List of Techniques Used:

Overview of Techniques:

1. Using radio buttons, combo boxes, and JCalendar for specific data entry

2. JTables for displaying data in a row-column format.

3. JTable Rowfilters using RegexFilter (Regex Library)

4. Create statements with appropriate primary key, nullable values, and appropriate data type restraints

5. Adding, Updating, and deleting records in a database table

6. SELECT statements with sort methods and JOIN statements

7. Error Trapping to make sure all data entered is valid

8. Dynamic polymorphism through constructor overloading

9. Automatic calculation of certain values.

10. Print JTable contents to a PDF

11. Emailing the user to do the payroll

1. Using radio buttons, combo boxes, and JCalendar for specific data entry
    a. JCalendar date selection

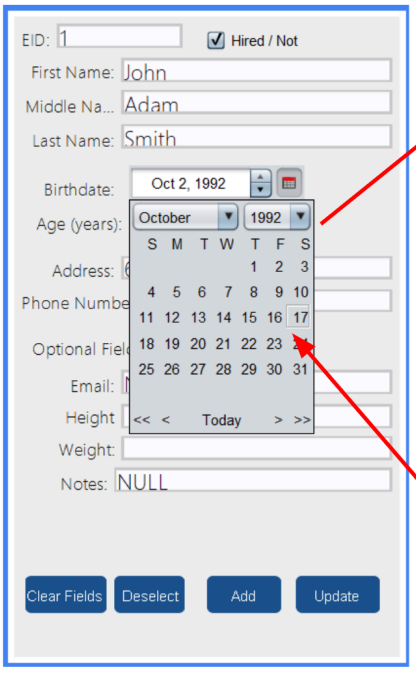
```
import java.util.Date;
    private de.wannawork.jcalendar.JCalendarComboBox jCalendarBirthdate;
private void jCalendarBirthdateStateChanged(javax.swing.event.ChangeEvent evt) {
    try{
        Date startDate = new Date(); //the current date
        Date endDate = jCalendarBirthdate.getDate(); //The date inputted by the user.

        // Calculate the difference in milliseconds
        long differenceInMillis = startDate.getTime() - endDate.getTime();

        // Calculate the number of years (approximate)
        long years = differenceInMillis / (1000L * 60 * 60 * 24 * 365);

        if (years<16){
            JOptionPane.showMessageDialog( parentComponent: null,  message: "Age must be at least 16.");
            txtAge.setText( t: "");
        }
        else{
            txtAge.setText( t: Long.toString( i: years));
        }
    }
    catch (Exception e){

    }

}
```

This uses the Date and jCalendar external libraries' built-in functions.

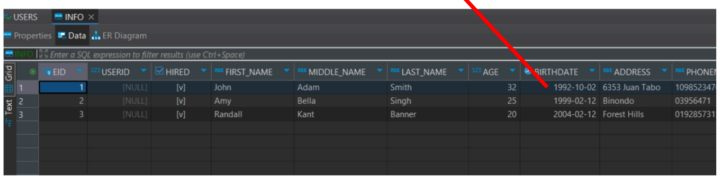
Gets the difference between the two date objects in milliseconds

Makes sure the age of the employee is at least 16.

Sets the Age textbox to the age in years calculated beforehand, showing the user the current age of the employee

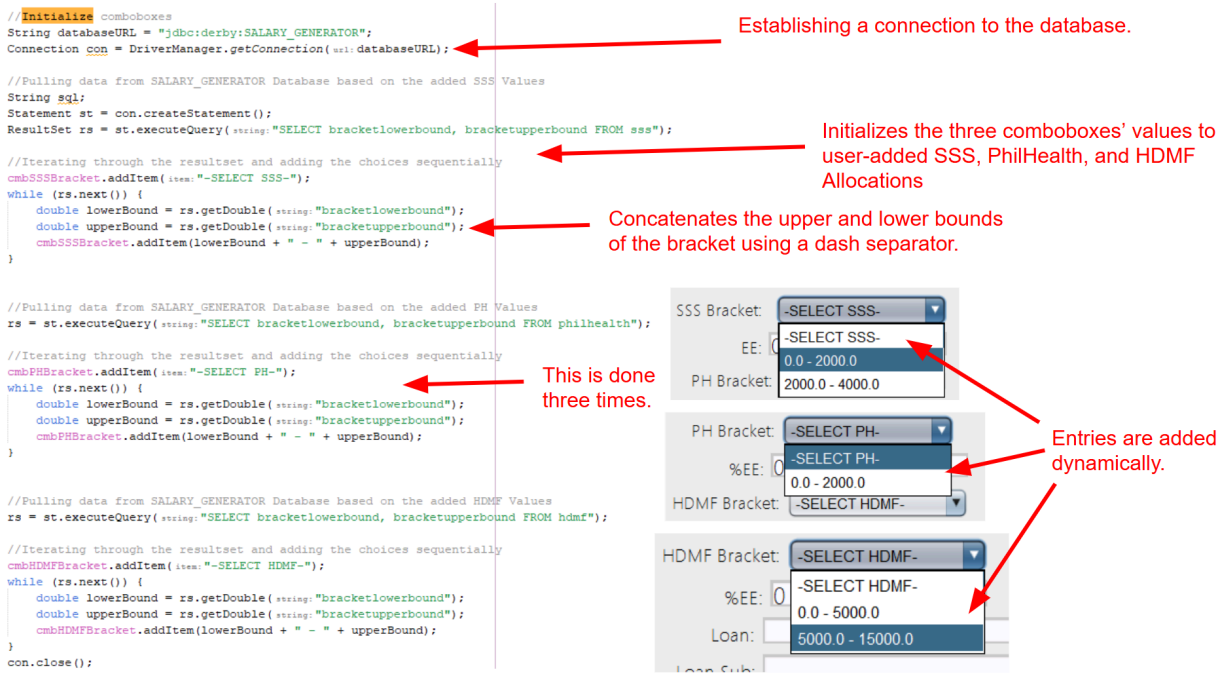


The JCalendar pop-up allows the user to select a specific date. This is saved as a date string in the INFO table

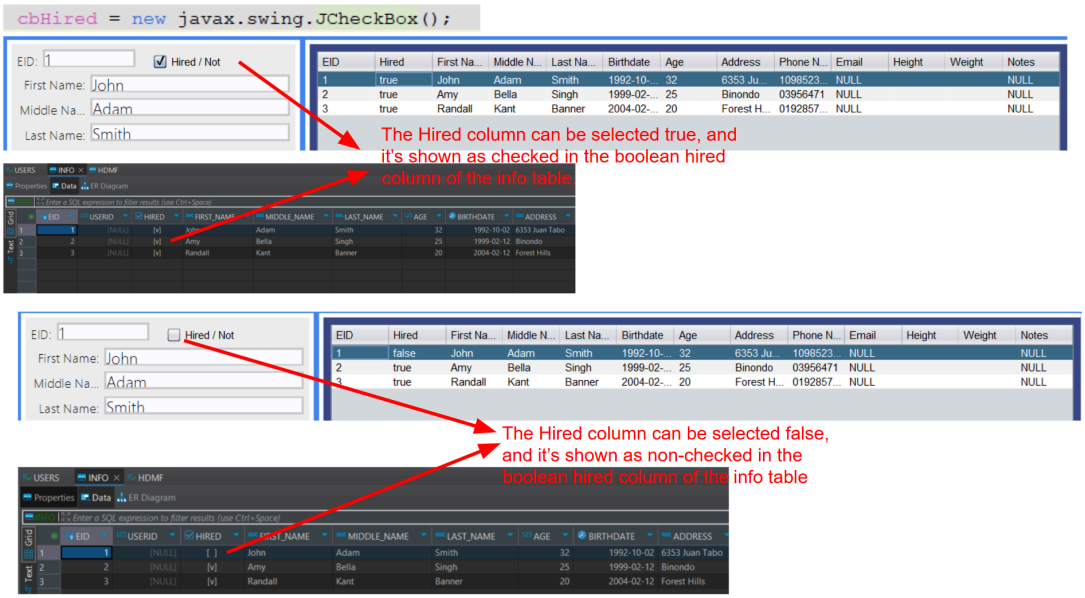


The JCalendar box handles this functionality. It generates the calendar of that month and year, and returns a date object of the user's selection

b. Using combo boxes for social security input presets



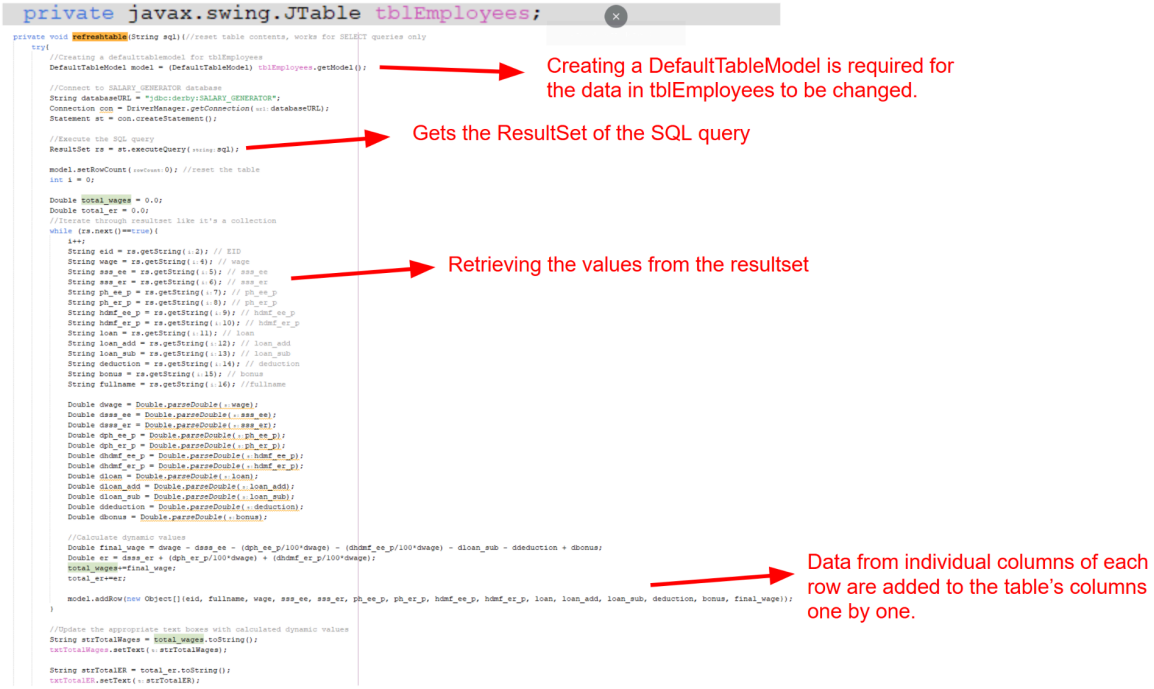
c. Checkbox for boolean values



Ingenuity: The GUI only allows a certain set of values to be input. The user is now unable to input invalid data into certain columns. This reduces error-checking, and increases the efficiency and speed of the application.

Appropriateness: The GUI's design connotes the type of data it accepts. (Boolean for checkbox, dates for JCalendar, Comboboxes for preset values). It's more intuitive for the client when they use the app, and they make fewer mistakes.

2. JTables for displaying data in a row-column format.



EID	Full N...	Wage	SSS EE	SSS ...	PH EE	PH ER	HDMF...	HDMF...	Loan	Add	Sub	Bonus	Deduc...	Final ...
2	Singh,...	4500.00	200.00	200.00	3.00	3.00	2.00	2.00	0.00	0.00	0.00	0.00	0.00	4075.0
3	Banne...	5000.00	0.00	0.00	0.00	0.00	0.00	0.00	600.00	0.00	0.00	0.00	0.00	5000.0

Data is shown in a row-column format.

Ingenuity: The tblEmployees table mirrors how the data is stored in the database. Data can be displayed by the system sequentially and easily. Additionally, specific rows can be updated or added, only requiring efficient SQL queries.

Appropriateness: The user understands how the data is stored within the database, and how the database changes depending on their query. This makes it much easier to interact with the system.

3. JTable Rowfilters using RegexFilter (Regex Library)

(The same code is used for the Employee Records and Salary Generator pages)

```
//Global Variables
int max = 0;
int payperiod = -1;
String email = "", uid = "";
String selectedroweid = "-1";
//Setting up the regexfilter
TableRowSorter sorter;
```

Defining a global TableRowSorter variable

```
//For the regexfilter later on
TableRowSorter sorter = new TableRowSorter<>((DefaultTableModel)tblEmployees.getModel());
this.sorter = sorter;
tblEmployees.setRowSorter(sorter);
```

Assigning a TableRowSorter variable tied to tblEmployees

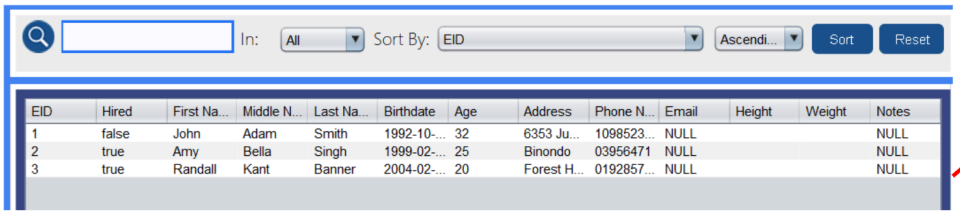
```
private void search(){
    try{
        int in = cmbIn.getSelectedIndex();

        String search = txtSearch.getText().trim();

        //Search using RowFilter and regex (regular expression)
        RowFilter<DefaultTableModel, Object> rf = null;
        try {
            if (in == 0){
                rf = RowFilter.regexFilter("(?i)" + search);   //("(?i)" for case insensitive filter
            }
            else{
                //to search a specific column
                rf = RowFilter.regexFilter("(?i)" + search, in - 1);
            }
        }
        catch (java.util.regex.PatternSyntaxException e) {
//              System.out.println("Error Found");
            return;
        }
        //The set RowFilter only shows table values that satisfy the search parameters
        sorter.setRowFilter(filter:rf);
    }
    catch (Exception e){
        e.printStackTrace();
    }
}
```

Trimming the user's search term

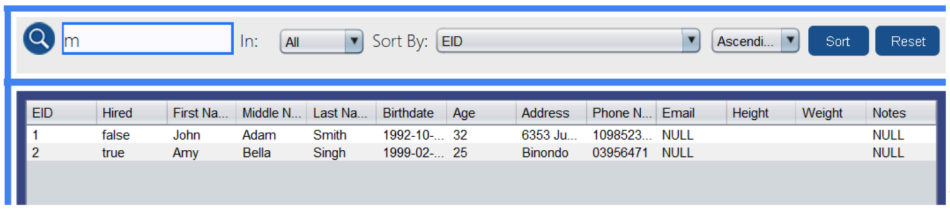
A Regex (Regular Expression) filter is applied to the jTable based on the search parameters in the text box.

Regex applied to just one specific selected column

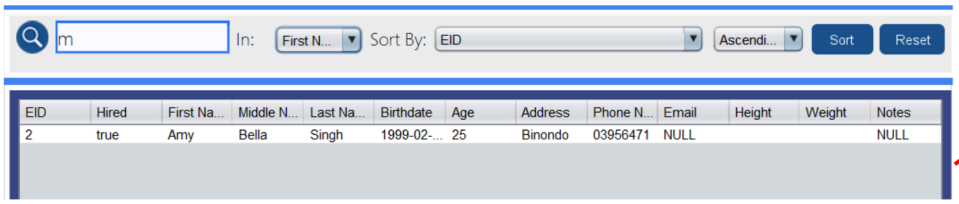
The jTable will now only show employees whose data match the search parameters. The RowFilter makes it so entries that don't meet the criteria are hidden, but not deleted. These can be selected, updated, deleted, etc.



With no search terms, no RowFilter applied



All records with an 'm' in any of their columns.



All records with an 'm' in the first name column. "A(m)y"

(Salary Generator Page)

(Anjali Sajeevan, 2021)

Ingenuity: The Regex Filter leverages Java's prebuilt libraries. The RowFilter removes the need to constantly query the SALARY_GENERATOR database. It's now much quicker than running linear searches across all columns

Appropriateness: The user can use the search function to sort the table extremely easily. With the large amount of data in the table, the search function is a necessity in ensuring usability and versatility.

4. Create statements with appropriate primary key, nullable values, and appropriate data type restraints

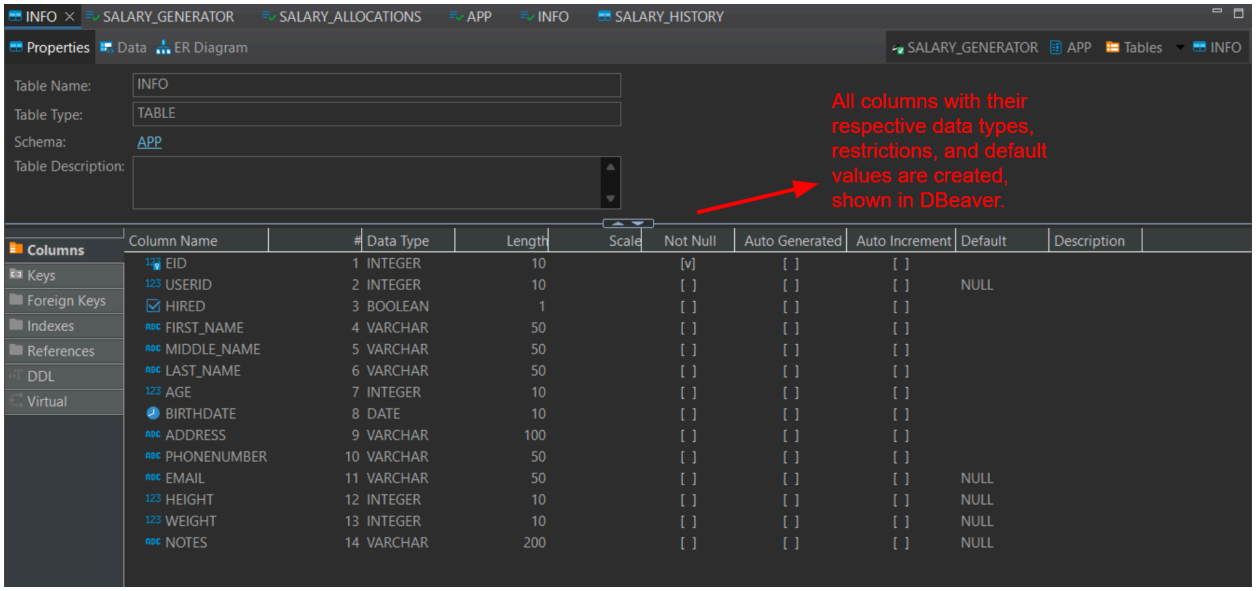
```
public SalaryGenerator(String uid, String email){ //CATCH ALL INCOMING DATA
    initComponents();
    try{

        //Assign email variable to a global variable to be used by other components
        this.email = email;

        //Establish a connection to the SALARY_GENERATOR database
        String currentURL = "jdbc:derby:SALARY_GENERATOR;create=true";
        Connection con = DriverManager.getConnection( url: currentURL);


        Statement st = con.createStatement();

        //Check if table exists, no need to overwrite an existing table
        if (!doesTableExists( tableName: "info",  conn: con)){
            //Create table for employee info
            String sql = "CREATE TABLE info "
                    + "(eid integer PRIMARY KEY, " //Primary key condition
                    + "userid integer DEFAULT NULL, " //Default null data type
                    + "hired boolean, " //boolean type for optimized data storage
                    + "first_name varchar(50), middle_name varchar(50), "
                    + "last_name varchar(50), age integer, "
                    + "birthdate date," //specialized date datatype
                    + "address varchar(100), phonenumber varchar(50), email varchar(50) DEFAULT NULL,"
                    + "height integer DEFAULT NULL, weight integer DEFAULT NULL, notes varchar(200) DEFAULT NULL)";
            st.execute( string: sql);
        }
```

Establishes a connection to the database and creates a table if one doesn't exist yet.

Specific data types, primary key restrictions, and default values are implemented.

Primary Key Condition

Default Null



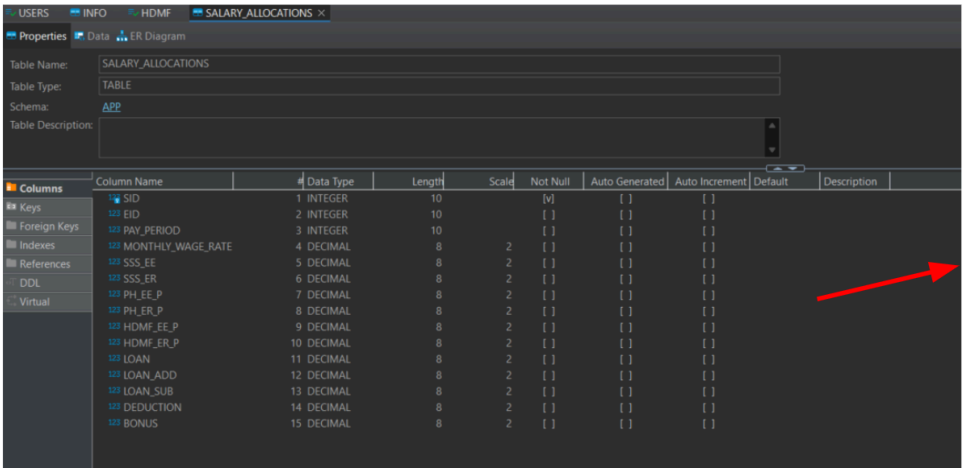
All columns with their respective data types, restrictions, and default values are created, shown in DBeaver.

```
if (!doesTableExists( tableName: "salary_allocations",  conn: con)){
    String sql = "CREATE TABLE salary_allocations "
            + "(sid integer PRIMARY KEY, eid integer, pay_period integer,"
            + "monthly_wage_rate decimal(8,2), sss_ee decimal(8,2)," //maximum bounds for currency values
            + "sss_er decimal(8,2), ph_ee_p decimal(8,2), ph_er_p decimal(8,2)," //8 whole number digits, 2 decimal places
            + "hdmf_ee_p decimal(8,2), hdmf_er_p decimal(8,2), loan decimal(8,2), "
            + "loan_add decimal(8,2), loan_sub decimal(8,2), deduction decimal(8,2),"
            + "bonus decimal(8,2))";
    st.execute( string: sql);
}
```

primary key condition

decimal with 8 whole number digits, and 2 decimal places

Specific data types, primary key restrictions, and default values are implemented.



All columns with their respective data types, restrictions, and default values are created, shown in DBeaver.

(W3Schools, 2024)

Ingenuity: The SQL create statements use SQL's default data type restrictions to optimize the memory use of the table. It automatically throws errors if the program tries to insert invalid values, making it much easier to debug and pinpoint errors in SQL statements.

Appropriateness: Using these data type restrictions and table schema allows the user to access data much more quickly and easily, as SQL queries run much more quickly. Additionally, it specifically fits the user's needs.

5. Adding, Updating, and deleting records in a database table

a. Adding a record

(Employee Records page)

```
boolean hired = false;
String firstname = "";
String middlename = "";
String lastname = "";
Date d = new Date();
String age = "";
String address = "";
String phonenumber = "";
String email = "";
String height = "";
String weight = "";
String notes = "";

//get values from the GUI buttons
hired = cbHired.isSelected();
firstname = txtFirstName.getText();
middlename = txtMiddleName.getText();
lastname = txtLastName.getText();
d = jCalendarBirthdate.getDate();
age = txtAge.getText();
address = txtAddress.getText();
phonenumber = txtPhoneNumber.getText();
email = txtEmail.getText();
height = txtHeight.getText();
weight = txtWeight.getText();
notes = txtNotes.getText();
```

Getting the values from the GUI buttons (error checking is included, but not shown here)

```
//Connect to the database
String databaseURL = "jdbc:derby:SALARY_GENERATOR";
Connection con = DriverManager.getConnection( url: databaseURL);
Statement st = con.createStatement();
Statement st2 = con.createStatement();

Date addeddate = new java.sql.Date( date: d.getTime());

//Insert SQL statement to insert a new row into the info system table
String sql = "INSERT INTO info (eid, userid, hired, first_name, middle_name, last_name, age, "
        + "birthdate, address, phonenumber, email, height, weight, notes) "
+ "VALUES (" + strmax + ", NULL, " + hired + ", '" + firstname + "', '" + middlename + "', '" + lastname + "', "
+ age + ", '" + new java.sql.Timestamp( time: d.getTime()) + "', '" + address + "', '" + phonenumber + "', '" + email
+ "', " + height + ", " + weight + ", '" + notes + "')";

maxfinder2();
this.salarymax++;

maxfinder3();
this.historymax++;
//Must insert a salary record into the salary allocations table as the two are linked.
//pay_period determined by the max from salary and loan history
String sql2 = "INSERT INTO salary_allocations (sid, eid, pay_period, monthly_wage_rate, sss_ee, sss_er, ph_ee_p, "
        + "ph_er_p, hdmf_ee_p, hdmf_er_p, loan, loan_add, loan_sub, deduction, bonus) "
+ "VALUES (" + this.salarymax + ", " + strmax + ", " + this.historymax + ", " + 0 + ", " + 0 + ", "
+ 0 + ", " + 0 + ", " + 0 + ", " + 0 + ", " + 0 //insert default values
+ ", " + 0 + ", " + 0 + ", " + 0 + "," + 0 +", "+ 0 +")";
st.execute( string: sql);
st2.execute( string: sql2);

con.close();
```

Establishes a connection to the database

INSERT statement to add a new row into the info table with the values entered by the user

INSERT statement to add a new row into salary allocations with automatically generated default values

Execute the SQL code statements

b. Updating a record

(Employee Records Page)

```
//Intitialize variables
String eid = "";
boolean hired = false;
String firstname = "";
String middlename = "";
String lastname = "";
Date d = new Date();
String age = "";
String address = "";
String phonenumber = "";
String email = "";
String height = "";
String weight = "";
String notes = "";

//Get values from textboxes and user-inputted data
eid = txtEID.getText();
hired = cbHired.isSelected();
firstname = txtFirstName.getText();
middlename = txtMiddleName.getText();
lastname = txtLastName.getText();
d = jCalendarBirthdate.getDate();
age = txtAge.getText();
address = txtAddress.getText();
phonenumber = txtPhoneNumber.getText();
email = txtEmail.getText();
height = txtHeight.getText();
weight = txtWeight.getText();
notes = txtNotes.getText();
```

Getting the values from the GUI buttons (error checking is included, but not shown here)

```
//Establish a connection with the database
String databaseURL = "jdbc:derby:SALARY_GENERATOR";
Connection con = DriverManager.getConnection( url:databaseURL);
Statement st = con.createStatement();

Date addeddate = new java.sql.Date( date:d.getTime());

//Update to the table using SQL statements, inserting error-trapped user-inputted data
String sql = "UPDATE info SET eid="+eid+", userid=NULL, hired="+hired+", first_name='"+firstname+"', middle_name='"+middlename+"',"
+ "last_name='"+lastname+"',age = "+age+", birthdate='"+ new java.sql.Timestamp( time:d.getTime())+"', address='" + address + "', "
+ "phonenumber='"+phonenumber+"', email = '"+email+"', height = "+height+", weight = "+weight+", notes='"+notes+"' "
+ "WHERE eid="+eid;
//Execute the query
st.execute( string:sql);
//Close the connection
con.close();
```

UPDATE statement to edit the selected row's info with user-inputted values

(Salary Generator page)

```
//Initializing Variables
String sid = "";
String eid = "";
String pay_period = "";
String monthly_wage_rate = "";
String sssee = "";
String ssser = "";
String phee = "";
String pher = "";
String hdmfee = "";
String hdmfer = "";
String loan = "";
String loanadd = "";
String loansub = "";
String deduction = "";
String bonus = "";
String finalwage = "";

//Get variables from user-input
eid = txtEID.getText();
monthly_wage_rate = txtWage.getText();
sssee = txtSSSEE.getText(); //SSS EE and ER must be validated on their respective pages
ssser = txtSSSER.getText();
phee = txtPHEE.getText();
pher = txtPHER.getText();
hdmfee = txtHDMFEE.getText();
hdmfer = txtHDMFER.getText();
loan = txtLoan.getText();
loanadd = txtLoanAdd.getText();
loansub = txtLoanSub.getText();
deduction = txtDeduction.getText();
bonus = txtBonus.getText();
finalwage = txtFinalWage.getText();
```

Getting the values from the GUI buttons (error checking is included, but not shown here)

```
//Establish connection with the database
String databaseURL = "jdbc:derby:SALARY_GENERATOR";
Connection con = DriverManager.getConnection( url: databaseURL);
Statement st = con.createStatement();

//Prepare the SQL statement
String sql = "UPDATE salary_allocations SET " +
     "monthly_wage_rate = " + monthly_wage_rate + ", " +
     "sss_ee = " + sssee + ", " +
     "sss_er = " + ssser + ", " +
     "ph_ee_p = " + phee + ", " +
     "ph_er_p = " + pher + ", " +
     "hdmf_ee_p = " + hdmfee + ", " +
     "hdmf_er_p = " + hdmfer + ", " +
     "loan = " + loan + ", " +
     "loan_add = " + loanadd + ", " +
     "loan_sub = " + loansub + ", " +
     "deduction = " + deduction + ", " +
     "bonus = " + bonus + " " +
       "= " + bonus + ", " +
     "WHERE eid = " + eid;
st.execute( string: sql);
con.close();

System.out.println( x: "Update successful.");
```

UPDATE statement to add a new row into info with user-inputted values

c. Deleting record(s)

(Employee Records Page)

```
DefaultTableModel model = (DefaultTableModel) tblEmployees.getModel();
int rowcount = model.getRowCount();

//Allows it to work with the filtered jtable (RowFilter)
int rows[] = tblEmployees.getSelectedRows();
for (int i = 0; i<rows.length; i++){
    rows[i] = tblEmployees.getRowSorter().convertRowIndexToModel(rows[i]);
}
```

This command “corrects” the row selected in the rowfilter by converting it to the actual row.

```
if (rows.length>0){//Delete multiple or single
    int eids[] = new int[rows.length];
    for (int i = 0; i<rows.length; i++){
        eids[i] = Integer.parseInt( s: model.getValueAt(rows[i], column: 0).toString());
    }

    //Establish connection to the database
    String databaseURL = "jdbc:derby:SALARY_GENERATOR";
    Connection con = DriverManager.getConnection( url: databaseURL);

    //iterate through the eids generated and delete them one by one
    String sql;
    for (int i = 0; i<eids.length; i++){
        Statement st = con.createStatement();

        sql = "DELETE FROM info WHERE eid=" + eids[i];
        st.execute( string: sql);
    }

    //Iterate through the employee's salaries in salary_allocations and delete them too
    for (int i = 0; i<eids.length; i++){
        Statement st = con.createStatement();

        sql = "DELETE FROM salary_allocations WHERE eid=" + eids[i];
        st.execute( string: sql);
    }
    con.close();
```

Placing the eids of the selected rows into an array to be looped through and deleted.

The DELETE statement only removes the specific row that matches the eid of the row selected by the user.

(HDMF page)

```
DefaultTableModel model = (DefaultTableModel) tblEmployees.getModel();
int rowcount = model.getRowCount();

//Allows it to work with the filtered jtable (RowFilter)
int rows[] = tblEmployees.getSelectedRows();
for (int i = 0; i<rows.length; i++){
    rows[i] = tblEmployees.getRowSorter().convertRowIndexToModel(rows[i]);
}

if (rows.length>0){//Delete multiple or single
    int eids[] = new int[rows.length];
    for (int i = 0; i<rows.length; i++){
        eids[i] = Integer.parseInt( s:model.getValueAt(rows[i], column: 0).toString());
    }

    //Establish connection to database
    String databaseURL = "jdbc:derby:SALARY_GENERATOR";
    Connection con = DriverManager.getConnection( url:databaseURL);

    //Delete statement to remove the specific row that lines up with the selected eid
    String sql;
    for (int i = 0; i<eids.length; i++){
        Statement st = con.createStatement();

        sql = "DELETE FROM hdmf WHERE bid=" + eids[i];
        st.execute( string: sql);
    }
```

This command “corrects” the row selected in the rowfilter by converting it to the actual row.

Placing the eids of the selected rows into an array to be looped through and deleted.

The DELETE statement only removes the specific row that matches the eid of the row selected by the user.

Ingenuity: SQL functionality makes it easy to add, update, or delete records. The queries are concise, requiring very few changes between them, making it easier to program and reuse code. It’s very time-efficient, and scales well as the database grows in size.

Appropriateness: These SQL statements allow the user to add, update, or delete records from the database by using SQL’s built-in functionality. The responsive interface in conjunction with the GUI abstract away complexity, decreasing the learning curve.

6. SELECT statements with sort methods and JOIN statements

(Salary Generator page)

```
refreshtable("SELECT sa.*, " +
"i.last_name || ', ' || i.first_name || ' ' || COALESCE(i.middle_name, '') AS full_name\n" + //concatenate name columns from emp records
"FROM salary_allocations sa\n" +
"JOIN info i ON sa.eid = i.eid\n" + //inner join statement to access data from salary_allocations and info
"WHERE i.hired = TRUE"); //making sure this is only done for hired employees.
```

SQL SELECT statement with an inner join to collect data values from both salary_allocations and info

```
private void refreshtable(String sql){//reset table contents, works for SELECT queries only
    try{
        //Creating a defaulttablemodel for tblEmployees
        DefaultTableModel model = (DefaultTableModel) tblEmployees.getModel();

        //Connect to SALARY_GENERATOR database
        String databaseURL = "jdbc:derby:SALARY_GENERATOR";
        Connection con = DriverManager.getConnection( url: databaseURL);
        Statement st = con.createStatement();

        //Execute the SQL query
        ResultSet rs = st.executeQuery( string: sql);

        model.setRowCount( rowCount: 0); //reset the table
        int i = 0;

        Double total_wages = 0.0;
        Double total_er = 0.0;
        //Iterate through resultset like it's a collection
        while (rs.next()==true){
            i++;
            String eid = rs.getString( i: 2); // EID
            String wage = rs.getString( i: 4); // wage
            String sss_ee = rs.getString( i: 5); // sss_ee
            String sss_er = rs.getString( i: 6); // sss_er
            String ph_ee_p = rs.getString( i: 7); // ph_ee_p
            String ph_er_p = rs.getString( i: 8); // ph_er_p
            String hdmf_ee_p = rs.getString( i: 9); // hdmf_ee_p
            String hdmf_er_p = rs.getString( i: 10); // hdmf_er_p
            String loan = rs.getString( i: 11); // loan
            String loan_add = rs.getString( i: 12); // loan_add
            String loan_sub = rs.getString( i: 13); // loan_sub
            String deduction = rs.getString( i: 14); // deduction
            String bonus = rs.getString( i: 15); // bonus
            String fullname = rs.getString( i: 16); //fullname

            Double dwage = Double.parseDouble( s: wage);
            Double dsss_ee = Double.parseDouble( s: sss_ee);
            Double dsss_er = Double.parseDouble( s: sss_er);
            Double dph_ee_p = Double.parseDouble( s: ph_ee_p);
            Double dph_er_p = Double.parseDouble( s: ph_er_p);
            Double dhdmf_ee_p = Double.parseDouble( s: hdmf_ee_p);
            Double dhdmf_er_p = Double.parseDouble( s: hdmf_er_p);
            Double dloan = Double.parseDouble( s: loan);
            Double dloan_add = Double.parseDouble( s: loan_add);
            Double dloan_sub = Double.parseDouble( s: loan_sub);
            Double ddeduction = Double.parseDouble( s: deduction);
            Double dbonus = Double.parseDouble( s: bonus);

            //Calculate dynamic values
            Double final_wage = dwage - dsss_ee - (dph_ee_p/100*dwage) - (dhdmf_ee_p/100*dwage) - dloan_sub - ddeduction + dbonus;
            Double er = dsss_er + (dph_er_p/100*dwage) + (dhdmf_er_p/100*dwage);
            total_wages+=final_wage;
            total_er+=er;

            //Add values to the jTable
            model.addRow(new Object[]{eid, fullname, wage, sss_ee, sss_er,
                ph_ee_p, ph_er_p, hdmf_ee_p, hdmf_er_p, loan, loan_add, loan_sub, deduction, bonus, final_wage});
        }

        //Update the appropriate text boxes with calculated dynamic values
        String strTotalWages = total_wages.toString();
        txtTotalWages.setText( s: strTotalWages);

        String strTotalER = total_er.toString();
        txtTotalER.setText( s: strTotalER);

        con.close();
```

Preparing and retrieving the values from the ResultSet

Applying the wage equations and aggregation calculations

Adding it to the jTable

(Employee Records Page)

```
String search = txtSearch.getText().trim();

int column = cmbColumn.getSelectedIndex();

//Set of column names to know which one to search in the database
String[] columnnames = {"eid", "hired", "first_name", "middle_name", "last_name",
    "birthdate", "age", "address", "phonenumber", "email", "height","weight", "notes"};

String strcolumn = columnnames[column];

//switch case to determine the order in which to sort the table
int sort = cmbOrder.getSelectedIndex();
String strsort = "";
switch (sort){
    case 0:
    strsort = "ASC";
    break;
    case 1:
    strsort = "DESC";
}

if (column==0 && sort==0){
    //The table is naturally sorted by eid, so a normal select query will do.
    //EmployeeRecords has its own implementation of refreshtable
    refreshtable( sql: "SELECT * FROM info");
}
else{
    //Send the select statement to refreshtable for processing
    refreshtable("select * from info order by "+strcolumn+" "+strsort);
}
```

refreshtable handles the SQL SELECT statements for the user, allowing it to process sort queries too.

```
private void refreshtable(String sql){//reset table contents, works for SELECT queries only
    try{
        DefaultTableModel model = (DefaultTableModel) tblEmployees.getModel();
        //Establish connection to the database
        String databaseURL = "jdbc:derby:SALARY_GENERATOR";
        Connection con = DriverManager.getConnection( url: databaseURL);

        Statement st = con.createStatement();

        //Place the results into a ResultSet
        ResultSet rs = st.executeQuery( string: sql);

        model.setRowCount( rowCount: 0); //reset the table
        while (rs.next()==true){
            //Add the data into the jTable
            model.addRow(new Object[]{rs.getString( i: 1),rs.getBoolean( i: 3),rs.getString( i: 4),rs.getString( i: 5),rs.getString( i: 6),
                rs.getString( i: 8),rs.getString( i: 7),rs.getString( i: 9),rs.getString( i: 10),rs.getString( i: 11),
                rs.getString( i: 12),rs.getString( i: 13),rs.getString( i: 14)});
        }
        con.close();

    }
    catch (Exception e){
        System.out.println( x: "Refresh table exception");
        e.printStackTrace();
    }
}
```

Establishing a connection to the database.

Adding the resultset values directly into the jTable.

(w3schools, 2019)

Ingenuity: The SELECT statement's versatile functionality allows the system to query the database quickly and easily. The programmer can reuse code and decrease the system's complexity.

Appropriateness: The user can sort and select data extremely easily. They can interact with the database without needing to learn SQL. Although this decreases some customizability, it's a reasonable trade-off for ease of use.

7. Dynamic polymorphism through constructor overloading
    a. Email checking code

```
String email = txtEmail.getText().trim().toUpperCase();
//Initializing variables
int atcount = 0;
int atpos = 0;
int firstdotpos = -1;
int dotpos = 0;
int finaldotpos = 0;
boolean invalidchars = false;

//Figuring out the positions of important characters
for (int i = 0; i<email.length(); i++){
    char character = email.charAt( index: i);
    //find position of the first at symbol
    if (character=='@'){
        atcount++; //count the nubmer of @ symbols
        atpos = i;
    }
    if (character=='.' && atpos!=0 && dotpos==0){
        dotpos = i; //the first . after the email
    }
    if (character=='.' && atpos==0 && firstdotpos==-1){
        firstdotpos = i; //the first . before the email
    }
    if (character=='.' && atpos!=0 && dotpos!=0){
        finaldotpos = i; //the final . symbol
    }
    int ascii = (int)character;
    if (ascii<=45 || ascii==47 || (ascii>=58 && ascii<=63) || ascii>=91){
        invalidchars = true; //check that all the ascii characters are valid
    }
}

//Email Error Trapping
if (email.length()==0){
    JOptionPane.showMessageDialog( parentComponent: null, message: "Email Field Empty. Please enter an email.");
}
else if (firstdotpos==0){
    JOptionPane.showMessageDialog( parentComponent: null, message: "Invalid Email. The '.' cannot be at the first position.");
}
else if (finaldotpos == email.length()-1){
    JOptionPane.showMessageDialog( parentComponent: null, message: "Invalid Email. A '.' cannot be at the final position.");
}
else if (atcount==0){
    JOptionPane.showMessageDialog( parentComponent: null, message: "Invalid Email. The email is missing an '@' symbol.");
}
else if (atcount>=2){
    JOptionPane.showMessageDialog( parentComponent: null, message: "Invalid Email. The email has too many '@' symbols.");
}
else if (atpos==0){
    JOptionPane.showMessageDialog( parentComponent: null, message: "Invalid Email. The email is lacking a prefix before the '@' symbol.");
}
else if (dotpos - atpos <=1){
    JOptionPane.showMessageDialog( parentComponent: null, message: "Invalid Email. The email is lacking a valid domain name.");
}
else if (invalidchars==true){
    JOptionPane.showMessageDialog( parentComponent: null, message: "Invalid Email. The email contains invalid characters.");
}
else if (email.length()>100){
    JOptionPane.showMessageDialog( parentComponent: null, message: "Invalid Email. The email is longer than 100 characters.");
}
```

For loop to find the important characters within the inputted email string, such as the first dot position, the @ symbol, and to detect invalid characters

Error trapping to tell the user exactly what's wrong with the inputted email.

b. Try and catch statements

```
public EmployeeRecords(String uid, String email){ //CATCH ALL INCOMING DATA
    initComponents();
    try{ //attempt to execute the code
        this.email = email;
        String currentURL = "jdbc:derby:SALARY_GENERATOR;create=true";
        Connection con = DriverManager.getConnection( url: currentURL);

        Statement st = con.createStatement();

        uid = "salary_allocation"; //table names can't start with a number
        this.uid = uid;

        con.close();

        maxfinder();

        refreshtable( sql: "select * from info");

        initComponents2();
    }
    catch (Exception e){ //catch any Exception, regardless of type
        e.printStackTrace(); //print the exact error
    }
}
```

Every function is run within a try-catch statement to make sure the application doesn't crash during runtime.

A stack trace function is called when an error is found

c. Inheritance through NumericDocumentFilter

```
import javax.swing.text.AttributeSet;
import javax.swing.text.BadLocationException;
import javax.swing.text.DocumentFilter;

/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */

/**
 * A DocumentFilter that allows only numeric input with an optional decimal point.
 */
public class NumericDocumentFilter extends DocumentFilter {

    // Inserts text only if it results in valid numeric input.
    @Override
    public void insertString(DocumentFilter.FilterBypass fb, int offset, String string, AttributeSet attr) throws BadLocationException {
        if (isValidInput( currentText: fb.getDocument().getText( offset: 0,  length: fb.getDocument().getLength()),  newText: string)) {
            super.insertString(fb, offset, string, attr);
        }
    }

    // Replaces text only if the new input remains numeric.
    @Override
    public void replace(DocumentFilter.FilterBypass fb, int offset, int length, String text, AttributeSet attrs) throws BadLocationException {
        if (isValidInput( currentText: fb.getDocument().getText( offset: 0,  length: fb.getDocument().getLength()),  newText: text)) {
            super.replace(fb, offset, length, text, attrs);
        }
    }

    // Allows removal of text without restrictions.
    @Override
    public void remove(DocumentFilter.FilterBypass fb, int offset, int length) throws BadLocationException {
        super.remove(fb, offset, length);
    }

    // Validates if the new input maintains a proper numeric format.
    private boolean isValidInput(String currentText, String newText) {
        String resultingText = currentText + newText;
        return resultingText.isEmpty() || resultingText.matches( regex: "\\d*\\.?\\d*"); // Allows digits and one decimal point
    }
}
```

NumericDocumentFilter inherits the DocumentFilter class

Text is only input into the textbox if it follows the regular expression

The following text boxes have the NumericDocumentFilter applied as their DocumentFilter.

```
//Numeric text fields
((AbstractDocument) txtEID.getDocument()).setDocumentFilter(new NumericDocumentFilter());
((AbstractDocument) txtWage.getDocument()).setDocumentFilter(new NumericDocumentFilter());
((AbstractDocument) txtSSSEE.getDocument()).setDocumentFilter(new NumericDocumentFilter());
((AbstractDocument) txtSSSER.getDocument()).setDocumentFilter(new NumericDocumentFilter());
((AbstractDocument) txtPHEE.getDocument()).setDocumentFilter(new NumericDocumentFilter());
((AbstractDocument) txtPHER.getDocument()).setDocumentFilter(new NumericDocumentFilter());
((AbstractDocument) txtHDMFEE.getDocument()).setDocumentFilter(new NumericDocumentFilter());
((AbstractDocument) txtHDMFER.getDocument()).setDocumentFilter(new NumericDocumentFilter());
((AbstractDocument) txtLoan.getDocument()).setDocumentFilter(new NumericDocumentFilter());
((AbstractDocument) txtLoanSub.getDocument()).setDocumentFilter(new NumericDocumentFilter());
((AbstractDocument) txtLoanAdd.getDocument()).setDocumentFilter(new NumericDocumentFilter());
((AbstractDocument) txtBonus.getDocument()).setDocumentFilter(new NumericDocumentFilter());
((AbstractDocument) txtDeduction.getDocument()).setDocumentFilter(new NumericDocumentFilter());
```

Ingenuity: Error trapping ensures that data entered by the user is validated and within proper ranges, allowing it to work with the database systems, Java's data types, and more. The try-catch statements allow the programmer to understand and address programming bugs.

Appropriateness: The error trapping provides a smooth user experience, as the user is informed of the errors they've made while using the application. It also ensures that the system doesn't crash during runtime.

8. Constructor overloading

```
public HDMF() {
    initComponents();

    try{

        maxfinder();

        refreshtable( sql: "SELECT * FROM hdmf");

        initComponents2();


    }
    catch (Exception e){
        e.printStackTrace();
    }
}

public HDMF(String uid, String email){ //CATCH ALL INCOMING DATA
    initComponents();
    try{

        maxfinder();

        refreshtable( sql: "SELECT * FROM hdmf");

        initComponents2();


    }
    catch (Exception e){
        e.printStackTrace();
    }
}
```

Same method name, different parameter list.
This is dynamic polymorphism

This allows it to receive the user id and email strings from any other page that calls it.


```
public PhilHealth() {
    initComponents();

    try{


        maxfinder();

        refreshtable( sql: "SELECT * FROM philhealth");

        initComponents2();


    }
    catch (Exception e){
        e.printStackTrace();
    }
}

public PhilHealth(String uid, String email){ //CATCH ALL INCOMING DATA
    initComponents();
    try{


        maxfinder();

        refreshtable( sql: "SELECT * FROM philhealth");

        initComponents2();


    }
    catch (Exception e){
        e.printStackTrace();
    }
}
```

Ingenuity: This allows important information such as email and user id to be passed between different pages. Additionally, the default constructor (no parameters) is a backup for the custom constructor (two parameters), allowing the system to run without these if necessary.

Appropriateness: There are only minor differences between the two constructors. Thus, if an error is raised during runtime, the page still generates correctly, ensuring a smooth user experience.

9. Automatic calculation of values:
    a. Final Wage and Final ER

```
private void refreshtable(String sql){//reset table contents, works for SELECT queries only
    try{
        //Creating a defaulttablemodel for tblEmployees
        DefaultTableModel model = (DefaultTableModel) tblEmployees.getModel();

        //Connect to SALARY_GENERATOR database
        String databaseURL = "jdbc:derby:SALARY_GENERATOR";
        Connection con = DriverManager.getConnection( url: databaseURL);
        Statement st = con.createStatement();

        //Execute the SQL query
        ResultSet rs = st.executeQuery( string: sql);

        model.setRowCount( rowCount: 0); //reset the table
        int i = 0;

        Double total_wages = 0.0;
        Double total_er = 0.0;
        //Iterate through resultset like it's a collection
        while (rs.next()==true){
            i++;
            String eid = rs.getString( i: 2); // EID
            String wage = rs.getString( i: 4); // wage
            String sss_ee = rs.getString( i: 5); // sss_ee
            String sss_er = rs.getString( i: 6); // sss_er
            String ph_ee_p = rs.getString( i: 7); // ph_ee_p
            String ph_er_p = rs.getString( i: 8); // ph_er_p
            String hdmf_ee_p = rs.getString( i: 9); // hdmf_ee_p
            String hdmf_er_p = rs.getString( i: 10); // hdmf_er_p
            String loan = rs.getString( i: 11); // loan
            String loan_add = rs.getString( i: 12); // loan_add
            String loan_sub = rs.getString( i: 13); // loan_sub
            String deduction = rs.getString( i: 14); // deduction
            String bonus = rs.getString( i: 15); // bonus
            String fullname = rs.getString( i: 16); //fullname

            Double dwage = Double.parseDouble( s: wage);
            Double dsss_ee = Double.parseDouble( s: sss_ee);
            Double dsss_er = Double.parseDouble( s: sss_er);
            Double dph_ee_p = Double.parseDouble( s: ph_ee_p);
            Double dph_er_p = Double.parseDouble( s: ph_er_p);
            Double dhdmf_ee_p = Double.parseDouble( s: hdmf_ee_p);
            Double dhdmf_er_p = Double.parseDouble( s: hdmf_er_p);
            Double dloan = Double.parseDouble( s: loan);
            Double dloan_add = Double.parseDouble( s: loan_add);
            Double dloan_sub = Double.parseDouble( s: loan_sub);
            Double ddeduction = Double.parseDouble( s: deduction);
            Double dbonus = Double.parseDouble( s: bonus);

            //Calculate dynamic values
            Double final_wage = dwage - dsss_ee - (dph_ee_p/100*dwage) - (dhdmf_ee_p/100*dwage) - dloan_sub - ddeduction + dbonus;
            Double er = dsss_er + (dph_er_p/100*dwage) + (dhdmf_er_p/100*dwage);
            total_wages+=final_wage;
            total_er+=er;

            //Add values to the jTable
            model.addRow(new Object[]{eid, fullname, wage, sss_ee, sss_er,
                ph_ee_p, ph_er_p, hdmf_ee_p, hdmf_er_p, loan, loan_add, loan_sub, deduction, bonus, final_wage});
        }

        //Update the appropriate text boxes with calculated dynamic values
        String strTotalWages = total_wages.toString();
        txtTotalWages.setText( t: strTotalWages);

        String strTotalER = total_er.toString();
        txtTotalER.setText( t: strTotalER);

        con.close();
```

Automatic calculation of final_wage for each employee, total_wages, and total_er

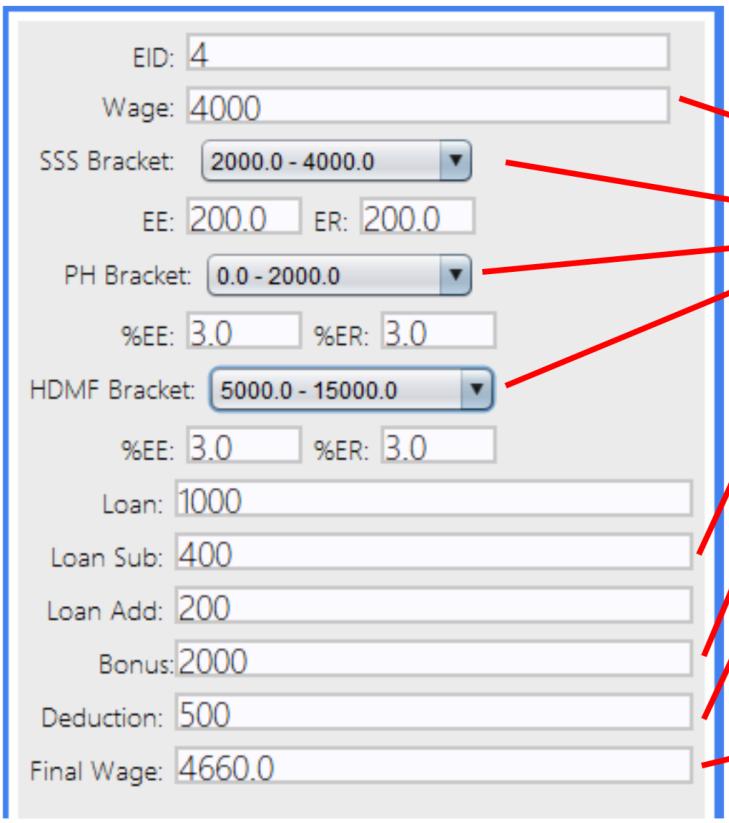
```
private void txtWageKeyReleased(java.awt.event.KeyEvent evt) {
    calcFinalWage();
}
private void txtLoanSubKeyReleased(java.awt.event.KeyEvent evt) {
    calcFinalWage();
}
```

```
private void txtBonusKeyReleased(java.awt.event.KeyEvent evt) {
    calcFinalWage();
}
private void txtDeductionKeyReleased(java.awt.event.KeyEvent evt) {
    calcFinalWage();
}
```

The function is only called when the user stops typing.

```
private void calcFinalWage(){
    //Try-catch makes it so that if there's any mistake, the final wage doesn't update its textfield
    try{
        //retrieves necessary values from the textboxes
        double wage = Double.parseDouble(s:txtWage.getText());
        double sssee = Double.parseDouble(s:txtSSSEE.getText());
        double phee = Double.parseDouble(s:txtPHEE.getText());
        double hdmfee = Double.parseDouble(s:txtHDMFEE.getText());
        double sub = Double.parseDouble(s:txtLoanSub.getText());
        double deduction = Double.parseDouble(s:txtDeduction.getText());
        double bonus = Double.parseDouble(s:txtBonus.getText());

        //applying the calculation dynamically
        double n = wage - sssee - (phee*wage/100) - (hdmfee*wage/100) - sub - deduction + bonus;
        String strn = Double.toString(d:n);

        //Updating the textbox
        txtFinalWage.setText(t:strn);
    }
    catch(Exception e){
        e.printStackTrace();
    }
}
```

The final_wage textbox is also automatically updated any time the above textfields or comboboxes change



When any of these values are changed, the final wage is updated.

Final wage is recalculated every time.

b. Calculation of age in years (Date and LocalDate library)

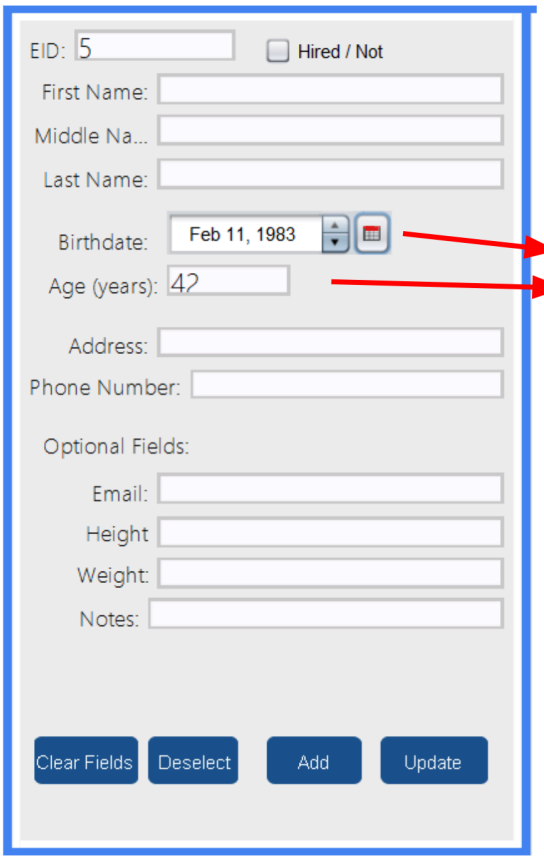
```
import java.util.Date;
    private de.wannawork.jcalendar.JCalendarComboBox jCalendarBirthdate;

private void jCalendarBirthdateStateChanged(javax.swing.event.ChangeEvent evt) {
    try{
        Date startDate = new Date(); //the current date
        Date endDate = jCalendarBirthdate.getDate(); //The date inputted by the user.

        // Calculate the difference in milliseconds
        long differenceInMillis = startDate.getTime() - endDate.getTime();

        // Calculate the number of years (approximate)
        long years = differenceInMillis / (1000L * 60 * 60 * 24 * 365);

        if (years<16){
            JOptionPane.showMessageDialog( parentComponent: null,  message: "Age must be at least 16.");
            txtAge.setText( t: "");
        }
        else{
            txtAge.setText( t: Long.toString( i: years));
        }
    }
    catch (Exception e){

    }
}
```

This uses the Date and jCalendar libraries' built-in functions.

The system automatically calculates the employee's age in years based on their birthdate and the current date.

Updates the value of the age textbox depending on the automatically calculated age

(GeeksforGeeks, 2016)



Selecting a new birthdate automatically updates the age textbox

c. Unique ID finder: generate a unique ID for the primary key

```
private void maxfinder(){ //figure out the highest eid
    try{
        //Connect to the database
        String databaseURL = "jdbc:derby:SALARY_GENERATOR";
        Connection con = DriverManager.getConnection( url: databaseURL);

        String sql;
        Statement st = con.createStatement();

        sql = "SELECT * FROM salary_allocations";
        ResultSet rs = st.executeQuery( string: sql);

        //db stands for database

        //Have a max variable begin at 0
        int max = 0;
        while (rs.next()==true){
            String dbeid = rs.getString( i: 1);
            int eid = Integer.parseInt( s: dbeid);
            if (eid>max){ //change max with the value in the ResultSet if the value is larger
               max = eid;
            }
        }
        //assign it to a global max variable to be used later.
        //Incrementing this.max by 1 guarantees a unique max value
        this.max = max;
        con.close();
    }
    catch (Exception e){

    }
}
```

The maxfinder algorithm guarantees finding the maximum value within the table's eid (Employee ID) column of the Salary Allocations table.

```
private int maxfinder2(){ //figure out the highest eid in salary history
    try{
        //establish connection to the database
        String databaseURL = "jdbc:derby:SALARY_GENERATOR";
        Connection con = DriverManager.getConnection( url: databaseURL);

        String sql;
        Statement st = con.createStatement();

        sql = "SELECT * FROM salary_history";
        ResultSet rs = st.executeQuery( string: sql);

        //db stands for database

        //Initialize a max variable
        int max = 0;
        while (rs.next()==true){
            String dbeid = rs.getString( i: 1);
            int eid = Integer.parseInt( s: dbeid);
            if (eid>max){ //compare the max variable to the shid in the resultset
               max = eid;
            }
        }

        con.close();

        //automatically returns the value of the highest record in salary history
        //incrementing this by 1 is a completely unique shid value.
        return max;
    }
    catch (Exception e){
        return -1;
    }

}
```

The maxfinder algorithm guarantees finding the maximum value within the salary history's shid (Salary History ID) column

Ingenuity: The max-finder algorithms can be used in other pages within the system. The generation of the unique ID is extremely easy, predictable, and works with the database systems.

Appropriateness: The user, who sees the unique ID, can interact with the system much more easily, as it's more human-readable than a random string or a hash. This increases ease of use and decreases the learning curve.

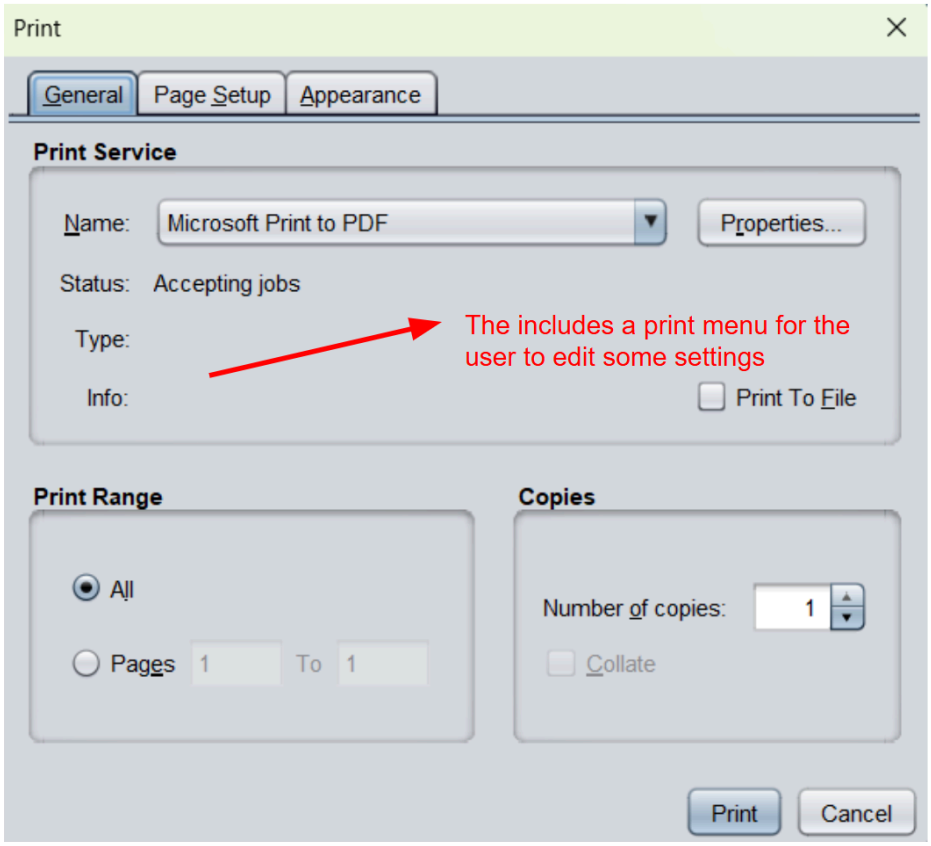
10. Print JTable contents to a PDF

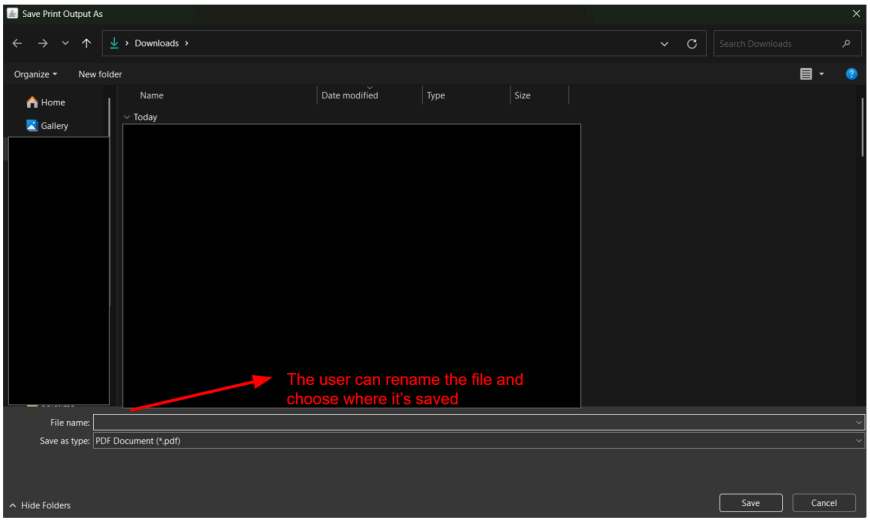
```
import javax.print.attribute.HashPrintRequestAttributeSet;
import javax.print.attribute.PrintRequestAttributeSet;
import javax.print.attribute.standard.OrientationRequested;
private void ktbtnPrintActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        // Create a print request attribute set to configure print settings
        PrintRequestAttributeSet set = new HashPrintRequestAttributeSet();

        // Get the current date and convert it to a string
        Date d = new Date();
        String sd = d.toString();

        // Create header and footer for the printout
        MessageFormat header = new MessageFormat( pattern: "Employee Salaries");
        MessageFormat footer = new MessageFormat( pattern: sd);

        // Set the print orientation to portrait mode
        set.add( attribute: OrientationRequested.PORTRAIT);

        // Print the JTable with the given header and footer
        tblEmployees.print( printMode: JTable.PrintMode.FIT_WIDTH, headerFormat: header,
                footerFormat: footer, showPrintDialog: true, attr: set, interactive: true);

        // Show a success message after printing
        JOptionPane.showMessageDialog( parentComponent: null, message: "\n Printed Successfully");
    }
    catch (java.awt.print.PrinterException e) {
        // Show an error message if printing fails
        JOptionPane.showMessageDialog( parentComponent: null, message: "\n Printing Failed");
    }
}
```

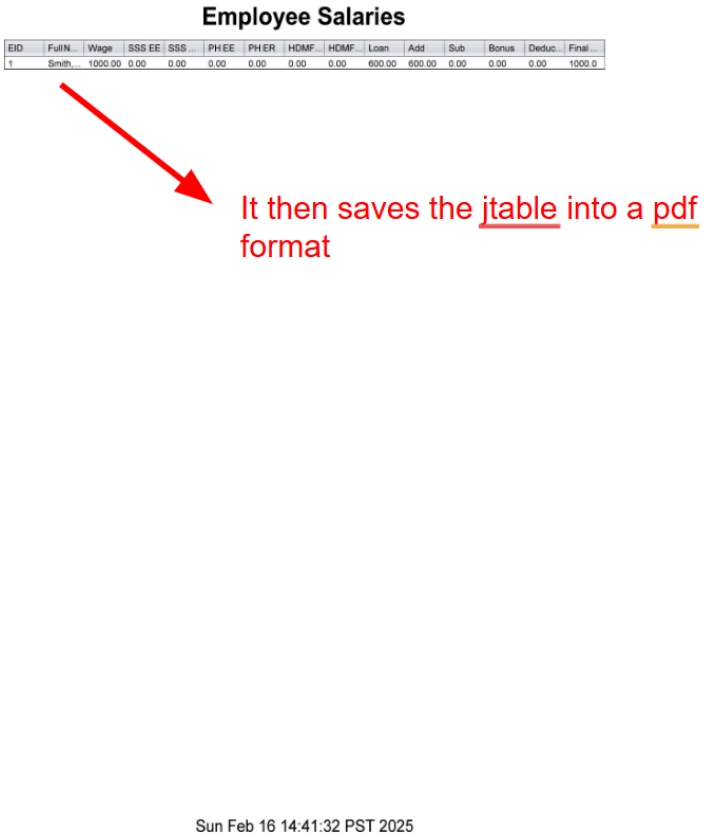
Changing specific settings (title, format, etc.) for the final document

Printing all the JTable's current contents

(GeeksforGeeks, 2020)


Print
General
Page Setup
Appearance
Print Service
Name:
Microsoft Print to PDF
Properties...
Status:
Accepting jobs
Type:
Info:
The includes a print menu for the user to edit some settings
Print To File
Print Range
All
Pages
1
To
1
Copies
Number of copies:
1
Collate
Print
Cancel


Save Print Output As
Downloads
Search Downloads
Organize
New folder
Home
Gallery
Name
Date modified
Type
Size
Today
The user can rename the file and choose where it's saved
File name:
Save as type:
PDF Document (*.pdf)
Hide Folders
Save
Cancel



Ingenuity: Java's built-in print functions save a lot of time for the programmer, as it's well-implemented and has been sufficiently tested beforehand. Additionally, code can be more versatile, increasing code reusability across pages.

Appropriateness: The interactive and clean UI allows the user to print the jTable's contents quickly and easily. They can specify where to save it on their device, how many copies to print, and edit the pdf's layout before printing.

11. Emailing the user to do the payroll if they haven't yet

(Salary Generator)

```
//Email sending code
String lastSentDateString = null;
try {
    // Read the last sent date from the file
    File file = new File( pathname: "filename.txt");
    if (file.exists()) {
        Scanner myReader = new Scanner( source: file);
        if (myReader.hasNextLine()) {
            lastSentDateString = myReader.nextLine();
        }
        myReader.close();
    }
} catch (FileNotFoundException e) {
    e.printStackTrace();
}

if (lastSentDateString == null) {
    // File doesn't contain any date, initialize with current date
    lastSentDateString = new SimpleDateFormat( pattern: "yyyy-MM-dd").format(new Date( date: 0));
    //if it's equal to null, write the current date to the file so an email doesn't get sent again
    String lastSentDateString2 = new SimpleDateFormat( pattern: "yyyy-MM-dd").format(new Date());
    writeDateToFile( filename: "filename.txt",  date: lastSentDateString2);
}
```

Checking a save file if an email has already been sent. Only one gets sent per day.

Sets the last sent date string as a default (Jan 1 1970) if an email has never been sent.

```
try {
    // Query the database for the most recent date salaries were saved
    ResultSet rs = st.executeQuery( string: "SELECT MAX(datesaved) FROM salary_history");
    String mostRecentSalaryDate = null;
    if (rs.next()) { // Move to the first (and only) row of the result
        mostRecentSalaryDate = rs.getString( i: 1);
    }

    SimpleDateFormat formatter = new SimpleDateFormat( pattern: "yyyy-MM-dd");
    Date lastSentDate = new Date();
    Date mostRecentDate = new Date();

    if (lastSentDateString == null){
        lastSentDateString = new SimpleDateFormat( pattern: "yyyy-MM-dd").format(new Date());
    }
    else{
        lastSentDate = formatter.parse( source: lastSentDateString);
    }

    if (mostRecentSalaryDate == null) {
        mostRecentSalaryDate = new SimpleDateFormat( pattern: "yyyy-MM-dd").format(new Date()); // Fallback to epoch date
    }
    else{
        mostRecentDate = formatter.parse( source: mostRecentSalaryDate);
    }

    // Parse dates for comparison
    // Check if salaries are up-to-date for this month
    Calendar cal = Calendar.getInstance();
    cal.setTime( date: lastSentDate);
    int currentDay = cal.get( field: Calendar.DAY_OF_MONTH);
    int currentMonth = cal.get( field: Calendar.MONTH) + 1; // Calendar months are 0-based
    int currentYear = cal.get( field: Calendar.YEAR);
```

Check when the most recent salaries were saved, based on the most recent date in Salary History

Extract the calendar, month, and day values of the lastsentdate.

```
// Check if mostRecentDate is in the current month and year
cal.setTime( date: mostRecentDate);
int salaryDay = cal.get( field: Calendar.DAY_OF_MONTH);
int salaryMonth = cal.get( field: Calendar.MONTH) + 1;
int salaryYear = cal.get( field: Calendar.YEAR);

// Determine if salaries have been done for the first and second halves of the month
boolean salaryDoneFirstHalf = salaryYear == currentYear && salaryMonth == currentMonth && salaryDay <= 15;
boolean salaryDoneSecondHalf = salaryYear == currentYear && salaryMonth == currentMonth && salaryDay > 15;

// Trigger the email if salaries for both halves of the current month are not done
if (!(salaryDoneFirstHalf && salaryDoneSecondHalf)) {
    // Ensure dates are compared without time
    SimpleDateFormat dateOnlyFormatter = new SimpleDateFormat( pattern: "yyyy-MM-dd");
    Date truncatedLastSentDate = dateOnlyFormatter.parse( source: dateOnlyFormatter.format( date: lastSentDate));
    Date truncatedCurrentDate = dateOnlyFormatter.parse( source: dateOnlyFormatter.format( date: mostRecentDate));

    // Check if the email has already been sent today
    boolean emailSentToday = truncatedLastSentDate.equals( obj: truncatedCurrentDate);

    if (!emailSentToday) {
        // Send reminder email

        email = email.toLowerCase();

        EmailSender send = new EmailSender( receiverEmail: email, "Wage Reminder " + truncatedCurrentDate.toString(),
                body: "Please do the wages for this month.");

        lastSentDateString = new SimpleDateFormat( pattern: "yyyy-MM-dd").format( date: truncatedCurrentDate);

        writeDateToFile( filename: "filename.txt",  date: lastSentDateString); //this doesn't work evidently
    }
}
```

Only triggers if salaries for both halves of the month aren't done.

Calls the custom EmailSender class to send the email.

(GeeksforGeeks, 2016)

(EmailSender.java)

```java
import java.util.Properties;
import javax.mail.*;
import javax.mail.internet.*;
import javax.net.ssl.SSLContext;
import javax.net.ssl.TrustManager;
import javax.net.ssl.X509TrustManager;
import java.security.cert.X509Certificate;

public class EmailSender {
    // Sender's email credentials
    private final String senderEmail = "ibcsiaemailsender@gmail.com";
    private final String senderPassword = "[redacted]";

    // Gmail SMTP server details
    private final String emailSMTPserver = "smtp.gmail.com";

    // Constructor for sending an email
    public EmailSender(String receiverEmail, String subject, String body) {
        try {
            // Trust all certificates (for testing purposes only)
            TrustManager[] trustAllCerts = new TrustManager[]{
                new X509TrustManager() {
                    public java.security.cert.X509Certificate[] getAcceptedIssuers() {
                        return null;
                    }
                    public void checkClientTrusted(X509Certificate[] certs, String authType) {}
                    public void checkServerTrusted(X509Certificate[] certs, String authType) {}
                }
            };
            SSLContext sc = SSLContext.getInstance( protocol: "TLS");
            sc.init( km: null,  tm: trustAllCerts, new java.security.SecureRandom());
            javax.net.ssl.HttpsURLConnection.setDefaultSSLSocketFactory( sf: sc.getSocketFactory());

            // Define SMTP server properties
            Properties props = new Properties();
            props.put( key: "mail.smtp.host",  value: emailSMTPserver);
            props.put( key: "mail.smtp.port",  value: "465");
            props.put( key: "mail.smtp.auth",  value: "true");
            props.put( key: "mail.smtp.socketFactory.port",  value: "465");
            props.put( key: "mail.smtp.socketFactory.class",  value: "javax.net.ssl.SSLSocketFactory");
            props.put( key: "mail.smtp.socketFactory.fallback",  value: "false");
            props.put( key: "mail.smtp.ssl.trust",  value: "smtp.gmail.com");

            // Create an Authenticator with the sender's credentials
            Authenticator auth = new Authenticator() {
                @Override
                protected PasswordAuthentication getPasswordAuthentication() {
                    return new PasswordAuthentication( userName: senderEmail,  password: senderPassword);
                }
            };

            // Create a new email session with the properties and authenticator
            Session session = Session.getInstance(props,  authenticator: auth);
            session.setDebug( debug: true); // Enable debug mode to see detailed logs

            // Create a new email message
            MimeMessage msg = new MimeMessage(session);
            msg.setFrom(new InternetAddress( address: senderEmail));
            msg.addRecipient( type: Message.RecipientType.TO, new InternetAddress( address: receiverEmail));
            msg.setSubject(subject);
            msg.setText( text: body);

            // Send the email
            Transport.send(msg);
        } catch (Exception e) {
            // Handle exceptions and print the stack trace
            e.printStackTrace();
        }
    }
```

Import libraries (javax.mail)

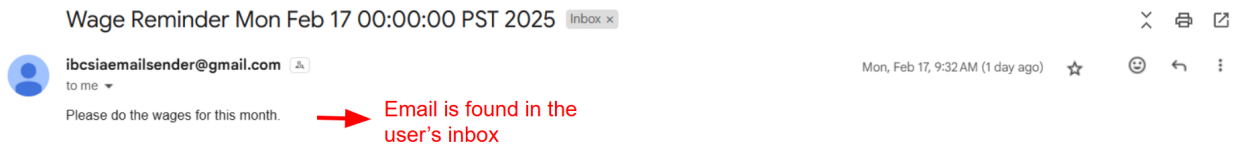
16-digit sender password obscured for security reasons

Define the Simple Mail Transfer Protocol properties

Create the email object

Editing the email's properties

Send the email



Ingenuity: The system uses pre-tested libraries to ensure smooth processing of the email feature, depending on the user's inputted email address. All email certificates and system requirements are handled.

Appropriateness: If the user forgets to do the payroll for the half-month, this is a helpful reminder. It sends a record of it outside the application, so that the user remembers to perform it, and has a record of all the times they've forgotten to do it.

Word Count: 1078

Works Cited:

Anjali Sajeevan (2021). *JAVA Swing-Single and Multiple Search - Anjali Sajeevan - Medium*. [online] Medium. Available at: https://anjalisajeev.medium.com/java-swing-netbeans-tips-and-tricks-search-single-and-multiple-criteria-part-2-ab21f526a265 [Accessed 27 Feb. 2025].

GeeksforGeeks (2016). *Date class in Java (With Examples)*. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/date-class-java-examples/ [Accessed 27 Feb. 2025].

GeeksforGeeks (2020). *MessageFormat format() method in Java with Example : Set 1*. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/messageformat-format-method-in-java-with-example-set-1/ [Accessed 27 Feb. 2025].

W3Schools (2024). *SQL PRIMARY KEY Constraint*. [online] W3Schools. Available at: https://www.w3schools.com/sql/sql_primarykey.ASP [Accessed 27 Feb. 2025].

W3schools (2019). *SQL INNER JOIN Keyword*. [online] W3schools.com. Available at: https://www.w3schools.com/sql/sql_join_inner.asp [Accessed 27 Feb. 2025].