# FPT University

# Java OOP Project

# Report

*Member: Trần Thành Nhân, Nguyễn Minh Tú*

## Problem Description

* 1. **Person**
     1. A person object has three attributes: code, name, address;

Some methods: constructors, getters, setters, and overriding method toString() returns formatted String: “*code* | *name* | *address*”.

* + 1. All person objects have a common attribute: TAX, with initial value is 10%.
  1. **Student** extends **Person** implements **Comparable**
     1. A student has two more attributes: grade (Double) and valid (Date). Field valid contains the enrolled date when that student started to study.
     2. Methods of student: constructors, getters, setters, overriding method toString() returns formatted String: “*code* | *name* | *address* | *grade*”, overriding method compareTo() compares the grade of this student to other student’s grade; method isValid() check if number of study years exceed 6 years to the current date.
  2. **Professor** extends **Person**
     1. Attributes of professor:
* Enum PositionEnum (PROFESSOR, ASSOCIATE\_PROFESSOR, HONOR\_PROFESSOR).
* Enum EducationLevel (BACHELOR, MASTER, DOCTOR).

Attributes: experience (int, > 0), basicSalary (int, default = 1000), position (PositionEnum), education (EducationLevel).

Attributes to manage a list of student: arr (List <**Person**>), count (int): holds the number of available students in arr.

* + 1. Basic methods of professor: constructors, getters, setters, overriding method toString() returns formatted String “*code* | *name* | *address* | *experience* | *realSalary*”.
    2. Common attributes of all professors:
* COF = 0.33; STEP = 3;
* SUPPOS = 0.5 or 1.25 or 0.75, depends on position is ASSOCIATE\_PROFESSOR, PROFESSOR or HONOR\_PROFESSOR correspondingly.
* SUPLEV = 0.1 or 0.25, depends on education is MASTER or DOCTOR, correspondingly.
  + 1. Common methods of all professors:
       - getRealSalary(Professor) = basicSalary \* (experience / STEP) \* (1+COF) \* (1+SUPPOS) \* (1+SUPLEV)
       - getAnnualIncome(Professor) = getRealSalary(Professor) \* 12 \* (1-TAX)
    2. Management methods of professor: to manage student list
       - boolean add(Student x): add a student x to arr and return true if added successfully, otherwise return false.
       - void addAllStudent (File f): access input file f and add all students to arr with student’s format.
       - void remove (String studentCode): remove a student with studentCode in arr.
       - void update (String studentCode): update information (name, address, grade, valid) of a student with studentCode in arr.
       - void removeInvalid(): method will find all students who have invalid date (more than 6 years from the enrolled date to the current date) and then remove them out of arr.
       - Student find(String findCode): This method will return a student who has code == findCode in the list arr.
       - void displayAllStudents(): display all students available in student list arr in ascending order of code.
       - void sortDescending(): use Bubble Sort to sort the list arr of students in descending order of code.
       - Student[] getTopGrade(): return a list of students who have the most grade in arr list.
       - Student[] getBottomGrade(): return a list of students who have the least grade in arr list.
       - void showDistribution(): display a statistic of the distribution of grade, with data retrieved from arr.
       - double averageGrade(): returns the average grade of all students in arr.
       - void showDistinctGrade(): display a statistic of all students whose grade are unique.
       - void showDuplicateName(): display all students’ names which are names of at least two students.
       - void showDistinctName(): display all students’ names which are name of exactly one student.
       - void outAllStudents(File f): store all student in arr and save in output file f.
  1. Data text file and Data format
     1. A text file, named *in\_students.txt* contains a number of lines.

The first line represents data of a professor: the owner of student list. Data of professor has format: “*code | name | address | experience | basicSalary | position | education | count*”, where count is the number of students in the list.

Each of the following lines represents a student’s data in format: “*code | name | address | grade | valid*” separated by a single vertical bar.

Valid has format: *day/month/year.*

Code has format: see the section below.

* + 1. A text file, named *out\_students.txt*, stores data from students list arr to the file when the method outAllStudents() performed. Stored data have the same format like data in file *in\_students.txt*.
    2. Student code has format: *STxxx*, where xxx is a 3-digit number in range [001, 999] inclusive.
    3. Professor code has format: *PRxxx*, where xxx is a 3-digit number in range [001, 999] inclusive.
    4. Person name must be in standard format: remove all leading and ending spaces; all spare spaces between words; only the first character of each word is uppercase.
  1. Application  
     Write a Java application with the following user interface, allows user to manage a professor’s list of students. Supported functions:
     + Add a student to professor’s list, retrieve data from keyboard.
     + Import all students from an input file to professor’s list.
     + Remove a student in the list by student’s code.
     + Remove all invalid students.
     + Update data of a student in the list by student’s code.
     + Find a specific student in the list by student’s code.
     + Display all students in ascending order of student’s code.
     + Display all students who have the most grade.
     + Display all students who have the least grade.
     + Display the average grade of all students.
     + Display distribution of grade.
     + Display the distinct grade of all students in the list.
     + Display all students who have the same name.
     + Display all students who have unique name.
     + Export the student list to an output file.
     + Quit.

## Analysis. From the problem description, the following user-cases are idendified:

User

Add a student

System

Export students to file

Display unique names

Display duplicate names

Display grade distribution

Display distinct grade

Display bottom students

Display top students

Display all student

Find a student

Import students from file

Remove a student

Remove invalid students

Update a student

Display average grade

## Design

|  |  |
| --- | --- |
| **Concept** | **Class** |
| Person | Person (code, name, address) |
| Student | Student (grade, valid) extends Person implements Comparable |
| Professor | Professor (experience, basicSalary, position, education, Student[]) extends Person |
| Menu | A manage student menu |
| Program | Manage program |

**Class detail**

|  |
| --- |
| **Person** |
| * static double TAX = 10% * code: String * name: String * address: String * Person() * Person (String, String, String) * Getters, Setters * String toString(): “code | name | address” * isCodeStandard(String, String): check the format code input * String trimName(String) |

|  |
| --- |
| **Menu** |
| * void add (String) * void addSubMenu(int index, String s): add a submenu line *s* to the super menu at index *index* * int getUserChoice() * void displayMenu() * void displaySubMenu() * int getChoice() |

|  |
| --- |
| **Manage Program** |
| * void loadFile(Professor): load student list to Professor * void main(String[] args) |

|  |
| --- |
| **Student extends Person** |
| * grade: double * valid: date * Student() * Student(String, String, String, double, SimpleDate) * Getters, Setters * String toString(): “code | name | address | grade” * int compareTo(Student s): compare grade between 2 students * Comparator compareCode(Object 1, Object 2) * Comparator compareName(Object 1, Object 2) * Comparator compareGrade(Object 1, Object 2) * boolean isValid(): valid date is less than 6 years to current date * boolean parseStudent(String): parse a string into data of student * boolean updateStudent(Student, Student):clone Student to update, return true if successful * Student newStudent(): return a new student with data retrieve from keyboard * Void updateStudent(Student st, Student original):update student *st* from student *original* * Void validGrade(Double): check grade in range of 0-10 |

|  |
| --- |
| **Professor extends Person** |
| * PositionEnum: PROFESSOR, ASSOCIATE\_PROFESSOR, HONOR\_PROFESSOR * EducationLevel: BACHELOR, MASTER, DOCTOR * experience: int * basicSalary: int * position: PositionEnum * education: EducationLevel * String [] student * count: int * Professor() * Professor(int, int, PositionEnum, EducationLevel, Student[], int, String, String, String) * Getters, Setters * String toString(): return “code | name | address | experience | realSalary” * COF = 0.33 * STEP = 3 * SUPPOS = * SUPLEV = * realSalary = basicSalary \* (experience/STEP) \* (1+COF) \* (1+SUPPOS) \* (1+SUPLEV) * annualIncome = realSalary \* 12 \* (1-TAX) * boolean parseProfessor(String): parse a string into data and assign to professor’s fields * boolean addStudent(Student s): return true if add successful * boolean addAllStudent(File f): read all Student from file, return true if successful * void removeStudent(String code): remove student by student code * void removeInvalid(): remove Students has valid date is more than 6 to current * Student findStudent(String fcode): return student match the fcode * void updateStudent(String code): update information of a student by student code. * void displayAllStudents(): display all student in ascending order of student code * void sortByCode(): sort list of students by code * void sortByName(): sort list of students by name * void sortByGrade(): sort list of students by grade * void displayHighestGradeStudent(): display student has highest grade * void displayLowestGradeStudent(): display student has lowest grade * void displayAverageGrade(): display average grade * void displayDistributionOfGrade(): display all distribution of grade * void displayDistinctGrade(): display all distinct grade * void dupStudentName(): display all student have the same name * void uniqueStudentName(): display all unique student name * void displayHigher5(): display students whose grade higher than 5 * void displayLower5(): display students whose grade lower than 5 * void displayNthYearStudent(): display students who’s in n-th year * int getNumberStudentsOfYear(String sYear): get number of students enroll in the given year * boolean updateProfessor(): update Professor information, return true if successful * void displayProfessor(): display Professor information * boolean isExist(String code, ArrayList<Person> arr): check if Student code exist or not in the array when add student to * void exportTo(File f): write all student into file |

|  |
| --- |
| **FormatException extends RuntimeException** |
| A user define exception allows program to catch an exception with custom message   * String message * FormatException() * FormatException(String) * String getMessage() * void setMessage(String) |

**SUPPORTED CLASSES**

|  |
| --- |
| **SimpleDate** |
| * int dayOfMonth * int monthValue * int year * SimpleDate() * SimpleDate(int, int, int) * Getters, Setters * String toString() * SimpleDate parseSimpleDate(String): parse a String into data day/month/year * Void isValidDate(SimpleDate) throws FormatException: check valid of a date |

(+: public members, -: private members)

**User Interface:**

*All students from initial data.txt have been added successfully*

========== Manage Student Program ===========

1. Add student
2. Remove student
3. Update student
4. Find student
5. Display students
6. Display statistic
7. Export student list to file
8. About Professor
9. More display options
10. Quit

*Sub-menu of each option:*

* Option 1 (Add):
  1. Add a new student
  2. Import student list from file
* Option 2 (Remove):
  1. Remove a student
  2. Remove invalid students
* Option 5 (Display):
  1. Display students sorted by student code (ascending)
  2. Display students sorted by student name (ascending)
  3. Display students sorted by student grade (descending)
  4. Display highest grade students
  5. Display lowest grade students
* Option 6 (Statistic):
  1. Display average grade
  2. Display distribution of grade
  3. Display distinct grade
  4. Display duplicate student names
  5. Display unique student names
* Option 8 (About professor):
  1. Update information of professor
  2. Display information of professor
* Option 9 (More display options):
  1. Display students have grades higher than average
  2. Display students have grades lower than average
  3. Display n-th year students
  4. Display number of students enrolled in a specific year

## Implementation

1. **Initial data of the program (file *initial data.txt*)**

Please explore the software structure

1. **Software**

Please run the program

## Testing

|  |  |
| --- | --- |
| **Validation of data** | |
| **Test** | **Result (pass / not pass)** |
| Format code of Student: STxxx | **Passed** |
| Format code of Professor: PRxxx | **Passed** |
| Name of Person has single space between words | **Passed** |
| Date of student must be valid | **Passed** |
| Grade of student must be in range [0, 9] | **Passed** |
| Student codes are unique | **Passed** |

|  |  |
| --- | --- |
| **Function** | |
| **Method** | **Result** |
| Load initial data from file ‘initial data.txt’ | **Success** |
| Add new Student  (Option 1 → 1)   * code: ST020 * name: Henry Watson * address: 221B Baker Street, London, England * grade: 8 * day: 1 * month: 1 * year: 2015   *Note: Choose 5 (display) and then choose 1 to display result* | **Success** |
| Add students from file  (Option 1 → 2)   * file name: in\_students.txt   *Note: Choose 5 (display) and then choose 1 to display result* | **Success** |
| Remove a student  (Option 2 → 1)   * student code: ST008   *Note: Choose 5 (display) and then choose 1 to display result* | **Success** |
| Remove all invalid students  (Option 2 → 2)  *Note: Choose 5 (display) and then choose 1 to display result* | **Success** |

|  |  |
| --- | --- |
| Update student  (Option 3)   * code: ST020 * name: Henry J. Watson * address: skip * grade: 9 * day: 10 * month: 10 * year: skip   *Note: Choose 5 (display) and then choose 1 to see result* | **Success** |
| Find student  (Option 4)   * code: ST020   → Result: information of student ST020 | **Success** |
| Display students sorted code ascending  (Option 5 → 1) | **Success** |
| Display students sorted name ascending  (Option 5 → 2) | **Success** |
| Display students sorted grade descending  (Option 5 → 3) | **Success** |
| Display highest grade students  (Option 5 → 4)  → Result: Two students with grade 9.9 | **Success** |
| Display lowest grade students  (Option 5 → 5)  → Result: Students ST009 with grade 2.0 | **Success** |
| Display average grade  (Option 6 → 1)  → Result: 6.42, suppose that all the above tests were executed. | **Success** |
| Display distribution of grade  (Option 6 → 2) | **Success** |
| Display distinct grade  (Option 6 → 3) | **Success** |
| Display duplicate student names  (Option 6 → 4) |  |
| Display unique student names  (Option 6 → 5) | **Success** |
| Export student list to file  (Option 7)   * file name: out\_students.txt | **Success** |
| Update professor  (Option 8 → 1)  *Note: Fist, choose 8 (about professor) and then choose 2 (display professor) to see the difference*   * name: skip * address: skip * position: HONOR\_PROFESSOR * edu: DOCTOR * experience: 30 * basic salary: 3000 | **Success** |
| Display professor  (Option 8 → 2) | **Success** |
| Display students with grade higher than 5  (Option 9 → 1) | **Success** |
| Display students with grade lower than 5  (Option 9 → 2) | **Success** |
| Display n-th year student  (Option 9 → 3)   * n: 5   → Result: display students: ST001, ST002, ST006, ST007 *(suppose that all above tests were excuted)* | **Success** |
| Display number of students enrolled in a year  (Option 9 → 4)   * year: 2016   → Result: 3 students | **Success** |

Other details: this program is able to handle invalid input data from user to avoid making program crash.

## Task list

1. **Member’s Tasks**
   1. Trần Thành Nhân
      1. Problem Description
      2. Analysis
      3. Implementation
         1. Enum EducationLevel (20%)
         2. Enum PositionEnum (20%)
         3. Class Student (30% methods)
         4. Class Person (20% methods)
         5. Class ManageProgram (include main class) (50% methods)
         6. Class Professor (50% methods)
         7. Additional support class (80% methods)
         8. Class Menu (80%)
      4. Testing (50%)
   2. Nguyễn Minh Tú
      1. Design
      2. Implementation
         1. Enum EducationLevel (80%)
         2. Enum PositionEnum (80%)
         3. Class Person (80%)
         4. Class Student (70%)
         5. Class ManageProgram (50%)
         6. Class Professor (50%)
         7. Additional support class (20%)
         8. Class Menu (20%)
      3. Testing (50%)
2. **Contributing rate**
3. Trần Thành Nhân: 50%
4. Nguyễn Minh Tú: 50%