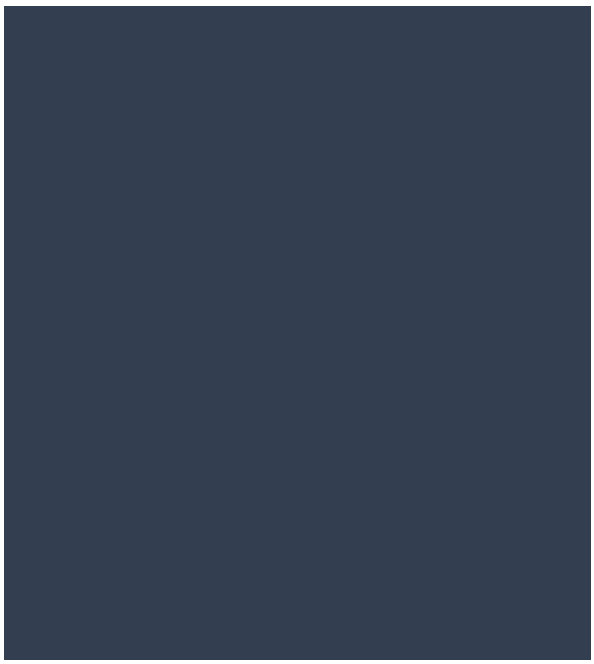

IT PAT

Phase 4: Technical Documentation

Name: Milaan Kassie

Table of Contents

4.1.1 Externally Sourced Code

- Validation Class
- DbManager Class
- populateJTable method
- Text formatting line
    - Removes all text characters, leaving only
- Date line
    - Gets the date before a certain date

Validation Class

```
// VALIDATION CLASS (c)
// CREATOR: S. Govender (Seatides Combined School)
// VERSION: 5 (2012)
// OTHER VERSIONS: 1 (2008), 2 (2009), 3 (2010), 4 (2011)
// This class or any of its parts MAY NOT be copied, printed, distributed by itself or as part of
// a package (including tuition) for which monetary gain is received, without receiving consent
from the creator.
// Note that I (Milaan Kassie) have edited or added methods to this class
package dataPkg;

import java.sql.ResultSet;
import java.sql.SQLException;
import javax.swing.JOptionPane;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.Scanner;

public class Validation
{

   static DbManager dbm = new DbManager();

   public static boolean vDouble(String value, String message)
   {
      boolean valid = false;
      if (message.equalsIgnoreCase(""))
      {
         message = "Invalid Number";
      }
      if (value.endsWith("d") || value.endsWith("D") || value.endsWith("f") ||
value.endsWith("F"))
      {
         message += " The value cannot end in a D or a F";
         JOptionPane.showMessageDialog(null, message, "Error",
JOptionPane.ERROR_MESSAGE);
```

```
        } else
        {
            try
            {
                double n = Double.parseDouble(value);
                valid = true;
            } catch (Exception e)
            {
                JOptionPane.showMessageDialog(null, message, "Error",
JOptionPane.ERROR_MESSAGE);
                valid = false;
            }
        }
        return valid;
    }

    public static boolean vDoublePositive(String value, String message)
    {
        if (message.equalsIgnoreCase(""))
        {
            message = "Invalid Number";
        }

        boolean valid = vDouble(value, message);
        if (valid)
        {
            double n = Double.parseDouble(value);
            if (n < 0)
            {
                JOptionPane.showMessageDialog(null, message, "Error",
JOptionPane.ERROR_MESSAGE);
                valid = false;
            }
        }

        return valid;
    }

    public static boolean vInteger(String value, String message)
    {
        if (message.equalsIgnoreCase(""))
        {
            message = "Invalid Integer";
        }

        boolean valid = false;
        try
        {
            int n = Integer.parseInt(value);
            valid = true;
```

```
        } catch (Exception e)
        {
            JOptionPane.showMessageDialog(null, message, "Error",
JOptionPane.ERROR_MESSAGE);
            valid = false;
        }

        return valid;
    }

    public static boolean vIntegerPositive(String value, String message)
    {
        if (message.equalsIgnoreCase(""))
        {
            message = "Invalid Integer";
        }

        boolean valid = vInteger(value, message);
        if (valid)
        {
            int n = Integer.parseInt(value);
            if (n < 0)
            {
                JOptionPane.showMessageDialog(null, message, "Error",
JOptionPane.ERROR_MESSAGE);
                valid = false;
            }
        }
        return valid;
    }

    public static boolean vDoubleRange(String value, String message1, String message2,
double min, double max)
    {
        boolean validrange = false;

        boolean valid = vDouble(value, message1);

        if (message2.equalsIgnoreCase(""))
        {
            message2 = "The Number does not fall between " + min + " and " + max;
        }

        if (valid == true)
        {
            double n = Double.parseDouble(value);
            if ((n >= min) && (n <= max))
            {
                validrange = true;
            } else
```

```
            {
                JOptionPane.showMessageDialog(null, message2, "Error",
JOptionPane.ERROR_MESSAGE);
                validrange = false;
            }
        }
        return validrange;
    }

    public static boolean vIntegerRange(String value, String message1, String message2, int
min, int max)
    {
        boolean validrange = false;

        boolean valid = vInteger(value, message1);

        if (message2.equalsIgnoreCase(""))
        {
            message2 = "The Integer does not fall between " + min + " and " + max;
        }

        if (valid == true)
        {
            int n = Integer.parseInt(value);
            if ((n >= min) && (n <= max))
            {
                validrange = true;
            } else
            {
                JOptionPane.showMessageDialog(null, message2, "Error",
JOptionPane.ERROR_MESSAGE);
                validrange = false;
            }
        }
        return validrange;
    }

    public static boolean v2IntegerRange(int entry, int min, int max) //checks if its between the
min and max values entered
    {
        boolean valid = false;
        if (entry >= min)
        {
            if (entry <= max)
            {
                valid = true;
            } else
            {
                JOptionPane.showMessageDialog(null, "Value entered is more than the maximum
value of " + max, "Invalid User Input", JOptionPane.ERROR_MESSAGE);
```

```
            }
        } else
        {
            JOptionPane.showMessageDialog(null, "Value entered is less than the minmum value
of  " + min, "Invalid User Input", JOptionPane.ERROR_MESSAGE);
        }

        return valid;
    }

    public static boolean vIntegerDigits(String value, String message1, String message2, int
digits)
    {
        boolean validdigits = false;

        boolean valid = vInteger(value, message1);

        if (message2.equalsIgnoreCase(""))
        {
            message2 = "The Integer Value is not a " + digits + " digit value";
        }

        if (valid == true)
        {
            int n = Integer.parseInt(value);
            double n1 = Math.pow(10, digits - 1);
            double n2 = Math.pow(10, digits) - 1;
            if ((n >= n1) && (n <= n2))
            {
                validdigits = true;
            } else
            {
                JOptionPane.showMessageDialog(null, message2, "Error",
JOptionPane.ERROR_MESSAGE);
                validdigits = false;
            }
        }
        return validdigits;
    }

    public static boolean vIntegerDigitsRange(String value, String message1, String message2,
String message3, int digits, int min, int max)
    {
        boolean validdigitsrange = false;

        boolean valid1 = vIntegerDigits(value, message1, message2, digits);

        if (valid1 == true)
        {
            boolean valid2 = vIntegerRange(value, message1, message3, min, max);
```

```
            if (valid2 == true)
            {
                validdigitsrange = true;
            } else
            {
                validdigitsrange = false;
            }
        }

        return validdigitsrange;
    }

    public static boolean vStringAZ(String value, String message1, String message2)
    {
        if (message1.equalsIgnoreCase(""))
        {
            message1 = "Only Letters (A to Z) are allowed";
        }

        if (message2.equalsIgnoreCase(""))
        {
            message2 = "Field cannot be left blank";
        }

        boolean valid = false;
        int lengthvalue = value.length() - 1;

        if (lengthvalue >= 0)
        {
            int k = 0;
            boolean invalid = false;
            while ((k <= lengthvalue) && (invalid == false))
            {
                if (!(Character.isLetter(value.charAt(k))))
                {
                    invalid = true;
                }
                k++;
            }

            if (invalid == true)
            {
                JOptionPane.showMessageDialog(null, message1, "Error",
JOptionPane.ERROR_MESSAGE);
                valid = false;
            } else
            {
                valid = true;
            }
        } else
```

```
        {
            JOptionPane.showMessageDialog(null, message2, "Error",
JOptionPane.ERROR_MESSAGE);
            valid = false;
        }

        return valid;
    }

    public static boolean vString09(String value, String message1, String message2)
    {
        if (message1.equalsIgnoreCase(""))
        {
            message1 = "Only Digits (0 to 9) are allowed";
        }

        if (message2.equalsIgnoreCase(""))
        {
            message2 = "Field cannot be left blank";
        }

        boolean valid = false;
        int lengthvalue = value.length() - 1;

        if (lengthvalue >= 0)
        {

            int k = 0;
            boolean invalid = false;
            while ((k <= lengthvalue) && (invalid == false))
            {
                if (!(Character.isDigit(value.charAt(k))))
                {
                    invalid = true;
                }
                k++;
            }

            if (invalid == true)
            {
                JOptionPane.showMessageDialog(null, message1, "Error",
JOptionPane.ERROR_MESSAGE);
                valid = false;
            } else
            {
                valid = true;
            }
        } else
        {
```

```
            JOptionPane.showMessageDialog(null, message2, "Error",
JOptionPane.ERROR_MESSAGE);
            valid = false;
        }

        return valid;
    }

    public static boolean vStringAZspace(String value, String message1, String message2)
    {
        if (message1.equalsIgnoreCase(""))
        {
            message1 = "Only Letters (A to Z) and Space/s are allowed";
        }

        if (message2.equalsIgnoreCase(""))
        {
            message2 = "Field cannot be left blank";
        }

        boolean valid = false;
        int lengthvalue = value.length() - 1;

        if (lengthvalue >= 0)
        {
            int k = 0;
            boolean invalid = false;
            while ((k <= lengthvalue) && (invalid == false))
            {
                if (!((Character.isLetter(value.charAt(k))) || (value.charAt(k) == ' ')))
                {
                    invalid = true;
                }
                k++;
            }

            if (invalid == true)
            {
                JOptionPane.showMessageDialog(null, message1, "Error",
JOptionPane.ERROR_MESSAGE);
                valid = false;
            } else
            {
                valid = true;
            }
        } else
        {
            JOptionPane.showMessageDialog(null, message2, "Error",
JOptionPane.ERROR_MESSAGE);
            valid = false;
```

```
        }

        return valid;
    }

    public static boolean vStringAZdash(String value, String message1, String message2)
    {
        if (message1.equalsIgnoreCase(""))
        {
            message1 = "Only Letters (A to Z) and Dash/es are allowed";
        }

        if (message2.equalsIgnoreCase(""))
        {
            message2 = "Field cannot be left blank";
        }

        boolean valid = false;
        int lengthvalue = value.length() - 1;

        if (lengthvalue >= 0)
        {
            int k = 0;
            boolean invalid = false;
            while ((k <= lengthvalue) && (invalid == false))
            {
                if (!((Character.isLetter(value.charAt(k))) || (value.charAt(k) == '-')))
                {
                    invalid = true;
                }
                k++;
            }

            if (invalid == true)
            {
                JOptionPane.showMessageDialog(null, message1, "Error",
JOptionPane.ERROR_MESSAGE);
                valid = false;
            } else
            {
                valid = true;
            }
        } else
        {
            JOptionPane.showMessageDialog(null, message2, "Error",
JOptionPane.ERROR_MESSAGE);
            valid = false;
        }

        return valid;
```

```
    }

    public static boolean vStringAZspaceDash(String value, String message1, String
message2)
    {
        if (message1.equalsIgnoreCase(""))
        {
            message1 = "Only Letters (A to Z), Space/s and Dash/es are allowed";
        }

        if (message2.equalsIgnoreCase(""))
        {
            message2 = "Field cannot be left blank";
        }

        boolean valid = false;
        int lengthvalue = value.length() - 1;

        if (lengthvalue >= 0)
        {
            int k = 0;
            boolean invalid = false;
            while ((k <= lengthvalue) && (invalid == false))
            {
                if (!((Character.isLetter(value.charAt(k))) || (value.charAt(k) == ' ') || (value.charAt(k)
== '-')))
                {
                    invalid = true;
                }
                k++;
            }

            if (invalid == true)
            {
                JOptionPane.showMessageDialog(null, message1, "Error",
JOptionPane.ERROR_MESSAGE);
                valid = false;
            } else
            {
                valid = true;
            }
        } else
        {
            JOptionPane.showMessageDialog(null, message2, "Error",
JOptionPane.ERROR_MESSAGE);
            valid = false;
        }

        return valid;
    }
```

```
    public static boolean vStringAZ09(String value, String message1, String message2)
    {
        if (message1.equalsIgnoreCase(""))
        {
            message1 = "Only Letters (A to Z) and Digits (0 - 9) are allowed";
        }

        if (message2.equalsIgnoreCase(""))
        {
            message2 = "Field cannot be left blank";
        }

        boolean valid = false;
        int lengthvalue = value.length() - 1;

        if (lengthvalue >= 0)
        {
            int k = 0;
            boolean invalid = false;
            while ((k <= lengthvalue) && (invalid == false))
            {
                if (!((Character.isLetter(value.charAt(k))) || (Character.isDigit(value.charAt(k)))))
                {
                    invalid = true;
                }
                k++;
            }

            if (invalid == true)
            {
                JOptionPane.showMessageDialog(null, message1, "Error",
JOptionPane.ERROR_MESSAGE);
                valid = false;
            } else
            {
                valid = true;
            }
        } else
        {
            JOptionPane.showMessageDialog(null, message2, "Error",
JOptionPane.ERROR_MESSAGE);
            valid = false;
        }

        return valid;
    }

    public static boolean vStringAZ09space(String value, String message1, String message2)
    {
```

```
        if (message1.equalsIgnoreCase(""))
        {
            message1 = "Only Letters (A to Z), Digits (0 - 9) and Space/s are allowed";
        }

        if (message2.equalsIgnoreCase(""))
        {
            message2 = "Field cannot be left blank";
        }

        boolean valid = false;
        int lengthvalue = value.length() - 1;

        if (lengthvalue >= 0)
        {
            int k = 0;
            boolean invalid = false;
            while ((k <= lengthvalue) && (invalid == false))
            {
                if (!((Character.isLetter(value.charAt(k))) || (Character.isDigit(value.charAt(k))) ||
(value.charAt(k) == ' ')))
                {
                    invalid = true;
                }
                k++;
            }

            if (invalid == true)
            {
                JOptionPane.showMessageDialog(null, message1, "Error",
JOptionPane.ERROR_MESSAGE);
                valid = false;
            } else
            {
                valid = true;
            }
        } else
        {
            JOptionPane.showMessageDialog(null, message2, "Error",
JOptionPane.ERROR_MESSAGE);
            valid = false;
        }

        return valid;
    }

    public static boolean vStringAZ09SpaceComma(String value, String message1, String
message2)
    {
        if (message1.equalsIgnoreCase(""))
```

```
        {
            message1 = "Only Letters (A to Z), Digits (0 - 9), Space/s and Dash/es are allowed";
        }

        if (message2.equalsIgnoreCase(""))
        {
            message2 = "Field cannot be left blank";
        }

        boolean valid = false;
        int lengthvalue = value.length() - 1;

        if (lengthvalue >= 0)
        {
            int k = 0;
            boolean invalid = false;
            while ((k <= lengthvalue) && (invalid == false))
            {
                if (!((Character.isLetter(value.charAt(k))) || (Character.isDigit(value.charAt(k))) ||
(value.charAt(k) == ' ') || (value.charAt(k) == ',')))
                {
                    invalid = true;
                }
                k++;
            }

            if (invalid == true)
            {
                JOptionPane.showMessageDialog(null, message1, "Error",
JOptionPane.ERROR_MESSAGE);
                valid = false;
            } else
            {
                valid = true;
            }
        } else
        {
            JOptionPane.showMessageDialog(null, message2, "Error",
JOptionPane.ERROR_MESSAGE);
            valid = false;
        }

        return valid;
    }

    public static boolean vStringAZrange(String value, String message1, String message2,
String message3, int size)
    {
        boolean validrange = false;
```

```
        boolean valid = vStringAZ(value, message1, message2);

        if (message3.equalsIgnoreCase(""))
        {
            message3 = "Please enter a value that has at least " + size + " character/s";
        }

        if (valid == true)
        {
            if (value.length() == size)
            {
                validrange = true;
            } else
            {
                JOptionPane.showMessageDialog(null, message3, "Error",
JOptionPane.ERROR_MESSAGE);
                validrange = false;
            }

        }
        return validrange;
    }

    public static boolean vString09range(String value, String message1, String message2,
String message3, int size)
    {
        boolean validrange = false;

        boolean valid = vString09(value, message1, message2);

        if (message3.equalsIgnoreCase(""))
        {
            message3 = "Please enter a value that has at least " + size + " character/s";
        }

        if (valid == true)
        {
            if (value.length() == size)
            {
                validrange = true;
            } else
            {
                JOptionPane.showMessageDialog(null, message3, "Error",
JOptionPane.ERROR_MESSAGE);
                validrange = false;
            }

        }
        return validrange;
    }
```

```
    public static boolean vStringAZspaceRange(String value, String message1, String
message2, String message3, int size)
    {
        boolean validrange = false;

        boolean valid = vStringAZspace(value, message1, message2);

        if (message3.equalsIgnoreCase(""))
        {
            message3 = "Please enter a value that has at least " + size + " character/s";
        }

        if (valid == true)
        {
            if (value.length() == size)
            {
                validrange = true;
            } else
            {
                JOptionPane.showMessageDialog(null, message3, "Error",
JOptionPane.ERROR_MESSAGE);
                validrange = false;
            }

        }
        return validrange;
    }

    public static boolean vStringAZ09range(String value, String message1, String message2,
String message3, int size)
    {
        boolean validrange = false;

        boolean valid = vStringAZ09(value, message1, message2);

        if (message3.equalsIgnoreCase(""))
        {
            message3 = "Please enter a value that has at least " + size + " character/s";
        }

        if (valid == true)
        {
            if (value.length() == size)
            {
                validrange = true;
            } else
            {
                JOptionPane.showMessageDialog(null, message3, "Error",
JOptionPane.ERROR_MESSAGE);
```

```
            validrange = false;
        }

    }
    return validrange;
}

public static boolean vEmail(String emailAdd, String message)
{
    boolean valid = false;

    //check if its blank
    if (emailAdd.equalsIgnoreCase(""))
    {

        message = "Email field cannot be left blank";
    } else
    {
        if (message.equalsIgnoreCase(""))
        {
            message = "Format for email address incorrect";
        }

        String pattern = "^([a-zA-Z0-9_\\-\\.]+)@((\\[[0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}\\.)|(([a-zA-
Z0-9\\-]+\\.)+))([a-zA-Z]{2,4}|[0-9]{1,3})(\\]?)$";
        if (emailAdd.matches(pattern))
        {
            valid = true;
        } else
        {
            JOptionPane.showMessageDialog(null, message, "Error",
JOptionPane.ERROR_MESSAGE);
        }
    }
    return valid;
}

//tweaked version of original method
public static boolean vEmail2(String emailAdd, String message)
{
    boolean valid = false;

    //check if its blank
    if (emailAdd.equalsIgnoreCase(""))
    {

        message = "Email field cannot be left blank";
    } else
    {
        message = "Format for email address incorrect";
```

```
        }

        String pattern = "^([a-zA-Z0-9_\\-\\.]+)@((\\[[0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}\\.)|(([a-zA-Z0-
9\\-]+\\.)+))([a-zA-Z]{2,4}|[0-9]{1,3})(\\]?)$";
        if (emailAdd.matches(pattern))
        {
            valid = true;
        } else
        {
            JOptionPane.showMessageDialog(null, message, "Error",
JOptionPane.ERROR_MESSAGE);
        }

        return valid;
    }

    public static boolean vWebsite(String webAdd, String message)
    {
        boolean valid = false;

        if (webAdd.equalsIgnoreCase(""))
        {
            message = "Please enter a web address";
        } else
        {
            if (message.equalsIgnoreCase(""))
            {
                message = "Format for web address incorrect";
            }

            String pattern = "^http(s{0,1})://[a-zA-Z0-9_/\\-\\.]+\\.([A-Za-z/]{2,5})[a-zA-Z0-
9_/\\&\\?\\=\\-\\.\\~\\%]*";
            if (webAdd.matches(pattern))
            {
                valid = true;
            } else
            {
                JOptionPane.showMessageDialog(null, message, "Error",
JOptionPane.ERROR_MESSAGE);
            }
        }
        return valid;
    }

    public static boolean vDateString(String theDate, String dateFormat, String message)
    {
        boolean valid = false;

        if (dateFormat.equalsIgnoreCase(""))
        {
```

```
            dateFormat = "dd-MM-yyyy";
        }

        if (message.equalsIgnoreCase(""))
        {
            message = "Invalid Date. Please enter a valid date in the format " + dateFormat;
        }

        try
        {
            SimpleDateFormat d = new SimpleDateFormat(dateFormat);
            d.setLenient(false);
            d.parse(theDate);

            if (vIntegerDigitsRange(theDate.substring(dateFormat.indexOf("y")), "There must be a
CORRECT year value", "The year must have FOUR digits", "Invalid year [1900 - " + (new
java.util.Date().getYear() + 1900) + "]", 4, 1900, new java.util.Date().getYear() + 1900))
            {
                valid = true;
            }

        } catch (Exception e)
        {
            JOptionPane.showMessageDialog(null, message, "Error",
JOptionPane.ERROR_MESSAGE);
        }

        return valid;
    }

    public static boolean vDateParts(String day, String month, String year)
    {
        boolean valid = false;

        if ((vIntegerRange(day, "There must be at least one day", "At most a month can only
have 31 days", 1, 31)) && (vIntegerRange(month, "There must be at least one month", "There
are only 12 Months in a year", 1, 12)) && (vIntegerDigitsRange(year, "There must be a year
value", "A year must have FOUR digits", "Invalid year [1900 - " + (new
java.util.Date().getYear() + 1900) + "]", 4, 1900, new java.util.Date().getYear() + 1900)))
        {
            valid = true;

            switch (Integer.parseInt(month))
            {
                case 4:
                case 6:
                case 9:
                case 11:
                {
                    if (Integer.parseInt(day) > 30)
```

```
                {
                    valid = false;
                    JOptionPane.showMessageDialog(null, "There are only THIRTY days in this
month", "Error", JOptionPane.ERROR_MESSAGE);
                }

                break;
            }

            case 2:
            {
                if (Integer.parseInt(year) % 4 == 0)
                {
                    if (Integer.parseInt(day) > 29)
                    {
                        valid = false;
                        JOptionPane.showMessageDialog(null, "There are only 29 days in February
for this year", "Error", JOptionPane.ERROR_MESSAGE);

                    }
                } else
                {
                    if (Integer.parseInt(day) > 28)
                    {
                        valid = false;
                        JOptionPane.showMessageDialog(null, "There are only 28 days in February
for this year", "Error", JOptionPane.ERROR_MESSAGE);
                    }
                }

                break;
            }
        }
    }
    return valid;
}

public static boolean vTimeParts(String hour, String minutes, String seconds)
{
    boolean valid = false;

    if ((vIntegerRange(hour, "Invalid hour value [0 - 23]", "There are 24 hours in a day [0 -
23]", 0, 23)) && (vIntegerRange(minutes, "Invalid minute value [0 - 59]", "There are 60
minutes in a hour [0 - 59]", 0, 59)) && (vIntegerRange(seconds, "Invalid second value [0 -
59]", "There are 60 seconds in minute [0 - 59]", 0, 59)))
    {
        valid = true;
    }
    return valid;
}
```

```
    public static boolean vTimeString(String theTime, String message)
    {
        boolean valid = false;

        String timeFormat = "HH:mm:ss";

        if (message.equalsIgnoreCase(""))
        {
            message = "Invalid Time. Please enter a valid time in the format  " + timeFormat + "
[0-23] : [0-59] : [0-59]";
        }

        try
        {
            SimpleDateFormat t = new SimpleDateFormat(timeFormat);
            t.setLenient(false);
            t.parse(theTime);
            valid = true;
        } catch (Exception e)
        {
            JOptionPane.showMessageDialog(null, message, "Error",
JOptionPane.ERROR_MESSAGE);
        }

        return valid;
    }

    public String CurrentAge(String birthDate, String dateFormat, String message)
    {
        String line = "";

        try
        {
            if (dateFormat.equalsIgnoreCase(""))
            {
                dateFormat = "dd-MM-yyyy";
            }

            if (message.equalsIgnoreCase(""))
            {
                message = "Invalid Date. Please enter a valid date in the format " + dateFormat;
            }

            if (vDateString(birthDate, dateFormat, message))
            {
                Date dd = new SimpleDateFormat(dateFormat).parse(birthDate);

                int dayP = dd.getDate();
                int monthP = dd.getMonth() + 1;
```

```
int yearP = dd.getYear() + 1900;

int day = new java.util.Date().getDate();
int month = new java.util.Date().getMonth() + 1;
int year = new java.util.Date().getYear() + 1900;

int y = year - yearP;

int m = 0;

int d = 0;

int numD = 0;

if (month < monthP)
{
   m = (month - monthP) + 12;
   y = y - 1;
} else
{
   m = month - monthP;
}

switch (month)
{
   case 4:
   case 6:
   case 9:
   case 11:
   {
      numD = 30;
      break;
   }

   case 1:
   case 3:
   case 5:
   case 7:
   case 8:
   case 10:
   case 12:
   {
      numD = 31;
      break;
   }

   case 2:
   {
      if (year % 4 == 0)
      {
```

```
                    numD = 29;
                } else
                {
                    numD = 28;
                }

                break;
            }
        }

        if (day < dayP)
        {
            d = (day - dayP) + numD;
            m = m - 1;
        } else
        {
            d = day - dayP;
        }

        line = y + " year/s,  " + m + " month/s, " + d + " day/s";

      }
    } catch (Exception ex)
    {

    }
    return line;

  }

}
```

DbManager Class

```
package dataPkg;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class DbManager
{

    Connection conn;

    public DbManager()
    {
        try
        {
            String filename = "CasinoRecordsV3.accdb";

            conn = DriverManager.getConnection("jdbc:ucanaccess://" + filename); //we were told
that all that will change is the filename over her. Nothign else needs to be changed for it to
work

        } catch (Exception e)
        {
            //System.out.println(e.toString());
            System.out.println(e.toString());
        }
    }

    public ResultSet query(String SQL) throws SQLException
    {
        Statement stmt = conn.createStatement();
        ResultSet result = stmt.executeQuery(SQL);
        return result;
    }

    public int update(String SQL) throws SQLException
    {
        Statement stmt = conn.createStatement();
        int done = stmt.executeUpdate(SQL);
        return done;
    }

    public int updateReturnID(String SQL) throws SQLException
    {
        Statement stmt = conn.createStatement();
```

```
        int id = -1;
        stmt.executeUpdate(SQL, Statement.RETURN_GENERATED_KEYS);
        ResultSet result = stmt.getGeneratedKeys();
        if (result.next())
        {
            id = result.getInt(1);
        }
        return id;
    }
}
```

populateJTable

The populateJTable method is used across the 3 classes but it is essentially the same code with different variables and methods being called in each as per the object class and table being populated

VisitsData

```
public void populateJTable(javax.swing.JTable table, int rowSelect)
   {
      DefaultTableModel model = (DefaultTableModel) table.getModel();

      model.setRowCount(0);

      for (int i = 0; i < visitsList.size(); i++)
      {
         dataPkg.Visits v = visitsList.get(i);
         Object[] rowData =
         {
            v.getVisitNo(), v.getEventID(), v.getPatronID(), v.getAmountSpent(),
v.getUsername(), v.getDateOfVisit()
         };
         model.addRow(rowData);
      }

      table.setModel(model);

      if (table.getRowCount() > 0)
      {
         table.setRowSelectionInterval(rowSelect, rowSelect);
      }
   }
```

EventsData

```
public void populateJTable(javax.swing.JTable table, int rowSelect)
   {
      DefaultTableModel model = (DefaultTableModel) table.getModel();

      model.setRowCount(0);

      for (int i = 0; i < eventsList.size(); i++)
      {
         dataPkg.Events e = eventsList.get(i);
         Object[] rowData =
         {
            e.getEventID(), e.getEventName(), e.getStartDate(), e.getEndDate(),
e.getLocation(), e.getCapacity(), e.getStatus(), e.getRegistrationDeadline()
         };
         model.addRow(rowData);
      }

      table.setModel(model);

      if (table.getRowCount() > 0)
      {
         table.setRowSelectionInterval(rowSelect, rowSelect);
      }
   }
```

PatronsData

```
public void populateJTable(javax.swing.JTable table, int rowSelect) //in this parameter we're getting a whoe table, javax.swing.JTable
   { //everytime you change the view of the table you have to call populateJTable().
      DefaultTableModel model = (DefaultTableModel) table.getModel();

      model.setRowCount(0); //clears the table

      for (int i = 0; i < patronsList.size(); i++)
      {
         dataPkg.Patrons p = patronsList.get(i); //fine
         Object[] rowData =
         {
            p.getPatronID(), p.getFirstName(), p.getSurname(), p.getGender(), p.getDateOfBirth(), p.getHomeAddress(), p.getEmailAddress(), p.getCardLevel(), p.getJoinDate()
         };
         model.addRow(rowData);
      }

      table.setModel(model);

      if (table.getRowCount() > 0) //if there is something in the first row, then
      {
         table.setRowSelectionInterval(rowSelect, rowSelect);
      }

   }
```

Text formatting line

```
int extractedNewPK = Integer.parseInt(newPK.replaceAll("[^\\d]", ""));
```

Date code line

```
Calendar calendar = Calendar.getInstance();
calendar.setTime(new Date());

//Subtract one day
calendar.add(Calendar.DAY_OF_MONTH, -1);

//Get the new date (day before today)
Date dayBeforeToday = calendar.getTime();
```

4.1.2 Explanation of critical algorithms

Adding a Visit

Explanation

The core functionality of the program is to log a new visit to the database. This feature tracks patronage and the events attended, providing essential insights into the casino's operations. It integrates data from both the TblEvents (EventsTable) and TblPatrons (PatronsTable), utilizing foreign keys to ensure data consistency and integrity. By adding a visit, the program creates a comprehensive record that combines information from various tables to accurately document each visit. This detailed tracking allows for better management and analysis of patron behavior and event popularity, ultimately contributing to more informed decision-making and improved casino operations.

Therefore, logging a visit can be said to be the core aspect of the program

Pseudocode

VisitsTableFrame – btnAdd

```
cbManualEdit.enabled ← true
cbManualEdit.selected ← false

navigation(false)
search(false)
options(false)
details(true)

btnSaveNew.enabled ← true
btnCancel.enabled ← true

txfVisitNo.enabled ← false

newPKrs ← dbm.query("SELECT Nz(Max(VISITNO), 0) + 1 FROM TBLVISITS")
newPK ← 0
IF newPKrs.next()
        newPK ← newPKrs.getInt(1)
ENDIF
txfVisitNo.text ← newPK

txfAmountSpent.text ← ""
jdcDateOfVisit.date ← current_date

EventIDPKrs ← dbm.query("SELECT EVENTID FROM TBLEVENTS")
firstEventID ← ""
IF EventIDPKrs.next() THEN
        firstEventID ← EventIDPKrs.getString("EVENTID")
ENDIF
cmbEventID.selectedItem ← firstEventID
```

```
patronIDPKrs ← dbm.query("SELECT PATRONID FROM TBLPATRONS")
firstPatronID ← ""
IF patronIDPKrs.next() THEN
        firstPatronID ← patronIDPKrs.getString("PATRONID")
ENDIF
cmbPatronID.selectedItem ← firstPatronID

usernamePKrs ← dbm.query("SELECT USERNAME FROM TBLUSERS")
firstUsername ← ""
IF usernamePKrs.next() THEN
        firstUsername ← usernamePKrs.getString("USERNAME")
ENDIF
cmbUsername.selectedItem ← firstUsername

txfAmountSpent.requestFocus()
```

VisitsTableFrame – btnSaveNew

```
cbManualEdit.enabled ← false
cbManualEdit.selected ← false

df ← new SimpleDateFormat("yyyy-MM-dd")

newVisitNo ← txfVisitNo.text.trim()
newPatronID ← cmbPatronID.selectedItem
newEventID ← cmbEventID.selectedItem
newAmountSpent ← txfAmountSpent.text.trim()
newUsername ← cmbUsername.selectedItem

newDateOfVisitRaw ← jdcDateOfVisit.date
newDateOfVisit ← df.format(newDateOfVisitRaw)

IF Validation.vVisitNoCheck(newVisitNo) THEN
        IF Validation.vDoublePositive(newAmountSpent, "") THEN
                vd.addVisit(newVisitNo.toInteger(), newEventID, newPatronID,
                newAmountSpent.toDouble(), newUsername, newDateOfVisit)
                vd.populateJTable(tblVisits, vd.getVisitPosition(newVisitNo.toInteger()))
                populateDetails()

                navigation(true)
                search(true)
                options(true)
                details(false)

                btnSaveNew.enabled ← false
                btnCancel.enabled ← false
        ELSE
                JOptionPane.showMessageDialog(null, "Entry in Amount Spent field is invalid",
                "Error", JOptionPane.ERROR_MESSAGE)
                txfAmountSpent.text ← ""
                txfAmountSpent.requestFocus()
        ENDIF
ELSE
        JOptionPane.showMessageDialog(null, "Entry in VisitNo field is invalid", "Error",
        JOptionPane.ERROR_MESSAGE)
        cbManualEdit.selected ← true
        txfVisitNo.enabled ← true
        txfVisitNo.text ← ""
        txfVisitNo.requestFocus()
ENDIF
```

VisitsDate Class – addVisit Method

```
IF db.update("INSERT INTO TBLVISITS(VISITNO, EVENTID, PATRONID, AMOUNTSPENT,
USERNAME, DATEOFVISIT) VALUES(" + visitNo + ", '" + eventID + "', '" + patronID + "', " +
amountSpent + ", '" + username + "', #" + dateOfVisit + "#)") > 0 THEN
        CALL getAllVisits()
        JOptionPane.showMessageDialog(null, "Visit successfully added to database",
        "INFORMATION", JOptionPane.INFORMATION_MESSAGE)
ELSE
        JOptionPane.showMessageDialog(null, "Visit NOT added to database", "ERROR",
        JOptionPane.ERROR_MESSAGE)
ENDIF
```

4.1.3 Advanced Techniques

Timer

Explanation

The code creates a timer that updates a progress bar and label to simulate a loading process. The loading process is complete when the percentage reaches 100, at which point the timer is stopped, the SplashScreen is disposed of, and the LoginFrame is made visible. The speed of the loading bar can be controlled by adjusting the third argument in the scheduleAtFixedRate method.

This code creates a timer that updates a progress bar and a label to simulate a loading process.

This code is for the splash screen. The progress bar increments every 20 milliseconds (controlled by the Timer and TimerTask classes). Once the progress bar reaches 100%, the splash screen is closed, and a new login frame is displayed.

Code

```
final Timer t = new Timer();

        TimerTask tt = new TimerTask()
        {
            public void run()
            {
                perc++;
                if (perc < 101)
                {
                    sf.pbLoading.setValue(perc);
                    sf.lblLoadingPerc.setText(perc + " %");
                } else
                {
                    try
                    {
                        t.cancel();
                        sf.dispose();
                        new LoginFrame().setVisible(true);
                    } catch (Exception e)
                    {
                        System.out.println(e.toString());
                    }

                }

            }
        };
        t.scheduleAtFixedRate(tt, 0, 20); //the second number here controls the speed at which the loading bar moves. 20 or 45 is a good speed tho
```

Bar Graph

Explanation

This code is for the ReportsFrame. It generates a bar chart using the JFreeChart library in Java. The chart displays the number of patrons for each card level.

The code connects to the database, retrieves data, and uses JFreeChart to generate a bar chart displaying the number of patrons for each card level.

Code

```
ResultSet rs1 = db.query("SELECT CARDLEVEL, COUNT(*) FROM TBLPATRONS GROUP BY CARDLEVEL;");

    List<String> categoryLabels = new ArrayList(); //in sirs one this is harcoded. We're getting rw vals from db for this
    List<Integer> values = new ArrayList();

    while (rs1.next())
    {
       //getting value for the first column and adds it to categoryLabels
       categoryLabels.add(rs1.getString(1));
       //getting value for the second column and adds it to values
       values.add(rs1.getInt(2)); //the number is the column name
    }

    DefaultCategoryDataset data = new DefaultCategoryDataset();

    for (int j = 0; j < categoryLabels.size(); j++)
    {
       data.addValue(values.get(j), "", categoryLabels.get(j));
    }
    JFreeChart barChart = ChartFactory.createBarChart("Card Types", "Card Types", "Number of Patrons", data, PlotOrientation.VERTICAL, true, true, false);

    ChartFrame graphFrame = new ChartFrame("Bar Graph", barChart);
    graphFrame.pack();
    graphFrame.setLocationRelativeTo(null);
    graphFrame.setVisible(true);
```

Pie Chart

Explanation

This code generates a pie chart using the JFreeChart library in Java. The chart displays the count of patrons by gender.

This code connects to the database, retrieves data on patrons' genders, and uses JFreeChart to generate a pie chart displaying the count of patrons by gender.

The ChartFactory class is used to create the pie chart, and the DefaultPieDataset class is used to store the data for the chart.

Code

```
ResultSet rs2 = db.query("SELECT GENDER, COUNT(*) FROM TBLPATRONS GROUP BY GENDER;");

    List<String> gender = new ArrayList(); //in sirs one this is harcoded. We're getting rw vals from db for this
    List<Integer> value = new ArrayList();

    while (rs2.next())
    {

       gender.add(rs2.getString(1));

       value.add(rs2.getInt(2)); //the number is the column name
    }

    DefaultPieDataset data = new DefaultPieDataset();

    for (int i = 0; i < gender.size(); i++)
    {
       data.setValue(gender.get(i), (int) value.get(i));
    }

    JFreeChart pieChart = ChartFactory.createPieChart("Patrons' Gender", data, true, true, false);

    ChartFrame graphFrame = new ChartFrame("Gender Pie Chart", pieChart);
    graphFrame.pack();
    graphFrame.setLocationRelativeTo(null);
    graphFrame.setVisible(true);
```

Playing a video

Explanation

This code opens the help video using the default video player. If the file is not found or cannot be opened, it displays an error message.

A trycatch statement is used in case the video file is not found.

Code

```
try
        {
            Desktop.getDesktop().open(new File("test.mp4"));
        } catch (Exception e)
        {
            JOptionPane.showMessageDialog(null, "Video not found", "Error",
JOptionPane.ERROR_MESSAGE);
        }
```