1. **Introduction:**

   (a) 1.1 Purpose of this Document:
   This document will be a detailed analysis of the project I will be working during the summer.

   (b) 1.2 Scope of this document:
   This document covers the core of the system. It includes the code that will do most of the calculations to find suitable schedules, the GUI for the applet, and the requirements for running the applet.

   (c) 1.3 Overview:
   The program is mainly for college students or those just entering college and need to create a schedule for their classes. The program will allow students to create effective schedules for their classes without becoming a burden. Also students will be able to fit their courses around their own day to day schedule, instead of fitting their schedule around their courses.

   (d) 1.4 Business Context:
   The Amp program gives students the chance to experience researching and using resources that are available to them, especially for minorities. It helps students prepare for graduate school.

2. **General Description:**

   (a) 2.1 Product Functions:
   The system should be able to take in multiple classes with different time slots, create a unique schedule out of every possible combination of classes, then display the schedules that have no conflicting time slots. The system should also be able to take in other schedules, such as work or any times the student would not like to have class, and determine whether it will conflict with the course schedule. The system should make it effortless for GMU students to create class schedules that will not interfere with their daily lives. The system should not jeopardize ease of use for performance.

   (b) 2.2 Similar System information:
   The product will be used along side a website that will hold the system. The system will be a web applet. If we can finish the project before the deadline we might be able to turn it into a web application.

   (c) 2.3 User Characteristics:
   Users do not need prior experience to run the software, but should be familiar with how scheduling works and day to day applications.

   (d) 2.4 User Problem Statement:
   In order to use the software, users will need to have prepared the classes they would like to schedule which might be difficult. Users will have to input every single potential timeslot which might become time consuming.

   (e) 2.5 User Objectives:
   Users will want an easy and fast way to input their courses and the individual timeslots. Inputting so many different timeslots can become very tedious, so the GUI would need to lessen this task. We can do this by asking for the class name once, then create textboxes for the users to input different times. A combo box might be best for the minutes. Finally the schedule should be displayed in a manner that is easy to read and comprehend. Somewhat like a calendar.

(f) 2.6 General Constraints:
Since the program will be a web applet it will not have access to system resources, so we cannot save files or run programs on the users computer. We will also need to make the program reusable incase we want to turn it into a stand alone application or a web application.

3. **Functional Requirements:**

(a) System must be able to compute working schedules efficiently and in a timely manner.

i. Description:
The applet should not take long to load the working schedules and display them to the user.

ii. Criticality:
Very critical

iii. Technical issues:
System will have to not only compare each course but also its labs. It will also have to compare the final schedules with other the timeslots the student would not like to have class.

iv. Risks:
If the user inputs wrong data, such as replacing the start time of a course with its end time, the system will not be able to process it.

(b) System should be able to store the data temporarily.

i. Description:
If the user wants to go back and enter another class or change a timeslot, he should not have to enter all the data in again. The user should be able to go back to his previous courses easily.

ii. Criticality:
Very critical.

iii. Technical issues:
The system will have to somehow display the current courses and timeslots in a way that is easy for the user to read and change.

iv. Risks:
none

(c) GUI should display the final results in a manner that is readable.

i. Description:
After the program calculates the working schedules, it should display a chart containing a schedule. As the user looks through the schedules the chart should redisplay the current schedule that the user is looking at.
The chart that is displayed should highlight the classes and the times that the user listed as not wanted.

ii. Criticality:
Very Critical.

iii. Technical issues:
I am very bad with GUI designs.

iv. Risks:
Fitting the chart with a schedule that has a lot of classes, in a predetermined size box.

(d) The program should ask for classes that are optional for the student.

i. Description:
Some classes students are not sure whether they should take them or not depending on their schedule. The system should show any schedule that contains or does not contain the optional classes.

ii. Criticality:
Somewhat critical.

iii. Technical issues:
Applet should have a check box, that users can check if the class is optional.

iv. Risks:
If there are too many optional classes it will greatly increase the time it takes to calculate the resulting schedules.

4. **Interface Requirements:**

(a) 4.1 User Interfaces:
The system will interface with the user mainly through the GUI.

(b) 4.1.1 GUI:

- The graphical user interface will first present the user with a couple of text boxes and combo boxes for the user to input the class name and its timeslots. It will also have 4 buttons, NEXT, PREVIOUS, REMOVE, and ADD, which will add that class to the list.
  - This will also contain check boxes next to the classes. The user can check these if the class is optional.
  - It will also have a PANEL that displays the classes that were added. The user can click on any of these to highlight a class, and change its properties.
- The next phase, the GUI will present the user with similar boxes for the user to input when he would not be available for classes. This phase might have a chart that the user can use to enter when he is unavailable instead of textboxes.
- The third phase, will display the resulting schedule to the student. It will have a PREVIOUS button incase the student wants to change any class.

(c) 4.1.2 API:
To be filled later.

5. **Performance Requirements:**
Since the system will be an applet the user will only need an internet connection. If we make it a stand alone application, then the user will need a java virtual machine.

6. **Design Constraints:**

- If the system will save users schedules for later use, the information cannot be shown to any other users.

- If user is to be asked for any personal information, the system will need a way to obtain proof of the user's permission.
- System does not have access to the user's computer so it will need some other way of storing the user's information and previous schedules. We can do this by using a database or asking the user if he would like to save or download his current work.

7. **Other Non-Functional Attributes:**

   (a) Security will not be a major issue, unless we will be keeping a database of all the users and their saved work. That will not be covered by this SRS.

   (b) Reliability will be a great issue in the system, mainly because many students will be using it to schedule their classes and small mistakes can greatly effect a student's standing.
   System must be able to keep track of the user's progress, incase an error occurs.

   (c) System should also be extensible since it will probably go through a lot of changes. Features and updates should be easy to setup, without having to change to much in the code.

8. **Premiliminary Object-Oriented Domain Analysis:**

   (a) Class Descriptions: (Course)

      i. *CLASS NAME:* Course

      ii. Concrete

      iii. *PURPOSE:* This class will hold a single course. It will also include the course's name, section, and three different timeslots.

      iv. *COLLABORATIONS:*
      - This class holds three TimeSlots. The time slot will hold the actual time and days for each course. Each course can have a lab, lecture, and recitation, so each one will have a seperate TimeSlot.

      v. *ATTRIBUTES:*
      - String name
      - int section
      - TimeSlot lab;
      - TimeSlot lecture;
      - TimeSlot recitation;

      vi. *OPERATIONS:*
      - getTimeSlot(integer type); returns: TimeSlot
        User can ask for the object reference to one of the TimeSlots. The argument specifies which TimeSlot he wants.
      - getLabTime(integer point); returns: integer
        User can ask what time a TimeSlot begins or ends. Point specifies if the he wants the start time or the end time of the TimeSlot and point must be either 0 or 1. "'getRecitationTime and getLectureTime are the same methods.
      - setLabTime(String time, String days)
        Allows the user to set the time slot for a lab. He can input the time in string format (ie. "'12:00PM-2:00PM"').
      - toString; returns: String
        Returns a string containing the name, section, and times for the different TimeSlots. It also prints out whether the object has a lab, lecture, or recitation.

(b) Class Descriptions: (TimeSlot)

i. *CLASS NAME:* TimeSlot

ii. Concrete

iii. *PURPOSE:* This class will hold the start time, end time, and days of a class. It will also hold the time in a string format, for testing.

iv. *COLLABORATIONS:*

- This class will be used by any object which needs to hold a certain time, like the NotAvailabel or Course objects.

v. *ATTRIBUTES:*

- String stringTime
- int startTime
- int endTime
- String days - it might be better to make this a char[].

vi. *OPERATIONS:*

- getDays; returns: String
  User can ask for the days for which this time slot occurs.
- getStringTime; returns: String
  This will mainly be used for testing purposes. This will allow me to check the time of this object in a standard format that is easy to read. (ie. "'12:00PM-2:00PM"').
- convertToString(integer startTime, integer endTime); return String
  This will allow me to send in the time this timeslot starts and ends and will return a string containing the time in a standard time format. The arguments must be in 2400 times though.
- convertToMilitary(String time); returns: void
  This will be the most important method which will allow me to convert the input from the user, which I am expecting to be a string, into two 2400 times. This will not return anything but will set the startTime and endTime attributes.

(c) Class Descriptions: (NotAvailable)

i. *CLASS NAME:* NotAvailable

ii. Concrete

iii. *PURPOSE:* This class will hold the time slot at which the user will not be available. The system will use this object when checking for workable schedules. The system will have to find schedules that do not conflict with this time slot.

iv. *COLLABORATIONS:*

- This class will contain one TimeSlot object. The Schedule object may also contain multiple NotAvailable objects.

v. *ATTRIBUTES:*

- String label
- TimeSlot time
- char[] days
- boolean optional

vi. *OPERATIONS:*

- getDays; returns: char[]

- isOptioanl; returns: boolean
- getStartTime; returns integer
- getEndTime; returns integer

(d) Class Descriptions: (Schedule)

i. *CLASS NAME:* Schedule

ii. Concrete

iii. *PURPOSE:* This class will be where most of the calculations take place. This will hold one schedule, which will contain one of each course the user inputs. This will test that schedule to see if it is not conflicting.

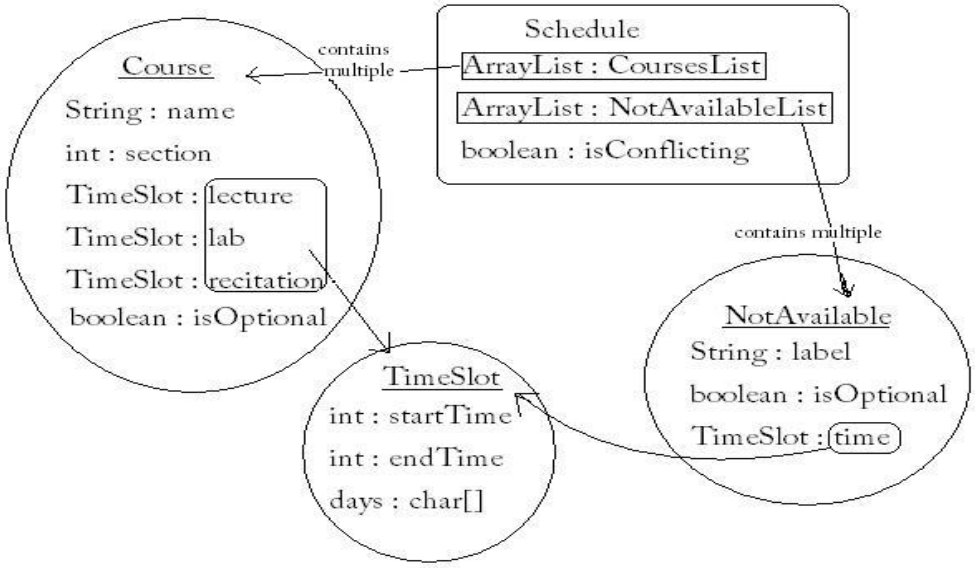
iv. *COLLABORATIONS:*

- This class will have an arraylist of timeslots and an arraylist of courses.

v. *ATTRIBUTES:*

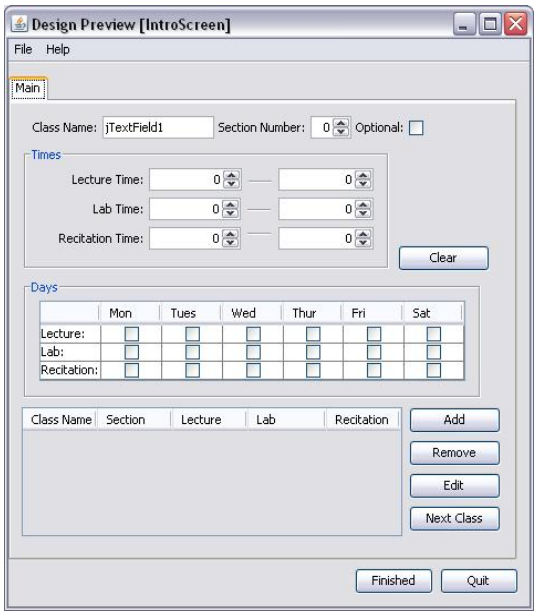
- ArrayList mainList
- ArrayList stretchedList
- ArrayList likeDaysList
- String currentDays
- boolean isConflicting
- ArrayList conflicting

9. **CLASS OVERVIEW:** This is a very brief visual overview of how the classes will interact with each other.



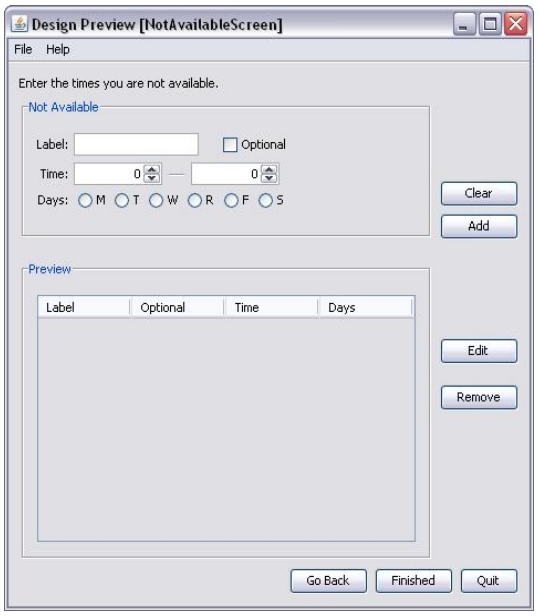
10. **Operational Scenarios:**

- Scenario One:
  User starts the applet and the Intro Screen will be displayed:

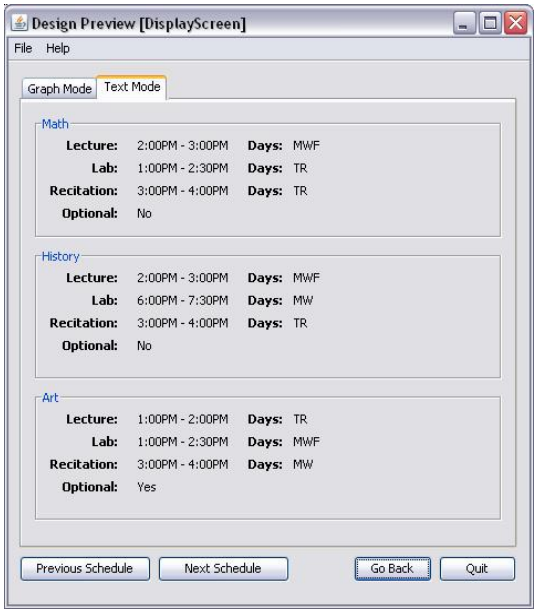


The user fills out the form for one class. He clicks the "'add"' button and the course is added to the bottom table. He adds some more sections for that course. Finally he clicks the "'Add Class"' button which clears the form and creates a tab containing the previous class' information. Like so:

This tab will only be a preview for the user, which will allow him to look at the time slots he has added. For every different class there will be a tab. He then adds some more classes, looks at the different preview tabs, then clicks "'Finished"'. This then takes him to the next page which asks for times that he is not available. User inputs "'Work"' for the label, sets the times and days, then clicks "'Add"' which adds that time slot to the bottom table. The page will look something like this:



User then looks at the table to make sure everything is correct then clicks "'Finished"'. This will calculate the schedules and display the DisplayScreen which will look like this:

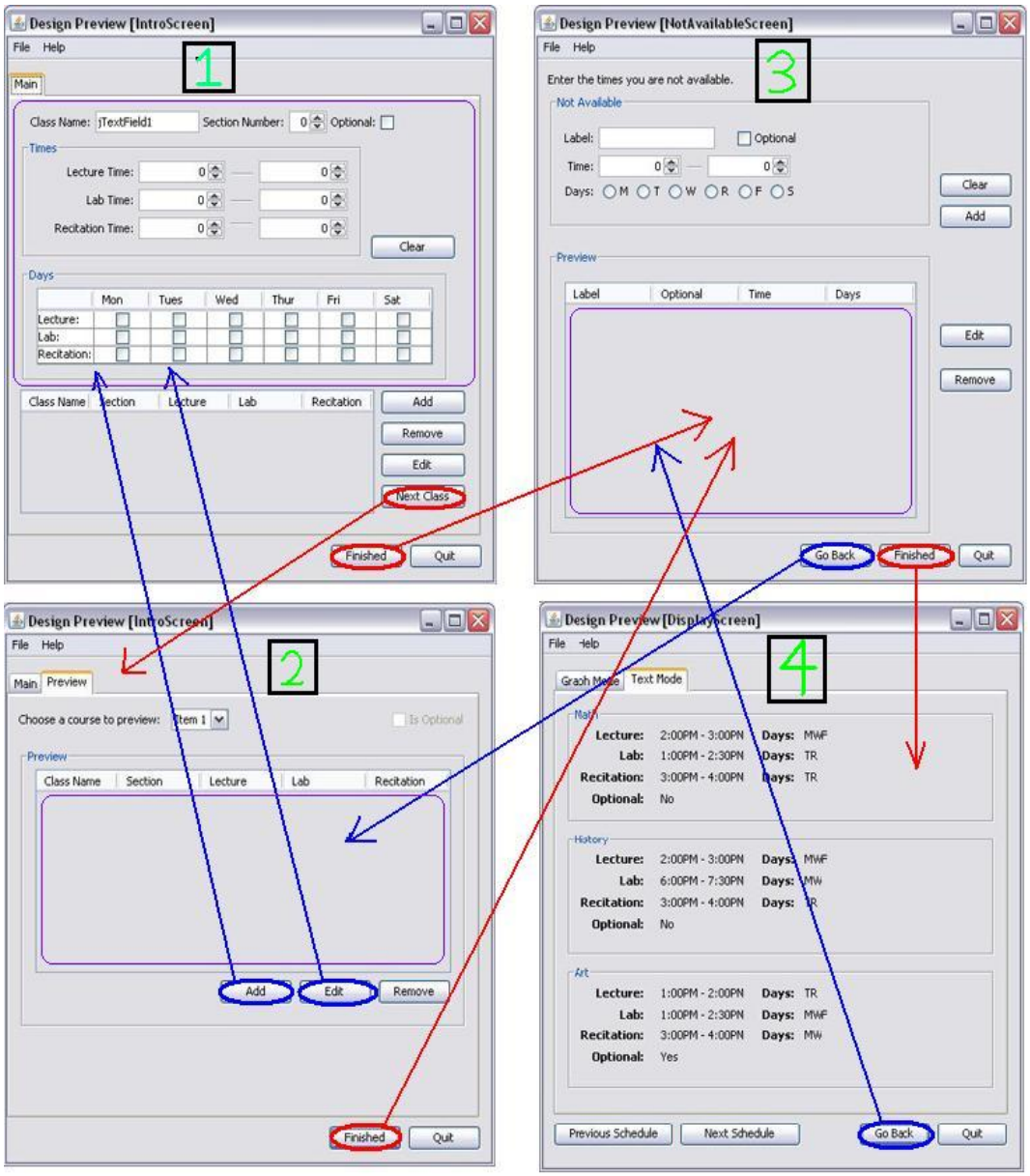


User looks at the first schedule that is shown but does not like it so he clicks "'Next Schedule"' which displays another schedule in the same format. To get a better view the user clicks on Graph mode to get a visual representation of the schedule. After coming to a schedule that he likes he presses "'Quit"' to exit the program.

- Scenario Two:
  User starts the applet and the Intro Screen is displayed. He fills out the form for one class, but half way through he changes his mind so he clicks "'Clear"' to clear the form and begins to add another class. He adds a few sections, which are now dispayed in the table. After checking the table he finds a section that he wants to edit, so he highlights that section and clicks "'Edit"'.

This reloads all the information for that section into the form and deletes it from the table. After the user has edited it he clicks on "'Add"' and it adds it again to the list.
The user clicks "'Finished"' and is met with the NotAvailable page. The user adds some times at which he is not available. He finds that he is available on one of the times he added so he highlights it and clicks "'Remove"' to remove it from the list. When he is done he clicks "'Finished"' which takes him to the DisplayScreen. Here the user can get both a visual and textual representation of his schedules. He then goes through each schedule til he finds the one he wants, writes it down, then clicks "'Quit"'.

- Scenario Three:
  User starts the applet and inputs all the classes and their courses. He looks at the preview then clicks "'Finished"'. He now inputs the times at which he is not available. He clicks "'Finished"'. The system finds no working schedules. The user decides to go back and add another Math course that might fit his schedule. The user clicks "'Go Back"' and he is met with the "'Not Available"' screen. He does not want to change anything here so he clicks "'Go Back"' and he comes to the Course preview screen. From the drop down box he selects the Math class. This displays all the possible courses for that class. The user then clicks "'Add"' and this takes him to the very first page. The page is blank except for the bottom portion which displays the same courses as the ones in the preview. He inputs the information for that course then clicks "'Finished"'. He clicks "'Finished"' again to go back to the Results screen. A schedule is found and it is displayed.
- Scenario Four:
  User starts the application and fills out the first form. He adds all the classes he needs but when he looks at the preview he changes his mind about one of the courses. He highlights that course and clicks "'Edit"'. This returns him to the previous tab, which is the First page. The top portion is already filled out with the information from that course. The course is deleted from the bottom section. The user edits the information he needs to then clicks "'Add"'. He then clicks "'Finished"' twice to get to the Display screen.
- Scenario Five:
  User starts the application and goes fills everything out. When he gets to the last page with the results there are no working schedules. The user is willing to change his personal schedule. He clicks "'Go Back"' which takes him to the "'Not Available"' page. He highlights the time he wants to change and clicks "'Edit"'. The top panel is filled with information from this time slot and it is removed from the bottom table. He changes the information and clicks "'Add"'. When he is done he clicks "'Finished"' and the results page is Displayed. When he is satisfied he clicks "'Quit"'.


Design Preview [IntroScreen]
File Help
1
Main
Class Name: jTextField1
Section Number: 0
Optional:
Times
Lecture Time: 0 0
Lab Time: 0 0
Recitation Time: 0 0
Clear
Days
Mon Tues Wed Thur Fri Sat
Lecture:
Lab:
Recitation:
Class Name Section Lecture Lab Recitation
Add
Remove
Edit
Next Class
Finished
Quit
Design Preview [NotAvailableScreen]
File Help
3
Enter the times you are not available.
Not Available
Label:
Optional
Time: 0 0
Days: M T W R F S
Clear
Add
Preview
Label Optional Time Days
Edit
Remove
Go Back
Finished
Quit
Design Preview [IntroScreen]
File Help
2
Main Preview
Choose a course to preview: Item 1
Is Optional
Preview
Class Name Section Lecture Lab Recitation
Add
Edit
Remove
Finished
Quit
Design Preview [DisplayScreen]
File Help
4
Graph Mode Text Mode
Math
Lecture: 2:00PM - 3:00PM Days: MWF
Lab: 1:00PM - 2:30PM Days: TR
Recitation: 3:00PM - 4:00PM Days: TR
Optional: No
History
Lecture: 2:00PM - 3:00PM Days: MWF
Lab: 6:00PM - 7:30PM Days: MW
Recitation: 3:00PM - 4:00PM Days: TR
Optional: No
Art
Lecture: 1:00PM - 2:00PM Days: TR
Lab: 1:00PM - 2:30PM Days: MWF
Recitation: 3:00PM - 4:00PM Days: MW
Optional: Yes
Previous Schedule
Next Schedule
Go Back
Quit