Final Project Report

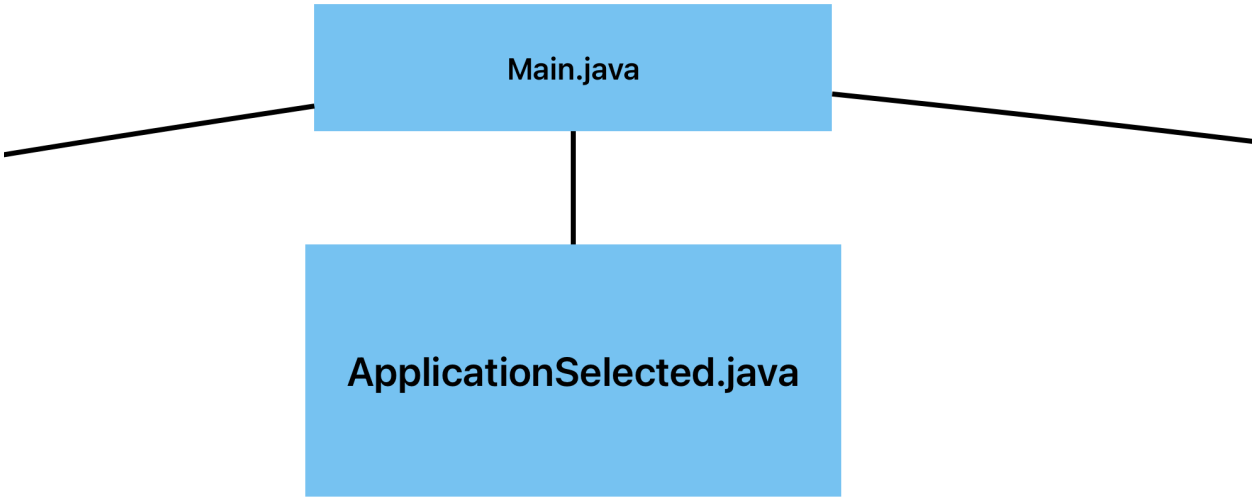
- **Namaz Application**

2022315800
Javokhir Zokirov

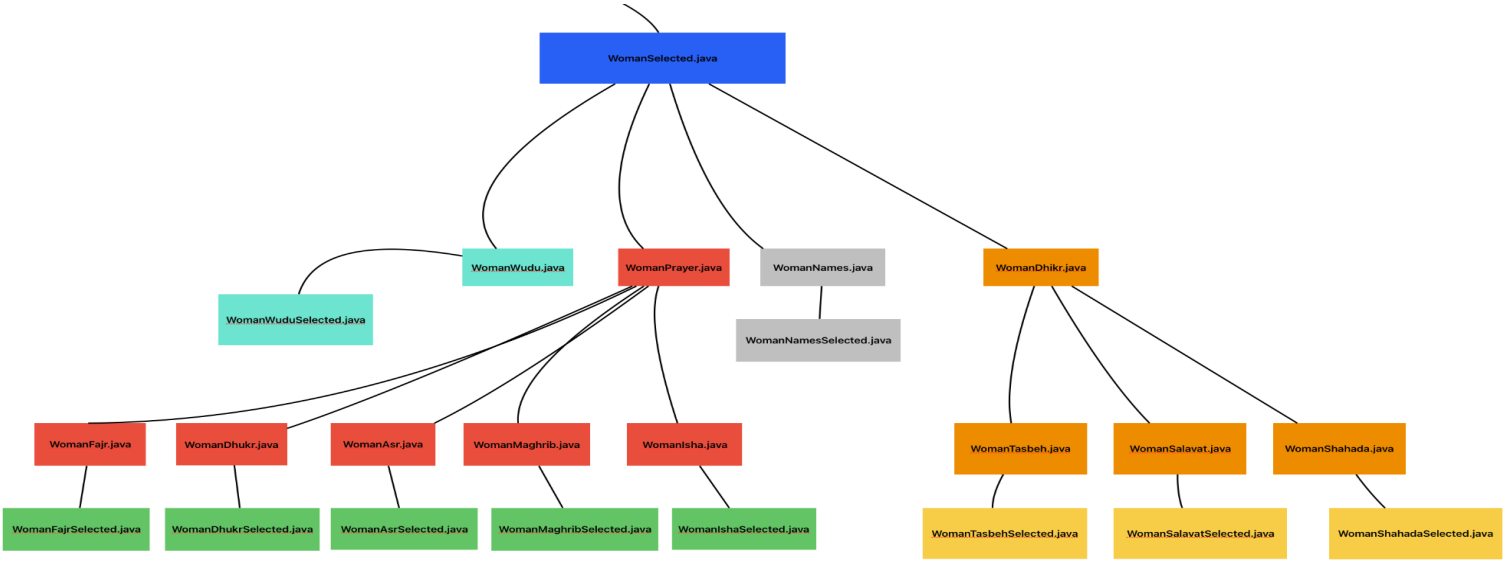
- ## Project Purpose

Many people ask me about Islam, sometimes they ask me questions that I don't have an answer. I decided to make a Namaz Application that will answer all questions and give more information about Islam.

- ## Draw the logic flow of the program

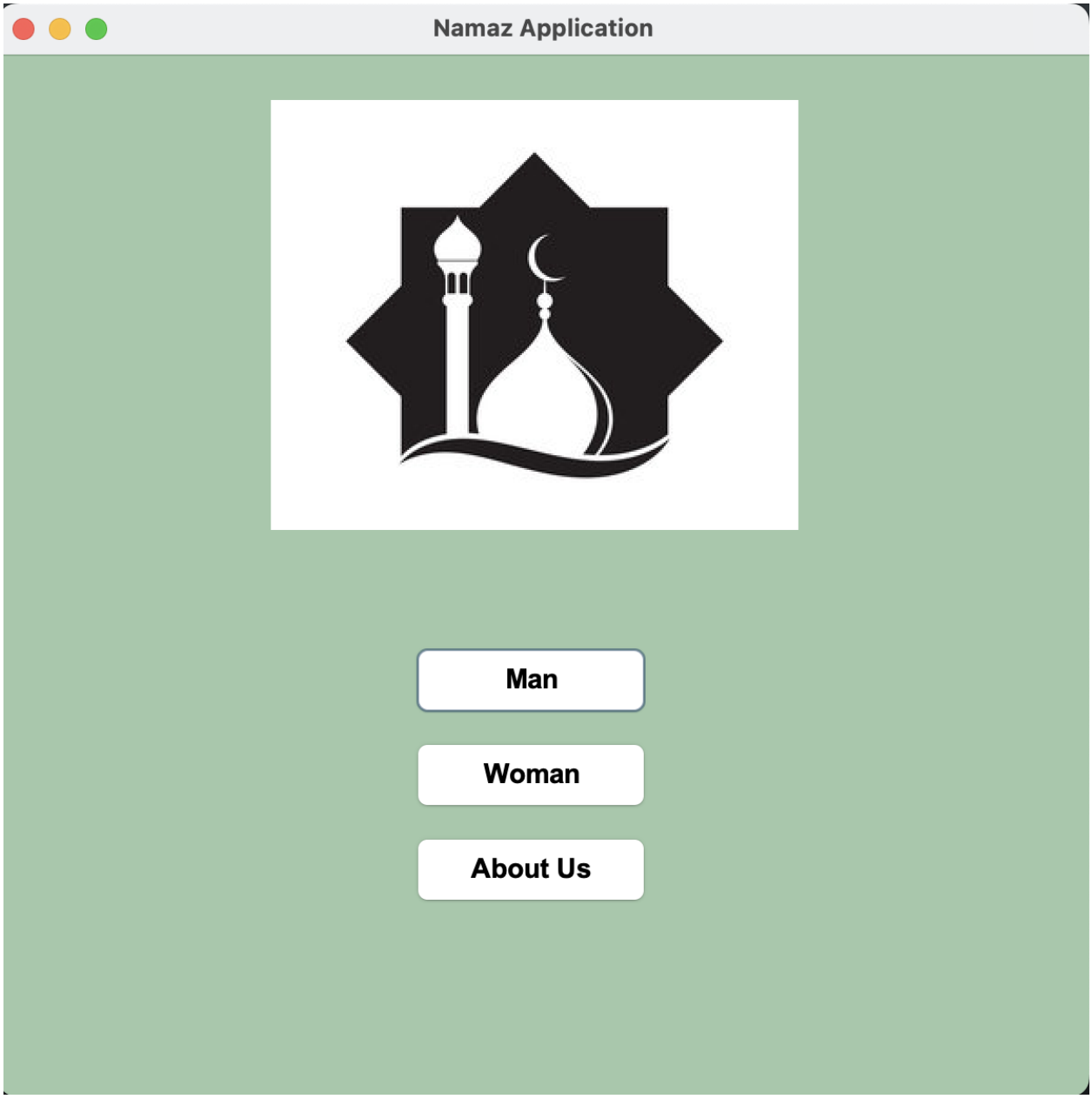

Main.java
ApplicationSelected.java


ManSelected.java
Wudu.java
Prayer.java
Names.java
Dhikr.java
WuduSelected.java
NamesSelected.java
Fajr.java
Dhukr.java
'Asr.java
Maghrib.java
'Isha.java
Tasbeh.java
Salavat.java
Shahada.java
Adhan.java
Iqamah.java
FajrSelected.java
DhukrSelected.java
'AsrSelected.java
MaghribSelected.java
'IshaSelected.java
TasbehSelected.java
SalavatSelected.java
ShahadaSelected.java
AdhanSelected.java
IqamahSelected.java



This flowchart indicates the overall flow of the program. All the tricky parts are inside of ApplicationSelected.java, I got the idea for this from previous assignments, user will write(optional) suggestion about this program, when filling out the application there is a requirements for each fields, if all the requirements are met user can send us a suggestion regarding our program

- **Provide Screenshots for each screen with brief description**



This is the Main part of my app. As you can see there are options for Man and Woman, in Islam there are different rules for Man and Woman in some cases. Also there is an About Us button that users can read about us and send us feedback.

Namaz Application

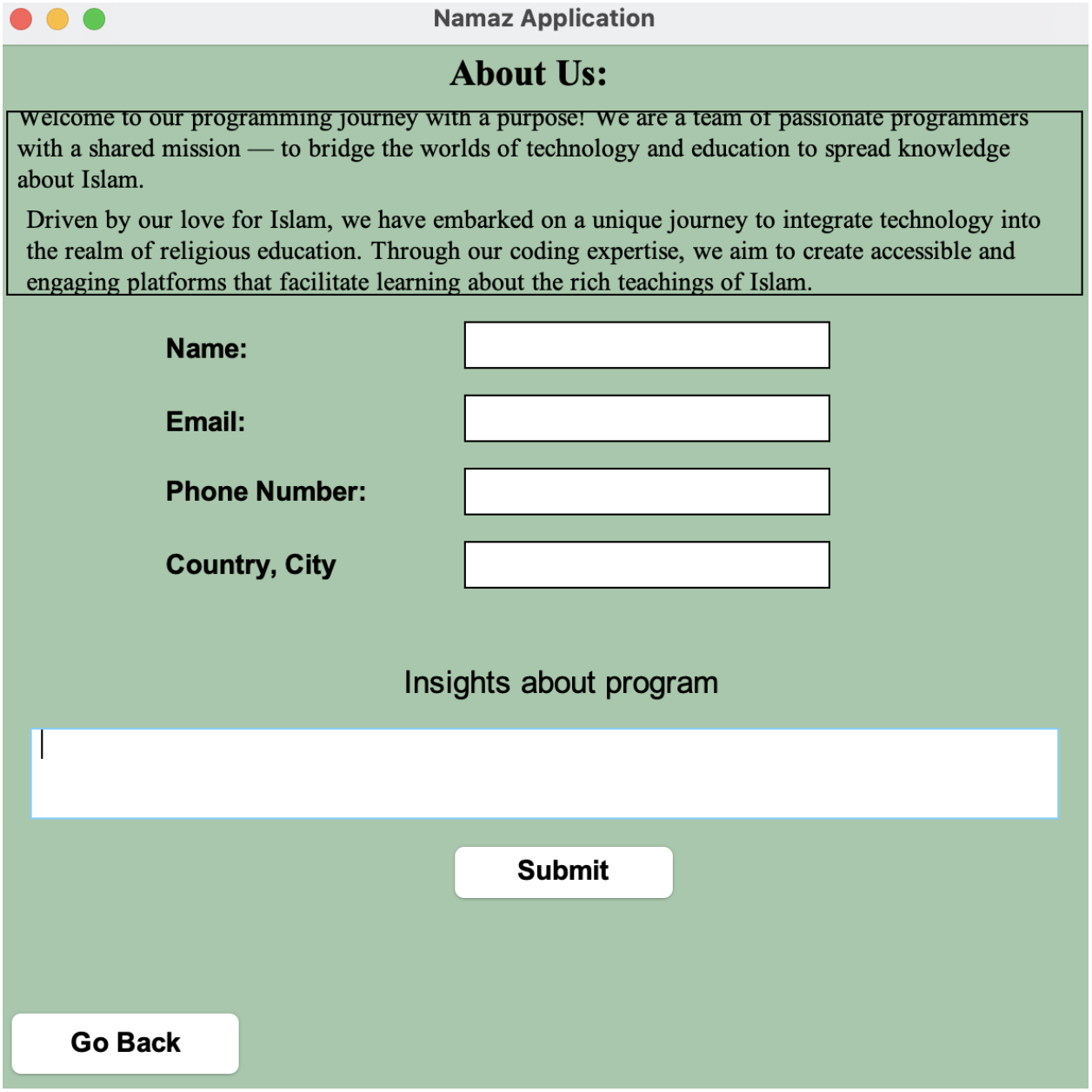
Wudu

5 Prayers

99 Names

Dhikr

Go Back



As you've seen in the video you know how this part works, you don't meet the requirements you cannot send us a feedback.

- **Explain the code of the main functionalities**

```
private void validateAndHandleFields() {
        List<String> errors = new ArrayList<>();
        int errorNumber = 1;

        if (!validateName(nameText.getText())) {
            errors.add(errorNumber + ". Invalid Input. Enter your name and surname");
            errorNumber++;
        }

        if (!validateEmail(emailText.getText())) {
            errors.add(errorNumber + ". Invalid Email. Enter a valid email address(name@name.name)");
            errorNumber++;
        }

        if (!validateNumber(numberText.getText())) {
            errors.add(errorNumber + ". Invalid Number. Enter number only(01030084056)");
            errorNumber++;
        }

        if (!validateAddress(countryText.getText())) {
            errors.add(errorNumber + ". Country format should be: Country, City");
            errorNumber++;
        }

        if (!validateStatement(insightArea.getText())) {
            errors.add(errorNumber + ". Enter at least 50 characters");
            errorNumber++;
        }



        //based on error it shows how many errors user made
        if (!errors.isEmpty()) {
            showErrorMessages(errors);
        } else {
            // if none, shows success message
            showSuccessMessage();
        }

    }
```

As you can see this part of the code in the ApplicationSelected.java does all the job for the About Us section.

```
private void showErrorMessages(List<String> errors) {
    StringBuilder errorMessage = new StringBuilder("Please fix the following errors:\n");
    for (String error : errors) {
        errorMessage.append("- ").append(error).append("\n");
    }
    JOptionPane.showMessageDialog(getApplicationSelectedFrame(), errorMessage.toString(), "Input Errors", JOptionPane.ERROR_MESSAGE);
}

private Component getApplicationSelectedFrame() {
    // TODO Auto-generated method stub
    return null;
}
private boolean validateName(String fullName) {
    String[] nameParts = fullName.split(" ");
    return nameParts.length >= 2 && fullName.matches("^[a-zA-Z ]+$");
}
private boolean validateEmail(String email) {
    return email.matches("^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}$");
}
private boolean validateNumber(String number) {
    return number.matches("^[0-9]+$");
}
private boolean validateAddress(String address) {
    if (address.length() < 5) {
        return false;
    }
    return true;
}
private boolean validateStatement(String statement) {
    if (statement.length() < 50) {
        return false;
    }
    return true;
}

private void showSuccessMessage() {

 JOptionPane.showMessageDialog(getApplicationSelectedFrame(), "Successfully submitted!", "Success", JOptionPane.INFORMATION_MESSAGE);
 // clears the text field or perform other actions if needed
 nameText.setText("");
 emailText.setText("");
 numberText.setText("");
 countryText.setText("");
 insightArea.setText("");
}
```

For exception handling, I used (if), (else) statements. If a user were to fail to meet the requirement, the program will check the problem and report it.

- **Explain what is included in your project and why it is used**

List of the topics included:

1. Event Dispatch Thread (EDT)
2. Polymorphism
3. Inheritance
4. Validation handling

- Event Dispatch Thread (EDT)

Suppose I did not use multithreading in my code however my program follows the standard Swing practices of updating the GUI components on the Event Dispatch Thread.

- Polymorphism

My program does not use polymorphism directly but there are elements that we can consider as polymorphism.

In my code:

1. ApplicationSelected class:

The validateAndHandleFields() method performs validation on various fields, and it can be considered polymorphic if it's overridden in a subclass with a different implementation.

The showErrorMessages(List<String> errors) and showSuccessMessage() methods could potentially be overridden in subclasses to provide different error handling or success messages.

2. Main class:

The showManSelectedPage(), showWomanSelectedPage(), and showAboutSelectedPage() methods are polymorphic in a sense because they override the same-named methods in the Main class to show different pages based on the button pressed.

I gave examples only from 2 files out of 30

- Inheritance

I have used inheritance almost in every file, for example:

```
public class ApplicationSelected {
    // code

  public class Main {
        // code
```

(**ApplicationSelected** and **Main**) make use of inheritance implicitly through the **extends** keyword, which indicates that these classes are inheriting from existing classes.

- Validation Handling

My code does not explicitly use exception handling through try-catch blocks. Instead,I've implemented a validation approach where the program checks for specific conditions using if statements. In my **validateAndHandleFields**() method, I check various conditions using if statements and build a list of errors if any condition is not met. Later, the program displays these errors to the user. While exceptions are a way to handle errors, my approach of validation using if statements is appropriate for this scenario.