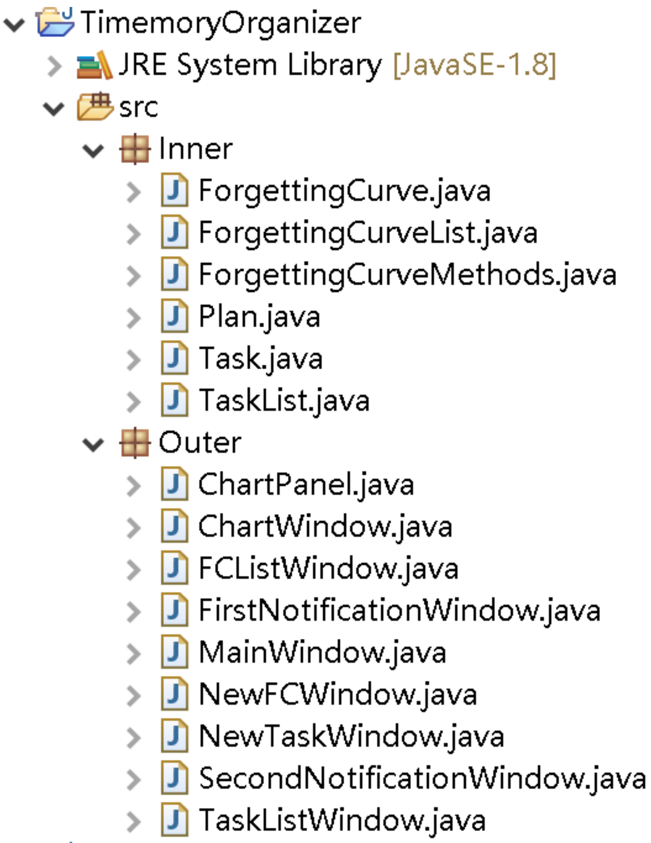
# Development

## Program Structure



```
TimemoryOrganizer
  JRE System Library [JavaSE-1.8]
  src
    Inner
      ForgettingCurve.java
      ForgettingCurveList.java
      ForgettingCurveMethods.java
      Plan.java
      Task.java
      TaskList.java
    Outer
      ChartPanel.java
      ChartWindow.java
      FCListWindow.java
      FirstNotificationWindow.java
      MainWindow.java
      NewFCWindow.java
      NewTaskWindow.java
      SecondNotificationWindow.java
      TaskListWindow.java
```

(Platform: Eclipse 2019-06)

# GUI – Java Swing

Using Swing which contains many sophisticated GUI components allows me to build a sophisticated program, the picture below shows which components were used for building “Create a task” window:

```
import java.awt.Color;
import java.awt.EventQueue;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.JTextPane;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
```

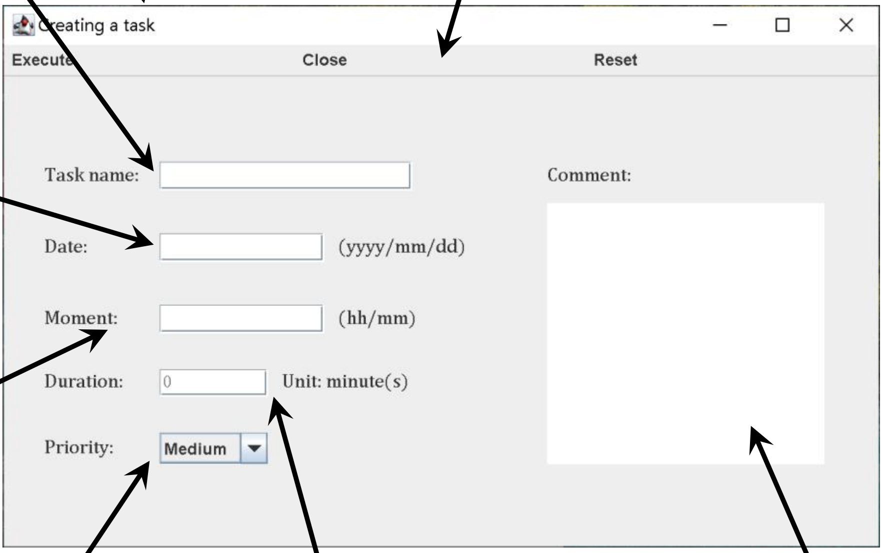
Some awt components were also adopted.

Since the GUI stuff in my program is too much, I will take NewTaskWindow class as an example to demonstrate how a window was built:

```java
frmCreatingATask = new JFrame();
frmCreatingATask.setTitle("Creating a task");
frmCreatingATask.getContentPane()
        .setFont(new Font("Cambria Math", Font.PLAIN, 14));
frmCreatingATask.setBounds(100, 100, 650, 400);
frmCreatingATask.getContentPane().setLayout(null);
```

```java
// Menu
JMenuBar menuBar = new JMenuBar();
frmCreatingATask.setJMenuBar(menuBar);

JMenuItem mntmExecute = new JMenuItem("Execute");
menuBar.add(mntmExecute);
mntmExecute.addActionListener(new ExecuteMnListener());

JMenuItem mntmClose = new JMenuItem("Close");
menuBar.add(mntmClose);
mntmClose.addActionListener(new CloseMnListener());

JMenuItem mntmReset = new JMenuItem("Reset");
menuBar.add(mntmReset);
mntmReset.addActionListener(new ResetMnListener());
```

```java
// Enter the name of the task
taskNameTextField = new JTextField();
taskNameTextField.setBounds(114, 62, 184, 21);
frmCreatingATask.getContentPane().add(taskNameTextField);
taskNameTextField.setColumns(10);

JLabel lblTaskName = new JLabel("Task name:");
lblTaskName.setFont(new Font("Cambria Math", Font.PLAIN, 14));
lblTaskName.setBounds(30, 66, 69, 15);
frmCreatingATask.getContentPane().add(lblTaskName);
```



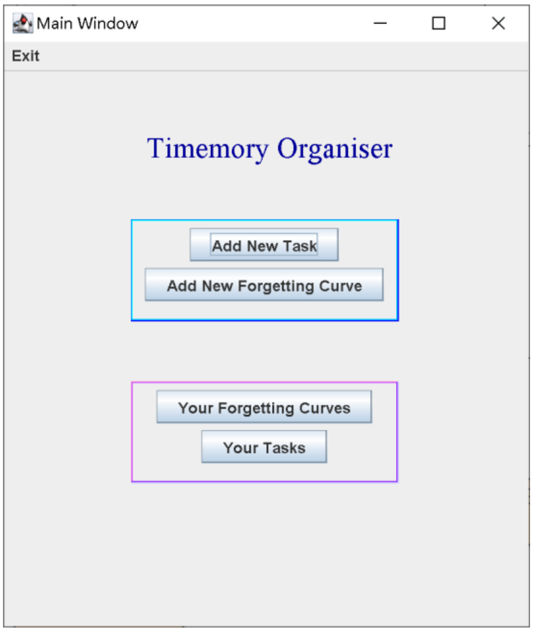
```java
// Date
textFieldDate = new JTextField();
textFieldDate.setBounds(114, 114, 120, 21);
frmCreatingATask.getContentPane().add(textFieldDate);
textFieldDate.setColumns(10);

JLabel lblDate = new JLabel("Date:");
lblDate.setFont(new Font("Cambria Math", Font.PLAIN, 14));
lblDate.setBounds(30, 118, 54, 15);
frmCreatingATask.getContentPane().add(lblDate);

JLabel lblYyyymmdd = new JLabel("(yyyy/mm/dd)");
lblYyyymmdd.setFont(new Font("Cambria Math", Font.PLAIN, 14));
lblYyyymmdd.setBounds(244, 118, 101, 15);
frmCreatingATask.getContentPane().add(lblYyyymmdd);

// Moment
textFieldMoment = new JTextField();
textFieldMoment.setBounds(114, 166, 120, 21);
frmCreatingATask.getContentPane().add(textFieldMoment);
textFieldMoment.setColumns(10);

JLabel lblMoment = new JLabel("Moment:");
lblMoment.setFont(new Font("Cambria Math", Font.PLAIN, 14));
lblMoment.setBounds(30, 170, 54, 15);
frmCreatingATask.getContentPane().add(lblMoment);

JLabel lblHhmm = new JLabel("(hh/mm)");
lblHhmm.setFont(new Font("Cambria Math", Font.PLAIN, 14));
lblHhmm.setBounds(244, 170, 73, 15);
frmCreatingATask.getContentPane().add(lblHhmm);
```

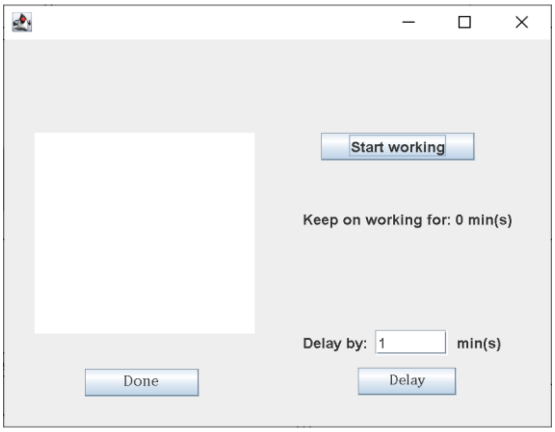
```java
// Add a comment
commentTextArea = new JTextArea();
commentTextArea.setWrapStyleWord(true);
commentTextArea.setLineWrap(true);
commentTextArea.setBounds(396, 92, 202, 190);
frmCreatingATask.getContentPane().add(commentTextArea);

JLabel lblComment = new JLabel("Comment:");
lblComment.setBounds(396, 66, 79, 15);
lblComment.setFont(new Font("Cambria Math", Font.PLAIN, 14));
frmCreatingATask.getContentPane().add(lblComment);
```

```java
// Choose priority
priorityComboBox = new JComboBox<String>();
priorityComboBox.addItem("Medium");
priorityComboBox.addItem("High");
priorityComboBox.addItem("Low");
priorityComboBox.setBounds(114, 259, 79, 23);
frmCreatingATask.getContentPane().add(priorityComboBox);
JLabel lblPriority = new JLabel("Priority:");
lblPriority.setFont(new Font("Cambria Math", Font.PLAIN, 14));
lblPriority.setBounds(30, 264, 54, 15);
frmCreatingATask.getContentPane().add(lblPriority);
```

```java
// Enter a duration
JLabel lblDuration = new JLabel("Duration:");
lblDuration.setFont(new Font("Cambria Math", Font.PLAIN, 14));
lblDuration.setBounds(30, 212, 73, 23);
frmCreatingATask.getContentPane().add(lblDuration);

JLabel lblUnitMinutes = new JLabel("Unit: minute(s)");
lblUnitMinutes.setFont(new Font("Cambria Math", Font.PLAIN, 14));
lblUnitMinutes.setBounds(203, 216, 120, 15);
frmCreatingATask.getContentPane().add(lblUnitMinutes);

txtDuration = new JTextField();
txtDuration.setFont(new Font("Cambria Math", Font.PLAIN, 13));
txtDuration.setText("0");
txtDuration.setForeground(Color.GRAY);
txtDuration.setBounds(114, 213, 79, 20);
frmCreatingATask.getContentPane().add(txtDuration);
txtDuration.setColumns(10);
```

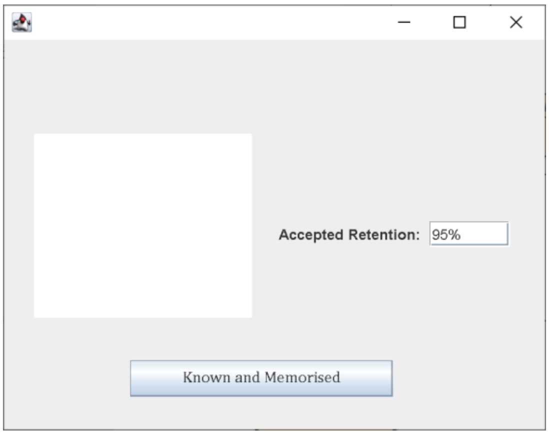
Some other GUI windows:



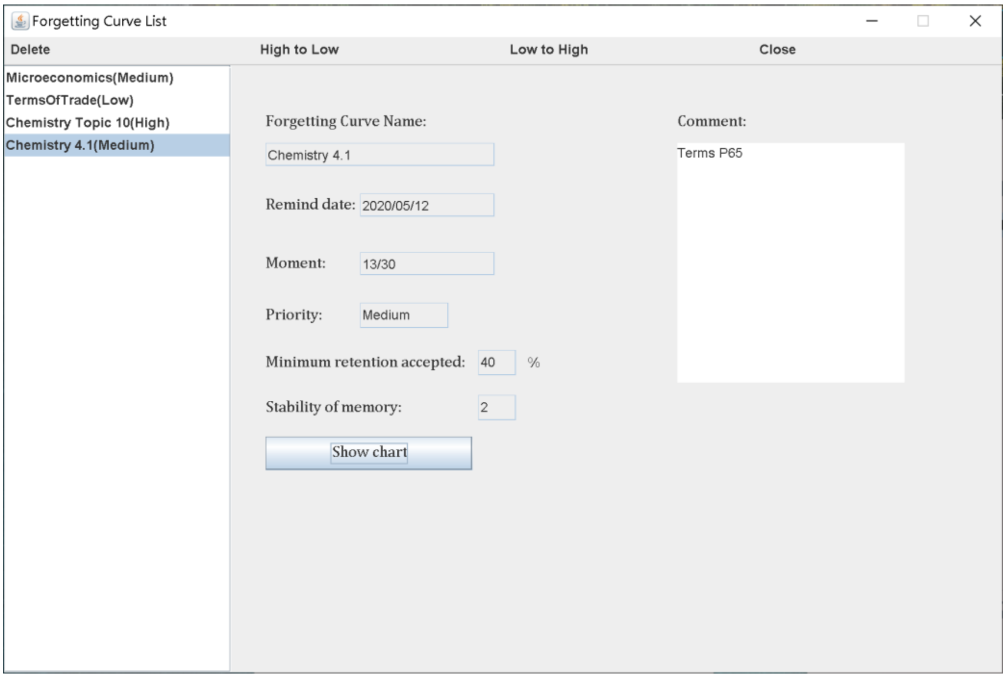


Keep on working for: 0 min(s)

Delay by: 1 min(s)



Accepted Retention: 95%



Forgetting Curve Name: Chemistry 4.1

Remind date: 2020/05/12

Moment: 13/30

Priority: Medium

Minimum retention accepted: 40 %

Stability of memory: 2

Comment: Terms P65

Moreover, there are action listeners to perform actions after end-users click buttons:

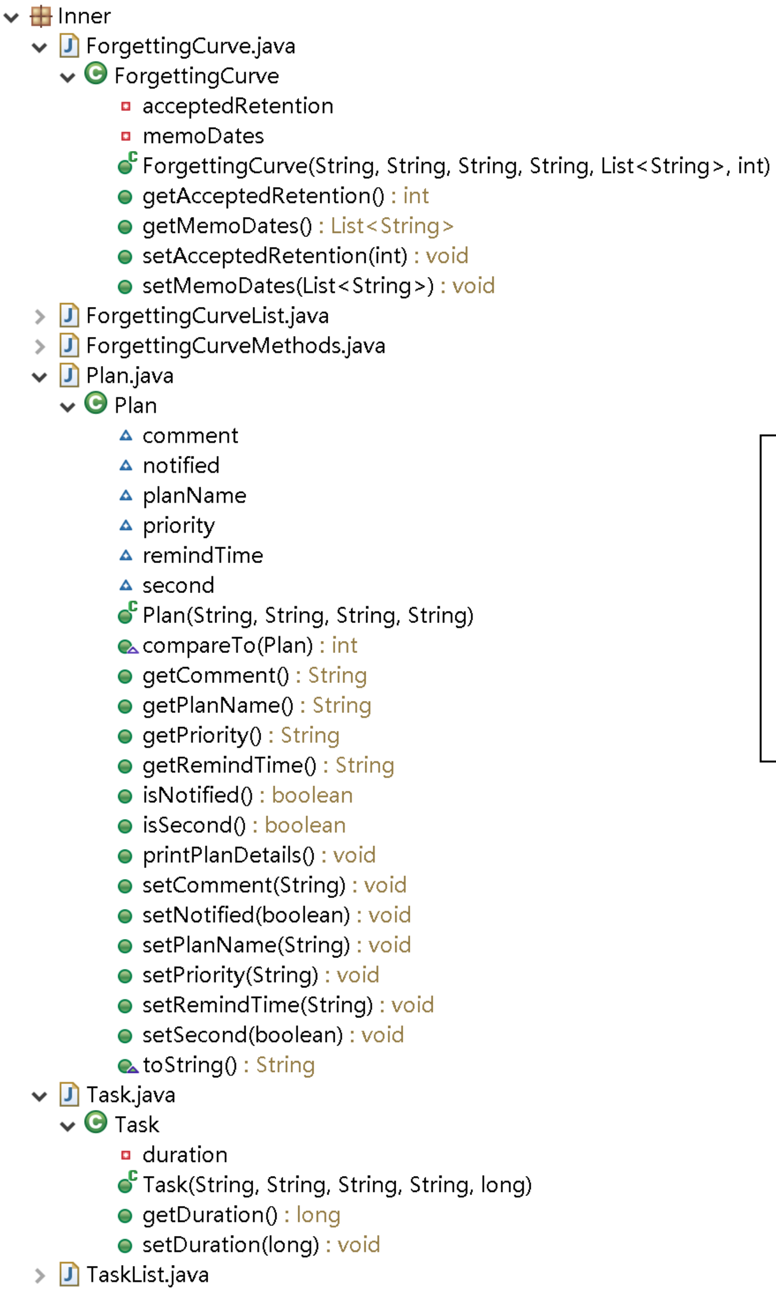
For example, in NewTaskWindow class:

```
// Reset
public class ResetMnListener implements ActionListener {
  public void actionPerformed(ActionEvent ev) {
    taskNameTextField.setText("");
    commentTextArea.setText("");
    txtDuration.setText("0");
    textFieldDate.setText("");
    textFieldMoment.setText("");
  }
}
// Close
public class CloseMnListener implements ActionListener {
  public void actionPerformed(ActionEvent ev) {
    frmCreatingATask.dispose();
  }
}
// Excute
public class ExecuteMnListener implements ActionListener {
  public void actionPerformed(ActionEvent ev) {
    try {
      Task tsk = new Task(taskNameTextField.getText(),
          commentTextArea.getText(),
          (String) priorityComboBox.getSelectedItem(),
          (textFieldDate.getText() + " "
              + textFieldMoment.getText() + "/" + "00"),
          Integer.parseInt(txtDuration.getText()));
      proceedTaskTimer(tsk);
      Inner.TaskList.taskList.add(tsk);
    } catch (ParseException e) {
      JOptionPane.showMessageDialog(null, e.toString());
    }
  }
}
```

These three actions represent the three buttons (3 actions) on the menu bar on the “Creating a task”

# Object-Orientation

Three key advantages of using Java, a OOP language that benefit my program were its reusability, extensibility, and security. By using OOP, I successfully reused the properties of parent classes in child classes, this helps to use the same code over and over. It also made my program more modular, meaning it is easier to extend and maintain the program. Moreover, I adopted Encapsulation to protect objects from unwanted access:



```
Inner
  ForgettingCurve.java
    ForgettingCurve
        acceptedRetention
        memoDates
        ForgettingCurve(String, String, String, String, List<String>, int)
        getAcceptedRetention() : int
        getMemoDates() : List<String>
        setAcceptedRetention(int) : void
        setMemoDates(List<String>) : void
  ForgettingCurveList.java
  ForgettingCurveMethods.java
  Plan.java
    Plan
        comment
        notified
        planName
        priority
        remindTime
        second
        Plan(String, String, String, String)
        compareTo(Plan) : int
        getComment() : String
        getPlanName() : String
        getPriority() : String
        getRemindTime() : String
        isNotified() : boolean
        isSecond() : boolean
        printPlanDetails() : void
        setComment(String) : void
        setNotified(boolean) : void
        setPlanName(String) : void
        setPriority(String) : void
        setRemindTime(String) : void
        setSecond(boolean) : void
        toString() : String
  Task.java
    Task
        duration
        Task(String, String, String, String, long)
        getDuration() : long
        setDuration(long) : void
  TaskList.java
```

ForgettingCurve and Task inherited the features from Plan class. Getters and setters were also made.

Note: *notified* and *second* variables are not related to the content of plan, their uses will be explained later.

# Algorithms

## 1. Getting retention values between starting date and current date

```
public static List<Double> GetRetentions (List<String> memoDates_Str) {

   retentionVaules = new ArrayList<Double>();
   List<Calendar> memoDates_cld  = StrListToCldList(memoDates_Str);

   for (int i=1; i <= (memoDates_Str.size()-1); i++) {
      retentionVaules.add(100.0);
      for (int index = 1; index <= ((int) ChronoUnit.DAYS.between(memoDates_cld.get(i-1).toInstant(),
            memoDates_cld.get(i).toInstant())-1); index++) {
         stabilityOfMemo = i;
         int daysSinceMemo = index;
         retentionVaules.add(CalculateRetention(daysSinceMemo, stabilityOfMemo));
      }
   }

   cld = Calendar.getInstance();
   stabilityOfMemo+=1.0;

   int diffCurrentLast =
         (int) ChronoUnit.DAYS.between(memoDates_cld.get(memoDates_cld.size()-1).toInstant(), cld.toInstant());

   int i = 0;
   while (i <= diffCurrentLast) {
      retentionVaules.add(CalculateRetention(i, stabilityOfMemo));
      i++;
   }
   return retentionVaules;
}
```

The inner loop is for getting each retention of date in one period, the outer loop is for getting retention values of all periods. In each period, the memory stability is the same, so here stabulityOfMemo = the outer loop counter.

This loop is for getting all retention values between the last date in *memoDates* and the current date. Thereafter, this method will return a list all retention values from start date to current date.

```
public static double CalculateRetention(int daysSinceMemo, int stabilityOfMemo) {
   df = new DecimalFormat("#.##");
   double retention = Double.valueOf(df.format(Math.exp((-daysSinceMemo)/stabilityOfMemo)*100));
   return retention;
}
```

CalculateRetention() refers to the equation mentioned in Planning:

$$R = e^{-t/S}$$

Where t = *daysSinceMemo*, S = *stabilityOfMemo*

```
public static List<Calendar> StrListToCldList(List<String> memoDates) {
   List<Calendar> memoDates_cld = new LinkedList<Calendar>();
   SimpleDateFormat ft = new SimpleDateFormat ("yyyy/MM/dd hh/mm");

   for (int i = 0; i<memoDates.size();i++) {
      Calendar cld = Calendar.getInstance();
      try {
         cld.setTime(ft.parse(memoDates.get(i)));
      } catch (ParseException e) {
         e.printStackTrace();
      }
      memoDates_cld.add(cld);
   }
   return memoDates_cld;
}
```

Since all dates stored in *memoDates* list are Strings and the algorithm requires to get the difference in days between two dates. So StrListToCldList() method is developed to convert dates in *memoDates* to type Calendar to do maths calculation.

After getting a list of retention values based on the dates that end-user chooses to memorise on, it is able to plot a chart showing how one's memory retention changes over time. This will be demonstrated in Graphing section later.

2. Count after how many days should user be reminded to revise a memorisation task:

```
public static int CountDaysAfterToRemind(int acceptedRetention) {
    int stabilityOfMemo = 1;
    int afterDays = 0;
    // Cast int to double
    double AR = (double) acceptedRetention;
     while (CalculateRetention(afterDays, stabilityOfMemo) > AR) {
         afterDays++;
     }
    return afterDays;
}
```

As long as the future retention value on a date is greater than *acceptedRetention*, *afterDays* will be increased by 1, then it will be put as parameter again in CalculateRetention() to compare.

```
public static int CountDaysAfterToRemind(List<String> memoDates_Str, int acceptedRetention) {
    int noOfItems = memoDates_Str.size();
    stabilityOfMemo = noOfItems;
     int afterDays = 1;
     double AR = (double) acceptedRetention;

     while (CalculateRetention(afterDays, stabilityOfMemo) > AR) {
         afterDays++;
     }
    return afterDays;
}
```

Over and over, after a number of days, the current retention value will be less than accepted, so return *afterDays*, meaning after *afterDays* days, user should be reminded to revise as on that date the current retention will be unacceptable as this algorithm predicted.

These two methods above share the same name but parameters. This indicates a Java's feature: Overloading, a form of Polymorphism. I decided to do so since the size of *memoDates* can be 1 or greater than 1. If the size is 1, then there is no need to take *memoDates* as parameter.

## After Executing: Task

After click “Execute” button, all user inputed data will be collected and used to instantiate a Task object, tsk. Then, proceedTaskTimer() will be called and inputted tsk as parameter. Finally, tsk will be added to *taskList* in TaskList class.

```
// Execute
public class ExecuteMnListener implements ActionListener {
   public void actionPerformed(ActionEvent ev) {
      try {
         Task tsk = new Task(taskNameTextField.getText(), commentTextArea.getText(),
               (String) priorityComboBox.getSelectedItem(),
               (textFieldDate.getText() + " "+ textFieldMoment.getText() + "/" + "00"),
               Integer.parseInt(txtDuration.getText()));
         proceedTaskTimer(tsk);
         Inner.TaskList.taskList.add(tsk);
      } catch (ParseException e) {
         JOptionPane.showMessageDialog(null, e.toString());
      }
   }
}

public static void proceedTaskTimer(Task tsk) throws ParseException {
   String timeToRemind = tsk.getRemindTime();
   ft = new SimpleDateFormat("yyyy/MM/dd hh/mm");
   Calendar remindTime = Calendar.getInstance();
   remindTime.setTime(ft.parse(timeToRemind));
   now = Calendar.getInstance();
   long currentDateMilli = now.getTimeInMillis();
   long remindDateMilli = remindTime.getTimeInMillis();
   long remindAfterMilli = Math.subtractExact(remindDateMilli,
         currentDateMilli);
   Timer timer = new Timer();

   timer.schedule(new TimerTask() {
      public void run() {
         TaskNotificationWindow(tsk);
         frmCreatingATask.setVisible(true);
         timer.cancel();
      }
   }, remindAfterMilli);
}
```

Getting the time interval (in milliseconds) between *remindTime* and the current time, then proceed a timer method stating after how many milliseconds (the interval) should perform an action, the action is about calling a method called TaskNotificationWindow(), which is the notification window for tasks.

```
// Notification Window
private static void TaskNotificationWindow(Task tsk) {

    frmCreatingATask = new JFrame();
    frmCreatingATask.setBounds(100, 100, 457, 355);
    frmCreatingATask.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frmCreatingATask.getContentPane().setLayout(null);

    JLabel lblNoticationTitle = new JLabel(tsk.getPlanName());
    lblNoticationTitle.setFont(new Font("Cambria Math", Font.BOLD, 16));
    lblNoticationTitle.setBounds(25, 36, 224, 30);
    frmCreatingATask.getContentPane().add(lblNoticationTitle);

    JButton btnDone = new JButton("Done");
    btnDone.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            frmCreatingATask.dispose();
            TaskList.taskList.remove(tsk);
        }
    });
    btnDone.setFont(new Font("Cambria Math", Font.PLAIN, 13));
    btnDone.setBounds(66, 272, 93, 23);
    frmCreatingATask.getContentPane().add(btnDone);

    JTextPane textPaneNotifComt = new JTextPane();
    textPaneNotifComt.setEditable(false);
    textPaneNotifComt.setBounds(25, 77, 179, 166);
    textPaneNotifComt.setText(tsk.getComment());
    frmCreatingATask.getContentPane().add(textPaneNotifComt);

    JLabel labelWorking = new JLabel("Keep on working for: "
            + String.valueOf(tsk.getDuration()) + " min(s)");
    labelWorking.setBounds(243, 141, 179, 15);
    frmCreatingATask.getContentPane().add(labelWorking);
    labelWorking.setVisible(false);
```

Clicking “Done” in the notification window, nominated task will be removed from *tasklist.*

```
JLabel labelWorking = new JLabel("Keep on working for: "
        + String.valueOf(tsk.getDuration()) + " min(s)");
labelWorking.setBounds(243, 141, 179, 15);
frmCreatingATask.getContentPane().add(labelWorking);
labelWorking.setVisible(false);

JButton btnStartWorking = new JButton("Start working");
btnStartWorking.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        labelWorking.setVisible(true);
    }
});
btnStartWorking.setBounds(258, 77, 125, 23);
frmCreatingATask.getContentPane().add(btnStartWorking);
```

Display user inputted *duration* of the task after clicking button “Starting working”.

```
JLabel lblDelayBy = new JLabel("Delay by:");
lblDelayBy.setBounds(243, 243, 54, 15);
frmCreatingATask.getContentPane().add(lblDelayBy);

textField = new JTextField();
textField.setText("1");
textField.setBounds(302, 240, 59, 21);
textField.setColumns(10);
frmCreatingATask.getContentPane().add(textField);

JLabel lblMins = new JLabel("min(s)");
lblMins.setBounds(368, 243, 54, 15);
frmCreatingATask.getContentPane().add(lblMins);

JButton btnDelay = new JButton("Delay");
btnDelay.setBounds(288, 271, 80, 23);
frmCreatingATask.getContentPane().add(btnDelay);
btnDelay.setFont(new Font("Cambria Math", Font.PLAIN, 12));
btnDelay.addActionListener(new ActionListener() {
    long remindAfterMilli = Integer.parseInt(textField.getText()) * 1000
            * 60;

    public void actionPerformed(ActionEvent e) {
        Timer timer = new Timer();

        timer.schedule(new TimerTask() {
            public void run() {
                TaskNotificationWindow(tsk);
                frmCreatingATask.setVisible(true);
                timer.cancel();
            }
        }, remindAfterMilli);
        frmCreatingATask.dispose();
    }
});
```

Getting how many minutes to delay, then proceed a timer calling TaskNoticationWindow() after this time interval. Note: this method can be recursively called if the user wants to delay the notification.

## After Executing: Memorisation Task (Forgetting Curve)

I initially decided to use the same timer method as how I did with Tasks. However, since the functionality of my program should be demonstrated in a short time, normally user will be informed to revise after more than 24 hours, and my used timer method will not be affected by manipulating the system clock. I therefore decide to develop a new timer method which will check if the remind time is later than the current system time frequently, if later, then perform an action. By using schedule() which can perform an action frequently, it is enabled to develop such method. However, in order to stop performing after reaching condition, I added *notified* and *second* variables of type Boolean (to indicate if notified. If it is true, then stop checking). There are two variables because there are two notification windows which will be displayed and the FirstNotificationWindow() for FC will only be displayed by once.

New timer method:

```
void setTimer() {
    Timer timer = new Timer();
    timer.schedule(new TimerTask() {
        @Override
        public void run() {
            Date now = Calendar.getInstance().getTime();
            SimpleDateFormat ft = new SimpleDateFormat("yyyy/MM/dd hh/mm");

            for (ForgettingCurve fc : ForgettingCurveList.fcList) {

                if (fc.isNotified()) {
                    continue;
                }
                try {
                    Date time = ft.parse(fc.getRemindTime());
                    if (now.compareTo(time) >= 0) {
                        fc.setNotified(true);
                        if (!fc.isSecond()) {
                            new FirstNotificationWindow(fc);
                            fc.setSecond(true);
                        } else {
                            new SecondNotificationWindow(fc);
                        }
                    }
                } catch (ParseException e) {
                    JOptionPane.showMessageDialog(frmMainWindow, e.toString());
                }
            }
        }
    }, 500, 500);
}
```

Get current time

Check for every FC in the list

Check if the *remindTime* is earlier than the current, if not, then perform an action.

Allowing the FirstNoticationWindow() will only be displayed by once.

Perform once per 500 milliseconds

# FirstNotificationWindow

setTimer() will always perform. After knowing the starting time (user-inputted *remindTime*) of a memorisation task, this method will display relative FirstNotificationWindow based on the time. Inside FirstNotificationWindow(), user can only choose to memorise:

Since choose to memorise, add today (*remindTime*) into *memoDates* list, then obtaining after how many days should remind based on *acceptedRetention*.

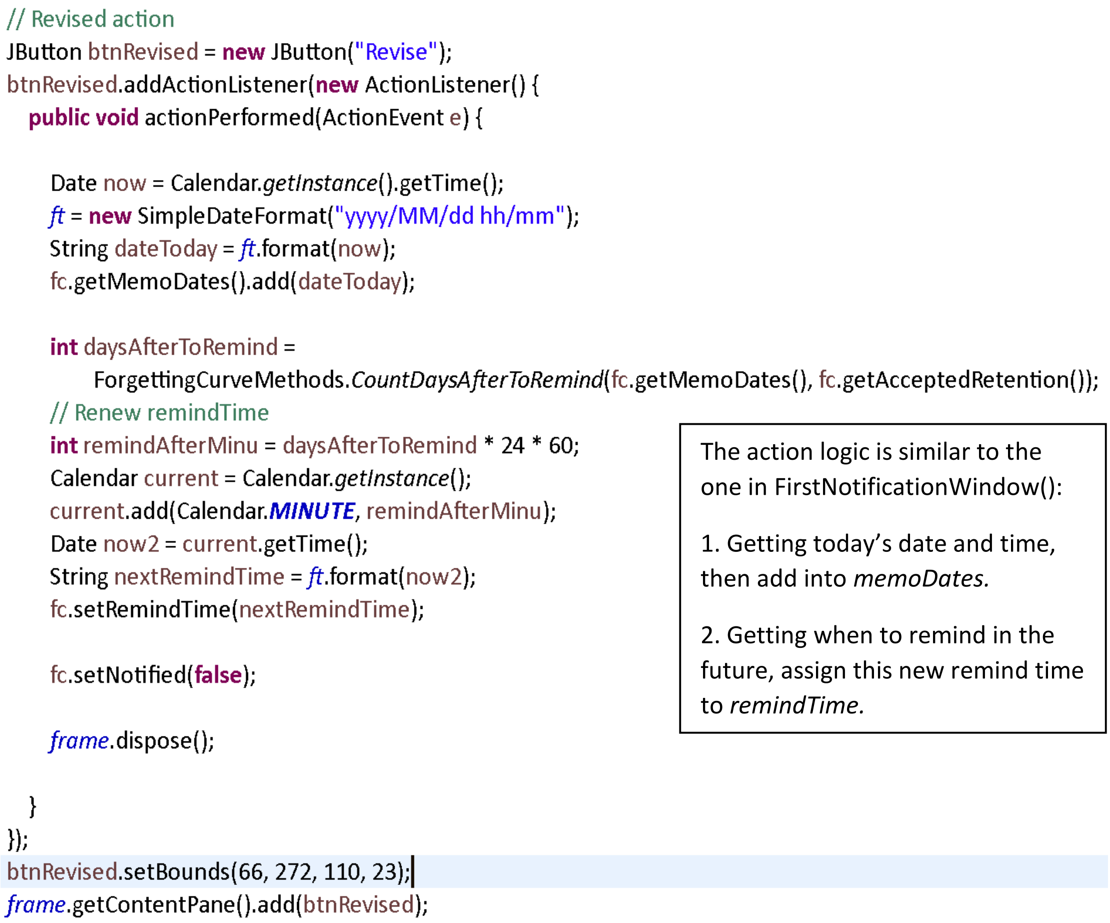
```
// Known and Memorised action
JButton btnRevised = new JButton("Known and Memorised");
btnRevised.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {

    fc.getMemoDates().add(fc.getRemindTime());

    int daysAfterToRemind = ForgettingCurveMethods.CountDaysAfterToRemind(fc.getAcceptedRetention());

    // Renew remindTime
    int remindAfterMinu = daysAfterToRemind * 24 * 60;
    Calendar current = Calendar.getInstance();
    current.add(Calendar.MINUTE, remindAfterMinu);
    Date now = current.getTime();
    String nextRemindTime = ft.format(now);
    fc.setRemindTime(nextRemindTime);
    fc.setNotified(false);

    frame.dispose();
  }
});
btnRevised.setFont(new Font("Cambria Math", Font.PLAIN, 13));
btnRevised.setBounds(104, 263, 216, 30);
frame.getContentPane().add(btnRevised);
```

Since knowing after how many days should remind, generate a new future remind time, then assign this new time to remindTime. Thereafter, setTime() will again sense when to display SecondNotificationWindow.

# SecondNotificationWindow

In this window, user can either choose to revise today or not. After displaying, there will be two buttons (**two actions**):

## 1st Action: Revise



```
// Revised action
JButton btnRevised = new JButton("Revise");
btnRevised.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {

        Date now = Calendar.getInstance().getTime();
        ft = new SimpleDateFormat("yyyy/MM/dd hh/mm");
        String dateToday = ft.format(now);
        fc.getMemoDates().add(dateToday);

        int daysAfterToRemind =
            ForgettingCurveMethods.CountDaysAfterToRemind(fc.getMemoDates(), fc.getAcceptedRetention());
        // Renew remindTime
        int remindAfterMinu = daysAfterToRemind * 24 * 60;
        Calendar current = Calendar.getInstance();
        current.add(Calendar.MINUTE, remindAfterMinu);
        Date now2 = current.getTime();
        String nextRemindTime = ft.format(now2);
        fc.setRemindTime(nextRemindTime);

        fc.setNotified(false);

        frame.dispose();

    }
});
btnRevised.setBounds(66, 272, 110, 23);
frame.getContentPane().add(btnRevised);
```

The action logic is similar to the one in FirstNotificationWindow():

1. Getting today's date and time, then add into *memoDates.*

2. Getting when to remind in the future, assign this new remind time to *remindTime.*

## 2$^{nd}$ Action: Not Revise

```
// Unable to revise today
JButton btnRemindMeTomorrow = new JButton("Unable to revise today");
btnRemindMeTomorrow.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {

        int daysAfterToRemind = 1;
        // Renew remindTime
        int remindAfterMinu = daysAfterToRemind * 24 * 60;
        Calendar current = Calendar.getInstance();
        current.add(Calendar.MINUTE, remindAfterMinu);
        String nextRemindTime = ft.format(current.getTime());
        fc.setRemindTime(nextRemindTime);
        fc.setNotified(false);

        frame.dispose();
    }
});
btnRemindMeTomorrow.setBounds(252, 271, 167, 23);
frame.getContentPane().add(btnRemindMeTomorrow);
```

This time, since the user chooses to not revise today, so today will not be added into *memoDate.*

Also, the new future *remindTime* become one day after the current date.

# Graphing

I initially planned to draw curves to visualise FCs, however, I found it would be better to plot bar charts since bar charts look better to indicate quantities and memory retentions are indeed quantities. I decided to adopt Graphics and Graphics2D (awt) to plot bar charts. In ChartPanel class:

```
private ForgettingCurve fc;
Font titleFont = new Font("Cambria Math", Font.BOLD, 20);
List<Double> record;

public ChartPanel(ForgettingCurve fc) {
  this.fc = fc;
  record = ForgettingCurveMethods.GetRetentions(fc.getMemoDates());
}

public void paint(Graphics g0) {
  super.paint(g0);
  Graphics2D g2 = (Graphics2D) g0;
  Font org = getFont();
  g2.setFont(titleFont);
  g2.drawString(fc.getPlanName(), getWidth() / 2 - 50, 20);
  g2.setFont(org);
  int count = record.size();
  if (count == 0) {
    g2.drawString("No data", getWidth() / 2 - 50, 200);
    return;
  }
  int w = (getWidth() - 30) / count;
  for (int i = 0; i <= 100; i += 10) {
    int height = (int) ((100 - i) / 100.0 * (getHeight() - 40 - 40))+ 40;
    g2.setColor(Color.blue);
    g2.drawString(i + "%", 10, height + 5);
    g2.setColor(Color.gray);
    g2.drawLine(w / 4 + 11, height, w * count, height);
  }
  g2.setColor(Color.gray);
  g2.drawLine(w / 4 + 11, 40, w / 4 + 11, getHeight() - 40);
  for (int i = 0; i < count; i++) {
    int top = (int) ((100 - record.get(i)) / 100.0* (getHeight() - 40 - 40)) + 40;
    int height = (int) (record.get(i) / 100.0* (getHeight() - 40 - 40));
    int x = i * w + w / 2 + 10;
    g2.setColor(Color.gray);
    g2.setColor(Color.blue);
    String s = "Day " + (i + 1);
    if (i == count - 1) {s = "Day Last";}
    g2.drawString(s, x + w / 4 - 10, getHeight() - 22);
    g2.drawString(String.format("%.1f", record.get(i)), x + w / 4 - 20, top - 5);
    g2.fillRect(x, top, w / 2, height);
  }
}
```

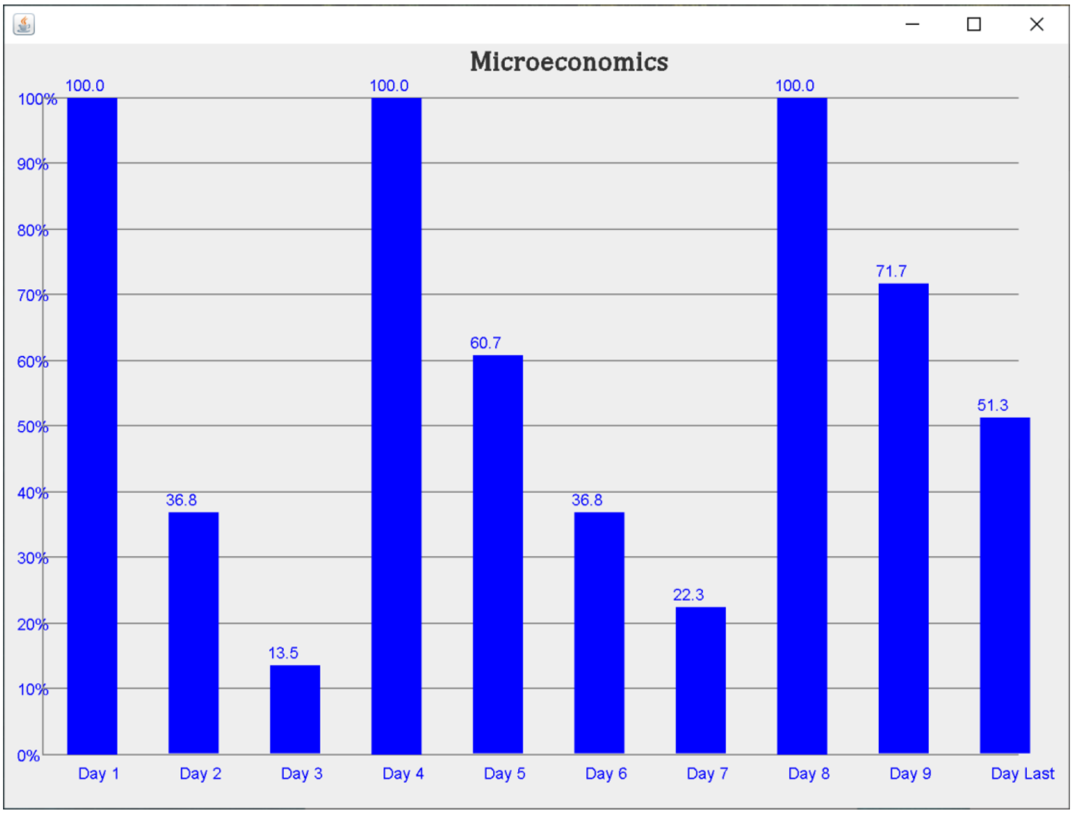
Firstly, obtaining retention values using GetRetentions().

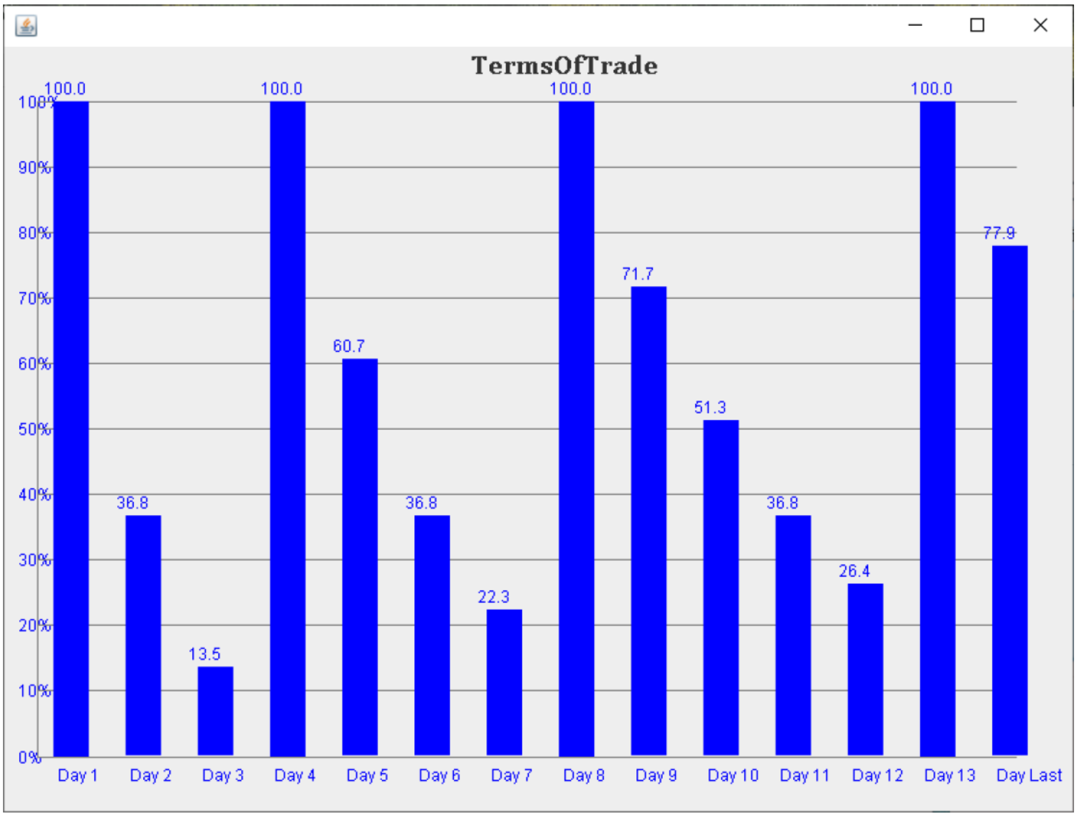
No-retention-situation is considered.

Bar heights depending on retention values.

Adding bars and their related days using for loop

Examples of completed FC bar charts:





Word count: 1102

# References

[1] Sedgewick R. and Wayne K., Princeton University (2011), “Algorithms”, p702 – 730.

[2] LogicBig.com, “Java - Scheduling tasks using java.util.Timer”, available at: https://www.logicbig.com/tutorials/core-java-tutorial/java-multi-threading/util-timer.html (Accessed: 2020/12/20)

[3] javatpoint.com, “Java Swing Tutorial”, available at: https://www.javatpoint.com/java-swing (Accessed: 2020/12/30 )

[4] stackoverflow.com, “calendar date to yyyy-mm-dd format in java”, anwered by MadProgrammer on Sep 25, 2012, available at https://stackoverflow.com/questions/6420623/how-to-bind-arraylist-to-jlist (Accessed: 2020/01/11)

[5] java2s.com, “Simple bar chart: Chart « 2D Graphics GUI « Java”, available at: http://www.java2s.com/Code/Java/2D-Graphics-GUI/Simplebarchart.htm (Accesses: 2020/03/07)

[6] stackoverflow.com, “how to bind ArrayList to JList”, anwered by Oleg Pavliv on Jun 11, 2011, available at https://stackoverflow.com/questions/6420623/how-to-bind-arraylist-to-jlist (Accessed: 2020/03/03 )

[7] stackoverflow.com, “how to add minutes to my date”, anwered by Marco Montelon Sep 20, 2016, available at https://stackoverflow.com/questions/9043981/how-to-add-minutes-to-my-date (Accessed: 2020/04/04 )