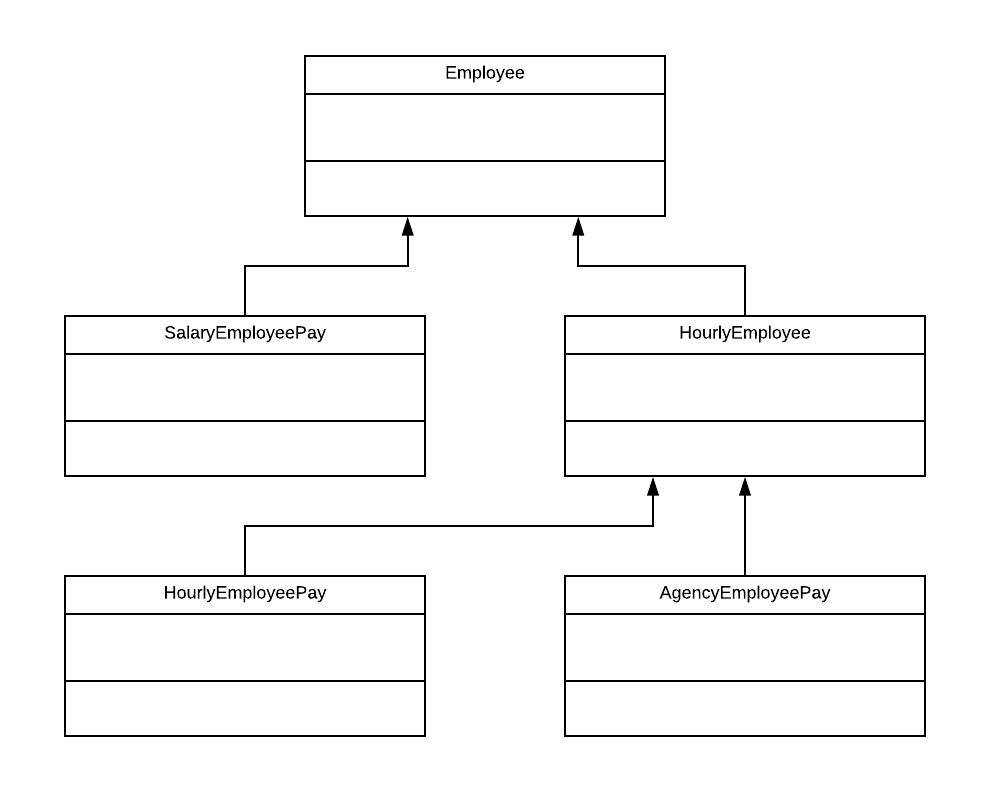
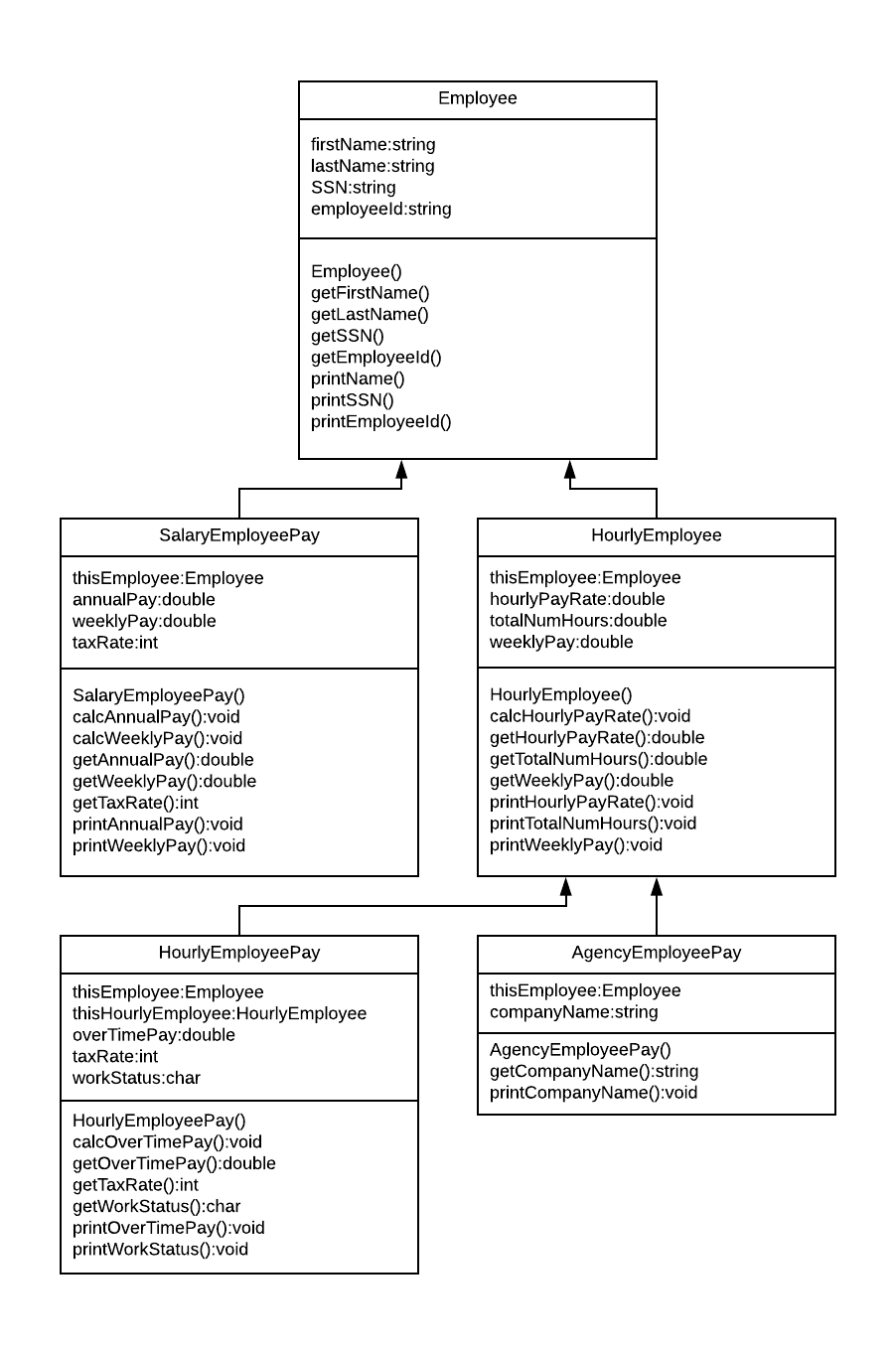
1. **Problem Statement**

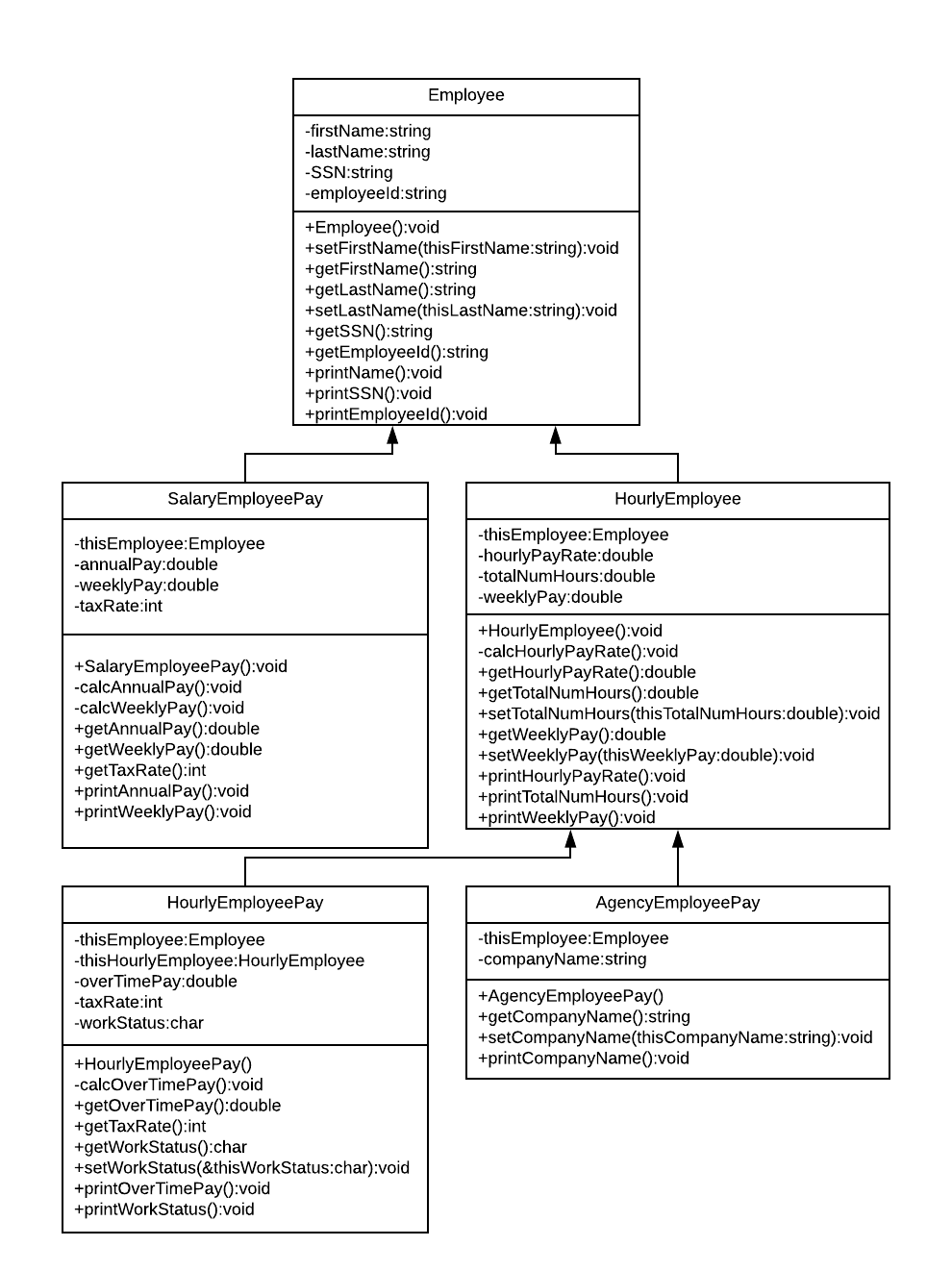
Program calculates and displays employee payments for a company using employees’ private information

1. **Requirements**
   1. **Assumptions**
      1. User types in appropriate integer number to select an option from the menu
      2. User types social security number without the dashes
      3. User types employee number without the dashes
      4. User types first and last name with space in the middle
      5. 40 hours per week for every employee
      6. User types either ‘F’ for Full time work status or ‘P’ for Part time work status
   2. **Specifications**
      1. Base class *Employee* has Employee name, Social Security Number, and Employee number stored in respective variables with a given format:
         1. Social Security Number format: xxx-xxx-xxxx (x as in number)
         2. Employee Number format: xxx-L (L as in letter)
      2. Getter an setter functions for each private variables in each class
      3. Print method to print each data in each class
      4. Constructor and destructor to be added in each class
      5. Class *SalaryEmployeePay*, derived from *Employee* class, has following data:
         1. Annual pay
         2. Weekly pay – Calculated from annual pay
         3. Tax rate code (1 = 0.25, 2 = 0.20, 3 = 0.15)
      6. Class *HourlyEmployee*, derived from *Employee* class, has following data:
         1. Hourly pay rate – based on weekly pay, assume a 40-hour work week
         2. # of hours worked
      7. Class *HourlyEmployeePay*, derived from *HourlyEmployee* class, has following data:
         1. Overtime pay rate – Calculated from *HourlyEmployee* hourly pay rate
         2. Tax rate code (1 = 0.25, 2 = 0.20, 3 = 0.15)
         3. Work Status (F = Full time, P = Part time)
      8. Class *AgencyEmployeePay*, derived from *HourlyEmployee* class, has a company name
      9. Output file “pay.dat” to store sample data for various employee and the calculations
2. **UML Classes Design Part 1:**
3. **Decomposition Diagram**

|  |  |  |
| --- | --- | --- |
| **Main** | | |
| **Input** | **Process** | **Output** |
| First & last name | Assign first and last name in separate variables | When function called, print first and last name |
| Social Security Number (SSN) | Assign SSN to its variable in the format xxx-xxx-xxxx | When function called, print SSN with dashes |
|  | Check if SSN is valid with no alphabetic characters |  |
| Employee Number | Assign employee # to its variable in the format xxx-L | When function called, print employee # with dash |
|  | Check if employee # is valid and matches its format |  |
| Menu Selection | Call the function with respect to number selected by user |  |
| Company name | Assign company name to its appropriate variable | When function called, print employee # with dash |
| Annual pay | Assign annual pay to its appropriate variable |  |
|  | Calculate weekly pay |  |
|  | Calculate hourly pay rate |  |
| Work Status | Assign the character to its appropriate variable |  |
| # of hours worked | Assign # of hours worked to its variable |  |
| Tax Rate Code | Check if tax rate code is between 1 and 3 |  |
|  | Use tax rate code to figure out the pay |  |

1. **UML Classes Design Part 2**
2. **Test Strategy**
   1. Valid Data
   2. Invalid Data
3. **Test Plan Version 1**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| Valid | 1 | SSN has no alphabetic characters. ONLY numbers. |  |  |  |  |
| Valid | 2 | Employee # has 3 numbers and 1 letter at the end |  |  |  |  |
| Valid | 3 | Only positive number accepted for annual pay |  |  |  |  |
| Valid | 4 | Only values between 1 and 3 are accepted for tax rate code |  |  |  |  |
| Valid | 5 | Hourly pay rate >= 10 |  |  |  |  |
| Valid | 6 | Hourly pay rate <= 75 |  |  |  |  |
| Valid | 7 | Only values ‘F’ or ‘P’ accepted for work status |  |  |  |  |
| Invalid | 1 | User inputs alphabets in SSN |  |  |  |  |
| Invalid | 2 | User inputs only numbers for employee number |  |  |  |  |
| Invalid | 3 | User inputs only letters for employee number |  |  |  |  |
| Invalid | 4 | User inputs negative # for annual pay |  |  |  |  |
| Invalid | 5 | User inputs value for tax rate code that is not between 1 and 3 |  |  |  |  |
| Invalid | 6 | Hourly pay rate is less than 10 |  |  |  |  |
| Invalid | 7 | Hourly pay rate is more than 75 |  |  |  |  |
| Invalid | 8 | User inputs more than 60 hours worked |  |  |  |  |
| Invalid | 9 | User inputs value for work status other than ‘F’ or ‘P’ |  |  |  |  |

1. **UML Classes Design Part 3**
2. **Initial Algorithm**
   1. In main function
      1. Include all the classes and libraries to this file
      2. Make various employee instances
         1. Get data from user and store it in a specific variable
         2. For each instance calculate employee’s pay
         3. Display her/his pay information on the screen by calling print functions
         4. Save the same displays in a file *pay.dat*
   2. In *Employee* class
      1. The constructor
         1. Ask user for employee name and store it in appropriate variable
         2. Ask for Social Security #
            1. Loop until social security format correct
         3. Ask for employee #
            1. Loop until employee # format is correct
   3. In *SalaryEmployeePay* – derived from *Employee* class
      1. Constructor
         1. Ask user for annual pay
            1. Loop until user enters positive number for annual pay
         2. Calculate and store weekly pay in its appropriate variable
            1. Weekly pay = annual pay / 52.142
         3. Ask user for a tax rate code
            1. Loop until user enters # between 1 and 3

1 = 0.25

2 = 0.20

3 = 0.15

* 1. In *HourlyEmployee* class – derived from *Employee* class
     1. Constructor
        1. Ask user for hours worked in a week and store it in a variable
        2. Ask user for weekly pay and store it in appropriate variable

1. **Test Plan Version 2**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| Valid | 1 | SSN has no alphabetic characters. ONLY numbers. | “123456789” | “SSN successfully taken” |  |  |
| Valid | 2 | Employee # has 3 numbers and 1 letter at the end | “123N” | “Employee # successfully taken” |  |  |
| Valid | 3 | Only positive number accepted for annual pay | “100” | “Annual pay successfully taken” |  |  |
| Valid | 4 | Only values between 1 and 3 are accepted for tax rate code | “1” | “Tax rate of 25% selected successfully” |  |  |
| Valid | 5 | Hourly pay rate between 10 and 75 | “50” | “Hourly pay rate successfully calculated” |  |  |
| Valid | 6 | Only values ‘F’ or ‘P’ accepted for work status | ‘F’ | “Work status successfully updated to Full Time” |  |  |
| Invalid | 1 | User inputs alphabets in SSN | “12345abcd” | “Invalid Input: incorrect SSN format” |  |  |
| Invalid | 2 | User inputs only numbers for employee number | “1234” | “Invalid Input: Incorrect Employment # format” |  |  |
| Invalid | 3 | User inputs only letters for employee number | “ABCD” | “Invalid Input: Incorrect Employment # format” |  |  |
| Invalid | 4 | User inputs negative # for annual pay | “-20” | “Invalid Input: Negative # not allowed” |  |  |
| Invalid | 5 | User inputs value for tax rate code that is not between 1 and 3 | “5” | “Invalid Input: Out of bounds tax rate code” |  |  |
| Invalid | 6 | Hourly pay rate is less than 10 | “1” | “Invalid Input: Hourly pay cannot be less than 10” |  |  |
| Invalid | 7 | Hourly pay rate is more than 75 | “100” | “Invalid Input: Hourly pay cannot be more than 75” |  |  |
| Invalid | 8 | User inputs more than 60 hours worked | “100” | “Invalid Input: Total hours worked in a week cannot exceed 60” |  |  |
| Invalid | 9 | User inputs value for work status other than ‘F’ or ‘P’ | “A” | “Invalid Input: Only values ‘F’ or ‘P’ is allowed for work status” |  |  |

1. **Code**
2. **Updated Algorithm**
3. **Test Plan Version 3**
4. **Screenshots**
5. **Status**