

Hasan Al-Habbobi 216315428  
Sharujan Rajakumar 216376410  
Alain Ballen 216341703  
Ahmed Hagi 215043896

## DESIGN DOCUMENT - GROUP 5

## **Purpose:**

The purpose of this document is to provide a high-level structure of our software system through the use of class and sequence diagrams. This document will in turn allow an experienced developer that is unfamiliar with the system to get a clear idea of the system's design (ideally also the rationale for the design).

## **Table of Contents**

### **1.0 = Classes and Methods**

#### **1.1 - Important Classes and Methods**

#### **1.2 - How Important Classes & Methods Interact with each other (includes Class Diagram)**

### **2.0 = Objects**

#### **2.1 - Important Objects (that get created at runtime)**

#### **2.2 - How the different Objects are connected to each other (includes Sequence Diagram)**

### **3.0 = Rationale and Scenarios**

#### **2.1 - Rationale (why did we choose this design/implementation)**

#### **2.2 - Maintenance Scenarios**

---

## **1.0 = Classes and Methods**

### **1.1 - Important Classes and Methods**

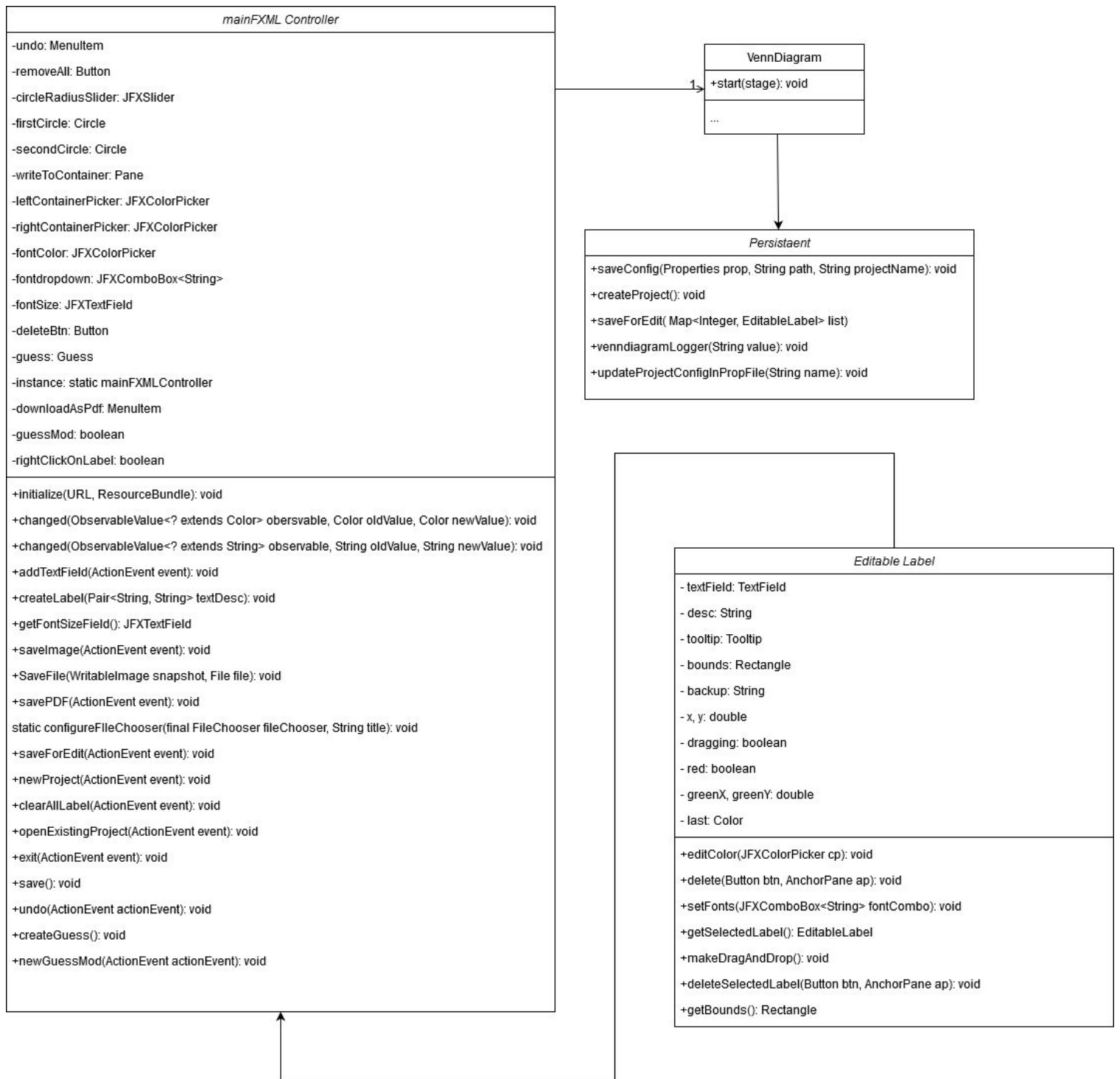
<b>Important Classes</b>	<b>Important Methods</b>
VennDiagram	start(Stage stage) throws Exception: void
mainFXML Controller	initialize(URL url, ResourceBundle rb): void changed(ObservableValue<? extends Color> observable, Color oldValue, Color newValue): void changed(ObservableValue<? extends String> observable, String oldValue, String newValue): void addTextField(ActionEvent event): void createLabel(Pair<String, String> textDesc): void

	getFontSizeField(): JFXTextField saveImage(ActionEvent event): void SaveFile(WritableImage snapshot, File file): void savePDF(ActionEvent event): void static configureFileChooser(final FileChooser fileChooser, String title): void saveForEdit(ActionEvent event): void newProject(ActionEvent event): void clearAllLabel(ActionEvent event): void openExistingProject(ActionEvent event): void exit(ActionEvent event): void save(): void undo(ActionEvent actionEvent): void createGuess(): void newGuessMod(ActionEvent actionEvent): void
<b>Guess</b> (inside mainFXMLController )	getOverLapPercent(): double getLeft(): String getMiddle(): String getRight(): String
<b>EditableLabel</b>	editColor(JFXColorPicker cp): void setFonts(JFXComboBox<String> fontCombo): void delete(Button btn, AnchorPane ap): void deleteSelectedLabel(Button btn, AnchorPane ap): void getSelectedLabel(): EditableLabel makeDragAndDrop(): void getBounds(): Rectangle
<b>Persistent</b>	saveConfig(Properties prop, String path, String projectName): void createProject(): void saveForEdit( Map<Integer, EditableLabel> list) venndiagramLogger(String value): void updateProjectConfigInPropFile(String name): void
<b>Values</b> (inside Persistent)	setText( EditableLabel label): void getText(): String setFont(EditableLabel label): void getFont(): Font

## 1.2 - How Important Classes & Methods Interact with each other (includes Class Diagram)

The following is a class diagram of how the important classes and methods mentioned above interact with each other (their relationship).

### CLASS DIAGRAM



## 2.0 = Objects

### 2.1 - Important Objects (that get created at runtime)

Below is a list of different objects that get created at runtime throughout different scenarios/aspects of the program.

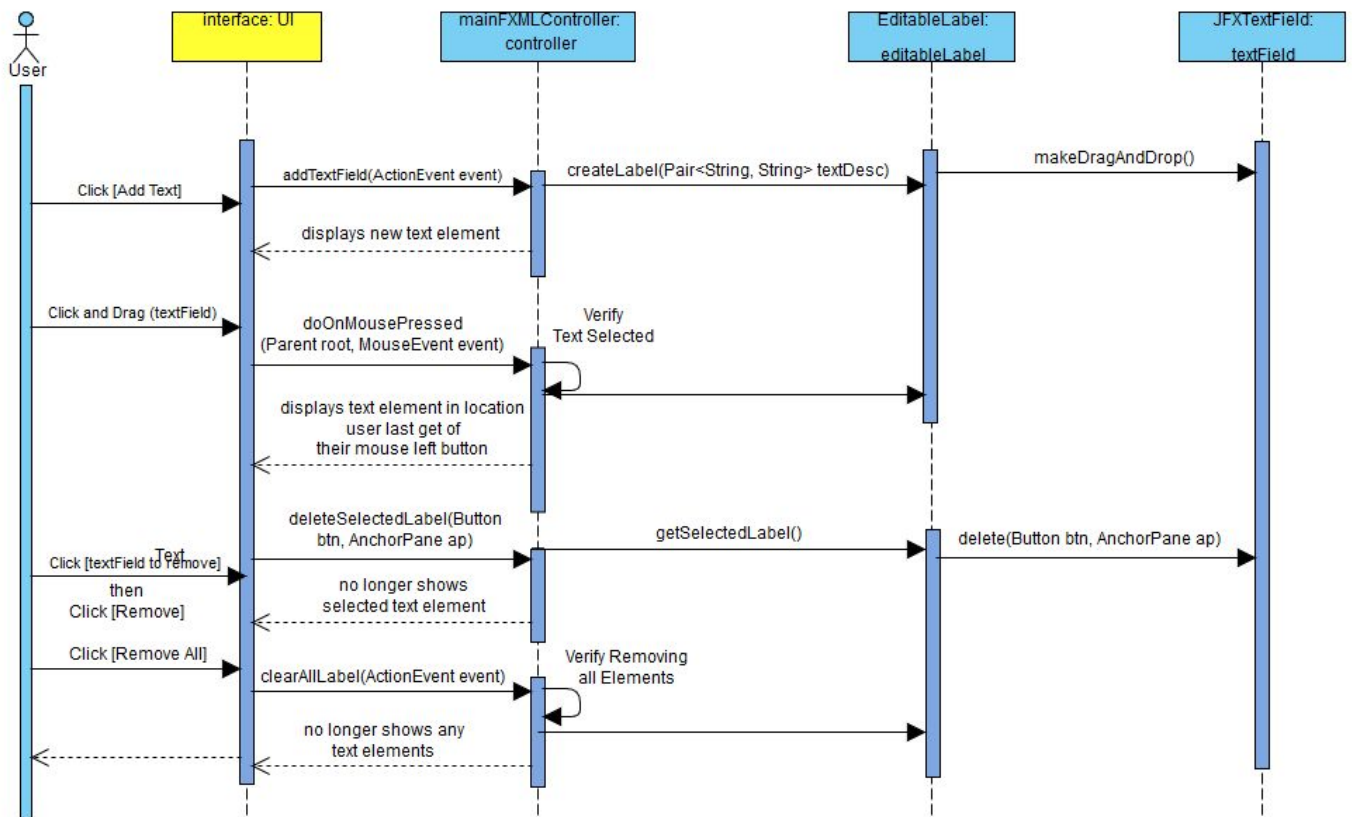
Scenario	Important Objects Created
<i>Adding and Dragging and Removing Text</i>	Interface: UI mainFXMLController: Controller EditableLabel: editableLabel TextField: textField
<i>Editing Text Properties (Font, Size, Color)</i>	Interface: UI mainFXMLController: Controller AddTextFields setFonts
<i>Saving Venn Diagram</i>	Interface: UI mainFXMLController: Controller
<i>Downloading Venn Diagram (as a pdf/png)</i>	Interface: UI mainFXMLController: Controller configureFileChooser saveImage

## 2.2 - How the different Objects are connected to each other (includes Sequence Diagram)

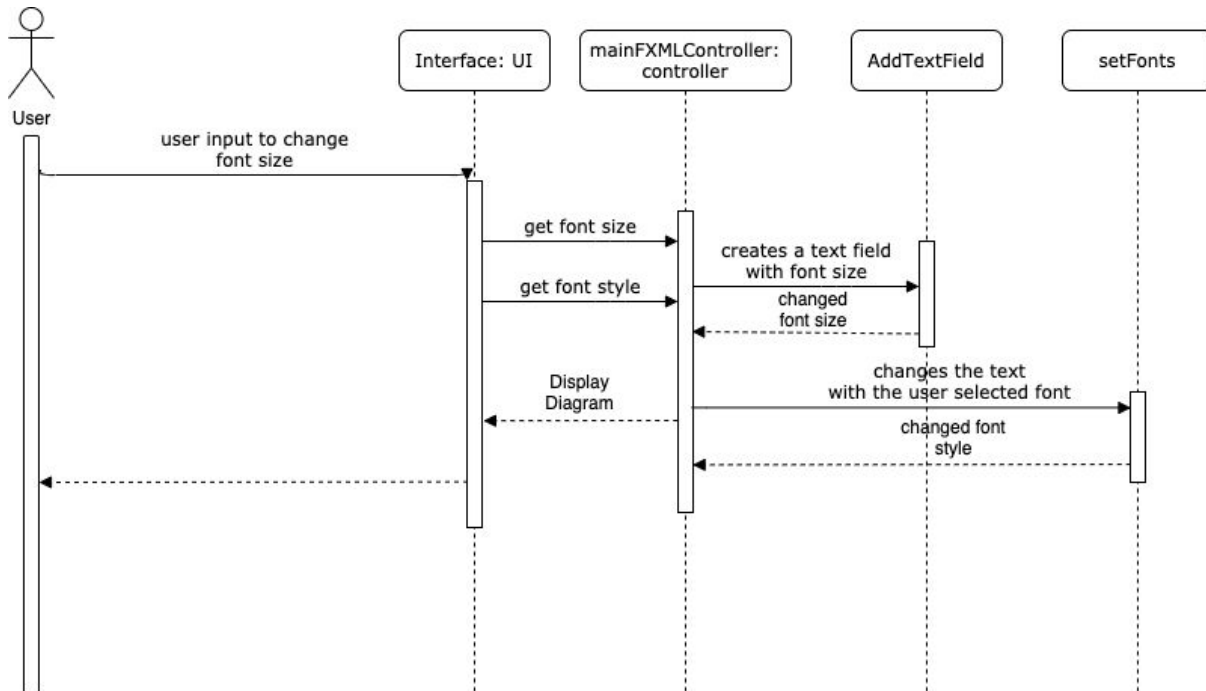
Below are different sequence diagrams each describing a different scenario or aspect of the program. Specifically they outline the different objects created and how they are connected to each other (in order to carry out the actions of a given scenario).

### SEQUENCE DIAGRAMS

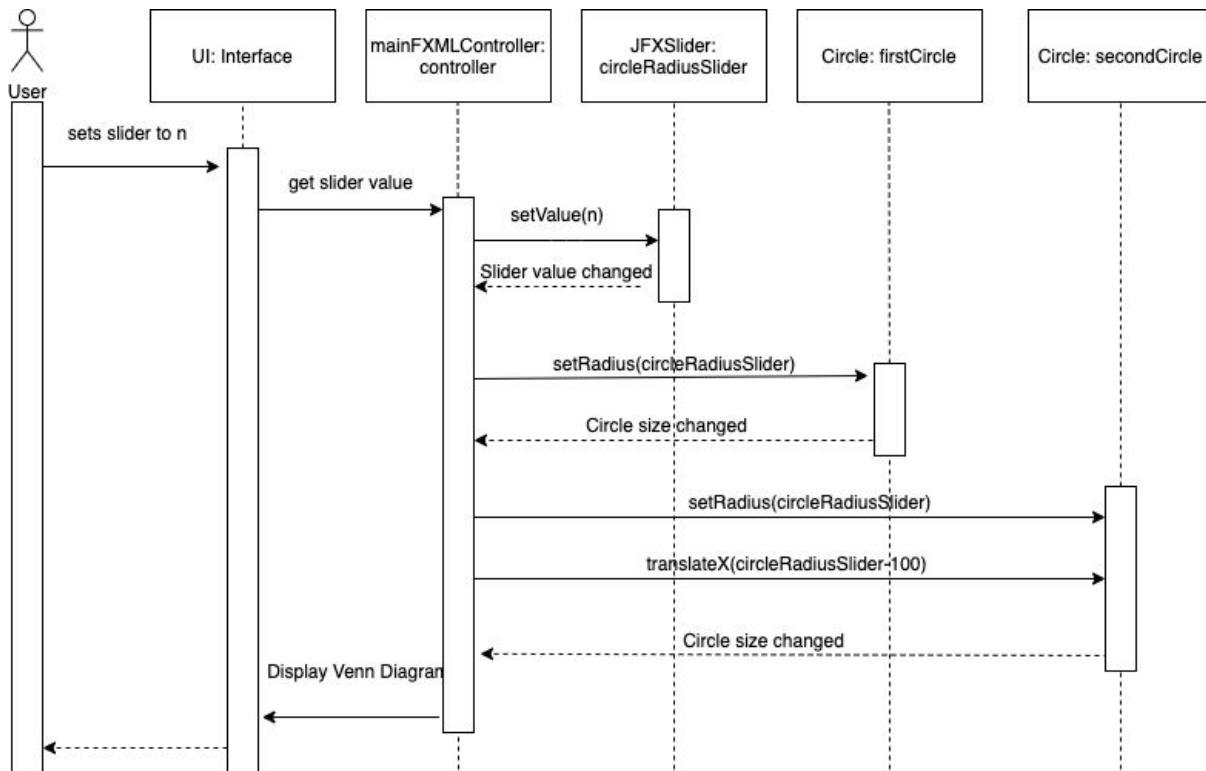
#### *Adding and Dragging and Removing Text*



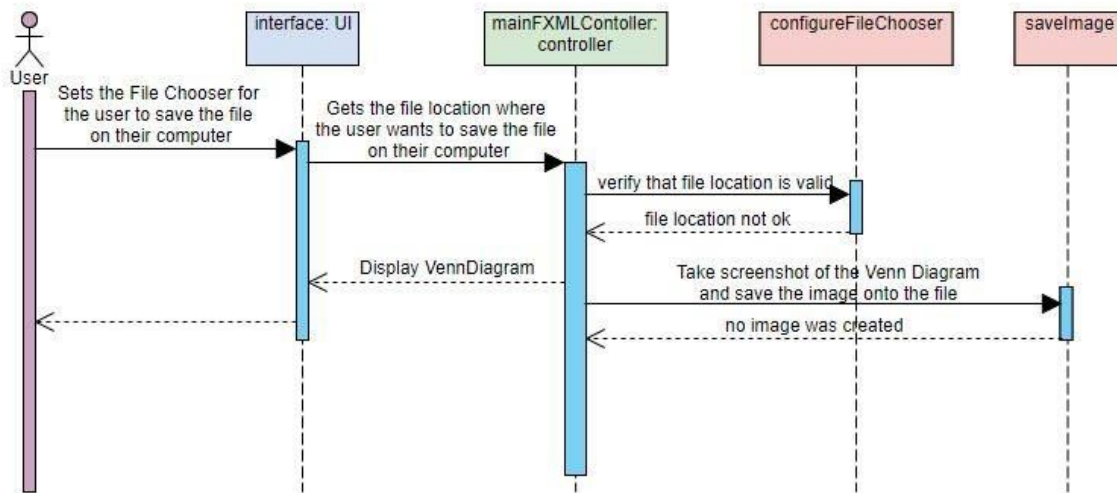
## Editing Text Properties (Font, Size, Color)



## Changing Container Size



### Downloading/Saving Venn Diagram (as a pdf/png)



## 3.0 = Rationale

### 2.1 - Rationale (why did we choose this design/implementation)

*Rationale/Explanation for why we:*

1. *Classified the Classes and Methods in 1.0 as Important*
2. *Classified the Objects in 2.0 as Important*
3. *Chose to implement this design in general*

---

#### 1. *Classified the Classes and Methods in 1.0 as Important*

The classes and methods we chose to implement are important because they each perform a number of critical tasks. For example the mainFXMLController manages all the containers and text fields. Inside of it are methods each one responsible for a certain task. getRadius for instance is responsible for getting the radius of the container.

Overall they are important because without them the fundamental principle of a Venn Diagram could not be achieved (e.g. users wouldn't be able to add text or edit or drag it etc).



## **2.     Classified the Objects in 2.0 as Important**

We have key objects which are the key components of the program : the user interface (UI) and the mainFXMLController objects. The user interface object is responsible for the GUI buttons the user sees and interacts with (front end). The mainFXMLController is responsible for managing all the backend work of the program. The mainFXMLController can create other objects which are also important such as textfield objects and selectionHandler objects to manage the text fields selection etc. In that regard these are all important for they all address key aspects of the program.


## **3.     Chose to implement this design in general**

In general a Venn Diagram Application can be broken down into a number of aspects: its user interface that deals with front end buttons and controls as well as its main controller which deals with backend and saving etc. Another way of looking at a Venn Diagram application is two aspects: the containers and the text. We applied these two principles/visions for our program and took it a step further with methods responsible for more specific aspects (e.g adding text vs deleting text, editing text, saving, etc). By doing this we were able to split up which member will program which method and we were able make our main code less cluttered and easier to read.

>>> Maintenance Scenarios >>> next page >>>

## 2.2 - Maintenance Scenarios

Below are a number of maintenance scenarios along with their respective descriptions/explanations (each of which explain what needs to be changed).

Maintenance Scenario	Description
What does one need to change to support <b>Container Shape Selector</b> in the Venn Diagram?	<p>Allow user to change the shape of their container to other shapes instead of just circle</p> <p>One will need to add a “change shape” button and program it so that it allows the user to choose from a number of shape options in a drop down menu (e.g. square, circle, triangle, etc). There should be premade layouts of various shapes of various numbers (see next Maintenance Scenario) that can be loaded based on the shape and number of shapes selected by the user</p>
What does one need to change to support <b>Edit Text Margins</b> in the Venn Diagram?	<p>Allow user to adjust text margins so that text can fit narrower/wider column styles based on the users needs/desires</p> <p>One will need to add a text margin slider similar to that seen on word/docs:</p>  <p>such that whenever a text element is created/selected a slider of its margin appears allowing the user to set the margins of the text. This will require new variables to be created to store the margins of each text element.</p>
What does one need to change to support <b>Extra Sets/Containers</b> in the Venn Diagram?	<p>Allow user to have more than 2 containers</p> <p>One will need to add a “add set” button and program it so that it creates a new container and loads the 3 container layout. There should be a total of 5 container layout options the user can select.</p>