

Introduction

The product being designed is a Java program that acts as a personal/ customized planner and organizer for the client, Ashish Narayan. It keeps track of tasks, sends reminders, and converts between timezones.

Word Count: 33

Summary List of All Techniques

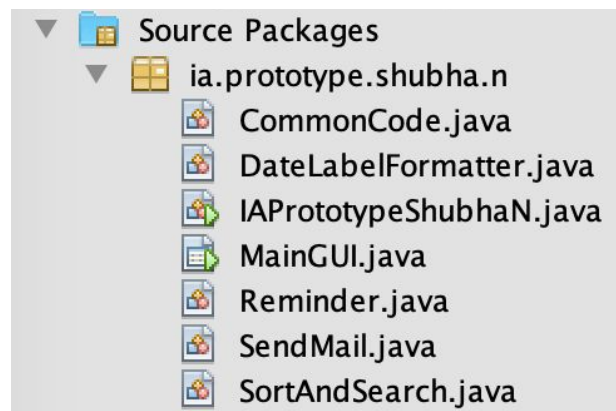
- For loop
- While loop
- Method returning a value
- ArrayLists
- User defined objects made from an OOP "template" class
- Encapsulation of private methods that work on public attribute of a "template" class
- Making an array of objects
- Simple and compound selection (if/else)
- Sorting an array of objects based on a key attribute (ie. bubble sort, etc.)
- Searching (ie, linear search, binary search, etc.)
- Error handling (for example catching a divide by 0 error, or a null pointer while using an array of objects...)
- GUI tabs
- Use of a flag value (such as -999, or "not set yet")
- Overloaded constructors, which work differently depending on the parameters sent
- Use of specialized library (ie. date)

Word Count: 0

Structure of the Program

What?

This program is heavily reliant on the **database** and it's sorting and searching functionalities. The database primarily stores an **ArrayList** of reminders, meetings, and tasks. This program is made up of the *Reminder* class which acts as a **template class**, the *Main GUI* class, the *TimeZone* class which converts between international time zones, and the *SortingAndSearching* which sorts and searches through the database for reminders.



This program has different components that aim to address different challenges faced by the client. In order to satisfy the client's requirements, **Object Oriented Programming** was used. This allowed me to use **decomposition**, dividing a large complex system into smaller classes or objects. The use of **Object Oriented Programming** has also resulted in the use of the object relationships of **dependency** ("uses") and **aggregation** ("has a").

Dependency is when one type of object *uses* another type of object. This relationship can be seen in this program between the *Main GUI* class and *SortingAndSearching* class. **Aggregation** is when one type of object contains or is composed of another. This relationship can be seen between the *Main GUI* class and the *Reminder* class as the *Main GUI* contains multiple *Reminder* classes. **Aggregation** is also seen between the *Main GUI* class and the *TimeZone* class as the *Main GUI* contains **multiple instances** of the *TimeZone* class.

Why?

The primary reason for using **Object Oriented Programming**, besides having learned it as a part of the IB Curriculum Option, was the many advantages that OOP can bring. For instance, the **decomposition** used in the program allows the use of **abstraction**, focusing on an individual problem while ignoring the other details. This allows faster development, easier debugging, easier updating procedure, and increases the program's re-usability.

Word Count: 283

Data Structures Used

ArrayLists were used to store objects such as instances of the *Reminder* class. *ArrayLists* provide the dynamic structure of a list as well as an array's ability to sort and search. Since both of these characteristics are essential to the nature of this program and the storing of the *Reminder* objects in the database, an *Array List* was selected. The dynamic structure ensures that the memory is not wasted but rather allocated efficiently.

Word Count: 73

Main Unique Algorithms

JDatePicker

```
public MainGUI() {
    initComponents();
    UtilDateModel model = new UtilDateModel();
    Properties P = new Properties();
    P.put("text.today", "Today");
    P.put("text.month", "Month");
    P.put("text.Year", "Year");
    JDatePanelImpl datePanel = new JDatePanelImpl(model,P);
    JDatePickerImpl datePicker = new JDatePickerImpl(datePanel, new DateLabelFormatter());
    this.add(datePicker);
    datePicker.setSize(300,100);
}
```

```
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.*;
import javax.swing.JFormattedTextField.AbstractFormatter;
/**
 *
 * @author 16484
 */
```

```
public class DateLabelFormatter extends AbstractFormatter {

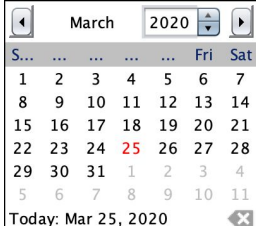
    DateLabelFormatter(){
        Calendar cal = Calendar.getInstance();
    }
    private String datePattern = "dd-MM-yyyy";
    private SimpleDateFormat dateFormatter = new SimpleDateFormat(datePattern);

    @Override
    public Object stringToValue(String text) throws ParseException {
        return dateFormatter.parseObject(text);
    }

    @Override
    public String valueToString(Object value) throws ParseException {
        if (value != null) {
            Calendar cal = (Calendar) value;
            return dateFormatter.format(cal.getTime());
        }

        return "";
    }
}
```

JDatePicker was used in order to have the ideal format for date selection as shown in the figure below. Everytime the user inputs a reminder, they have the ability to look through and select the date through this format rather than inserting the date, increasing consistency and reducing any input error.



March		2020				
S...	Fri	Sat	
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4
5	6	7	8	9	10	11
Today: Mar 25, 2020						

Database

```
/* Database Code */
boolean saveReminder() {
    try {

        String dt = date.get(Calendar.YEAR) + "-" + (date.get(Calendar.MONTH) + 1) + "-" + date.get(Calendar.DATE);
        String tm = date.get(Calendar.HOUR_OF_DAY) + ":" + date.get(Calendar.MINUTE);

        String sql = "insert into Reminders values ("
            + "" + name + ","
            + "" + dt + ", " + tm + ","
            + "" + category + ")";

        if (createConnection()) {

            stmt = cnn.createStatement();
            stmt.executeUpdate(sql);
            cnn.close();
            return true;
        }
    } catch (SQLException ex) {
        ex.printStackTrace(); // to know what error happened
        return false;
    }
    return false;
}

static ArrayList<Reminder> getAllReminders() {
    // Retrieval from database
    ArrayList<Reminder> R = new ArrayList();
    try {
        String sql = "select * from Reminders "; // * is for "All columns of the reminder table"
        Reminder T;

        if (createConnection()) {

            stmt = cnn.createStatement();

            ResultSet rs = stmt.executeQuery(sql);
            while (rs.next()) {
                SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd");
                java.util.Date date = formatter.parse(rs.getString(2));
                Calendar C = Calendar.getInstance();
                C.setTime(date);
                C.set(Calendar.HOUR_OF_DAY, rs.getTime(3).getHours());
                C.set(Calendar.MINUTE, rs.getTime(3).getMinutes());
                //C.set(Calendar.HOUR_OF_DAY, 0);
                T = new Reminder(rs.getString(1), rs.getString(4), C, 1);
                R.add(T);
            }
            cnn.close();
        }
    } catch (Exception ex) {
        ex.printStackTrace();
        return null;
    }
    return R;
}
```

The following code allows each reminder to be saved to a reminders database. This was done in order for the client to have maximum usability. The use of a database allows the client to refer back to previous reminders even when the program is restarted.

Email Notification

```
public class SendMail {  
1   public static void send(String msg) {  
    //1. Import the javax.mail.jar and then run this program to send message  
    final String username = "narayan.ash@gmail.com";  
    // this is username and pswd from which you are sending the email  
    // In security and privacy , you need to enable "Less Secure Apps access"  
    final String password = "xxxx";  
  
    Properties props = new Properties();  
    props.put("mail.smtp.starttls.enable", "true");  
    props.put("mail.smtp.auth", "true");  
    props.put("mail.smtp.host", "smtp.gmail.com");  
    props.put("mail.smtp.port", "587");  
  
    Session session = Session.getInstance(props,  
1    new javax.mail.Authenticator() {  
1    protected PasswordAuthentication getPasswordAuthentication() {  
        return new PasswordAuthentication(username, password);  
        }  
    });  
  
    try {  
  
        Message message = new MimeMessage(session);  
        message.setFrom(new InternetAddress("narayan.ash@gmail.com")); // MAKE CHANGES HERE  
        message.setRecipients(Message.RecipientType.TO,  
            InternetAddress.parse("ashish.narayan@itu.int")); // MAKE CHANGES HERE  
        message.setSubject("Testing Subject");  
        message.setText(msg);  
  
        Multipart multipart = new MimeMultipart();  
        message.setContent(multipart);  
  
        Transport.send(message);  
        System.out.println("Mail has been sent");  
  
    } catch (MessagingException e) {  
        e.printStackTrace();  
        throw new RuntimeException(e);  
    }  
}
```

There is a program constantly running, checking to see if there are any upcoming reminders for that day. If reminders are identified, the program displayed above allows the program to send an email to the user/client with a list of all their tasks for that day.

Word Count: 145

User Interface/GUI Work

The purpose of the Graphical User Interface (GUI) is to make the program as user friendly as possible. The Java Swing API was used in order to fulfil this purpose. Some of the most frequent features used in order to develop the GUI were the following:

- JButton

- JTextField
- JLabel
- JDisplayTable
- JTabbedPane
- JMenuBar
- JRadioButton
- JComboBox

Word Count: 54

Software Tools Used

The primary software tool that was used for this program is NetBeans IDE. This was the ideal program as it allows the use of Object Oriented Programming in Java. This allowed me to use decomposition and abstraction, dividing the program into different classes. NetBeans also provided the necessary tools to build the ideal GUI in a very simple and direct fashion.

Word Count: 61

Total Criterion C Word Count: 649