to the user selected value	P	
11	Exiting the program	Selecting file> exit closes the application	Application closes	P	
11		Pressing CTRL+Q exits the application		P	
11		Clicking the close window button closes the application		P	
11		Closing or cancelling child dialogues doesn't exit the program	Application doesn't close	F	Using the X button on the Relationship dialogue window causes the application to exit. **Corrected**

Running Our Program

Command Line:

We have provided the necessary Windows 'Batch' and Linux 'Shell' scripts to automate the processes of compiling/running our application.

Simply run 'build' and 'run' scripts for your platform located in the top directory of our project submission.

Otherwise, the project can be built manually from the 'src' directory, using the "javac" command.

Jar File:

We have also supplied an execution-able '.jar' file which can be run by simply double-clicking on the file. However we have identified that it is "temperamental", and as a result may fail to run, however due to time constraints, we were not able to identify the source of the problem.

Evaluation

Year: 2011 - 2012
Module: Software Development – CS12420
Assignment Number: 1
Assignment Description: Design then Code with Blue-J
Worth 20% of final mark for this module
How many hours (approx.) did you spend on this assignment? 120

Expected Letter Grade: B (2:1) (76/100)

And why?

1. 'Quality of code & Javadoc commenting' (Worth 5/100 marks):

In this project we believe that this criterion has been fulfilled to a high level. Throughout the project the source code has been correctly formatted (using appropriate indentation where necessary) and to produced informative and clear 'JavaDoc' documentation that is easily readable by anyone who may want to view our code at a later date. We therefore propose a mark of 4/5 for this criterion.

2. 'Degree and correctness of the basic implementation' (Worth 40/100 marks):

After reviewing the project specification, we are confident that our solution fulfils every main requirement specified. Our program allows the user to create full class diagrams using interactive tools and dialog boxes, including adding classes (complete with variables and methods of multiples types with the function of adding parameters to those methods), and adding relationships (of many types) between those classes onto a fully interactive canvas that allows the user to move and adjust the diagram as they see fit.

The program also provides the function to automatically export all the class objects in the class diagram into '.java' files that contain the skeleton Java code of those objects and also ensures that any relationships between classes are represented correctly in Java code (using ArrayLists if necessary). The user is also able to clear the canvas.

We have also fulfilled the non-functional requirement of making the GUI of our program look fairly similar to that of the 'BlueJ' GUI due to layout and style of the program looking very similar to the layout of BlueJ in many ways. With all this in mind, we propose a mark of 34/40.

3. ‘Design and analysis’ (Worth 15/100 marks):

We feel we have thoroughly analysed the project specification, and as a result have identified the key objectives in order to meet that specification. We were therefore able to produce a range of UML diagrams including Use Case, Class and Activity diagrams that accurately represent our design, and detail the implementation of the proposed solution design, as well as providing clear descriptions for each. We also attempted to justify why we felt our designs were the best way to fulfil the specification objectives.

On reflection however, we feel that we should have spent more time ensuring that all the specification objects had been identified BEFORE beginning designing our solution and coding, as this would have ensured we did not waste time having to re-engineer our design to accommodate more objectives solutions at a later stage. We therefore propose a mark of 12/15 for this criterion.

4. ‘Testing’ (Worth 15/100 marks):

As a group we are happy with the amount of testing that was performed on our program, and the evidence that has been collated to show whether the tests were successful or not. Thorough testing was carried out on all core functionality of the program, including testing of practically all aspects of the GUI and Data Model layers of the software. However, we feel that due to lack of time, we did not collect screenshots, which would have been useful to support our testing; however we are happy that we have provided evidence for the most fundamental tests. Propose a mark of 12/15.

5. ‘WOW Marks’ (Worth 20/100 marks):

After completing the core functionality of the program, the group worked hard to include useful additional functionality and features to make the program as useful (and professional) as possible in the time available. We therefore decided to implement a fully working ‘Save/Load’ feature that allows the user to save the entire project they are working on to an XML file, and the load that same file back in to continue at their leisure. We felt this was an extremely useful feature and was very important to add to the program.

We also added the intuitive feature of allowing users to ‘click’ on and ‘drag’ diagram objects around the drawing canvas as they required and in addition developed a complex ‘line drawing’ algorithm that automatically re-draws relationship lines between classes as the user moves the objects around the canvas in a way that provides a line that is the shortest distance possible, without overlapping any of the objects on the canvas.
We propose a mark of 14/20 for this criterion.

What did you learn?

We have all learned a tremendous amount in this project. The most significant skill we have developed through all of us is learning to work together and communicating with each other as a team, to produce a good result. We have all learned how to delegate different tasks equally out to each team member while ensuring that the different strengths of team members are utilized when solving problems and completing specific tasks.

In terms of technical skills we have learned how the ‘Swing’ framework can be used to create complex and intuitive user interfaces (e.g. MousePositionListeners & SpringLayout) and we now also have an appreciation of how ‘abstract’ classes can be very useful when it comes to creating methods and variables that need to be used over more than one type of object or class.

In conclusion, we feel overall that we all worked very well as a team, but by doing this group project have developed better team working and communication skills that are going to be vital in the future when we work in groups to design other software solutions.

Connor Goddard Self Evaluation

I have been very happy with the team I was placed into and the solution we have created as a result of working together. My main responsibilities in the project included coding the majority of the GUI for the project including working on the line drawing algorithm and some of the input dialogs. I was also in charge of linking all the separately coded parts of the project by my team members together to create the full program. In addition I have written this evaluation.

Samuel Jackson – Self Evaluation

My experience from the group project has mostly been a positive one. I've had no previous experience designing and implementing an application with other people involved and little experience developing an application of this magnitude. My main responsibility within the project was to come up with the initial class diagram for the project. I also gave a fair bit of coding support and was responsible for storing data to be drawn to the screen, XML encoding and decoding and the ParamWindow, among other things.

Apart from the technical challenges involved with this assignment (in particular, I think we can all agree that drawing relationships was defiantly a big one), I found that working in a group was relatively easy on the whole. The only challenges I found were letting other people be responsible for different parts of the code, when you are normally used to having absolute control over the design and implementation. I think that being able to let others be handling different parts of the application will defiantly be something that I take away from this project.

Another challenge that I think our group faced was the fact that I was often difficult for more than one person to be editing to code at any one time. This meant that I occasionally became difficult to keep track of what code version was currently begin worked on. It also meant that only small parts of the code could be worked on outside of the main application. Despite this however, I feel that we have managed to produce a decent application that meets all the requirement s laid down by the specification and I am very pleased with the result.

Craig Heptinstall Self Evaluation

Through the duration of this project, I have learnt and contributed mainly to the Swing section of the code, for instance using buttons, textboxes, combo boxes and layout managers which I put to use in creating the input dialogs for both relationship and class objects to appear on our canvas. This practice has added significantly to my java skills also in a way to use more organized code, and the way in which the code as packaged. My responsibilities within this project have involved and included creating the input class dialog box, input for the relationship dialog, and the integral design and analysis for our program. By doing this, the entire group was able to roll out our finished program based on the design plans. I was also responsible for producing this final documentation from contributions from the other group members.