School Application

Objectives

* Be able to write Java code to implement and utilize enumerated types
* Be able to write Java code to implement and utilize a class and instance methods
* Be able to write Java code to store, maintain and iterate through a collection of objects
* Be able to effectively troubleshoot, test and debug Java code
* Be able to write Java code to sort a collection
* Be able to write Java code to read a text file and store data in RAM
* Be able to write Java code to effectively validate data entered into a system
* Be able to write Java code to effectively implement exception handling
* Be able to write Java code that allows classes to collaborate within a system
* Be able to write Java code to implement the OOP concepts of encapsulation, composition, inheritance and polymorphism

The following outlines a set of classes designed to collaborate to allow a user to manage a School. The instance variables and methods of the classes have been defined but feel free to improve the design and add features if you wish.

Four text files populated with data have been prepared so that a School can be instantiated with sample data.

Pay close attention to the structure of the data within each file to determine how the file should be read.

**Teachers.txt**

Sample Record – how the data appears in the file, what each of the data values represents and what Scanner method must be used to read it:

TeacherID (nextInt()) FirstName (next()) LastName (next())

40000 Mark Fuhrer

data

Street (nextLine())

1 Ave A

data

City,State (next()) can use String.split method to process

Brooklyn,NY

data

ZipCode (nextInt())

data

11204

TelNumber (next()) Gender (next().charAt(0) HireDate(next()) BirthDate(next())

data

9173051234 M 11/2/2005 5/23/1970

EmployeType (next()) DepartmentCode(next()) SocialSecNum(next())

data

PROFESSOR D100 000112222

Degree(next()) Major(next()) Salary(nextDouble())

data

Phd MATH 80000.00

**Departments.txt**

Sample Record – data values are delimited by ; read in Scanner.next() and split the string into separate components.

DepartmentID, DepartmentName, location, phoneNumber,faxNumber , teacherID (of department chair)

data

D105;ACCOUNTING;200 Whitehead Hall;7189811500;7189811501;40013

**Courses.txt**

Sample Record – data values are delimited by ; - read in with Scanner.next() and split string into separate components.

CourseID, CourseName, DepartmentID, Number of Credits

data

HIS220;Survey of Modern History 1;D103;3

**Students.txt**

StudentID LastName FirstName MidInitial

10179 Euler Lennie L

City,State (next()) can use String.split method to process

Three Rivers,Michigan

PhoneNumber Gender Major

2786290017 M MATH

1. Define an enumerated type Major that will store the code that represents each of the valid Majors that a student can choose and a String value that describes the major. Feel free to add any majors that aren’t included in this list.

Valid Major Codes and associated description are as follows:

ACCT (“Accounting”), ART (“ART”), BIOL (“Biology”), CHEM (“Chemistry”), CPSC (“Computer Science”), ECON (“Economics”), EDUC (“Education”), ENGL (“English”), ENGR (“Engineering”) ,FRCH (“French”), GEOG (“Geology”), GERM (“German”), GREE (“Greek”), HIST (“HISTORY”), MATH (“MATH”) , MUSC (“Music”), NURS (“Nursing”), PHIL (“Philosophy”), PE (“Physical Ed”),PHYS (“Physics”), POLS (“Political Science”), PSYC (“Psychology”), RELI (“Religion”), SOCI (“Sociology”), SPEE (“Speech”) , UDCD (“Undecided”)

1. Define an enumerated type EmployeeType that will store the following employee types. You can add others that you identify.

ACCOUNTANT, CHAIRPERSON, CLERK, DEAN, PROFESSOR, INSTRUCTOR, MAINTENANCE, SECRETARY

1. Design an enumerated type Degree that will store the following types of degrees

BA,BS, MA,MS,Phd,CPA

1. Design an enumerated type Grade that will store the following types of Grades and their associated point value

APLUS (4.0) , A(4.0), A- (3.7), BPLUS (3.3), B(3.0), BMINUS(2.7),C+(2.3), C(2.0),CMINUS(1.7),DPLUS(1.3),D(1.0),DMINUS(.7), F(0.0)

1. Design an enumerated type Semester that will store the following semester values

FALL, SPRING, SUMMER1, SUMMER2

1. Design an enumerated type Section that will store the following section values

BA,BB,BC, FA, FB,FD, OL,FC,FD,FE

1. Design a Department class that can store the following data about each Department

DepartmentID, Department Name, location, phone number, fax number, department chairperson

* Implement constructors. Note: location, phone number, fax number and department chairperson are not required to define a new department but may be available when setting up a new department
* Implement setters and getters
  + Department location, phonenumber, faxnumber and department chairperson might be modified.
* Data validation should always be performed to verify that phonenumber and fax number consist of ten digits of numeric data.
* Implement toString () method
* Implement compareTo() and equals() methods that will compare two instances of Department based on their respective DepartmentIDs

1. Design an Address class that will include the following data

Street , City, State, ZipCode

* Implement constructor
  + Verify that ZipCode consists of 5 or 9 numeric digits
  + Verify that state is a valid abbreviation of State in the US.
    - Note : you can set up an enumerated type that will identify each state abbreviation.
* Implement setters and getters
* Implement toString () method

1. Design a Person class that will include the following data about each Person associated with a given university.

ID, FirstName, LastName, MidInitial(optional), Address, PhoneNumber(optional), Gender

* Implement constructors. Note: MidInitial, PhoneNumber are optional but may be available when setting up a new Person
* Implement setters and getters
  + Lastname, address and phoneNumber may be modified
* Data validation should always be performed to verify that phonenumber consists of ten digits of numeric data.
* Implement toString () method
* Implement compareTo() and equals() methods that will compare two instances of Person based on their respective IDs

1. Design a Student class that “is a” Person class that will include the ***additional*** data about each Student currently enrolled at given university.

Major, EnrolledDate, DateOfBirth, GPA(default null), CreditsEarned(default 0), SocialSecurityNum, ArrayList < CompletedCourse>

* Implement constructors. Note: MidInitial, PhoneNumber, Major are optional but may be available when setting up a new Student. GPA, CreditsEarned should be set to default values when an instance of Student is instantiated.
  + If Major isn’t provided it should be set to UDCD (undecided)
* Implement setters and getters
  + Major,GPA, CreditsEarned may be modified
* Data validation should always be performed to verify that Major is a valid value, GPA is a value between 0-4.0 and CreditsEarned is a reasonable value
* Implement toString () method
* Implement the following additional methods
  + void CompleteCourse(Course c, Grade g)
    - This method should
      * Instantiate a new CompletedCourse
      * Add this CompletedCourse to the ArrayList of CompletedCourses
      * Recalculate the creditsEarned and GPA of this Student based on the Grade earned for the Course ‘c’ and the number of credits of Course ‘c’
  + CompletedCourse findCompletedCourse (String courseID)
    - return a deep copy of CompletedCourse with given CourseID. Throw NotFoundException if course wasn’t completed
  + Grade getGradeofCourse(String courseID)
    - return the Grade earned for completedCourse with given CourseID
  + ArrayList<CompletedCourse> getCoursesbyDepartment(String departmentID)
    - return a list of CompletedCourses that were offered by the departmentID
  + ArrayList<CompletedCourse> getCoursesbyGrade(Grade g)
    - Return a list of CompletedCourses in which Student earned the specific Grade.

1. Design an Employee class that “is a “ Person class and will include the following ***additional*** data about each Employee

ID, HireDate, DateOfBirth, EmployeeTypeID(enum)

* Implement constructors. Note: MidInitial, PhoneNumber, Major are optional but may be available when setting up a new Employee.
  + HireDate must be > DateOfBirth and must be reasonable (Employee must be at least 18 years of age), EmployeeTypeID must be a valid EmployeeType
* Implement setters and getters
  + EmployeeType may be modified – assuming Employee switch positions
* Implement toString () method

1. Design a Teacher class that “is an” Employee and will include the following ***additional*** data about each Teacher

ID, DepartmentID, SocialSecurityNum, Degree(enum), MajorID(enum), Salary, ArrayList<TaughtCourse>

* Implement constructors. Note: MidInitial, PhoneNumber are optional but may be available when setting up a new Teacher.
  + Salary must be a reasonable amount – you can decide on the range of values that are considered to be reasonable
* Implement setters and getters
  + Degree and MajorID may be modified – assuming Teacher earns an additional higher level degree
  + Salary may be modified, assuming Teacher gets a raise. Salaries are not decreased.
  + Ensure that the Degree and Major are valid values
* Implement toString () method
* Implement the following additional methods
  + void applyRaise(Double percent)
    - modify the current salary by adding salary \* percent to current salary
  + void taughtCourse(Course c, Integer Year, Semester semester, Section sectionID)
    - instantiate a TaughtCourse based on the information provided.
    - Add the TaughtCourse reference to the ArrayList<TaughtCourse>
      * Make sure that this particular TaughtCourse hasn’t already been added to this Teacher’s list of TaughtCourses.
  + int howManyCoursesPerSemester(Integer year, Semester semesterID)
    - return how many courses this Teacher taught during a given semester
  + int howManyDifferentCourses()
    - return how many different Courses this Teacher has taught over time. A Course differs if it has a different CourseID

1. Design a Course class that will include the following data about each Course

CourseID, Description, NumCredits, DepartmentID

* Implement constructor.
  + NumCredits must be a reasonable amount – 0-4 is one possible range of values
* Implement getters
* Implement toString () method
* Implement compareTo and equals methods , two instances of Course are compared based on the CourseID

1. Design a TaughtCourse class that “is a” Course and will include the following ***additional*** data about the TaughtCourse

TeacherID, Year, SemesterID,SectionID

* Implement constructor that will be supplied with the following parameters: Course course, Integer Year, Semester semesterID, Section sectionID
  + Instantiate a TaughtCourse by adding the TeacherID of the given teacher and data provided by the parameters
* Implement getters
* Implement toString () method

1. Design a CompletedCourse class that “is a “ Course and will include the following ***additional*** data about each CompletedCourse

StudentID, Grade, CompletedDate

1. Design a School class that will maintain the following data

SchoolName , Address , PhoneNumber

ArrayList<Person> to store references to instances of Students and Teachers,

ArrayList<Course> to store references to each Course the School may offer

ArrayList<Department> to store references to each Department in the School

* Implement constructors.
  + School(String schoolname, Address address, String PhoneNumber)
  + School(String schoolname, Address address, String phoneNumber, String teachFileName, String studentFileName, String departmentFileName, String courseFileName)
    - Read in the data from each file, instantiate an instance of the appropriate class (Teacher, Student, Department, or Course) and add the instance to the appropriate ArrayList
* Implement setters and getters
  + PhoneNumber may be modified
* Implement toString () method
* Implement the following additional methods
  + addTeacher()
  + addStudent() ensure that no duplicates are added
  + addCourse()
  + addDepartment()
  + removeTeacher()
  + removeStudent()
  + removeCourse()
  + modifyTeacherLastName(Integer teacherID, String newLastName)
  + modifyTeacherAddress(Integer teacherID, Address address)
  + modifyTeacherDegree(Integer teacherID, Degree degree, Major major)
  + giveTeacherRaise(Integer teacherID, Double percent)
  + giveTeacherRaise(Integer teacherID, Integer amount)
  + modifyStudentLastName(Integer studentID, String newLastName)
  + modifyStudentPhoneNumber(Integer studentID, String newPhoneNumber)
  + addCompletedCourse(Integer studentID, Integer courseID, Grade grade)
    - search through ArrayList of Courses and find the courseID , then based on the data you have instantiate a CompletedCourse and add that course to record of the Student with id, studentID.
  + getStudentGPA(Integer studentID)
  + getGradeofCourse(Integer studentID, String courseID)
    - Search through the ArrayList of Persons to find the Student with studentID, then return the Grade earned for completedCourse with given CourseID for student with id, studentID.
  + getCoursesbyDepartment(Integer studentID, String departmentID)
    - return a list of CompletedCourses that were offered by the departmentID and completed by student with id, studentID
  + getCoursesbyGrade(Integer studentID, Grade g)
    - Return a list of CompletedCourses in which Student with id, studentID, earned the specific Grade.
  + getTeachersSortedByName()
    - return a list of Teachers sorted by lastname, firstname
  + getTeachers()
    - return list of Teachers sorted by teacherid
  + getStudents()
    - return list of Students sorted by studentid
  + getStudentsByName()
    - return list of Students sorted by lastname, firstname
  + addTaughtCourse (Integer teacherID, String courseID, Integer year, Semester semester, Section section)
    - search through ArrayList<Person> to find the Teacher with id, teacherID. Then search for the Course in ArrayList<Course> that has courseID. Then invoke the Teacher.taughtCourse sending it the Course with courseID, year, semester, and section.
  + howManyCoursesPerSemester(Integer teacherID, Integer Year, Semester semester)
    - search for the Teacher with teacherID and then invoke the Teacher.howManyCoursesPerSemester to find out how many courses that particular Teacher taught that semester.

1. Implement a class ManageSchool
   1. Implement a static method menu() that will provide the user with a list of choices that will allow him to access all the methods of School outlined above
   2. Implement a main() method that will
      1. Instantiate a School using the data that is provided in the four text files that are included with this project
      2. invoke the appropriate School method and display the results to the user.
2. Test all code before submitting your work.