Group Report

This document outlines the details of each group member and the part(s) of the project they contributed to. It outlines the Classes and a comprehensive outline of what makes up each class, detailing what the Author(s) had to establish. Please make note of all developments.

Percival Roberts 2202974

Dongie King 2203350

Jordayne Price 2202939

Joshuawn Johnson 2205652

Contents

OOA & OOD

Dongie King – Responsible for implementing the OOA & OOD

Coad and Yourdon’s six selection characteristics:

Selection Characteristics	Explanation of Characteristics
Retained Information	the potential object will be useful during analysis only if information about it must be remembered so that the system can function
Needed Services	the potential object must have a set of identifiable operations that can change the value of its attributes in some way
Multiple Attributes	during requirements analysis, the focus should be on “major” information (an object with a single attribute may, in fact, be useful during design, but is probably better represented as an attribute of another object during the analysis activity)
Common Attributes	a set of attributes can be defined for the potential object, and these attributes apply to all occurrences of the object
Common Operations	a set of operations can be defined for the potential object, and these operations apply to all occurrences of the object
Essential Requirements	external entities that appear in the problem space and produce or consume information that is essential to the operation of any solution for the system will almost always be defined as objects in the requirements model

Object Oriented Analysis:

Potential Class	Retained Information	Needed Services	Multiple Attributes	Common Attributes	Common Operations	Essential Requirements	Accepted As Class
Department	departmentCode, departmentName, regularRate; overtimeRate,	assignment operator , not equal	Department,addDepartmentRecord,viewDepartmentRecord,updateEmployeeRecord	departmentCode	assignment operator , not equal	FileManager,Employee,Main	Department
Employee	departmentCode,employeeID,firstName ,lastName ,employeeDepartmentCode,position,taxRegistrationNumber,nationalInsuranceScheme,dateOfBirth,dateOfHire,hrsWorked	equal, not equal	Employee,addEmployeeRecord,updateEmployeeRecord,viewemployeeRecord,deleteEmployeeRecord	isValidEmployeeCode.	equal, not equal	Department,File Manager, Main	Employee
FileManager	-	assignment operator	writeToFile,readFromoFile,updateRecord	readFromFile,writetoFile	assignment operator	Department,Employee,Main	FileManager

4. Payroll	regularPay, overtimePay;
	
	grossPay,dateOfProcessing,chequeNumber;
		equal , not equal to,greater than, multiplication, assignment operator	setRegularPay,setOvertimePay, calculateRegularPay,calculateOvertimePay,view DepartmentPayroll	calculateRegularPay.calculateOvertimePay,viewDepartmentPayroll,Department,addDepartmentRecord,view DepartmentRecord,updateEmployeeRecord	equal to	Employee	Payroll

5. Main		assignment operator,not equal	payrollMenu,employeemenu,departmentMenu,mainMenu	Employee,addEmployeeRecord ,updateEmployeeRecord,viewemployeeRecord ,deleteEmployeeRecord,	assignment operator	Department,Employee,Payroll	Main
6. GUI	mainMenuPanel ,departmentMenuPanel.employeeMenuPanel, payrollMenuPanel	equal to, arrow operator	createMainMenuPanel,createMenuButton,createDepartmentMenuPanel,createEmployeeMenuPanel, createPayrollMenuPanel	createMainMenuPanel,createMenuButton,create Department MenuPanel,createEmployeeMenuPanel,createPayrollMenuPanel	equal to	Main,Department,Employee, Payroll	GUI

The Object Oriented Analysis is an approach used by software developers to understand, define and map requirements a system will need to be fully functional. In this project five Potential Classes were identified, theses classes are:

- Department
- Employee
- FileManager
- Payroll
- Main
- GUI

The Department Class is needed to store information about the department, validate employee input and set employee rate, whether it is regular rate or overtime rates. Hence values such as regularRate,overRate,departmentName and departmentCode are needed to fulfill these requirements. The operations needed through this program are the ,not equal to sign (!=) and the assignment operator(=).The assignment operator is used to instantiate occurrences of object classes,arrays as well as boolean values.The data from the Department class will be used in the Employee class since a department is made up of employees(composition relationship),File Manager will also need to store information about each department(department name and department code) , so there will be a direct association between these two classes, and Main will provide the application menu for this class, to add, update and view data in this class.

The Employee Class is required to receive employee information such as, date of birth,first name, last name, NIS, TRN, employee id, position in the department, date of hire as well as hours worked. Operations such as the assignment operator will be needed to format fields that require date, by using

Java/C++ defined functions. The not equal to operator will be needed to validate conditions, before the program accept and stores user input , which will lower the debugging time.This employee class would form a composition relation between two other classes, the Payroll class and Department Class, because neither the Payroll Class or Department Class cannot exist without the Employee who makes up the department and has a payroll.

The Payroll Class is needed to calculate and store the regular pay as well as the overtime pay for employees and the date the pay is processed .
Numerous operations are required in this class because of the extensive arithmetic needed to calculate employee pay. Multiplication will be used to calculate regular pay by finding the product of the hourly rate and number of hours worked, as well as adding any overtime inclusive of a different overtime rate, to display the total pay. The Payroll class is reliant on the Employee class, because each record from payroll would be needed to be assigned to an employee. The Main class provides the men option to select and modify the Payroll Class, this class would also extend the Employee Class, because of the needed attributes from the parent class such as hours worked, which will be needed in the Payroll class.

FileManager, this class will store all the data that has been validated in the Payroll,Department and Employee Class. This class uses a common attribute with the mentioned classes that will need to serialize the various data types into a singular string , so it can be stored into a file as well as arrays, and evidently it will need to deserialize the stored data to display information to the user. In this class because of the extensive use of files, exception handling has to be done throughout, therefore the not equal sign to validate contents of the file before data is added,deleted or updated is required. This class will be associated with the Department class, because the Department Class will cover any other relationship with other classes since the employee in particular belongs to the department and the employee has a payroll.
The Main class is fundamental in giving accessibility features to the software, by displaying a menu option to select the needed class of amendment . This will display a menu option to select department ,employee and payroll submenu. Each sub-menu will give direct access to the class of the same of each menu option, that will allow, add, update and a view of stored data. This class will also allow a seamless flow of the algorithm as well as a gracious termination of the program rather than an abrupt finish.

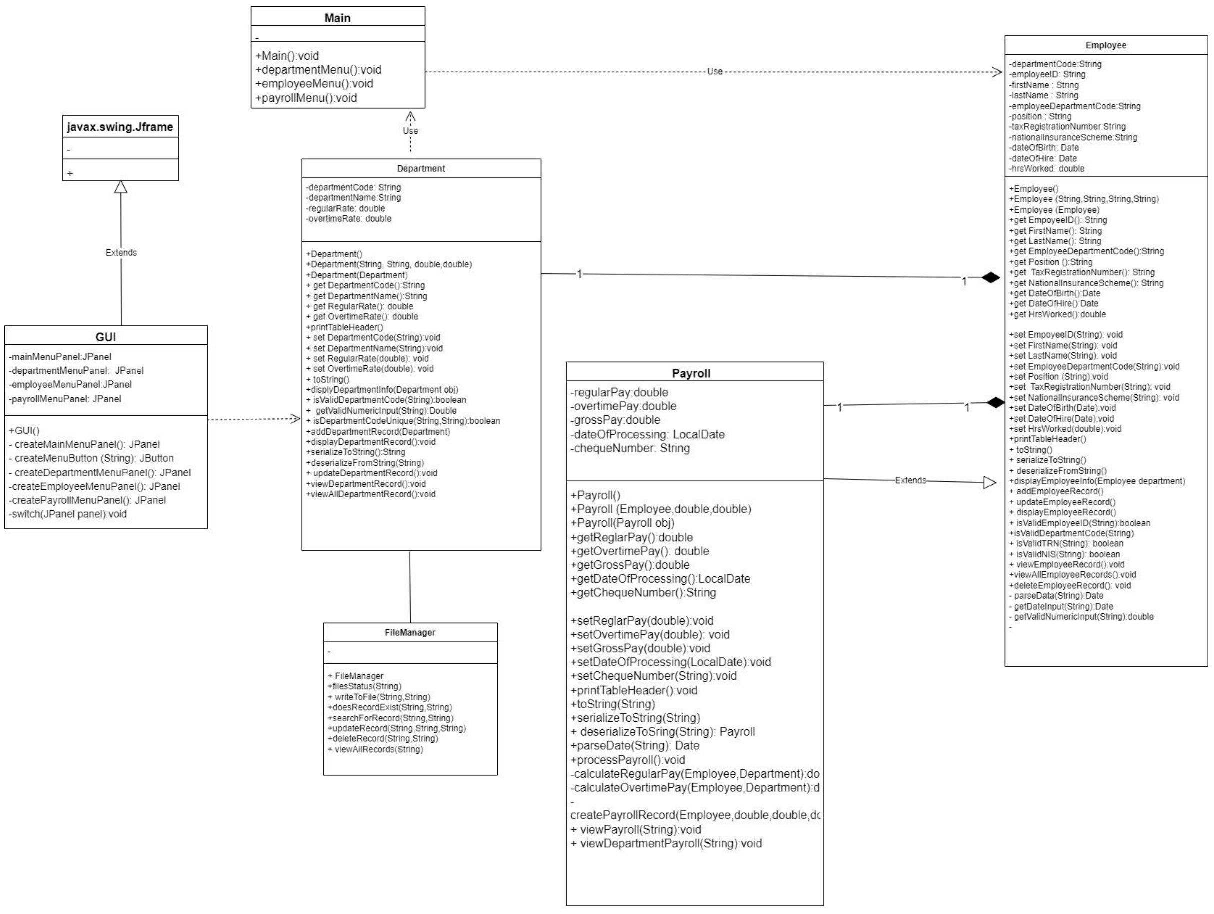
The GUI class is not a functional requirement, because without it the software is still usable, however it is needed for the ease of use by the user as well as the aesthetic display of information. The GUI will be associated with the Main class since, main is the only class that users will directly interact with. If this model is implemented in JAVA, the GUI class will extend the JAVA defined functionality, Jframe that is responsible for the window display, labels, buttons and all other GUI related features.

When this is being implemented in JAVA/C++ , the fundamental principles of OOP will be obeyed these include but are not limited to:

- Variable naming consistency with camel case
- Class naming consistency with pascal case
- Correct Formatting
- Potential Bug Fixes and updates prior to code development and implementation

OBJECT ORIENTED DESIGN

The following outline is displayed in the Object Oriented Design, using UML that displays class attributes, relationships and cardinality.


Main
-
+Main():void
+departmentMenu():void
+employeeMenu():void
+payrollMenu():void
Use
javax.swing.Jframe
-
+
Extends
Use
Department
-departmentCode: String
-departmentName:String
-regularRate: double
-overtimeRate: double
+Department()
+Department(String, String, double,double)
+Department(Department)
+ get DepartmentCode():String
+ get DepartmentName():String
+ get RegularRate(): double
+ get OvertimeRate(): double
+printTableHeader()
+ set DepartmentCode(String):void
+ set DepartmentName(String):void
+ set RegularRate(double): void
+ set OvertimeRate(double): void
+ toString()
+displyDepartmentInfo(Department obj)
+ isValidDepartmentCode(String):boolean
+ getValidNumericInput(String):Double
+ isDepartmentCodeUnique(String,String):boolean
+addDepartmentRecord(Department)
+displayDepartmentRecord():void
+serializeToString():String
+deserializeFromString(String)
+ updateDepartmentRecord():void
+viewDepartmentRecord():void
+viewAllDepartmentRecord():void
GUI
-mainMenuPanel:JPanel
-departmentMenuPanel: JPanel
-employeeMenuPanel:JPanel
-payrollMenuPanel: JPanel
+GUI()
- createMainMenuPanel(): JPanel
- createMenuButton (String): JButton
- createDepartmentMenuPanel(): JPanel
-createEmployeeMenuPanel(): JPanel
-createPayrollMenuPanel(): JPanel
-switch(JPanel panel):void
FileManager
-
+ FileManager
+filesStatus(String)
+ writeToFile(String,String)
+doesRecordExist(String,String)
+searchForRecord(String,String)
+updateRecord(String,String,String)
+deleteRecord(String,String)
+ viewAllRecords(String)
1
1
Payroll
-regularPay:double
-overtimePay:double
-grossPay:double
-dateOfProcessing: LocalDate
-chequeNumber: String
+Payroll()
+Payroll (Employee,double,double)
+Payroll(Payroll obj)
+getReglarPay():double
+getOvertimePay(): double
+getGrossPay():double
+getDateOfProcessing():LocalDate
+getChequeNumber():String
+setReglarPay(double):void
+setOvertimePay(double): void
+setGrossPay(double):void
+setDateOfProcessing(LocalDate):void
+setChequeNumber(String):void
+printTableHeader():void
+toString(String)
+serializeToString(String)
+ deserializeToSring(String): Payroll
+parseDate(String): Date
+processPayroll():void
-calculateRegularPay(Employee,Department):do
-calculateOvertimePay(Employee,Department):d
-
createPayrollRecord(Employee,double,double,dc
+ viewPayroll(String):void
+ viewDepartmentPayroll(String):void
1
1
Extends
Use
Employee
-departmentCode:String
-employeeID: String
-firstName : String
-lastName : String
-employeeDepartmentCode:String
-position : String
-taxRegistrationNumber:String
-nationalInsuranceScheme:String
-dateOfBirth: Date
-dateOfHire: Date
-hrsWorked: double
+Employee()
+Employee (String,String,String,String)
+Employee (Employee)
+get EmpoyeeID(): String
+get FirstName(): String
+get LastName(): String
+get EmployeeDepartmentCode():String
+get Position ():String
+get TaxRegistrationNumber(): String
+get NationalInsuranceScheme(): String
+get DateOfBirth():Date
+get DateOfHire():Date
+get HrsWorked():double
+set EmpoyeeID(String): void
+set FirstName(String): void
+set LastName(String): void
+set EmployeeDepartmentCode(String):void
+set Position (String):void
+set TaxRegistrationNumber(String): void
+set NationalInsuranceScheme(String): void
+set DateOfBirth(Date):void
+set DateOfHire(Date):void
+set HrsWorked(double):void
+printTableHeader()
+ toString()
+ serializeToString()
+ deserializeFromString()
+displayEmployeeInfo(Employee department)
+ addEmployeeRecord()
+ updateEmployeeRecord()
+ displayEmployeeRecord()
+ isValidEmployeeID(String):boolean
+isValidDepartmentCode(String)
+ isValidTRN(String): boolean
+ isValidNIS(String): boolean
+ viewEmployeeRecord():void
+viewAllEmployeeRecords():void
+deleteEmployeeRecord(): void
- parseData(String):Date
- getDateInput(String):Date
- getValidNumericInput(String):double
-

Department.java

Jordayne Price and Dongie King - Responsible for the implementation of the Department class. class was designed to represent department-related information, including department code, name, regular rate, and overtime rate. Additionally, it provides methods to interact with and manage department records through serialization, validation, and file operations. Here's a summary of its functionalities:

1. **Attributes: here are some examples**
    - **departmentCode**: Represents the unique code for a department.
    - **departmentName**: Represents the name of the department.
    - **regularRate**: Represents the regular rate associated with the department.
    - **overtimeRate**: Represents the overtime rate associated with the department.
2. **Constructors: here are some examples**
    - Default Constructor: Initializes attributes with default values.
    - Primary Constructor: Initializes attributes with provided values.
    - Copy Constructor: Creates a new **Department** object by copying an existing one.
3. **Getters and Setters:**
    - Accessors and mutators for the attributes to retrieve and update their values.
4. **Table Display:**
    - **printTableHeader**: Prints a formatted table header for displaying department records.
    - **toString**: Overrides the **toString** method to provide a formatted string representation of a department record.
5. **Display Department Information:**
    - **displayDepartmentInfo**: Displays information about a department, including the header and record.
6. **Attribute Validations:**
    - **isValidDepartmentCode**: Validates whether a given department code follows a specific pattern.
    - **getValidNumericInput**: Ensures valid numeric input, handling exceptions.
7. **Serialization:**
    - **serializeToString**: Converts a **Department** object to a string for storage in a file.
    - **deserializeToString**: Converts a string from a file to a **Department** object.
8. **File Operations:**
    - **addDepartmentRecord**: Adds a new department record to a file after input validation.

- **updateDepartmentRecord**: Updates an existing department record in a file after input validation.
- **viewDepartmentRecord**: Views a single department record based on user input.
- **viewAllDepartmentRecords**: Views all department records stored in a file.

9. **Menu-Driven Interface:**
   - The class was designed to support a menu-driven interface for managing department records with options like add, update, view, and view all.
10. **Input Validation:**
    - It includes input validation for department codes and numeric values.

Employee.java

Dongie King

- Responsible for the implementation of the Employee class. The class represents individual employee records and provides functionality for managing employee data. Here's a summary of its key features:

1. **Attributes:**
   - **employeeID**: Unique identifier for an employee.
   - **firstName**, **lastName**: First and last name of the employee.
   - **employeeDepartmentCode**: Code representing the department to which the employee belongs.
   - **position**: Job position of the employee.
   - **taxRegistrationNumber**, **nationalInsuranceScheme**: Tax and insurance-related identifiers.
   - **dateOfBirth**, **dateOfHire**: Dates of birth and hire for the employee.
   - **hrsWorked**: Hours worked by the employee.

2. **Constructors:**
   - Default constructor with empty values.
   - Primary constructor initializing all attributes.
   - Copy constructor for creating a new instance based on an existing one.

3. **Getters and Setters:**
   - Provides methods for accessing and modifying each attribute.

4. **Serialization and Deserialization:**
   - **serializeToString**: Converts employee data into a tab-delimited string.
   - **deserializeToString**: Creates an Employee object from a serialized string.
5. **User Input and Validation:**
   - Methods for validating employee ID, department code, tax registration number, and national insurance scheme.
   - Utilizes a **Scanner** object for user input.

6. **CRUD Operations:**

- **addEmployeeRecord**: Adds a new employee record by taking user input.
- **updateEmployeeRecord**: Updates an existing employee record based on user input.
- **viewEmployeeRecord**: Displays information for a single employee based on user input.
- **viewAllEmployeeRecords**: Displays information for all employees.
- **deleteEmployeeRecord**: Deletes an employee record based on user input.

7. **Table Display:**
   - **printTableHeader**: Prints a formatted table header for displaying employee information.

8. **Date Parsing:**
   - **setDateOfBirth** and **setDateOfHire**: Methods for parsing date strings into **Date** objects.

9. **Attribute Validation:**
   - Validation methods for employee ID, department code, tax registration number, and national insurance scheme.

10. **Utility Methods:**
    - Various utility methods for handling numeric and date input.

FileManager.java

Jordayne Price - Responsible for the implementation of the FileManager class. The class was designed to handle various file operations related to reading, writing, updating, and deleting records in a file. Here is a summary of its functionalities:

1. **File Status Check:**
   - The **fileStatus** method checks the status of a given file.
   - It logs whether the file is created, readable, and writable.
2. **Write to File:**
   - The **writeToFile** method appends data to a specified file.
   - It uses a **BufferedWriter** to write the provided data to the file.
3. **Record Existence Check:**
   - The **doesRecordExist** method checks if a specified record (search string) exists in a file.
   - It returns **true** if the record is found, and **false** otherwise.
4. **Search for Record:**
   - The **searchForRecord** method searches for a specified record (search string) in a file.
   - It returns the first occurrence of the record as a string.

5. **Update Record:**
   - The **updateRecord** method updates a record in a file based on a search string.
   - It creates a temporary file, replaces the existing record with new data, and then renames the temporary file to the original file.
6. **Delete Record:**
   - The **deleteRecord** method deletes a specified record (search string) from a file.
   - Similar to updating, it uses a temporary file to remove the specified record.
7. **View All Records:**
   - The **viewAllRecords** method reads all records from a file and returns them as an **ArrayList<String>**.
8. **Logging:**
   - The class uses a **Logger** for logging informational messages and errors.
9. **Exception Handling:**
   - The methods handle **IOException** and log relevant error messages.

Main.java

Percival Roberts – Responsible for the implementation of the Main class. The main class serves as the entry point of the Program. The **Main** class serves as the entry point for the program, providing a user interface to navigate through different functionalities related to department record management, employee record management, and payroll processing. Here's a summary of its key features:

1. **Main Menu:**
   - Displays a menu with options for department record management, employee record management, payroll processing management, and program exit.
   - Utilizes color-coding (green for menu options) for better user experience.
2. **Menu Handling:**
   - The **mainMenu** method processes the user's choice and invokes the corresponding functionality based on the selected option.
   - Utilizes a switch-case structure for option handling.
3. **Department Record Management:**
   - The **departmentMenu** method provides options to add, update, view, view all department records, and return to the main menu.
   - Uses the **Department** class methods for managing department records.
4. **Employee Record Management:**
   - The **employeeMenu** method offers options to add, update, view, view all employees in a department, delete, and return to the main menu.

- Relies on the **Employee** class methods for employee record management.

5. **Payroll Processing Management:**
   - The **payrollMenu** method presents options for processing payroll, viewing individual payroll records, viewing department payroll, returning to the main menu, and exiting the program.
   - Interacts with the **Payroll** class for payroll-related operations.
6. **Input Handling:**
   - Utilizes a **Scanner** object for user input.
   - Handles invalid input gracefully, providing feedback to the user.
7. **Exit Handling:**
   - Closes the **Scanner** object and exits the program when the user chooses the exit option.
8. **Color Codes:**
   - Uses ANSI escape codes for color-coding certain text in the console (e.g., green for menu options).

Payroll.java

Joshuawn Johnson – was responsible for the development of the payroll class.

1. **Payroll Processing:**
   - Handles payroll processing with options to calculate payroll and generate a Processed Payroll File.
   - Allows viewing of single employee payroll records and all employee payroll records for a specific department.
2. **Attributes:**
   - Includes attributes such as regular pay, overtime pay, gross pay, date of processing, and cheque number.
   - Utilizes a primary constructor, default constructor, and copy constructor for object creation.
3. **Serialization:**
   - Implements methods to convert Payroll objects to a formatted string and deserialize a string to a Payroll object.
   - Uses a helper method to parse date strings during deserialization.
4. **Table Header:**
   - Defines a method (**printTableHeader()**) to print a formatted table header for displaying payroll records.

5. **Processing Logic:**
    - Contains methods to process payroll for all employees, calculate regular pay, calculate overtime pay, and create a payroll record.
    - Utilizes FileManager class methods for file operations.
6. **Viewing Payroll:**
    - Implements methods to view a single employee payroll record and view all employee payroll records for a specific department.
7. **Random Cheque Number:**
    - Generates a unique 6-digit random cheque number for each payroll record.
8. **Date Formatting:**
    - Utilizes **DateTimeFormatter** for consistent date formatting in string representations.
9. **Error Handling:**
    - Implements exception handling during date parsing and provides error messages when necessary.

GUI.java

Percival Roberts & Joshuawn Johnson – Both members were responsible for the implement of the GUI class.

1. **Functionality:**
    - Implements a graphical user interface (GUI) for the SSN Payroll Management System.
2. **Main Menu:**
    - Displays a main menu with options for Department Record Management, Employee Record Management, Payroll Processing Management, and Exit.
    - Each option is represented by a button with corresponding actions.
3. **Department Record Management:**
    - Subpanel for managing Department Records with options:
        - Add: Allows the user to add a new department rates record to the system.
        - Update: Allows the user to update an existing department record.
        - View: Allows the user to view a single department record.
        - View All: Allows the user to view all department records.
        - Return to Main Menu: Takes the user back to the main menu.
4. **Employee Record Management and Payroll Processing Management:**

- Placeholder panels for Employee Record Management and Payroll Processing Management are present, with room for implementation.

5. **Button Actions:**
   - Each button in the GUI has associated actions using event listeners.
   - Switches between panels based on user actions.
   - Exit button terminates the program.
6. **Panel Management:**
   - Utilizes a **switchToPanel** method to dynamically switch between different panels.
   - Enhances modularity and improves the user experience.
7. **Layout and Design:**
   - The GUI has a layout with a central content area using a **BorderLayout**.
   - Buttons have an aesthetically pleasing font for better readability.
8. **Size and Appearance:**
   - The GUI is set to a size of 1920x1080 pixels.
   - The appearance is visually appealing and follows a clean design.

Code Review

Percival Roberts & Joshuawn Johnson- After each development was made for the respective files both reviewers went through the code to analyse and ensure the following standards were met:

1. Variable naming consistency with Camel Case
2. Class naming consistency with Pascal Case
3. Correct formatting
4. Potential Bug fixes

NB: For files with multiple authors they were responsible for peer reviewing the code before passing it on to the code reviewers