CSE 215: Programming Language-II

Term Project [Part-02]

Sec: 09
Faculty: Dr. Mohammad Rashedur Rahman (RRn)
Semester: Summer-2020
Posted On: 9 September 2020
Due: 23 September 2020

Lab Instructor

- **Name:** Md. Mustafizur Rahman
- **Email:** rahman.mustafizur@northsouth.edu
- **Office Hours:**
  - ST - 9:00 AM - 11:00 AM
  - ST - 3:00 PM - 04:00 PM

General Instructions:

1. Go through the **Problem**, **Class Descriptions**, **UML Class Diagrams**, and **Methods Descriptions** carefully.
2. The solution that you submit must be your own work. You are expected to write the required methods by yourself, that is, without copying anyone else's coded solution. **Plagiarism will lead you to a straight zero.**
3. **You should follow the provided architecture to solve this problem.**
4. Use proper **naming conventions** for naming fields. Most of the IDE's and code editors do proper indentation automatically. So, please use it.

Problem:

Once the registration for the courses are completed, the university authority wants to provide a facility to generate a bill for that specific student. As the university is in its earliest stages of development, it charges some extra fees for its development purposes. They also charge **BD Tax** @15% of the total amount during the **Summer** semester and **Development Fee** @10% of the total amount for the other semesters. The university authorities also want to have the feature of imposing extra fees (**Development Fee** or **Govt. Tax**) on students' bills each semester. The university not only charges extra fees on students' bills but also gives some discounts based on **Academic Excellence**, **Freedom Fighter**, and **Minority Group**. So the students' billing system software must incorporate the above criteria (Extra Fee and Discount) when generating a bill for a student. **Learn More**

Class Descriptions:

- ❏ **Admin:** The university has an admin who manages the courses offered in a semester and publishes the offered courses list to the students. The admin can increase the seat capacity of a specific course, and can see the status (how many students registered for that course, its seat capacity, etc.) of any of the courses.

- ❑ **Course:** Each course consists of id, title, credit, tuition per credit, number of students, and seat capacity. In order to get each course's fee, you should multiply the tuition per credit by the credit of that course.

- ❑ **CurrentOfferedCourse:** From the pool of academic courses, the university offers different courses each semester. In this class, you are going to store all the offered courses in a semester.

- ❑ **Student:** Each **Student** object consists of name, id, cgpa, and statuses (freedom fighter and minority group) information. If a student belongs to a specific class like the son of a freedom fighter or a particular minority group, then we use the characters 'Y'/'N' to specify those statuses. This **Student** class also uses the **Registration** class as one of its instance variable types to keep track of the **Registration** information of a student. When a student will register for course(s), a **Registration** object will get created and that reference will be assigned to the student's instance variable of **Registration** type. When a student adds or drops a course, his/her **Registration** object gets updated. A student may be eligible for multiple discounts like Academic Excellence (CGPA > 3.5) and Freedom Fighter, in that case, his/her cgpa will determine if he/she is eligible for Academic Excellence discount or not and the rest discount criteria will be determined by the student's statuses. All the applicable discounts will be added to the student's **Registration** object**.**

- ❑ **Registration:** Whenever a student wants to register for the course(s) in a specific semester, a **Registration** object gets created for that student. This object keeps track of all the registration-related information for that student. When a student adds a course, this gets added to the **ArrayList<Course>** courseList of **Registration** class. This class is used to store the registered course(s) of a student. Using the instance of this class, you can get the total amount, which is the fee for the courses a student registered for. Not only to determine the course fees but also this class provides useful methods to get the extra fee and discount amount for that specific instance. This class is also used to store the eligible discount criteria of a student into an **ArrayList<IDiscountStrategy>** applicableDisounts.

- ❑ **DevelopmentFeeCalculator:** This class provides us a means to calculate the development fee which will be added on the student's bill.

- ❑ **BDTaxCalculator:** The **BD Tax** will be calculated by this class and unfortunately the method of this class returns the tax amount in **double.**

- ❑ **BDTaxAdapter:** As your program only works with **integer** types (when it comes to calculating total), the tax amount returned by the method in **BDTaxCalculator** class, has to be converted into the **integer** type. This class has the method to do so.

- ❑ **AcademicExcellenceDiscount:** The university is also generous enough to give discounts to students. As a part of this endeavor, the university offers Academic excellence discounts to students who have CGPA > 3.5. This class has the method to calculate the discount amount which is 20% of the total amount.

- ❑ **FreedomFighterDiscount:** The university offers discounts to students who are the sons of freedom fighters. This class has the method to calculate the discount amount which is 25% of the total amount.

- ❑ **MinorityGroupDiscount:** The university offers discounts to students who belong to the minority group for the smooth continuation of their study. This class has the method to calculate the discount amount which is 10% of the total amount.

ArrayList in Java

In part-01 of your project, you were solving the problem using fixed size of an array. You saw that you had to do some extra work in order to increase the size of the array if that size limit exceeded. But now, you can work more efficiently in terms of using an array in your program. Java **ArrayList** class uses a dynamic array for storing the elements. It is like an array, but there is no size limit. We can add or remove elements anytime. So, it is much more flexible than the traditional array.
You can learn more about it here.

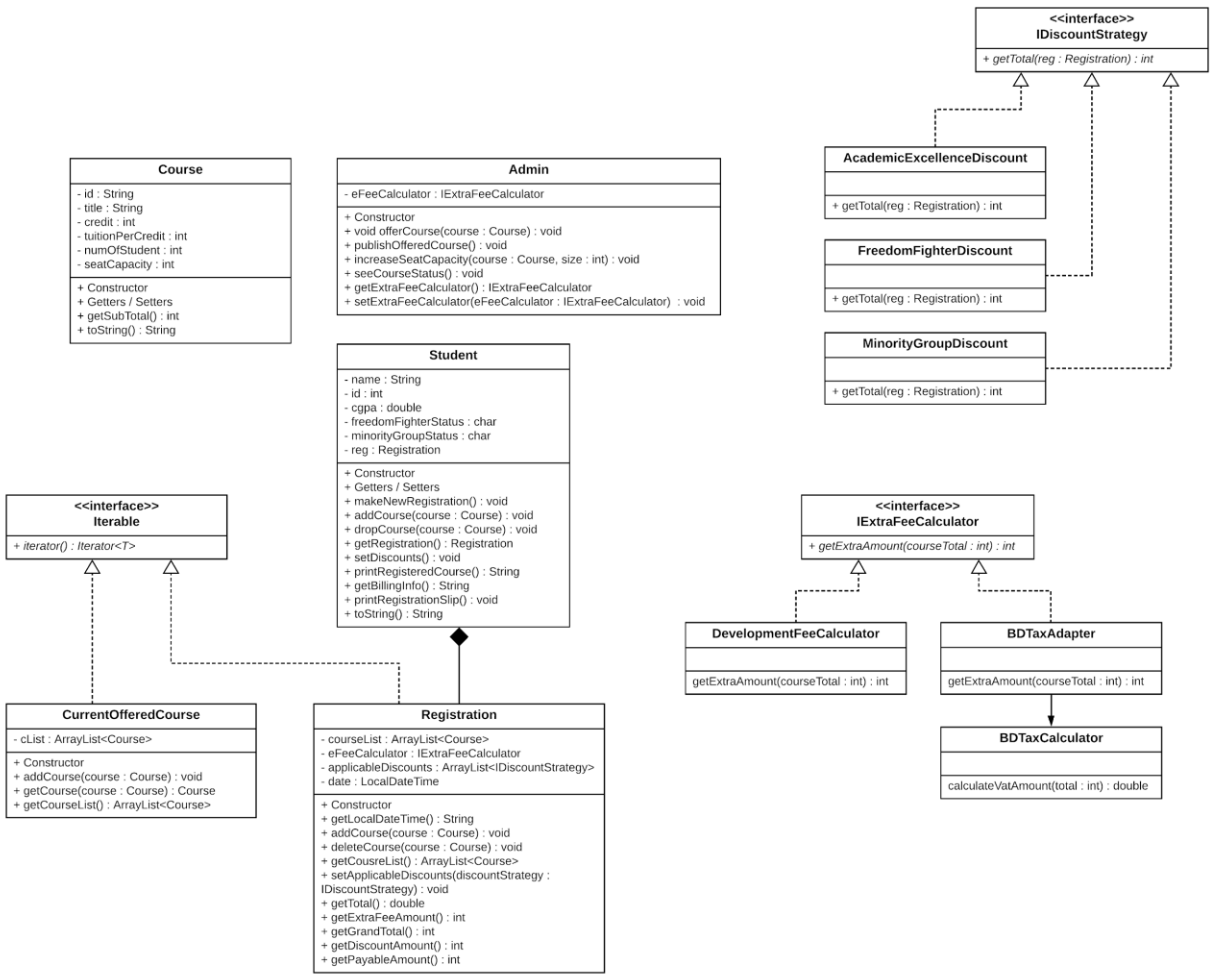
Iterable Interface in Java

The Java Iterable interface represents a collection of objects which is iterable - meaning which can be iterated. This means that a class that implements the Java Iterable interface can have its elements iterated. You can iterate the objects of a Java Iterable in the following ways:

1. by obtaining a Java Iterator from the Iterable
2. by calling the Java Iterable forEach() method.

You can learn more about it here.

UML Class Diagrams:



<<interface>>
IDiscountStrategy
+ getTotal(reg : Registration) : int
AcademicExcellenceDiscount
+ getTotal(reg : Registration) : int
FreedomFighterDiscount
+ getTotal(reg : Registration) : int
MinorityGroupDiscount
+ getTotal(reg : Registration) : int
Course
- id : String
- title : String
- credit : int
- tuitionPerCredit : int
- numOfStudent : int
- seatCapacity : int
+ Constructor
+ Getters / Setters
+ getSubTotal() : int
+ toString() : String
Admin
- eFeeCalculator : IExtraFeeCalculator
+ Constructor
+ void offerCourse(course : Course) : void
+ publishOfferedCourse() : void
+ increaseSeatCapacity(course : Course, size : int) : void
+ seeCourseStatus() : void
+ getExtraFeeCalculator() : IExtraFeeCalculator
+ setExtraFeeCalculator(eFeeCalculator : IExtraFeeCalculator) : void
Student
- name : String
- id : int
- cgpa : double
- freedomFighterStatus : char
- minorityGroupStatus : char
- reg : Registration
+ Constructor
+ Getters / Setters
+ makeNewRegistration() : void
+ addCourse(course : Course) : void
+ dropCourse(course : Course) : void
+ getRegistration() : Registration
+ setDiscounts() : void
+ printRegisteredCourse() : String
+ getBillingInfo() : String
+ printRegistrationSlip() : void
+ toString() : String
<<interface>>
Iterable
+ iterator() : Iterator<T>
<<interface>>
IExtraFeeCalculator
+ getExtraAmount(courseTotal : int) : int
DevelopmentFeeCalculator
getExtraAmount(courseTotal : int) : int
BDTaxAdapter
getExtraAmount(courseTotal : int) : int
BDTaxCalculator
calculateVatAmount(total : int) : double
CurrentOfferedCourse
- cList : ArrayList<Course>
+ Constructor
+ addCourse(course : Course) : void
+ getCourse(course : Course) : Course
+ getCourseList() : ArrayList<Course>
Registration
- courseList : ArrayList<Course>
- eFeeCalculator : IExtraFeeCalculator
- applicableDiscounts : ArrayList<IDiscountStrategy>
- date : LocalDateTime
+ Constructor
+ getLocalDateTime() : String
+ addCourse(course : Course) : void
+ deleteCourse(course : Course) : void
+ getCousreList() : ArrayList<Course>
+ setApplicableDiscounts(discountStrategy : IDiscountStrategy) : void
+ getTotal() : double
+ getExtraFeeAmount() : int
+ getGrandTotal() : int
+ getDiscountAmount() : int
+ getPayableAmount() : int

Figure: UML Class Diagram (**Better View**)

Some Important Methods Descriptions:

Class: Course	
getSubTotal() : int	This method returns a specific course's fee. Sub Total = Credit * Tuition Per Credit

Interface: Iterable
This is a default interface of java and by implementing this interface you can make your objects to be the target of the for-each loop statement. **You can learn more about it here**.

Class: CurrentOfferedCourse	
It implements the **Iterable** interface.	
addCourse(course : Course) : void	This method will take a **Course** object as a parameter and will add it to the ArrayList.
getCourse(course : Course) : Course	This method returns a **Course** object if it is offered in a semester.
getCourseList() : ArrayList<Course>	This method will return an ArrayList<Course> containing all the offered courses in a semester.

Class: Student	
makeNewRegistration() : void	Once a student starts his/her course registrati on process for a semester, an instance of **Registration** gets created for that student. So, this method creates a **Registration** object for a student.
addCourse(course : Course) : void	A student will be able to add a course by this method. All the registration related tasks of a student are handled by the **Registration** class. When a course is added by a student, it is basically added to the **ArrayList<Course>** in the **Registration** class. Make sure to check that the student is eligible for adding a course because there is a credit limit for students depending on CGPA and increase the number of students for that course. **Hint:** Call the addCourse() method of **Registration** class from this method.
dropCourse(course : Course) : void	It is very much similar to addCourse(course :Course) method. Here, you need to decrease the number of students for that course and delete that course from the **ArrayList<Course>**.
getRegistration() : Registration	This method will return a **Registration** object of a student which was created during makeRegistration() method call.

setDiscounts() : void	It will set different discounts applicable for a student. In the **Registration** class, an **ArrayList<IDiscountStrategy>** is there for holding different instances of discount classes. So, in this method, you will be checking which discounts can be applied to a student and pass the corresponding discount class's instance to the setApplicableDiscount s (discountStrategy:IDiscountStrategy) metho d of **Registration** class.
	
	Hint:
	if(cgpa > 3.5) {
	reg.setApplicableDiscounts(new AcademicEx cellenceDiscount());
	}
	if(freedomFighterStatus == 'Y') {
	reg.setApplicableDiscounts(new FreedomFig hterDiscount());
	}

printRegisteredCourse() : String	It will return the **course id** and **course title** of all the registered courses of a student.
getBillingInfo() : String	It will return the breakdown of the bill. Ex: Total Course Fee: Extra Fee: Discount: Payable Amount:
printRegistrationSlip() : void	This method prints all the basic information of a student including the billing info and courses registered for.
toString() : String	It returns the basic information of a student.

Class: Registration	
It implements the **Iterable** interface.	
getLocalDateTime() : String	It returns the date and time of registration for a student. **Learn More**
addCourse(course : Course):void	A Student will want to register for the course(s) in a specific semester. This method will add a course to a student's course list.
deleteCourse(course : Course):void	This method will remove a specific course from the student's course list.

getCourseList() : ArrayList<Course>	This method will return an **ArrayList<Course>** containing all the courses a student registered for in a semester.
setApplicableDiscounts(discountStrategy:IDiscountStrategy) : void	You know that the University applies different discounts to students. This method is used to add instances of discount classes to the **ArrayList<IDiscountStrategy>** applicableDiscounts.
getTotal() : double	It will return a total amount based on a student's registered courses. Each course has a fee depending on its credit and tuition per credit. So, the amount is calculated by adding all the registered course fees.

getExtraFeeAmount() : int	This method will return the extra fee that will be applied by the University depending on a specific semester. This method will call the getExtraFeeCalculator() of **Admin** class and that method will return the instance of **DevelopmentFeeCalculator** or **BDTaxAdapter** depending on which was set by the admin. By using this instance, we can call the getExtraAmount(courseTotal : int) method to get the fee.
	
	Hint:
	IExtraFeeCalculator eFeeCalculator = Admin object.getExtraFeeCalculator();
	
	return eFeeCalculator.getExtraAmount((int)(getTotal()));

getGrandTotal() : int	It will return the grand total amount for a specific student. Grand Total = Courses Fees + Extra Fee

getDiscountAmount() : int	This method returns the discount amount for a student based on his/her credentials. A student may be eligible for multiple discounts, and in that case, the university will apply the one which is the greatest.
	
	Hints:
	From the **ArrayList<IDiscountStrategy>** applicableDiscounts, get all the instances of the discount classes applied for a student. For each of the instances of discount class in the **ArrayList<IDiscountStrategy>**, calculate the discount amount and store it in the corresponding variable like the following.
	int academicExcellenceDiscount = 0;
	int freedomFighterDiscount = 0;

	int minorityGroupDiscount = 0; Now return the maximum value of the three.

Interface: IExtraFeeCalculator
This interface has an abstract method which will be overridden by the implementing classes.

Class: DevelopmentFeeCalculator	
It implements the **IExtraFeeCalculator** interface.	
getExtraAmount(courseTotal : int)	This is an overridden method from the interface **IExtraFeeCalculator.** It will return the development fee amount based on the total course fees of a student. Development Fee = Course Fees * 0.10 which is 10% of the total course fees amount.

Class: BDTaxAdapter	
It implements the **IExtraFeeCalculator** interface.	
getExtraAmount(course total : int)	This is an overridden method from the interface **IExtraFeeCalculator.** It will return the BD tax fee amount based on the total course fees of a student. This method calls another method calculateVatAmount(total:int) of the **BDTaxCalculator** class to get the tax amount. This third-party tax calculator returns the tax amount in **double** but we need it to be an **integer** value. So, the adapter class provides a method to convert and returns the tax amount in **integer**.

Class: BDTaxCalculator	
calculateVatAmount(total : int) : double	It will return the tax amount calculated for a specific student. BD Tax = Course Fees * 0.15 which is 15% of the total course fees.

Interface: IDiscountStrategy
This interface has an abstract method which will be overridden by the implementing classes.

Class: AcademicExcellenceDiscount	
It implements the **IDiscountStrategy** interface.	
getTotal(reg : Registration) : int	It will return the discount amount calculated for a specific student. Discount = Course Fees * 0.20 which is 20% of the total course fees.

Class: FreedomFighterDiscount	
It implements the **IDiscountStrategy** interface.	
getTotal(reg : Registration) : int	It will return the discount amount calculated for a specific student. Discount = Course Fees * 0.25 which is 25% of the total course fees.

Class: MinorityGroupDiscount	
It implements the **IDiscountStrategy** interface.	
getTotal(reg : Registration) : int	It will return the discount amount calculated for

	a specific student. Discount = Course Fees * 0.10 which is 10% of the total course fees.

Class : Admin	
offerCourse(course : Course) : void	This method will add a course in the current offered courses list.
publishOfferedCourse() : void	It will print all the courses offered in a semester with course id.
increaseSeatCapacity(course : Course, size : int) : void	This method will increase the seat capacity of a specified course by that size.
seeCourseStatus() : void	It will print all the offered courses with course id, number of students enrolled in and seat capacity of that course.
setExtraFeeCalculator(eFeeCalculator IExtraFeeCalculator) : void	Either the instance of **DevelopmentFeeCalculator** or **BDTaxAdapter** will be passed as a parameter for this method. This method will set the value of eFeeCalculator.
getExtraFeeCalculator() : IExtraFeeCalculator	It will return the instance of **DevelopmentFeeCalculator** or **BDTaxAdapter**.

Tasks:

Objects
Link: Objects.txt Follow the above link. These are the test objects you will be working with for all the tasks.

Sl.	Tasks	Expected output

1	Print the billing info for s1.
	Call the printBillingInfo() method on s1.
		```
		Billing Info: (ID: 1631728042)
		--------------------------------------------
		Total Course Fees: 36000
		Extra Fee:          +    3600
		----------------------------------------
		Grand Total:        39600
		Discount:          --  9000
		----------------------------------------
		Payable Amount:  30600
		```

2.	Print the billing info for s2. Call the printBillingInfo() method on s2.	Billing Info: (ID: 1821347042) -- Total Course Fees: 54000 Extra Fee: + 5400 -- Grand Total: 59400 Discount: -- 5400 --- Payable Amount: 54000
3.	Call the setExtraFeeCalculator() method of the **Admin** class and pass new BDTaxAdapter() as the parameter.	
4.	Print the billing info for s3. Call the printBillingInfo() method on s3	Billing Info: (ID: 2021746042) -- Total Course Fees: 90000 Extra Fee: + 13500 -- Grand Total: 103500 Discount: -- 18000 --- Payable Amount: 85500
5.	Print the billing info for s4. Call the printBillingInfo() method on s4	Billing Info: (ID: 1923147042) -- Total Course Fees: 36000 Extra Fee: + 5400 --- Grand Total: 41400 Discount: -- 7200 --- Payable Amount: 34200
6.	Print the Registration slip for s5. Call the printRegistrationSlip() method on s5.	Registration Time: 2020-08-25 01:49:03 -- Name: Mahmudul Hoque, ID: 1524137042, CGPA: 2.14 -- Course Id: Course Title: ======================================= Billing Info: (ID: 1524137042) -- Total Course Fees: 72000 Extra Fee: + 10800 --- Grand Total: 82800 Discount: -- 18000 --- Payable Amount: 64800