# Design Document

for

# Billing Interface Linked Library B.I.L.L.

Version 1.0

Prepared by Heiru Wu, Hassan Alamri, & Rick Stroud

CSCE – Software Engineering

5 November 2017

## Table of Contents

## Revision History

| Name | Date | Reason For Changes | Version |
|---|---|---|---|
| **Document Creation** | **20-Oct-17** | **n.a.** | **1.0** |

# 1 Introduction

This document provides a detailed overview of the design of the B.I.L.L. system and is intended primarily for the system developer to bridge the gap from requirements to implementation.

## 1.1 Purpose

The primary purpose of this document it to provide a design for the B.I.L.L system detailed to the extent that the system developer may implement the application in a manner that satisfies the requirements.

Design decisions are both documented and justified to allow a comprehensive understanding to the system developer as to what to do, the reasons for the chosen path, and the trade-offs involved.

Additional audiences who will benefit from this document are the software tester to identify boundary conditions, and the project manager to allocate resources and to plan development tasks.

## 1.2 System Overview

At the highest level the Billing Interface Linked Library delivers functionality which allows students to view billing information about their current semester and to make payments against their outstanding balance while administrative users may perform the same actions on-behalf-of students who fall under their managerial control.

The scope of capabilities of a student include making a payment and viewing the components that comprise the billing information which are charges, payments, refunds, and any balances from past semesters. Note, the in-scope function of "making a payment" entails only the recording of the payment and does not include any payment processing; actual payment processing is an out-of-scope activity.

Regarding the semester charges, the BILL system is responsible for determining which of a set of standard university fees are applicable for each student as well as the dollar amount of the fee for each student. For example, determining if the Cohort Study Abroad Fee of $300 is to be charged to a student, likewise, determining if the Technology Fee charged to every student should have an amount of $200 for full-time students or the $17 amount for part-time students.

This dynamic pricing of charges places in-scope the activities of managing & maintaining student profiles as they will impact the charges incurred and to this end, the student may view their profile and to a limited extent make modifications.

Administrative users possess all of the capabilities of a student user provided the system has determined the administrator has privilege to function on-behalf-of the student. Administrative users also possess the ability to issue refunds towards a student's account and may modify a student's profile to a greater degree than is permitted to the student.

To ensure that access to the BILL system is only granted to known users and that users are given only the student or administrative permission level they are entitled to, the system is also responsible for providing log-in & log-out functionality. However, the BILL system is not responsible for the actual authentication of the user credentials. Rather, the log-in request assumes the user has already been authenticated and will simply verify if the requesting user is contained in a list of registered users and if so record the user as "logged in" until such time the user requests to log out. All other requests must verify first if the requesting user is logged in.

All of the aforementioned functionality is provided to the users via function calls to a dynamic link library.

## 1.3 Design Objectives

Two overall system design goals exist which this document seeks to address. First, to provide a comprehensive design such that the functional and non-functional requirements may be satisfied. Second that beyond satisfying the requirements that the design anticipates changes in the future and affords the ability, through both extensible abstractions and sound justifications for the decisions made, to accommodate these changes.

### 1.3.1 Functional Requirements

The functional requirements which must be addressed by the design are summarized in the BILL Requirements document in section 1.3 Project Scope, and section 2.2 Product Functions, and itemized in detail in section 3 Functional Features of the same document. To restate, the functional requirements are fairly compact and may be grouped into 5 succinct categories and 9 discrete customer facing functions, as shown below.

| Category | Function | |
|---|---|---|
| **1 - Session Management** | | |
| | 1 | Log In |
| | 2 | Log Out |
| **2 - Student Profile Management** | | |
| | 3 | View Profile |
| | 4 | Update Profile |
| **3 - Viewing Payment Related Information** | | |
| | 5 | View Account Balance |
| | 6 | View Charges |
| | 7 | View Payments |
| **4 - Modifying Payment Related Information** | | |
| | 8 | Make Payment |
| **5 - Supporting Functions** | | |
| | 9 | Check On-Behalf-Of Privilege |

Of note, two areas of functionality not in scope of the design is the authentication of user credentials and the actual making of payments with a financial gateway. Users requesting to create or end a session with the BILL system will have already been authenticated when the login & logout requests are made. Similarly, any payments made to the system, or in the case of administrative users' refunds made, are only recorded by the system as having been made and the associated balances updated accordingly.

### 1.3.2 Nonfunctional Requirements

In Section 4 Other Nonfunctional Requirements of the BILL Requirements document, only security requirements are discretely listed as to a large extent these relate to other functional requirements listed in the document. This includes the maintenance of user sessions, authorization and role based access, the need to maintain confidentiality of information, and the requirement to maintain an audit stream. See section 4.2 Security Requirements of the BILL Requirements document for more detail.

As yet, no further specific nonfunctional requirements for performance, in the way of defined response times, or availability, in the way of defined system up-time, have been stated.

For other nonfunctional requirements, it is incumbent upon any design to also support nonfunctional requirements which will allow the design to be understandable, verifiable, and maintainable in the face of expected but unknown future change.

For these reasons it is a design goal that the system be abstracted using techniques of encapsulation and information hiding, and that components promote strong cohesion and are loosely coupled with other components. And lastly, that the design provides a clear flow of common scenarios to demonstrate traceability as to the satisfaction of all requirements.

### 1.4 References

| # | Item | Detail |
|---|---|---|
| 1 | BILL Requirements | The BILL Requirements Specification |
| 2 | BILL Use-Cases | The BILL Use-Case document |
| 3 | Current Semester Charges | The list of charges which may be applicable to students for the current semester and the amount which the student will be charged.<br>http://sc.edu/bursar/fees.shtml |

### 1.5 Definitions, Acronyms, and Abbreviations

| # | Item | Detail |
|---|---|---|
| 1 | API | Application Programming interface, a method or function exposed by the system which may be accessed via code, i.e. programmatically. |
| 2 | BILL or B.I.L.L. | The Billing Interface Linked Library, i.e. the product, system, or solution described by this document. |

# 2 Design Overview

### 2.1 Introduction

An object-oriented design approach is used for the BILL system with objects segregated into a core stack of four primary layers supported by adjacent supporting non-core layers.

### 2.2 Environment Overview

The BILL is developing in Java platform version 8 as a library which will be imported as a dependency in any external system.

### 2.3 System Architecture

The primary four layers of the BILL system architecture provide and group abstractions beginning with the application interface, proceeding through business logic, and concluding with two data centric layers for of data access control and persistence. Supporting this core stack are adjacent layers used to map incoming data from external systems and to provide infrastructure support.

Beginning with the core stack, at the top an interface layer isolates system access and presentation from business logic and data storage. Moving down the stack, a business layer next encapsulates domain logic and is itself modularized into three separate services to deliver the necessary functionality described in section 1.3.1 of this document. Concluding the stack, layers to provide management of data is next separated into two layers, one for providing data access control and a second to management data persistence.

Supporting the core stack are two additional layers. First a data transformation object layer which maps data from external systems in a manner which can be used by the core stack which in turn isolates the core stack from changes in external data representation. Second, an infrastructure component layer abstracts common features necessary to the core stack for providing security, logging, and exception handling. These infrastructure components in turn allow the core stack to be more focused on delivering their primary abstractions and relieves the implementation of the redundancy that separate delivery of these features would require.

The six layers of the BILL system architecture are depicted in Figure 1, below.

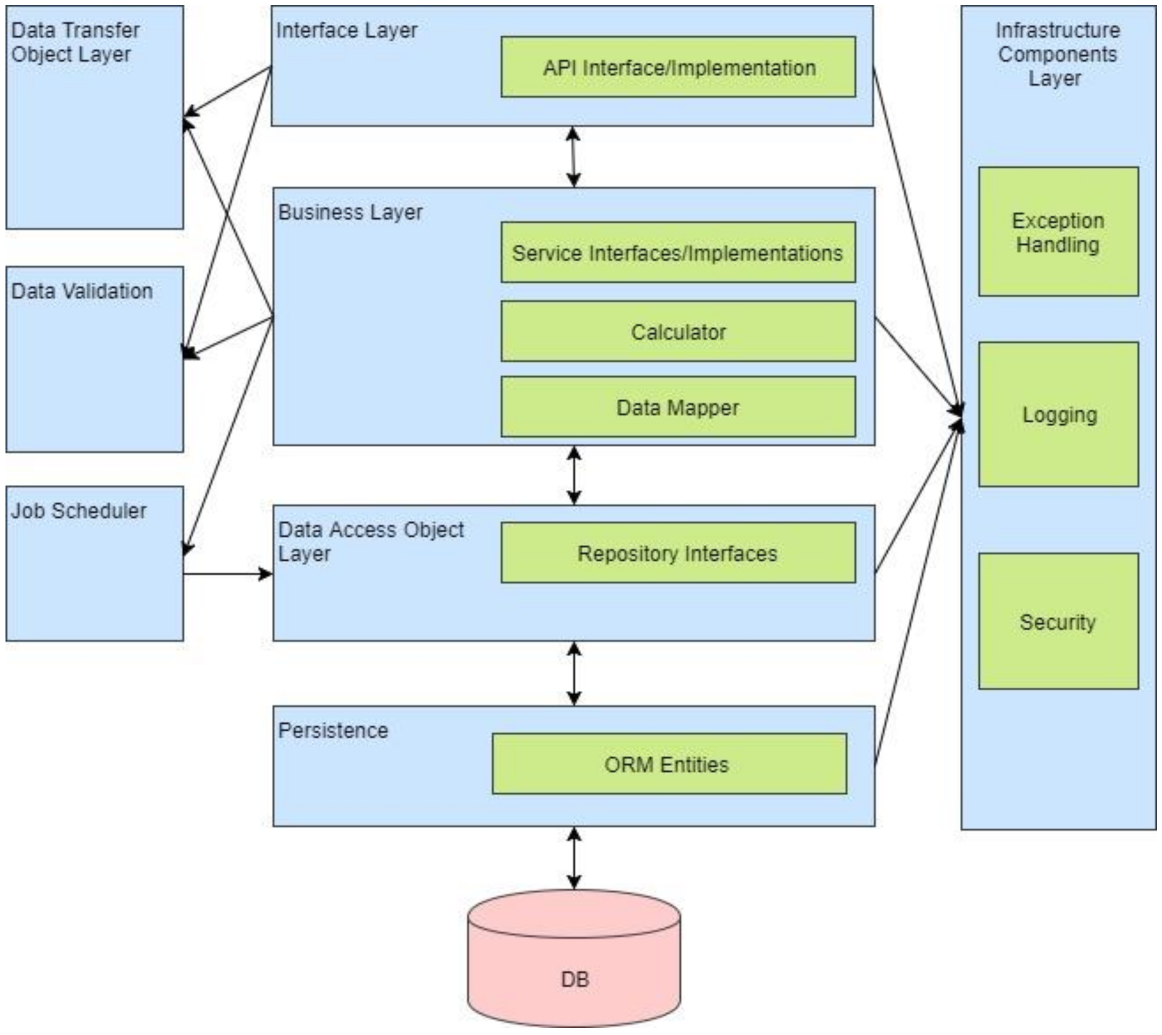

Figure 1 – Bill Layer Overview

### 2.3.1 Interface Layer

The Interface Layer is the first and only point of contact for any user requests to the BILL system and has a goal of limiting and defining all client access to the BILL system. This is illustrated in Section 5 – Dynamic Model of this document by showing all client requests being directed to constructs residing in the Interface Layer.

While a graphical user interface is out of scope for the BILL system, only an API is provided, by defining a separate Interface layer this provides a bulwark against any requirements or changes in future for presentation or access needs

The required Java implementation for the BILL system also uniquely supports the interface layer via the Java construct of Interfaces. An Interfaces allows for the method signatures of objects to be defined separately from the implementation of the methods, which is provided in the classes, and this ensures the overall goal of isolation which is the intent of the interface layer.

### 2.3.2 Business Layer

The Business Layer contains the domain logic of the BILL system. Notice the Business Layer is sandwiched, and isolated, between the layers used for presentation (above) and data storage (below). This allows the business layer to be free from requirements unique to these areas and to focus on delivering modular components solely to capture the domain logic.

The business layer itself is further modularized by providing three services which in turn will provide the functionality of the five required areas listed in section 1.3.1 of this document, which are defined.

These services and their functional areas of responsibility are:

- The **User Management Service** - Defines functionality for loading users into the system and establishing their available role, session management, and determining on-behalf-of privileges for user making requests of the BILL system. These are categories 1 & 5 of the required functionality.

- The **Student Profile Service** - Defines functionality for loading student profiles into the system, accessing student profiles, and updating student profiles within the constraints of the business logic. This is category 2 of the required functionality.

- The **Bill Service** - Defines all functionality for accessing the components of a student's bill, including current amount, past & present charges, and past & present payments. Additionally, the Bill Service defines functionality for users to make payments and refunds. These are categories 3 & 4 of the required functionality.

The access to the Business Layer and the isolation of these three services and associated freedom from direct user requests and data storage is depicted in Section 5 – Dynamic Model of this document by requests to the services originating from Interface Layer objects and requests being made to the repository objects of the data access layer.

### 2.3.3 Data Access Object Layer

The Data Access Object Layer contains a set of repository objects. The purpose of these repositories it to facilitate storage and retrieval of the items in the repositories. For example, the Payment History Repository provides the functionality only to store & retrieve payments. This isolates the storage of such objects from their domain logic and utilization. In the prior example of the Payment History Repository, this isolates the use of the contained payments in the calculation of the bill as this functionality remains in the Business Layer.

### 2.3.4 Persistence Layer

The Persistence Layer contains only the object relationship mapping of the repository items from the Data Access Object Layer to a true relational database.

#### 2.3.5 Data Transfer Object Layer

The Data Transfer Object Layer is the conduit for data which is fed into the BILL system on a periodic basis and provides abstractions for this purpose. Currently, this contains constructs defined as the “Bill of Materials” and is comprised of user and student profile records.

#### 2.3.6 Infrastructure Component Layer

The Infrastructure Component Layer is a supporting abstraction in the areas of exception handling, security, and logging.

#### 2.3.7 Data Validation Component

The data validation is a crucial component that is responsible for checking the inputs from the external interfaces to ensure the validity and integrity of data provided

#### 2.3.8 Job Scheduler Component

The Job Scheduler is a configurable component that is supported by Spring Framework. It’s responsible for running and executing scheduled tasks periodically in order gather the charges-related data by end of semester and dump into the database

# 3 Interfaces and Data Stores

## 3.1 System Interfaces

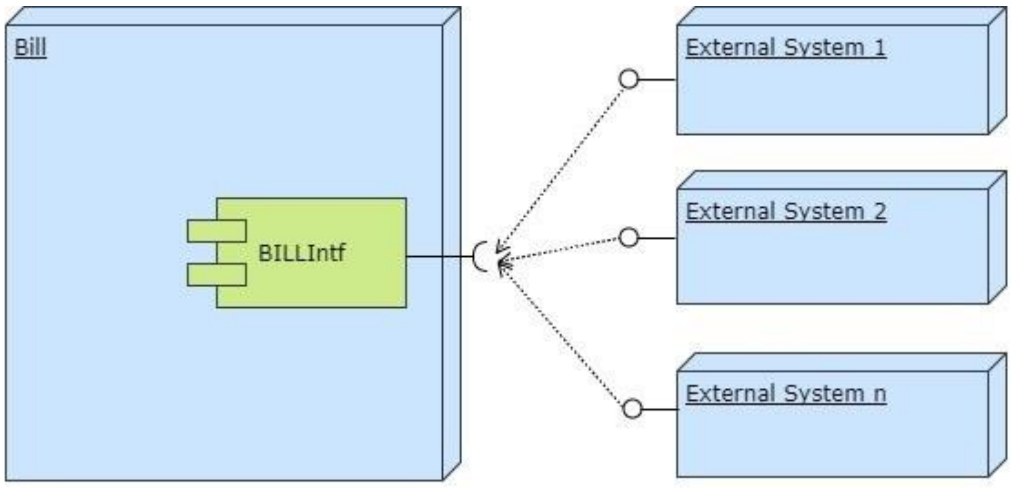

Figure 2 – Bill Interface Overview

As a linked library, the BILL system does not provide a graphical user interface. All external access to the BILL system is provided only via API calls. This access includes both end user requests to request the functionality of the BILL system as defined in section 1.3.1, as well as access from supporting systems that provide data to the BILL system so that it can provide this functionality.

All supported APIs are exposed as part of the architectural interface layer, see section 2.3.1 Interface Layer. It is the responsibility of the Interface Layer to limit access to the domain logic below as defined in the Business Layer.

Conformity to the definition of the BILL APIs is assured by the Java construct of an Interface, with provides a layer of separate to the actual implementation of the BILL methods. This in turn allows for systems interacting with the BILL system to program to this isolated interface, rather than to the implementation, and removes any implementation specific dependencies from the external systems.

See section 4.2.4 API Package for the supporting Interface and Implementation classes for the BILL system interface.

## 3.2 Data Stores

For data storage the BILL system utilizes Spring JPA framework, the Java Persistence API, and the H2 embedded, in-memory, database.

JPA was selected to relieve the developer from the load of object relationship mappings which would be required if only using previous technologies such as JDBC were used. This provides both a time saver to the developer and reduces the chance of developer error versus manually mapping objects to tables. Together this provides both more rapid and less error prone development of the persistence layer.

Likewise, the H2 database was chosen for its ability to accelerate development and reduce development errors. H2 provides an open-source relational database with integration to Java and the available database console of H2 assists the developer during both implementation and unit testing.

# 4 Structural Design

## 4.1 Package Diagram

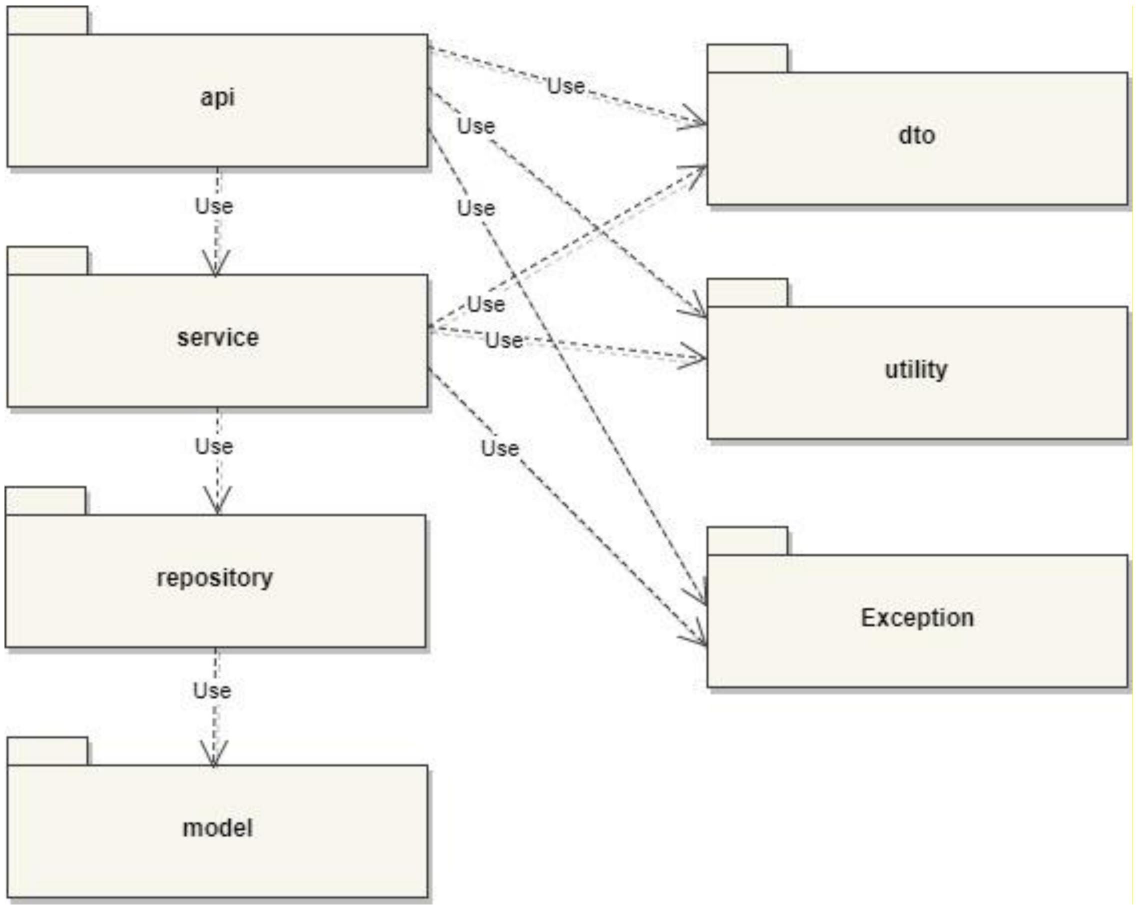

Figure 4 – Package Diagram

## 4.2 Class Diagram

### 4.2.1 Class Diagram in Model Package

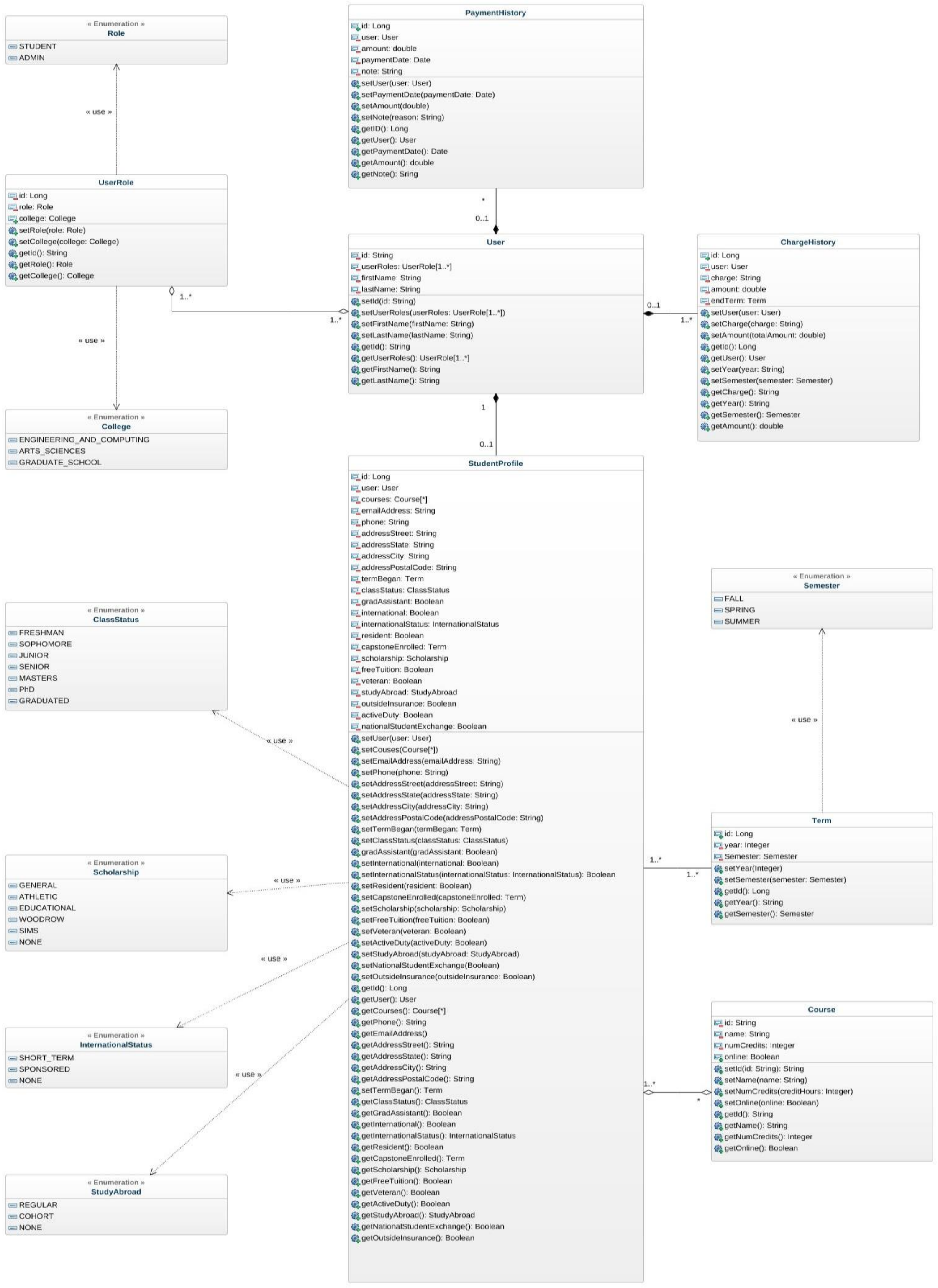

Figure 5 – Model Package Class Diagram

### 4.2.2 Class Diagram in Repository Package

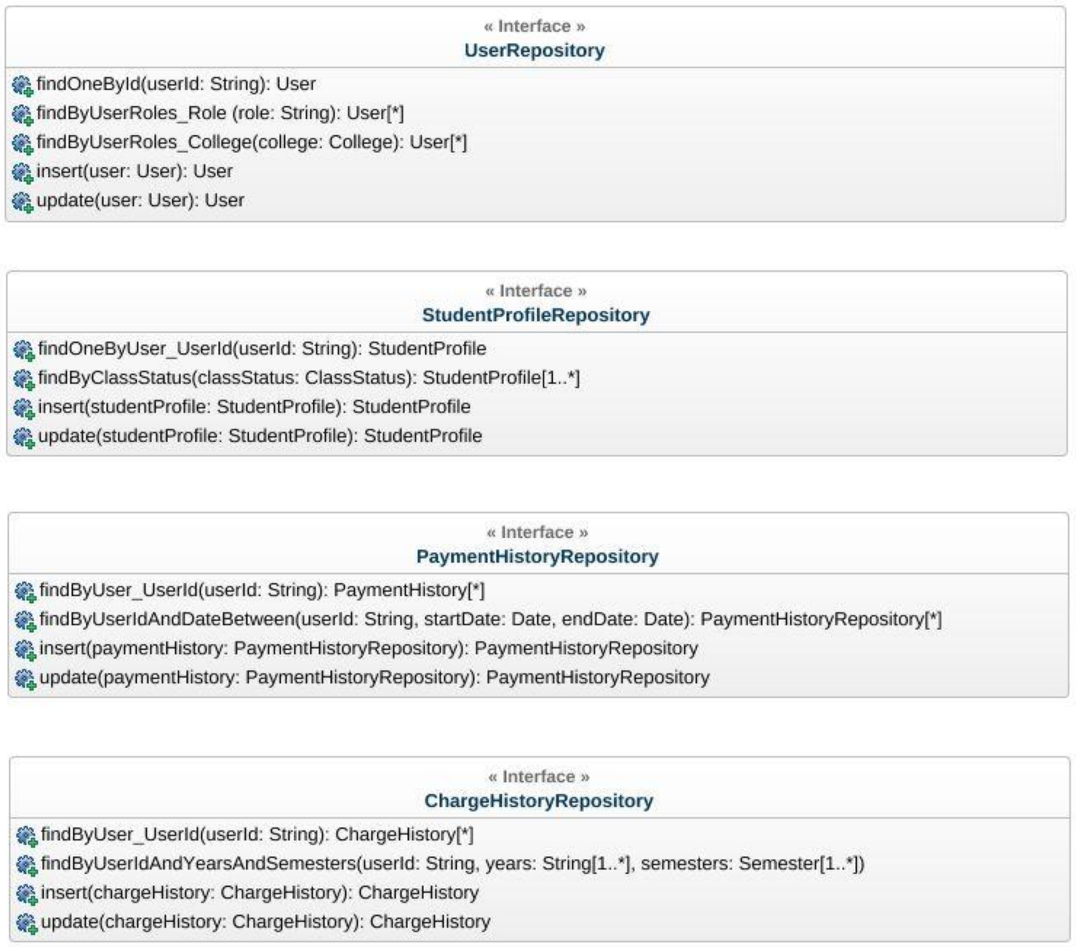
« Interface »
UserRepository
findOneById(userId: String): User
findByUserRoles_Role (role: String): User[*]
findByUserRoles_College(college: College): User[*]
insert(user: User): User
update(user: User): User
« Interface »
StudentProfileRepository
findOneByUser_UserId(userId: String): StudentProfile
findByClassStatus(classStatus: ClassStatus): StudentProfile[1..*]
insert(studentProfile: StudentProfile): StudentProfile
update(studentProfile: StudentProfile): StudentProfile
« Interface »
PaymentHistoryRepository
findByUser_UserId(userId: String): PaymentHistory[*]
findByUserIdAndDateBetween(userId: String, startDate: Date, endDate: Date): PaymentHistoryRepository[*]
insert(paymentHistory: PaymentHistoryRepository): PaymentHistoryRepository
update(paymentHistory: PaymentHistoryRepository): PaymentHistoryRepository
« Interface »
ChargeHistoryRepository
findByUser_UserId(userId: String): ChargeHistory[*]
findByUserIdAndYearsAndSemesters(userId: String, years: String[1..*], semesters: Semester[1..*])
insert(chargeHistory: ChargeHistory): ChargeHistory
update(chargeHistory: ChargeHistory): ChargeHistory

Figure 6 – Repository Package Class Diagram

### 4.2.3 Class Diagram in the Service Package

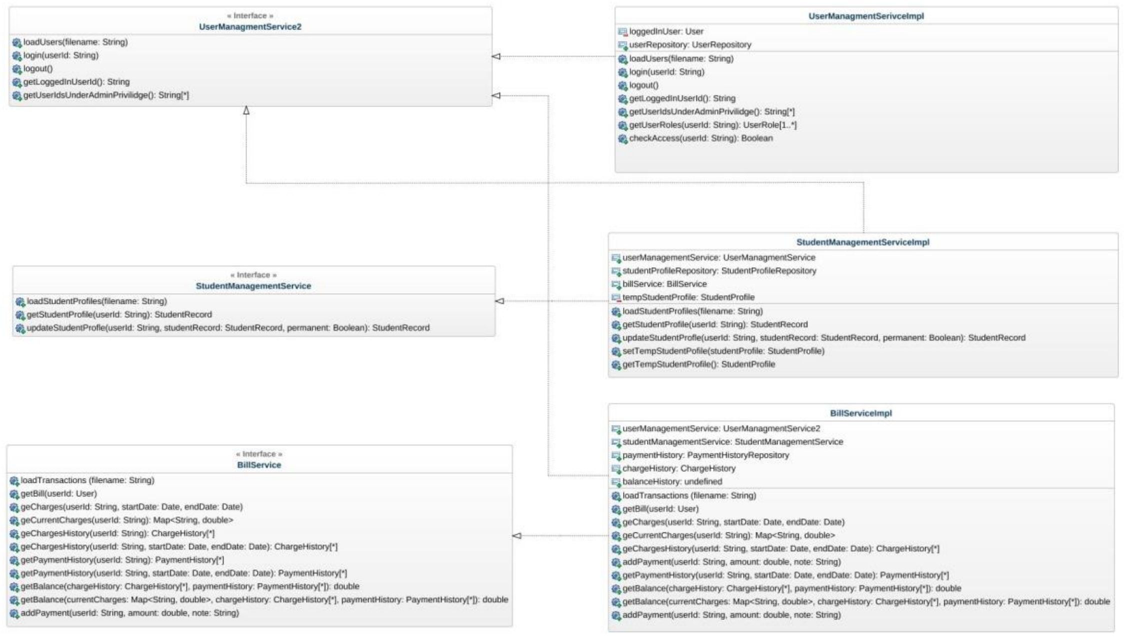
« Interface »
UserManagmentService2
loadUsers(filename: String)
login(userId: String)
logout()
getLoggedInUserId(): String
getUserIdsUnderAdminPrivilidge(): String[*]
UserManagmentSerivceImpl
loggedInUser: User
userRepository: UserRepository
loadUsers(filename: String)
login(userId: String)
logout()
getLoggedInUserId(): String
getUserIdsUnderAdminPrivilidge(): String[*]
getUserRoles(userId: String): UserRole[1..*]
checkAccess(userId: String): Boolean
« Interface »
StudentManagementService
loadStudentProfiles(filename: String)
getStudentProfile(userId: String): StudentRecord
updateStudentProfle(userId: String, studentRecord: StudentRecord, permanent: Boolean): StudentRecord
StudentManagementServiceImpl
userManagementService: UserManagmentService
studentProfileRepository: StudentProfileRepository
billService: BillService
tempStudentProfile: StudentProfile
loadStudentProfiles(filename: String)
getStudentProfile(userId: String): StudentRecord
updateStudentProfle(userId: String, studentRecord: StudentRecord, permanent: Boolean): StudentRecord
setTempStudentPofile(studentProfile: StudentProfile)
getTempStudentProfile(): StudentProfile
« Interface »
BillService
loadTransactions (filename: String)
getBill(userId: User)
geCharges(userId: String, startDate: Date, endDate: Date)
geCurrentCharges(userId: String): Map<String, double>
geChargesHistory(userId: String): ChargeHistory[*]
geChargesHistory(userId: String, startDate: Date, endDate: Date): ChargeHistory[*]
getPaymentHistory(userId: String): PaymentHistory[*]
getPaymentHistory(userId: String, startDate: Date, endDate: Date): PaymentHistory[*]
getBalance(chargeHistory: ChargeHistory[*], paymentHistory: PaymentHistory[*]): double
getBalance(currentCharges: Map<String, double>, chargeHistory: ChargeHistory[*], paymentHistory: PaymentHistory[*]): double
addPayment(userId: String, amount: double, note: String)
BillServiceImpl
userManagementService: UserManagmentService2
studentManagementService: StudentManagementService
paymentHistory: PaymentHistoryRepository
chargeHistory: ChargeHistory
balanceHistory: undefined
loadTransactions (filename: String)
getBill(userId: User)
geCharges(userId: String, startDate: Date, endDate: Date)
geCurrentCharges(userId: String): Map<String, double>
geChargesHistory(userId: String, startDate: Date, endDate: Date): ChargeHistory[*]
addPayment(userId: String, amount: double, note: String)
getPaymentHistory(userId: String, startDate: Date, endDate: Date): PaymentHistory[*]
getBalance(chargeHistory: ChargeHistory[*], paymentHistory: PaymentHistory[*]): double
getBalance(currentCharges: Map<String, double>, chargeHistory: ChargeHistory[*], paymentHistory: PaymentHistory[*]): double
addPayment(userId: String, amount: double, note: String)

Figure 7 – Service Package Class Diagram

### 4.2.4 Class Diagram in the API Package

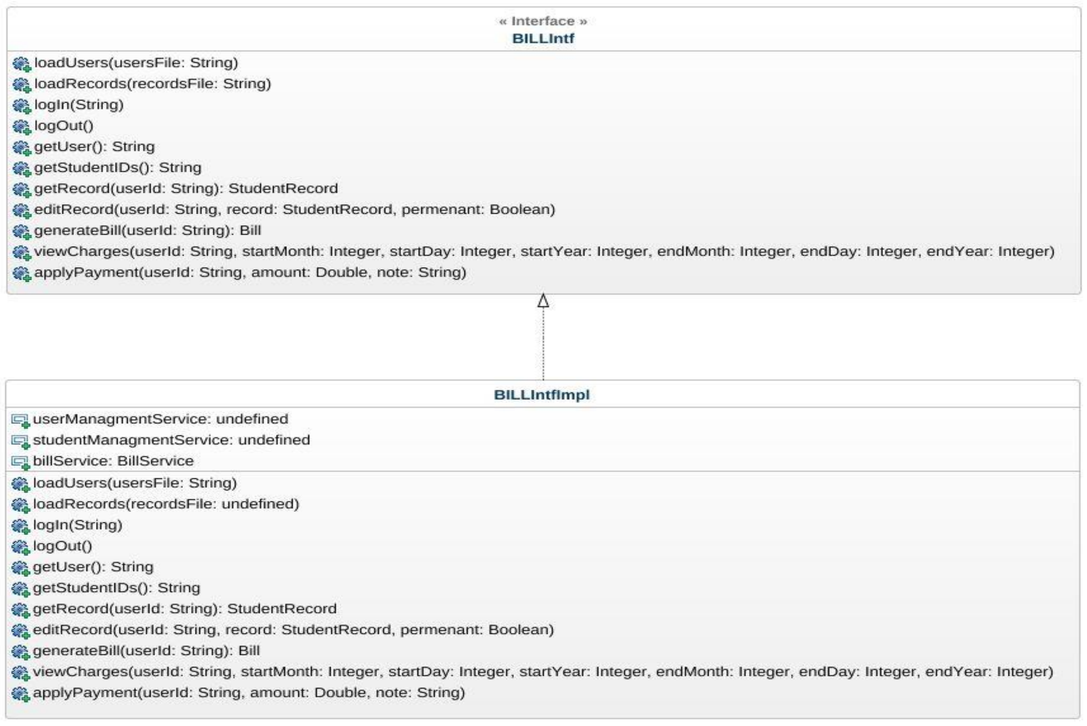
« Interface »
BILLIntf
loadUsers(usersFile: String)
loadRecords(recordsFile: String)
logIn(String)
logOut()
getUser(): String
getStudentIDs(): String
getRecord(userId: String): StudentRecord
editRecord(userId: String, record: StudentRecord, permenant: Boolean)
generateBill(userId: String): Bill
viewCharges(userId: String, startMonth: Integer, startDay: Integer, startYear: Integer, endMonth: Integer, endDay: Integer, endYear: Integer)
applyPayment(userId: String, amount: Double, note: String)
BILLIntfImpl
userManagmentService: undefined
studentManagmentService: undefined
billService: BillService
loadUsers(usersFile: String)
loadRecords(recordsFile: undefined)
logIn(String)
logOut()
getUser(): String
getStudentIDs(): String
getRecord(userId: String): StudentRecord
editRecord(userId: String, record: StudentRecord, permenant: Boolean)
generateBill(userId: String): Bill
viewCharges(userId: String, startMonth: Integer, startDay: Integer, startYear: Integer, endMonth: Integer, endDay: Integer, endYear: Integer)
applyPayment(userId: String, amount: Double, note: String)

Figure 8 – API Package Class Diagram

### 4.2.5 Class Diagram in the DTO Package

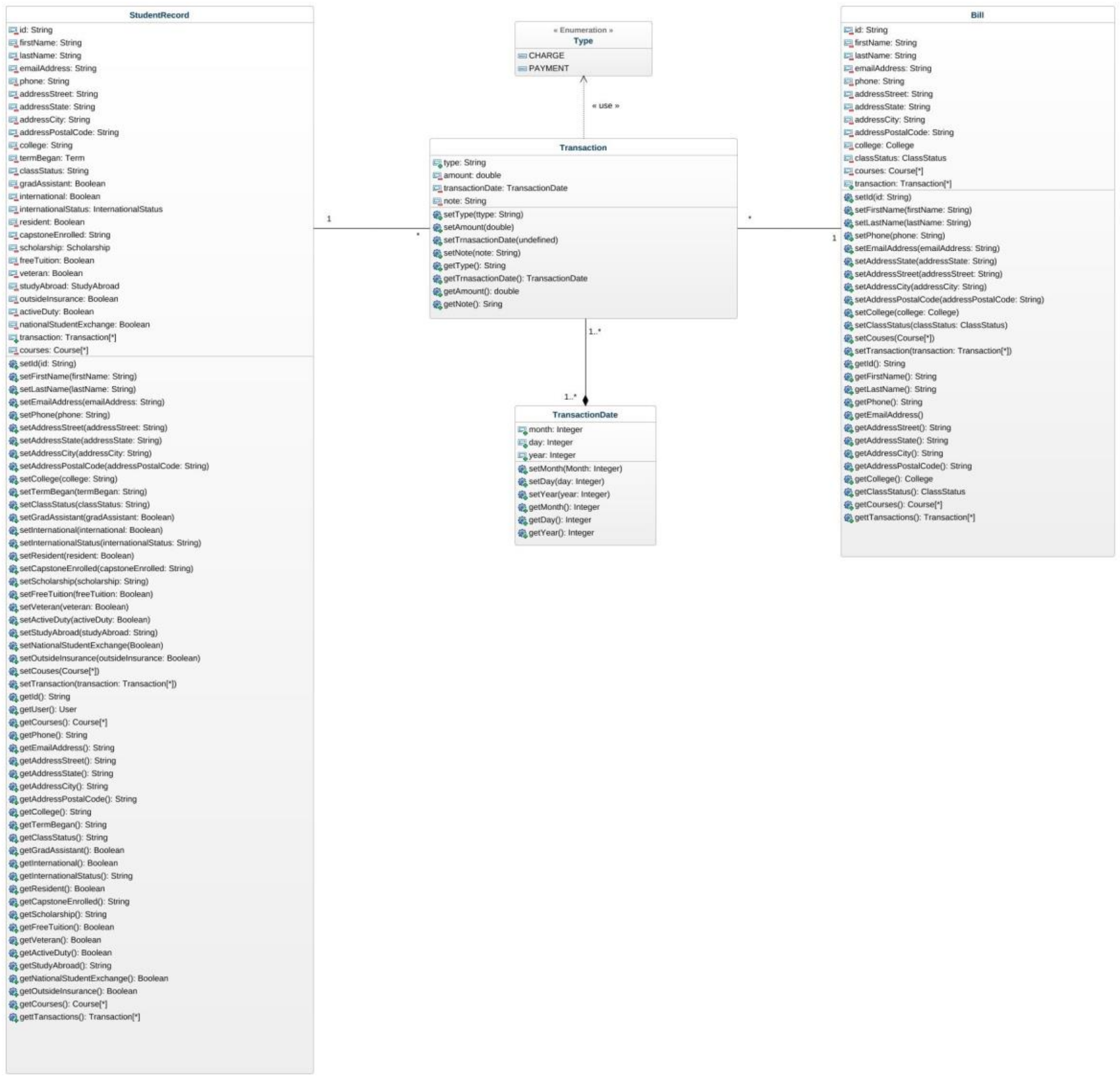
StudentRecord
id: String
firstName: String
lastName: String
emailAddress: String
phone: String
addressStreet: String
addressState: String
addressCity: String
addressPostalCode: String
college: String
termBegan: Term
classStatus: String
gradAssistant: Boolean
international: Boolean
internationalStatus: InternationalStatus
resident: Boolean
capstoneEnrolled: String
scholarship: Scholarship
freeTuition: Boolean
veteran: Boolean
studyAbroad: StudyAbroad
outsideInsurance: Boolean
activeDuty: Boolean
nationalStudentExchange: Boolean
transaction: Transaction[*]
courses: Course[*]
setId(id: String)
setFirstName(firstName: String)
setLastName(lastName: String)
setEmailAddress(emailAddress: String)
setPhone(phone: String)
setAddressStreet(addressStreet: String)
setAddressState(addressState: String)
setAddressCity(addressCity: String)
setAddressPostalCode(addressPostalCode: String)
setCollege(college: String)
setTermBegan(termBegan: String)
setClassStatus(classStatus: String)
setGradAssistant(gradAssistant: Boolean)
setInternational(international: Boolean)
setInternationalStatus(internationalStatus: String)
setResident(resident: Boolean)
setCapstoneEnrolled(capstoneEnrolled: String)
setScholarship(scholarship: String)
setFreeTuition(freeTuition: Boolean)
setVeteran(veteran: Boolean)
setActiveDuty(activeDuty: Boolean)
setStudyAbroad(studyAbroad: String)
setNationalStudentExchange(Boolean)
setOutsideInsurance(outsideInsurance: Boolean)
setCouses(Course[*])
setTransaction(transaction: Transaction[*])
getId(): String
getUser(): User
getCourses(): Course[*]
getPhone(): String
getEmailAddress(): String
getAddressStreet(): String
getAddressState(): String
getAddressCity(): String
getAddressPostalCode(): String
getCollege(): String
getTermBegan(): String
getClassStatus(): String
getGradAssistant(): Boolean
getInternational(): Boolean
getInternationalStatus(): String
getResident(): Boolean
getCapstoneEnrolled(): String
getScholarship(): String
getFreeTuition(): Boolean
getVeteran(): Boolean
getActiveDuty(): Boolean
getStudyAbroad(): String
getNationalStudentExchange(): Boolean
getOutsideInsurance(): Boolean
getCourses(): Course[*]
gettTansactions(): Transaction[*]
1
*
« Enumeration »
Type
CHARGE
PAYMENT
« use »
Transaction
type: String
amount: double
transactionDate: TransactionDate
note: String
setType(ttype: String)
setAmount(double)
setTrnasactionDate(undefined)
setNote(note: String)
getType(): String
getTrnasactionDate(): TransactionDate
getAmount(): double
getNote(): Sring
1..*
1..*
TransactionDate
month: Integer
day: Integer
year: Integer
setMonth(Month: Integer)
setDay(day: Integer)
setYear(year: Integer)
getMonth(): Integer
getDay(): Integer
getYear(): Integer
*
1
Bill
id: String
firstName: String
lastName: String
emailAddress: String
phone: String
addressStreet: String
addressState: String
addressCity: String
addressPostalCode: String
college: College
classStatus: ClassStatus
courses: Course[*]
transaction: Transaction[*]
setId(id: String)
setFirstName(firstName: String)
setLastName(lastName: String)
setPhone(phone: String)
setEmailAddress(emailAddress: String)
setAddressState(addressState: String)
setAddressStreet(addressStreet: String)
setAddressCity(addressCity: String)
setAddressPostalCode(addressPostalCode: String)
setCollege(college: College)
setClassStatus(classStatus: ClassStatus)
setCouses(Course[*])
setTransaction(transaction: Transaction[*])
getId(): String
getFirstName(): String
getLastName(): String
getPhone(): String
getEmailAddress()
getAddressStreet(): String
getAddressState(): String
getAddressCity(): String
getAddressPostalCode(): String
getCollege(): College
getClassStatus(): ClassStatus
getCourses(): Course[*]
gettTansactions(): Transaction[*]

Figure 9 – DTO Package Class Diagram

### 4.2.6 Class Diagram in the Exception Package

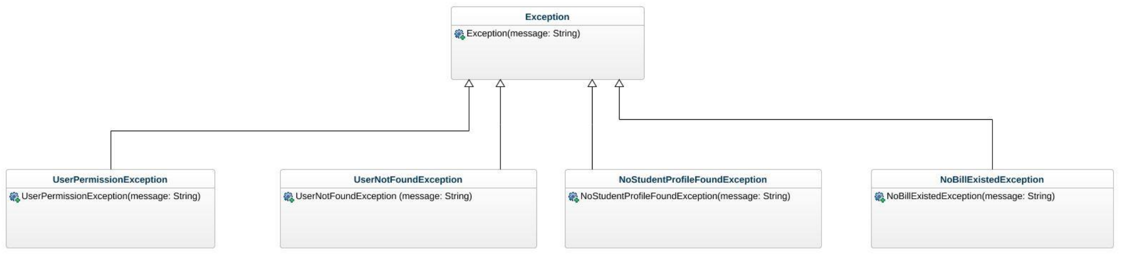
Exception
Exception(message: String)
UserPermissionException
UserPermissionException(message: String)
UserNotFoundException
UserNotFoundException (message: String)
NoStudentProfileFoundException
NoStudentProfileFoundException(message: String)
NoBillExistedException
NoBillExistedException(message: String)

Figure 10 – Exception Package Class Diagram

#### 4.2.7 Class Diagram in the Utility Package

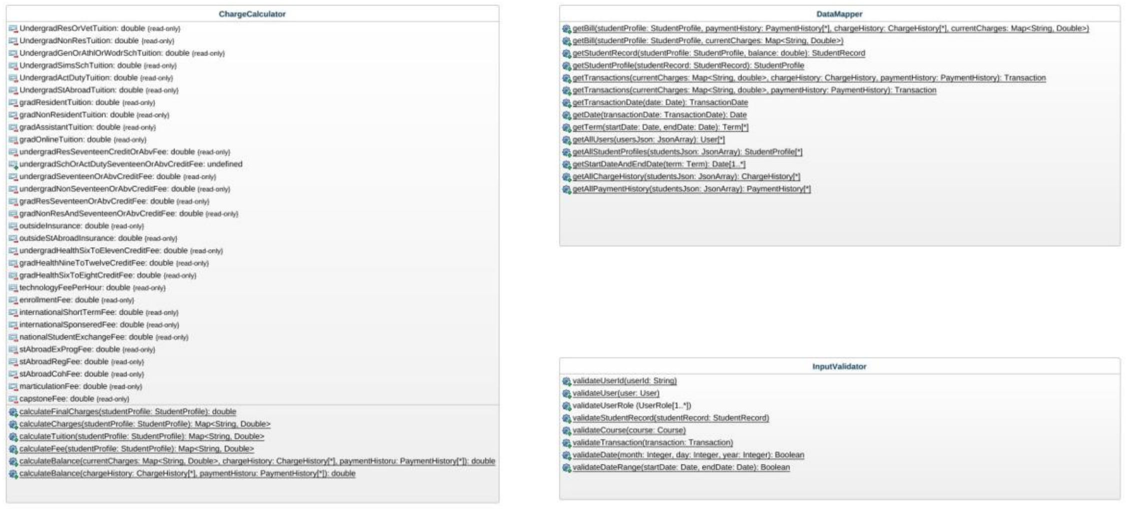
ChargeCalculator
UndergradResOrVetTuition: double (read-only)
UndergradNonResTuition: double (read-only)
UndergradGenOrAthlOrWodrSchTuition: double (read-only)
UndergradSimsSchTuition: double (read-only)
UndergradActDutyTuition: double (read-only)
UndergradStAbroadTuition: double (read-only)
gradResidentTuition: double (read-only)
gradNonResidentTuition: double (read-only)
gradAssistantTuition: double (read-only)
gradOnlineTuition: double (read-only)
undergradResSeventeenCreditOrAbvFee: double (read-only)
undergradSchOrActDutySeventeenOrAbvCreditFee: undefined
undergradSeventeenOrAbvCreditFee: double (read-only)
undergradNonSeventeenOrAbvCreditFee: double (read-only)
gradResSeventeenOrAbvCreditFee: double (read-only)
gradNonResAndSeventeenOrAbvCreditFee: double (read-only)
outsideInsurance: double (read-only)
outsideStAbroadInsurance: double (read-only)
undergradHealthSixToElevenCreditFee: double (read-only)
gradHealthNineToTwelveCreditFee: double (read-only)
gradHealthSixToEightCreditFee: double (read-only)
technologyFeePerHour: double (read-only)
enrollmentFee: double (read-only)
internationalShortTermFee: double (read-only)
internationalSponseredFee: double (read-only)
nationalStudentExchangeFee: double (read-only)
stAbroadExProgFee: double (read-only)
stAbroadRegFee: double (read-only)
stAbroadCohFee: double (read-only)
marticulationFee: double (read-only)
capstoneFee: double (read-only)
calculateFinalCharges(studentProfile: StudentProfile): double
calculateCharges(studentProfile: StudentProfile): Map<String, Double>
calculateTuition(studentProfile: StudentProfile): Map<String, Double>
calculateFee(studentProfile: StudentProfile): Map<String, Double>
calculateBalance(currentCharges: Map<String, Double>, chargeHistory: ChargeHistory[*], paymentHistoru: PaymentHistory[*]): double
calculateBalance(chargeHistory: ChargeHistory[*], paymentHistoru: PaymentHistory[*]): double
DataMapper
getBill(studentProfile: StudentProfile, paymentHistory: PaymentHistory[*], chargeHistory: ChargeHistory[*], currentCharges: Map<String, Double>)
getBill(studentProfile: StudentProfile, currentCharges: Map<String, Double>)
getStudentRecord(studentProfile: StudentProfile, balance: double): StudentRecord
getStudentProfile(studentRecord: StudentRecord): StudentProfile
getTransactions(currentCharges: Map<String, double>, chargeHistory: ChargeHistory, paymentHistory: PaymentHistory): Transaction
getTransactions(currentCharges: Map<String, double>, paymentHistory: PaymentHistory): Transaction
getTransactionDate(date: Date): TransactionDate
getDate(transactionDate: TransactionDate): Date
getTerm(startDate: Date, endDate: Date): Term[*]
getAllUsers(usersJson: JsonArray): User[*]
getAllStudentProfiles(studentsJson: JsonArray): StudentProfile[*]
getStartDateAndEndDate(term: Term): Date[1..*]
getAllChargeHistory(studentsJson: JsonArray): ChargeHistory[*]
getAllPaymentHistory(studentsJson: JsonArray): PaymentHistory[*]
InputValidator
validateUserId(userId: String)
validateUser(user: User)
validateUserRole (UserRole[1..*])
validateStudentRecord(studentRecord: StudentRecord)
validateCourse(course: Course)
validateTransaction(transaction: Transaction)
validateDate(month: Integer, day: Integer, year: Integer): Boolean
validateDateRange(startDate: Date, endDate: Date): Boolean

Figure 11 – Utility Package Class Diagram

## 4.3 Class Descriptions

### 4.3.1 Classes in the Model package

#### 4.3.1.1 Class: User

- Purpose: *To model user-related information.*
  - Constraints: *should be annotated as @Entity in order for the class to be mapped and created in database*
  - Persistent: *Yes. Data is persisted at either system initialization or loading from other available data. I.e. from files or upon client request.*

##### 4.3.1.1.1 Attribute Descriptions

1. Attribute: *id*
   Type: *String*
   Description: *the unique id of user*
   Constraints: *the length of id should be 8 characters, and consists of both letter and number*
2. Attribute: *userRoles[1..*]*
   Type: *UserRole*
   Description: *role/roles of user in the system either one or both of admin or student*
   Constraints: *should be no less than one role*
3. Attribute: *firstName*
   Type: *String*
   Description: *first name of user*
   Constraints: *shouldn't be empty and be consisted of only letters*
4. Attribute: *middleName*
   Type: *String*
   Description: *middle name of user*
   Constraints: *shouldn't be empty and be consisted of only letters*
5. Attribute: *lastName*
   Type: *String*
   Description: *middle name of user*

Constraints: *shouldn't be empty and be consisted of only letters*

#### 4.3.1.1.2 Methods Descriptions

1. Method: setId(id)
   Return Type: *void*
   Parameters: *id*
   Return value: no return value
   Pre-condition: *the length of id should be 8 characters consists of both letter and number*
   Post-condition:
   Attributes read/updated: *id*
   Processing logic: *set/update the id of user*

2. Method: setUserRoles( userRoles) : void
   Return Type: *void*
   Parameters: UserRole [1..*]
   Return value: no return value
   Pre-condition: *user roles should be no less than one role.*
   Post-condition:
   Attributes read/updated: UserRole [1..*]
   Processing logic: *set/update the roles of user*

3. Method: setFirstName ( firstName )
   Return Type: *void*
   Parameters: firstName
   Return value: no return value
   Pre-condition: *firstName shouldn't be empty and be consisted of only letters.*
   Post-condition: no post-condition
   Attributes read/updated: firstName
   Processing logic: *set/update the first name of user*

4. Method: setMiddleName( middleName )
   Return Type: *void*
   Parameters: middleName
   Return value: no return value
   Pre-condition: *middleName shouldn't be empty and be consisted of only letters.*
   Post-condition: no post-condition
   Attributes read/updated: middleName
   Processing logic: *set/update the middle name of user*

5. Method: setLastName ( lastName )
   Return Type: *void*
   Parameters: lastName
   Return value: no return value
   Pre-condition: *lastName shouldn't be empty and be consisted of only letters.*
   Post-condition: no post-condition
   Attributes read/updated: lastName
   Processing logic: *set/update the last name of user*

6. Method: getId()
   Return Type: *String*
   Parameters: no parameter
   Return value: id

Pre-condition: *no pre-condition*
Post-condition: no post-condition
Attributes read/updated: id
Processing logic: get the id of user

7. Method: getUserRoles()
Return Type: *UserRole*
Parameters: no parameter
Return value: UserRole [1..*]
Pre-condition: *no pre-condition*
Post-condition: no post-condition
Attributes read/updated: UserRole [1..*]
Processing logic: get the roles of user

8. Method: getFirstName()
Return Type: *String*
Parameters: no parameter
Return value: firstName
Pre-condition: *no pre-condition*
Post-condition: no post-condition
Attributes read/updated: firstName
Processing logic: *get the first name of user*

9. Method: getMiddleName()
Return Type: *String*
Parameters: no parameter
Return value: middleName
Pre-condition: *no pre-condition*
Post-condition: no post-condition
Attributes read/updated: middleName
Processing logic: *get the l name of user*

10. Method: getLastName()
Return Type: *String*
Parameters: no parameter
Return value: lastName
Pre-condition: *no pre-condition*
Post-condition: no post-condition
Attributes read/updated: lastName
Processing logic: *get the last name of user*

### 4.3.1.2 Class: Course

- Purpose: A course is an academic class associated with a Student, the number of courses and attributes such as the number of credit hours will be used to calculate the student's bill.
- Constraints: Every instance of Course is associated with a Student Profile. Should be annotated as @Entity in order for the class to be mapped and created in database
- Persistent: Yes. Data is persisted at either system initialization or loading from other available data. I.e. from files or upon client request.

#### 4.3.1.2.1 Attribute Descriptions

1. Attribute: id

Type: String
Description: The unique identifier of the Course
Constraint: Consist of 4-letter followed by 3-digit

2. Attribute: name
   Type: String
   Description: The name, or title, of the class
   Constraint: Non Null string, less than 25 characters

3. Attribute: numCredits
   Type: Integer
   Description: The number of credit hours for the class
   Constraint: May not be negative

4. Attribute: onLine
   Type: Boolean
   Description: Flag to indicate if the course is taught online
   Constraint: True or False

#### 4.3.1.2.2 Method Descriptions

1. Method: setId(id)
   Return Type: void
   Parameters: id
   Return value: no return value
   Pre-condition: id may not be NULL
   Post-condition: The id is associated to the Course object
   Attributes read/updated: id
   Processing logic: set/update the id value of the Course object.

2. Method: setName(name)
   Return Type: void
   Parameters: name
   Return value: no return value
   Pre-condition: name may not be NULL
   Post-condition: name is associated to the Course object
   Attributes read/updated: name
   Processing logic: set/update the name value associated to Course object

3. Method: setNumCredits(numCredits)
   Return Type: void
   Parameters: numCredits
   Return value: no return value
   Pre-condition: NumCredits may not be negative
   Post-condition: NumCredits is associated to the Course object
   Attributes read/updated: NumCredits
   Processing logic: set/update the NumCredits value associated to Course object

4. Method: setOnline(online)
   Return Type: void
   Parameters: online
   Return value: no return value
   Pre-condition: None
   Post-condition: online is associated to the Course object
   Attributes read/updated: oline

Processing logic: set/update the online value associated to Course object

5. Method: getId()
   Return Type: String
   Parameters:  None
   Return value: Id
   Pre-condition: None
   Post-condition: None
   Attributes read/updated: Id
   Processing logic: retrieve the Id of Course object.

6. Method: getName()
   Return Type: String
   Parameters:  None
   Return value: name
   Pre-condition: None
   Post-condition: None
   Attributes read/updated: name
   Processing logic: retrieve the name of Course object.

7. Method: getNumCredits()
   Return Type: Integer
   Parameters:  None
   Return value: numCredits
   Pre-condition: None
   Post-condition: None
   Attributes read/updated: numCredits
   Processing logic: retrieve the numCredits of the Course object,

8. Method: getOnline ()
   Return Type: Boolean
   Parameters:  None
   Return value: online
   Pre-condition: None
   Post-condition: None
   Attributes read/updated: online
   Processing logic: retrieve the online of the Course object.
   year of the Course object.

#### 4.3.1.3 Class: PaymentHistory

- Purpose: A payment history is an amount paid by a user against a student's bill.
- Constraints: Every instance of PaymentHistory is associated with a User.
- Persistent: Yes. Data is persisted at either system initialization or loading from other available data. I.e. from files or upon client request.

##### 4.3.1.3.1 Attribute Descriptions

1. Attribute: id
   Type: Long
   Description: The unique identifier of the PaymentHistory
   Constraint: Maximum eight digits numeric

2. Attribute: user

Type: User
Description: The user making the payment
Constraint: Non NULL

3. Attribute: amount
Type: Double
Description: The amount of the payment
Constraint: Must be positive

4. Attribute: paymentDate
Type: Date
Description: Chronologically, when the payment was made
Constraint: None

5. Attribute: note
Type: String
Description: Ancillary note comment about the payment
Constraint: None

#### 4.3.1.3.2 Method Descriptions

1. Method: setUser(user)
Return Type: void
Parameters:  user
Return value: no return value
Pre-condition: User object value may not be NULL
Post-condition: User is associated to the PaymentHistory object
Attributes read/updated: User
Processing logic: set/update the User value of the PaymentHistory object.

2. Method: setPaymentDate (paymentDate)
Return Type: void
Parameters:  paymentDate
Return value: no return value
Pre-condition: paymentDate may not be NULL
Post-condition: paymentDate is associated to the PaymentHistory object
Attributes read/updated: paymentDate
Processing logic: set/update the paymentDate value associated to PaymentHistory object

3. Method: setAmount(amount)
Return Type: void
Parameters:  amount
Return value: no return value
Pre-condition: Amount may not be negative
Post-condition: Amount is associated to the PaymentHistory object
Attributes read/updated: Amount
Processing logic: set/update the Amount value associated to PaymentHistory object

4. Method: setNote(note)
Return Type: void
Parameters:  reason
Return value: no return value
Pre-condition: None
Post-condition: Reason is associated to the PaymentHistory object
Attributes read/updated: note

Processing logic: set/update the note value associated to PaymentHistory object

5. Method: getId()
Return Type: Long
Parameters: None
Return value: id
Pre-condition: None
Post-condition: None
Attributes read/updated: id
Processing logic: retrieve the id of PaymentHistory object.

6. Method: getUser()
Return Type: User
Parameters: None
Return value: user
Pre-condition: None
Post-condition: None
Attributes read/updated: user
Processing logic: retrieve the user of PaymentHistory object.

7. Method: getPaymentDate()
Return Type: Date
Parameters: None
Return value: paymentDate
Pre-condition: None
Post-condition: None
Attributes read/updated: paymentDate
Processing logic: retrieve the paymentDate of the PaymentHistory object,

8. Method: getAmount()
Return Type: Double
Parameters: None
Return value: amount
Pre-condition: None
Post-condition: None
Attributes read/updated: amount
Processing logic: retrieve the amount of the PaymentHistory object.

9. Method: getNote ()
Return Type: String
Parameters: None
Return value: reason
Pre-condition: None
Post-condition: None
Attributes read/updated: reason
Processing logic: retrieve the reason of the PaymentHistory object.

#### 4.3.1.4 Class: ChargeHistory

- Purpose: A charge history is final amount charged against the user by the end of semester.
- Constraints: Every instance of ChargeHistory is associated with a User.
- Persistent: Yes. Data is persisted at either system initialization or loading from other available data. I.e. from files or upon client request.

#### 4.3.1.4.1 Attribute Descriptions

1. Attribute: id
Type: Long
Description: The unique identifier of the ChargeHistory
Constraint: Maximum eight digits numeric

2. Attribute: user
Type: User
Description: The user getting the charge
Constraint: Non NULL

3. Attribute: amount
Type: Double
Description: The amount of the charge
Constraint: Must be positive

4. Attribute: endTerm
Type: Term
Description: The end of term where the final charges calculated
Constraint: None

5. Attribute: note
Type: String
Description: Ancillary note comment about the payment
Constraint: None

#### 4.3.1.4.2 Method Descriptions

1. Method: setUser(user)
Return Type: void
Parameters: user
Return value: no return value
Pre-condition: User object value may not be NULL
Post-condition: User is associated to the ChargeHistory object
Attributes read/updated: User
Processing logic: set/update the User value of the ChargeHistory object

2. Method: setAmount(amount)
Return Type: void
Parameters: amount
Return value: no return value
Pre-condition: Amount may not be negative
Post-condition: Amount is associated to the ChargeHistory object
Attributes read/updated: Amount
Processing logic: set/update the Amount value associated to ChargeHistory object

3. Method: setEndTerm (endTerm)
Return Type: void
Parameters: endTerm
Return value: no return value
Pre-condition: endTerm may not be NULL
Post-condition: endTerm is associated to the ChargeHistory object
Attributes read/updated: endTerm
Processing logic: set/update the endTerm value associated to ChargeHistory object

4. Method: setNote(note)

Return Type: void
Parameters: reason
Return value: no return value
Pre-condition: None
Post-condition: Reason is associated to the ChargeHistory object
Attributes read/updated: note
Processing logic: set/update the note value associated to ChargeHistory object

5. Method: getId()

Return Type: Long
Parameters: None
Return value: id
Pre-condition: None
Post-condition: None
Attributes read/updated: id
Processing logic: retrieve the id of ChargeHistory object.

6. Method: getUser()

Return Type: User
Parameters: None
Return value: user
Pre-condition: None
Post-condition: None
Attributes read/updated: user
Processing logic: retrieve the user of ChargeHistory object.

7. Method: getAmount()

Return Type: Double
Parameters: None
Return value: amount
Pre-condition: None
Post-condition: None
Attributes read/updated: amount
Processing logic: retrieve the amount of the ChargeHistory object

1. Method: getEndTerm()

Return Type: Date
Parameters: None
Return value: endTerm
Pre-condition: None
Post-condition: None
Attributes read/updated: endTerm
Processing logic: retrieve the paymentDate of the ChargeHistory object

2. Method: getNote ()

Return Type: String
Parameters: None
Return value: reason
Pre-condition: None
Post-condition: None
Attributes read/updated: reason

Processing logic: retrieve the reason of the ChargeHistory object.

#### 4.3.1.5 Class: Student Profile

- Purpose: For users with the role of Student, to provide a centralized point for the information related to their role as a student. The values stored in the student profile will be initially provided from the external Bill of Materials and be used to calculate the applicable fees and their charges for a Student's Bill.
- Constraints: Student Profiles are always associated with a User. Should be annotated as @Entity in order for the class to be mapped and created in database
- Persistent: Yes. Data is persisted at either system initialization or loading from other available data. I.e. from files or upon client request.

##### 4.3.1.5.1 Attribute Descriptions

1. Attribute: id
   Type: Long
   Description: The unique identifier of the student profile
   Constraint: Maximum eight digits numeric
2. Attribute: user
   Type: User object
   Description: The User object associated with the student profile
   Constraint: Non-NULL
3. Attribute: Courses
   Type:  Array of Course objects
   Description: Array of Course objects associated with the student profile
   Constraint: Array contains zero or more Courses
4. Attribute: emailAddress
   Type: String
   Description: Email address of the student
   Constraint: Alpha-numeric characters following standard email limitations, e.g. length greater than 5, must contain '@', etc.
5. Attribute: phone
   Type: String
   Description: Phone number, represented as a string, for contacting the student
   Constraint: Numeric, plus characters (, ), -, and space.
6. Attribute: addressStreet
   Type: String
   Description: Street address of the student, e.g. "123 Maple St, Apt B"
   Constraint:None
7. Attribute: addressState
   Type: String
   Description: Two-character abbreviation of the state associated with the student
   Constraint: Two-character alpha, valid state abbreviation.
8. Attribute: addressCity
   Type: String
   Description: City name of the student
   Constraint: None
9. Attribute: addressPostalCode
   Type: String

Description: Postal code for the student
Constraint: Either 5 numeric characters or 5 plus a hyphen character and 4 more numeric

10. Attribute: termBegan
Type: Term
Description: The term the student started college
Constraint: Valid value of the enumerated type Term

11. Attribute: classStatus
Type: ClassStatus
Description: Current class categorization for the student, e.g. Freshman, Sophomore, etc.
Constraint: Valid value in the enumerated type ClassStatus

12. Attribute: gradAssistant
Type: Boolean
Description: Flag to indicate if the student is a graduate assistant
Constraint: True or False

13. Attribute: international
Type: Boolean
Description: Flag to indicate if the student is an international student
Constraint: True or False

14. Attribute: internationalStatus
Type: InternationalStatus
Description: Value to indicate if the student is an international student, what type of international student they are.
Constraint: Must have a value of "None" if international=False. May not have a value of "None" if international=True.

15. Attribute: resident
Type: Boolean
Description: Flag to indicate if the student is a resident of South Carolina
Constraint: True or False

16. Attribute: capstoneEnrolled
Type: Term
Description: Term when the student enrolled in the capstone program, e.g. "Spring2017","Fall2015", etc.
Constraint: Valid value of the enumerated type Term.

17. Attribute: scholarship
Type: Scholarship
Description: Value of the scholarship, if any, the student is entitled to.
Constraint: Valid value in enumerated type Scholarship. If IsResident=True then Scholarship must equal "None".

18. Attribute: freeTuition
Type: Boolean
Description: Flag to indicate if entire tuition charges should be waived for the student
Constraint: True or False

19. Attribute: veteran
Type: Boolean
Description: Flag to indicate if the student is a family member of a veteran
Constraint: True or False

20. Attribute: studyAbroad
Type: StudyAbroad

Description: Value to indicate if the student is studying abroad and if so what type of abroad status
Constraint: Valid value in enumerated type StudyAbroad

21. Attribute: outsideInsurance
Type: Boolean
Description: Flag to indicate if the student has health insurance through an outside 3rd party
Constraint: True or False

22. Attribute: activeDuty
Type: Boolean
Description: Flag to indicate if the student is active duty military
Constraint: True or False

23. Attribute: nationalStudentExchange
Type: Boolean
Description: Flag to indicate if the student is an exchange student
Constraint: True or False

#### 4.3.1.5.2 Method Descriptions

1. Method: setUser(User)
Return Type: void
Parameters: User
Return value: no return value
Pre-condition: user object may not be NULL
Post-condition: user is associated with the Student Profile object
Attributes read/updated: user
Processing logic: set/update the user of Student Profile.

2. Method: setCourses(Course[*])
Return Type: void
Parameters: Course [1..*]
Return value: no return value
Pre-condition: Course Array may not be NULL
Post-condition: Course Array is associated to the Student Profile
Attributes read/updated: Course [1..*]
Processing logic: set/update the Courses associated to Student Profile

3. Method: setEmailAddress(emailAddress)
Return Type: void
Parameters: emailAddress
Return value: no return value
Pre-condition: None
Post-condition: emailAddress is associated to the Student Profile
Attributes read/updated: emailAddress
Processing logic: set/update the emailAddress associated to Student Profile

4. Method: setPhone(phone)
Return Type: void
Parameters: phone
Return value: no return value
Pre-condition: None
Post-condition: phone is associated to the Student Profile
Attributes read/updated: phone

Processing logic: set/update the phone associated to Student Profile

5. Method: setAddressStreet(addressStreet)
   Return Type: void
   Parameters: addressStreet
   Return value: no return value
   Pre-condition: None
   Post-condition: addressStreet is associated to the Student Profile
   Attributes read/updated: addressStreet
   Processing logic: set/update the addressStreet associated to Student Profile

6. Method: setAddressState(addressState)
   Return Type: void
   Parameters: addressState
   Return value: no return value
   Pre-condition: None
   Post-condition: addressState is associated to the Student Profile
   Attributes read/updated: addressState
   Processing logic: set/update the addressState associated to Student Profile

7. Method: setAddressCity(addressCity)
   Return Type: void
   Parameters: addressCity
   Return value: no return value
   Pre-condition: None
   Post-condition: addressCity is associated to the Student Profile
   Attributes read/updated: addressCity
   Processing logic: set/update the addressCity associated to Student Profile

8. Method: setAddressPostalCode(addressPostalCode)
   Return Type: void
   Parameters: addressPostalCode
   Return value: no return value
   Pre-condition: None
   Post-condition: addressPostalCode is associated to the Student Profile
   Attributes read/updated: addressPostalCode
   Processing logic: set/update the addressPostalCode associated to Student Profile

9. Method: setTermBegan(termBegan)
   Return Type: void
   Parameters: termBegan
   Return value: no return value
   Pre-condition: None
   Post-condition: termBegan is associated to the Student Profile
   Attributes read/updated: termBegan
   Processing logic: set/update the termBegan associated to Student Profile

10. Method: setClassStatus(classStatus)
    Return Type: void
    Parameters: classStatus
    Return value: no return value
    Pre-condition: None
    Post-condition: classStatus is associated to the Student Profile

Attributes read/updated: classStatus
Processing logic: set/update the classStatus associated to Student Profile

11. Method: setAssistant(gradAssistant)
Return Type: void
Parameters: gradAssistant
Return value: no return value
Pre-condition: None
Post-condition: gradAssistant is associated to the Student Profile
Attributes read/updated: gradAssistant
Processing logic: set/update the gradAssistant associated to Student Profile

12. Method: setInternational(international)
Return Type: void
Parameters: international
Return value: no return value
Pre-condition: None
Post-condition: international is associated to the Student Profile
Attributes read/updated: international
Processing logic: set/update the international associated to Student Profile

13. Method: setInternationalStatus(internationalStatus)
Return Type: void
Parameters: internationalStatus
Return value: no return value
Pre-condition: None
Post-condition: internationalStatus is associated to the Student Profile
Attributes read/updated: internationalStatus
Processing logic: set/update the internationalStatus associated to Student Profile

14. Method: setResident(resident)
Return Type: void
Parameters: resident
Return value: no return value
Pre-condition: None
Post-condition: resident is associated to the Student Profile
Attributes read/updated: resident
Processing logic: set/update the resident associated to Student Profile

15. Method: setCapstoneEnrolled(capstoneEnrolled)
Return Type: void
Parameters: capstoneEnrolled
Return value: no return value
Pre-condition: None
Post-condition: capstoneEnrolled is associated to the Student Profile
Attributes read/updated: capstoneEnrolled
Processing logic: set/update the capstoneEnroled associated to Student Profile

16. Method: setScholarship(scholarship)
Return Type: void
Parameters: scholarship
Return value: no return value
Pre-condition: None

Post-condition: scholarship is associated to the Student Profile
Attributes read/updated: scholarship
Processing logic: set/update the scholarship associated to Student Profile

17. Method: setFreeTuition(freeTuition)
    Return Type: void
    Parameters:  freeTuition
    Return value: no return value
    Pre-condition: None
    Post-condition: freeTutition is associated to the Student Profile
    Attributes read/updated: freeTuition
    Processing logic: set/update the freeTuition associated to Student Profile

18. Method: setVeteran(veteran)
    Return Type: void
    Parameters:  veteran
    Return value: no return value
    Pre-condition: None
    Post-condition: veteran is associated to the Student Profile
    Attributes read/updated: veteran
    Processing logic: set/update the veteran associated to Student Profile

19. Method: setactiveDuty(activeDuty)
    Return Type: void
    Parameters:  activeDuty
    Return value: no return value
    Pre-condition: None
    Post-condition: activeDuty is associated to the Student Profile
    Attributes read/updated: activeDuty
    Processing logic: set/update the activeDuty associated to Student Profile

20. Method: setStudyAbroad (StudyAbroad)
    Return Type: void
    Parameters:  StudyAbroad
    Return value: no return value
    Pre-condition: None
    Post-condition: StudyAbroad is associated to the Student Profile
    Attributes read/updated: StudyAbroad
    Processing logic: set/update the StudyAbroad associated to Student Profile

21. Method: setNationalStudentExchange(nationalStudentExchange)
    Return Type: void
    Parameters:  nationalStudentExchange
    Return value: no return value
    Pre-condition: None
    Post-condition: vationalStudentExchange is associated to the Student Profile
    Attributes read/updated: nationalStudentExchange
    Processing logic: set/update the nationalStudentExchange associated to Student Profile

22. Method: setOutsideInsurance(outsideInsurance)
    Return Type: void
    Parameters:  outsideInsurance
    Return value: no return value

Pre-condition: None
Post-condition: outsideInsurance is associated to the Student Profile
Attributes read/updated: outsideInsurance
Processing logic: set/update the outsideInsurance associated to Student Profile

23. Method: getId()
    Return Type: Long
    Parameters: no parameter
    Return value: id
    Pre-condition: no pre-condition
    Post-condition: no post-condition
    Attributes read/updated: id
    Processing logic: get the id of Student Profile

24. Method: getUser()
    Return Type: User object
    Parameters: no parameter
    Return value: User
    Pre-condition: no pre-condition
    Post-condition: no post-condition
    Attributes read/updated: User
    Processing logic: get the User of Student Profile

25. Method: getCourses()
    Return Type: Array of Courses
    Parameters: no parameter
    Return value: Array of Courses
    Pre-condition: no pre-condition
    Post-condition: no post-condition
    Attributes read/updated: Courses[*]
    Processing logic: get the array of Courses of Student Profile

26. Method: getPhone()
    Return Type: String
    Parameters: no parameter
    Return value: phone
    Pre-condition: no pre-condition
    Post-condition: no post-condition
    Attributes read/updated: phone
    Processing logic: get the phone of Student Profile

27. Method: getEmailAddress()
    Return Type: String
    Parameters: no parameter
    Return value: EmailAddress
    Pre-condition: no pre-condition
    Post-condition: no post-condition
    Attributes read/updated: EmailAddress
    Processing logic: get the EmailAddress of Student Profile

28. Method: getAddressStreet()
    Return Type: String
    Parameters: no parameter

Return value: AddressStreet
Pre-condition: no pre-condition
Post-condition: no post-condition
Attributes read/updated: AddressStreet
Processing logic: get the AddressStreet of Student Profile

29. Method: getAddressState()
Return Type: String
Parameters: no parameter
Return value: AddressState
Pre-condition: no pre-condition
Post-condition: no post-condition
Attributes read/updated: AddressState
Processing logic: get the AddressState of Student Profile

30. Method: getAddressCity()
Return Type: String
Parameters: no parameter
Return value: AddressCity
Pre-condition: no pre-condition
Post-condition: no post-condition
Attributes read/updated: AddressCity
Processing logic: get the AddressCity of Student Profile

31. Method: getAddressPostalCode()
Return Type: String
Parameters: no parameter
Return value: AddressPostalCode
Pre-condition: no pre-condition
Post-condition: no post-condition
Attributes read/updated: AddressPostalCode
Processing logic: get the AddressPostalCode of Student Profile

32. Method: getTermBegan ()
Return Type: Term
Parameters: no parameter
Return value: termBegan
Pre-condition: no pre-condition
Post-condition: no post-condition
Attributes read/updated: termBegan
Processing logic: get the termBegan of Student Profile

33. Method: getClassStatus()
Return Type: ClassStatus
Parameters: no parameter
Return value: ClassStatus
Pre-condition: no pre-condition
Post-condition: no post-condition
Attributes read/updated: ClassStatus
Processing logic: get the ClassStatus of Student Profile

34. Method: getGradAssistant()
Return Type: Boolean

Parameters: no parameter
Return value: gradAssistant
Pre-condition: no pre-condition
Post-condition: no post-condition
Attributes read/updated: gradAssistant
Processing logic: get the gradAssistant of Student Profile

35. Method: geInternational()
Return Type: Boolean
Parameters: no parameter
Return value: international
Pre-condition: no pre-condition
Post-condition: no post-condition
Attributes read/updated: international
Processing logic: get the international of Student Profile

36. Method: getInternationalStatus()
Return Type: InternationalStatus
Parameters: no parameter
Return value: InternationalStatus
Pre-condition: no pre-condition
Post-condition: no post-condition
Attributes read/updated: InternationalStatus
Processing logic: get the InternationalStatus of Student Profile

37. Method: geResident()
Return Type: Boolean
Parameters: no parameter
Return value: resident
Pre-condition: no pre-condition
Post-condition: no post-condition
Attributes read/updated: resident
Processing logic: get the resident of Student Profile

38. Method: getCapstoneEnrolled()
Return Type: Term
Parameters: no parameter
Return value: capstoneEnrolled
Pre-condition: no pre-condition
Post-condition: no post-condition
Attributes read/updated: capstoneEnrolled
Processing logic: get the capstoneEnrolled of Student Profile

39. Method: getScholarship()
Return Type: Scholarship
Parameters: no parameter
Return value: Scholarship
Pre-condition: no pre-condition
Post-condition: no post-condition
Attributes read/updated: Scholarship
Processing logic: get the Scholarship of Student Profile

40. Method: getFreeTuition()

Return Type: Boolean
Parameters: no parameter
Return value: freeTuition
Pre-condition: no pre-condition
Post-condition: no post-condition
Attributes read/updated: freeTuition
Processing logic: get the freeTuition of Student Profile

41. Method: getVeteran()
Return Type: Boolean
Parameters: no parameter
Return value: veteran
Pre-condition: no pre-condition
Post-condition: no post-condition
Attributes read/updated: veteran
Processing logic: get the veteran of Student Profile

42. Method: getActiveDuty()
Return Type: Boolean
Parameters: no parameter
Return value: activeDuty
Pre-condition: no pre-condition
Post-condition: no post-condition
Attributes read/updated: activeDuty
Processing logic: get the activeDuty of Student Profile

43. Method: getStudyAbroad()
Return Type: StudyAbroad
Parameters: no parameter
Return value: studyAbroad
Pre-condition: no pre-condition
Post-condition: no post-condition
Attributes read/updated: studyAbroad
Processing logic: get the studyAbroad of Student Profile

44. Method: getNationalStudentExchange()
Return Type: Boolean
Parameters: no parameter
Return value: nationalStudentExchange
Pre-condition: no pre-condition
Post-condition: no post-condition
Attributes read/updated: nationalStudentExchange
Processing logic: get the nationalStudentExchange of Student Profile

45. Method: getOutsideInsurance()
Return Type: Boolean
Parameters: no parameter
Return value: outsideInsurance
Pre-condition: no pre-condition
Post-condition: no post-condition
Attributes read/updated: outsideInsurance
Processing logic: get the outsideInsurance of Student Profile

#### 4.3.1.6 Class: UserRole

- Purpose: Defines the access and permissions which a user has within the system
- Constraints: Every instance of UserRole is associated with a User object, conversely every instance of a User object is associated with 1 or more UserRoles. Should be annotated as @Entity in order for the class to be mapped and created in database
- Persistent: Yes. Data is persisted at either system initialization or loading from other available data. I.e. from files or upon client request.

##### 4.3.1.6.1 Attribute Descriptions

1. Attribute: id
   Type: String
   Description: The unique identifier of the User Role
   Constraint: Maximum eight digits numeric
2. Attribute: role
   Type: Role
   Description: Permission level for the user, i.e. Student or Admin
   Constraint: Valid value in the enumerated type Role
3. Attribute: college
   Type: College
   Description: School to which the user is associated, e.g. Computer Science, Electrical Engineering
   Constraint: Valid value in the enumerated type College

##### 4.3.1.6.2 Method Descriptions

1. Method: setRole(role)
   Return Type: void
   Parameters: role
   Return value: no return value
   Pre-condition: None
   Post-condition: Role is associated to the User Role object
   Attributes read/updated: Role
   Processing logic: set/update the Role of User Role object.

2. Method: setCollege(college)
   Return Type: void
   Parameters: college
   Return value: no return value
   Pre-condition: None
   Post-condition: college is associated to the User Role object
   Attributes read/updated: college
   Processing logic: set/update the college associated to Student Profile

3. Method: getId()
   Return Type: String
   Parameters: None
   Return value: id
   Pre-condition: None
   Post-condition: None
   Attributes read/updated: id
   Processing logic: retrieve the id of User Role object.

4. Method: getRole()
   Return Type: Role
   Parameters: None
   Return value: role
   Pre-condition: None
   Post-condition: None
   Attributes read/updated: role
   Processing logic: retrieve the role of User Role object.

5. Method: getCollege()
   Return Type: College
   Parameters: None
   Return value: college
   Pre-condition: None
   Post-condition: None
   Attributes read/updated: college
   Processing logic: retrieve the college of User Role object

### 4.3.2 Classes in the DTO package

#### 4.3.2.1 Class: Bill

- Purpose: Like all DTO objects, the Bill exists to perform data transformation. In this case from data fed which represents the student's bill, which represents information about the student and the transactions against their account.
- Constraints: None.
- Persistent: No, Bill is a data transformation object

##### 4.3.2.1.1 Attribute Descriptions

1. Attribute: Id
   Type: String
   Description: the id of user
   Constraint: *the length of id should be 8 characters, and consists of both letter and number*

2. Attribute: firstName
   Type: String
   Description: First name of the student
   Constraint: None

3. Attribute: lastName
   Type: String
   Description: Last name of the student
   Constraint: None

4. Attribute: EmailAddress
   Type: String
   Description: Email address of the student
   Constraint: Alpha-numeric characters following standard email limitations, e.g. length greater than 5, must contain '@', etc.

5. Attribute: Phone
   Type: String
   Description: Phone number, represented as a string, for contacting the student
   Constraint: Numeric, plus characters (, ), -, and space.

6. Attribute: AddressStreet
   Type: String
   Description: Street address of the student, e.g. "123 Maple St, Apt B"
   Constraint:None

7. Attribute: AddressState
   Type: String
   Description: Two-character abbreviation of the state associated with the student
   Constraint: Two-character alpha, valid state abbreviation.

8. Attribute: AddressCity
   Type: String
   Description: City name of the student
   Constraint: None

9. Attribute: AddressPostalCode
   Type: String
   Description: Postal code for the student
   Constraint: Either 5 numeric characters or 5 plus a hyphen character and 4 more numeric

10. Attribute: college
    Type: College
    Description: The school which the student attends, e.g. College of Engineering.
    Constraint: Valid value for the enumerated type College

11. Attribute: classStatus
    Type: ClassStatus
    Description: Current class categorization for the student, e.g. Freshman, Sophomore, etc.
    Constraint: Valid value in the enumerated type ClassStatus

12. Attribute: courses
    Type: Array of Courses
    Description: List of course objects associated with the Student Record
    Constraint: None

13. Attribute: transaction
    Type: Array of Transactions
    Description: List of Transaction objects associated with the Student Record
    Constraint: None

#### 4.3.2.1.2 Method Descriptions

1. Method: setId(id)
   Return Type: void
   Parameters: id
   Return value: no return value
   Pre-condition: the length of id should be 8 characters, and consists of both letter and number
   Post-condition: user id is associated to the Student Record
   Attributes read/updated: id
   Processing logic: set/update the id associated to **Bill**

2. Method: setFirstName(firstName)
   Return Type: void
   Parameters: firstName
   Return value: no return value
   Pre-condition: none
   Post-condition: firstName is associated to the Bill

Attributes read/updated: firstName
Processing logic: set/update the firstName associated to Bill

3. Method: setLastName(lastName)
   Return Type: void
   Parameters:  lastName
   Return value: no return value
   Pre-condition: none
   Post-condition: lastName is associated to the Bill
   Attributes read/updated: lastName
   Processing logic: set/update the lastName associated to Bill

4. Method: setPhone(phone)
   Return Type: void
   Parameters:  phone
   Return value: no return value
   Pre-condition: none
   Post-condition: phone is associated to the Bill
   Attributes read/updated: phone
   Processing logic: set/update the phone associated to Bill

5. Method: setEmailAddress(emailAddress)
   Return Type: void
   Parameters:  emailAddress
   Return value: no return value
   Pre-condition: None
   Post-condition: emailAddress is associated to the Bill
   Attributes read/updated: emailAddress
   Processing logic: set/update the emailAddress associated to Bill

6. Method: setAddressStreet(addressStreet)
   Return Type: void
   Parameters:  addressStreet
   Return value: no return value
   Pre-condition: None
   Post-condition: addressStreet is associated to the Bill
   Attributes read/updated: addressStreet
   Processing logic: set/update the addressStreet associated to Bill

7. Method: setAddressState(addressState)
   Return Type: void
   Parameters:  addressState
   Return value: no return value
   Pre-condition: None
   Post-condition: addressState is associated to the Bill
   Attributes read/updated: addressState
   Processing logic: set/update the addressState associated to Bill

8. Method: setAddressCity(addressCity)
   Return Type: void
   Parameters:  addressCity
   Return value: no return value
   Pre-condition: None

Post-condition: addressCity is associated to the Bill
Attributes read/updated: addressCity
Processing logic: set/update the addressCity associated to Bill

9. Method: setAddressPostalCode(addressPostalCode)
   Return Type: void
   Parameters: addressPostalCode
   Return value: no return value
   Pre-condition: None
   Post-condition: addressPostalCode is associated to the Bill
   Attributes read/updated: addressPostalCode
   Processing logic: set/update the addressPostalCode associated to Bill

10. Method: setCollege(college)
    Return Type: void
    Parameters: college
    Return value: no return value
    Pre-condition: None
    Post-condition: college is associated to the Bill
    Attributes read/updated: college
    Processing logic: set/update the college associated to Bill

11. Method: setClassStatus(classStatus)
    Return Type: void
    Parameters: classStatus
    Return value: no return value
    Pre-condition: None
    Post-condition: classStatus is associated to the Bill
    Attributes read/updated: classStatus
    Processing logic: set/update the classStatus associated to Bill

12. Method: setCourses(Course[*])
    Return Type: void
    Parameters: Array of Courses
    Return value: no return value
    Pre-condition: None
    Post-condition: Course Array is associated to the Bill
    Attributes read/updated: Course [1..*]
    Processing logic: set/update the Courses associated to Bill

13. Method: setTransaction(Transaction[*])
    Return Type: void
    Parameters: Array of Transactions
    Return value: no return value
    Pre-condition: None
    Post-condition: Transaction Array is associated to the Bill
    Attributes read/updated: Transaction [1..*]
    Processing logic: set/update the Transaction Array associated to Bill

14. Method: getId()
    Return Type: String
    Parameters: no parameter
    Return value: id

Pre-condition: None
Post-condition: None
Attributes read/updated: id
Processing logic: return the id associated to Bill

15. Method: getFirstName()
    Return Type: String
    Parameters: no parameter
    Return value: firstName
    Pre-condition: None
    Post-condition: None
    Attributes read/updated: firstName
    Processing logic: return the firstName associated to Bill

16. Method: getLastName()
    Return Type: String
    Parameters: no parameter
    Return value: lastName
    Pre-condition: None
    Post-condition: None
    Attributes read/updated: lastName
    Processing logic: return the lastName associated to Bill

17. Method: getPhone()
    Return Type: String
    Parameters: no parameter
    Return value: phone
    Pre-condition: None
    Post-condition: None
    Attributes read/updated: phone
    Processing logic: return the phone associated to Bill

18. Method: getEmailAddress()
    Return Type: String
    Parameters: no parameter
    Return value: emailAddress
    Pre-condition: None
    Post-condition: None
    Attributes read/updated: emailAddress
    Processing logic: return the emailAddress associated to Bill

19. Method: getAddressStreet()
    Return Type: String
    Parameters: no parameter
    Return value: addressStreet
    Pre-condition: None
    Post-condition: None
    Attributes read/updated: addressStreet
    Processing logic: return the addressStreet associated to Bill

20. Method: getAddressState()
    Return Type: String
    Parameters: no parameter

Return value: addressState
Pre-condition: None
Post-condition: None
Attributes read/updated: addressState
Processing logic: return the addressState associated to Bill

21. Method: getAddressCity()
    Return Type: String
    Parameters: no parameter
    Return value: addressCity
    Pre-condition: None
    Post-condition: None
    Attributes read/updated: addressCity
    Processing logic: return the addressCity associated to Bill

22. Method: getAddressPostalCode()
    Return Type: String
    Parameters: no parameter
    Return value: addressPostalCode
    Pre-condition: None
    Post-condition: a None
    Attributes read/updated: addressPostalCode
    Processing logic: return the addressPostalCode associated to Bill

23. Method: getCollege()
    Return Type: String
    Parameters: no parameter
    Return value: college
    Pre-condition: None
    Post-condition: None
    Attributes read/updated: college
    Processing logic: return the college associated to Bill

24. Method: getClassStatus()
    Return Type: String
    Parameters: no parameter
    Return value: classStatus
    Pre-condition: None
    Post-condition: None
    Attributes read/updated: classStatus
    Processing logic: return the classStatus associated to Bill

25. Method: getCourses(Course[*])
    Return Type: Array of Courses
    Parameters: no parameter
    Return value: Courses
    Pre-condition: None
    Post-condition: None
    Attributes read/updated: Course [1..*]
    Processing logic: return the Courses associated to Bill

26. Method: getTransaction(Transaction[*])
    Return Type: Array of Transactions

Parameters: no parameter
Return value: Transaction
Pre-condition: None
Post-condition: None
Attributes read/updated: Transaction [1..*]
Processing logic: return the Transaction Array associated to Bill

### 4.3.2.2 Class: TransactionDate

- Purpose: Like all DTO objects, the TransactionDate exists to perform data transformation. In this case from data which includes a day, month, and year, i.e. a date.
- Constraints: The TransactionDate must define a valid date on a Gregorian Calendar.
- Persistent: No, TransactionDate is a data transformation object

#### 4.3.2.2.1 Attribute Descriptions

1. Attribute: month
   Type: Integer
   Description: The index of the month, e.g. Jan=1, Feb=2, etc.
   Constraint: Value between 1-12, inclusive
2. Attribute: day
   Type: Integer
   Description: The index of the day in the month
   Constraint: Value from 1-31, depending upon the month & year and the rules of a Gregorian calendar, day is possibly further limited, e.g. day cannot equal 31 if month equals 11.
3. Attribute: year
   Type: Integer
   Description: The year value of the date
   Constraint: Positive value greater than 1950.

4.3.2.2.2 Method Descriptions

1. Method: setMonth(month)
   Return Type: void
   Parameters: month
   Return value: no return value
   Pre-condition: none
   Post-condition: month is associated to the TransactionDate
   Attributes read/updated: month, day, & year to prevent invalid overall date when setting month
   Processing logic: set/update the date associated to TransactionDate

2. Method: setDay(day)
   Return Type: void
   Parameters: day
   Return value: no return value
   Pre-condition: none
   Post-condition: day is associated to the TransactionDate
   Attributes read/updated: date, month, & year to prevent invalid overall date when setting day
   Processing logic: set/update the day associated to TransactionDate

3. Method: setYear(year)
   Return Type: void
   Parameters: year

Return value: no return value
Pre-condition: none
Post-condition: year is associated to the TransactionDate
Attributes read/updated: year, month, & day to prevent invalid overall date when setting year
Processing logic: set/update the year associated to TransactionDate

4. Method: getMonth()
   Return Type: Integer
   Parameters:  no parameter
   Return value: month
   Pre-condition: None
   Post-condition: None
   Attributes read/updated: month
   Processing logic: return the month associated to TransactionDate

5. Method: getDay()
   Return Type: Integer
   Parameters:  no parameter
   Return value: day
   Pre-condition: None
   Post-condition: None
   Attributes read/updated: user
   Processing logic: return the User associated with the TransactionDate

6. Method: getYear()
   Return Type: Integer
   Parameters:  no parameter
   Return value: year
   Pre-condition: None
   Post-condition: None
   Attributes read/updated: year
   Processing logic:  return the amount associated to TransactionDate

### 4.3.2.3 Class: StudentRecord

- Purpose: Like all DTO objects, the StudentRecord exists to perform data transformation. In this case from data fed into the system which represent students, to the Student Profile objects.
- Constraints: None.
- Persistent: No, Student Record is a data transformation object

#### 4.3.2.3.1 Attribute Descriptions

1. Attribute: Id
   Type: String
   Description: the length of id should be 8 characters, and consists of both letter and number
   Constraint: Maximum eight digit numeric

2. Attribute: firstName
   Type: String
   Description: First name of the student
   Constraint: None

3. Attribute: lastName
   Type: String

Description: Last name of the student
Constraint: None

4. Attribute: EmailAddress
Type: String
Description: Email address of the student
Constraint: Alpha-numeric characters following standard email limitations, e.g. length greater than 5, must contain '@', etc.

5. Attribute: Phone
Type: String
Description: Phone number, represented as a string, for contacting the student
Constraint: Numeric, plus characters (, ), -, and space.

6. Attribute: AddressStreet
Type: String
Description: Street address of the student, e.g. "123 Maple St, Apt B"
Constraint:None

7. Attribute: AddressState
Type: String
Description: Two-character abbreviation of the state associated with the student
Constraint: Two-character alpha, valid state abbreviation.

8. Attribute: AddressCity
Type: String
Description: City name of the student
Constraint: None

9. Attribute: AddressPostalCode
Type: String
Description: Postal code for the student
Constraint: Either 5 numeric characters or 5 plus a hyphen character and 4 more numeric

10. Attribute: college
Type: String
Description: The school which the student attends, e.g. College of Engineering.
Constraint: Valid value for the enumerated type College

11. Attribute: termBegan
Type: String
Description: The term which the student attends
Constraint: Valid value for the enumerated type Term

12. Attribute: capstoneEnrolled
Type: String
Description: The term which the student begins capstone program
Constraint: Valid value for the enumerated type Term

13. Attribute: classStatus
Type: String
Description: Current class categorization for the student, e.g. Freshman, Sophomore, etc.
Constraint: Valid value in the enumerated type ClassStatus

14. Attribute: graduateAssistant
Type: Boolean
Description: Flag to indicate if the student is a graduate assistant
Constraint: True or False

15. Attribute: international

Type: Boolean
Description: Flag to indicate if the student is an international student
Constraint: True or False

16. Attribute: internationalStatus
Type: String
Description: Value to indicate if the student is an international student, what type of international student they are.
Constraint: Must have a value of "None" if IsInternational=False. May not have a value of "None" if IsInternational=True.

17. Attribute: resident
Type: Boolean
Description: Flag to indicate if the student is a resident of South Carolina
Constraint: True or False

18. Attribute: activeDuty
Type: Boolean
Description: Flag to indicate if the student is active duty military
Constraint: True or False

19. Attribute: veteran
Type: Boolean
Description: Flag to indicate if the student is a family member of a veteran
Constraint: True or False

20. Attribute: tuitionFree
Type: Boolean
Description: Flag to indicate if entire tuition charges should be waived
Constraint: True or False

21. Attribute: scholarship
Type: String
Description: Value of the scholarship, if any, the student is entitled to.
Constraint: Valid value in enumerated type Scholarship. If IsResident=True then Scholarship must equal "None".

22. Attribute: StudyAbroad
Type: StudyAbroad
Description: Value to indicate if the student is studying abroad and if so what type of abroad status
Constraint: Valid value in enumerated type StudyAbroad

23. Attribute: nationalStudentExchange
Type: Boolean
Description: Flag to indicate if the student is an exchange student
Constraint: True or False

24. Attribute: outsideInsurance
Type: Boolean
Description: Flag to indicate if the student has health insurance through a 3rd party
Constraint: True or False

25. Attribute: courses
Type: Array of Courses
Description: List of course objects associated with the Student Record
Constraint: None

26. Attribute: transaction

Type: Array of Transactions
Description: List of Transaction objects associated with the Student Record
Constraint: None

#### 4.3.2.3.2 Method Descriptions

1. Method: setId(id)
   Return Type: void
   Parameters: id
   Return value: no return value
   Pre-condition: none
   Post-condition: id is associated to the Student Record
   Attributes read/updated: id
   Processing logic: set/update the id associated to Student Record

2. Method: setFirstName(firstName)
   Return Type: void
   Parameters: firstName
   Return value: no return value
   Pre-condition: none
   Post-condition: firstName is associated to the Student Record
   Attributes read/updated: firstName
   Processing logic: set/update the firstName associated to Student Record

3. Method: setLastName(lastName)
   Return Type: void
   Parameters: lastName
   Return value: no return value
   Pre-condition: none
   Post-condition: lastName is associated to the Student Record
   Attributes read/updated: lastName
   Processing logic: set/update the lastName associated to Student Record

4. Method: setPhone(phone)
   Return Type: void
   Parameters: phone
   Return value: no return value
   Pre-condition: none
   Post-condition: phone is associated to the Student Record
   Attributes read/updated: phone
   Processing logic: set/update the phone associated to Student Record

5. Method: setEmailAddress(emailAddress)
   Return Type: void
   Parameters: emailAddress
   Return value: no return value
   Pre-condition: None
   Post-condition: emailAddress is associated to the Student Record
   Attributes read/updated: emailAddress
   Processing logic: set/update the emailAddress associated to Student Record

6. Method: setAddressStreet(addressStreet)
   Return Type: void

Parameters: addressStreet
Return value: no return value
Pre-condition: None
Post-condition: addressStreet is associated to the Student Record
Attributes read/updated: addressStreet
Processing logic: set/update the addressStreet associated to Student Record

7. Method: setAddressState(addressState)
   Return Type: void
   Parameters: addressState
   Return value: no return value
   Pre-condition: None
   Post-condition: addressState is associated to the Student Record
   Attributes read/updated: addressState
   Processing logic: set/update the addressState associated to Student Record

8. Method: setAddressCity(addressCity)
   Return Type: void
   Parameters: addressCity
   Return value: no return value
   Pre-condition: None
   Post-condition: addressCity is associated to the Student Record
   Attributes read/updated: addressCity
   Processing logic: set/update the addressCity associated to Student Record

9. Method: setAddressPostalCode(addressPostalCode)
   Return Type: void
   Parameters: addressPostalCode
   Return value: no return value
   Pre-condition: None
   Post-condition: addressPostalCode is associated to the Student Record
   Attributes read/updated: addressPostalCode
   Processing logic: set/update the addressPostalCode associated to Student Record

10. Method: setCollege(college)
    Return Type: void
    Parameters: college
    Return value: no return value
    Pre-condition: None
    Post-condition: college is associated to the Student Record
    Attributes read/updated: college
    Processing logic: set/update the college associated to Student Record

11. Method: setTermBegan(termBegan)
    Return Type: void
    Parameters: termBegan
    Return value: no return value
    Pre-condition: None
    Post-condition: termBegan is associated to the Student Record
    Attributes read/updated: termBegan
    Processing logic: set/update the termBegan associated to Student Record

12. Method: setCapstoneEnrolled(capstoneEnrolled)

Return Type: void
Parameters: capstoneEnrolled
Return value: no return value
Pre-condition: None
Post-condition: capstoneEnrolled is associated to the Student Record
Attributes read/updated: capstoneEnrolled
Processing logic: set/update the capstoneEnrolled associated to Student Record

13. Method: setClassStatus(classStatus)
    Return Type: void
    Parameters: classStatus
    Return value: no return value
    Pre-condition: None
    Post-condition: classStatus is associated to the Student Record
    Attributes read/updated: classStatus
    Processing logic: set/update the classStatus associated to Student Record

14. Method: setGraduateAssistant(graduateAssistant)
    Return Type: void
    Parameters: graduateAssistant
    Return value: no return value
    Pre-condition: None
    Post-condition: graduateAssistant is associated to the Student Record
    Attributes read/updated: graduateAssistant
    Processing logic: set/update the graduateAssistant associated to Student Record

15. Method: setInternational(international)
    Return Type: void
    Parameters: international
    Return value: no return value
    Pre-condition: None
    Post-condition: international is associated to the Student Record
    Attributes read/updated: international
    Processing logic: set/update the international associated to Student Record

16. Method: setInternationalStatus(internationalStatus)
    Return Type: void
    Parameters: internationalStatus
    Return value: no return value
    Pre-condition: None
    Post-condition: internationalStatus is associated to the Student Record
    Attributes read/updated: internationalStatus
    Processing logic: set/update the internationalStatus associated to Student Record

17. Method: setResident(resident)
    Return Type: void
    Parameters: resident
    Return value: no return value
    Pre-condition: None
    Post-condition: resident is associated to the Student Record
    Attributes read/updated: resident
    Processing logic: set/update the resident associated to Student Record

18. Method: setFreeTuition(freeTuition)
    Return Type: void
    Parameters: freeTuition
    Return value: no return value
    Pre-condition: None
    Post-condition: freeTuition is associated to the Student Record
    Attributes read/updated: freeTuition
    Processing logic: set/update the freeTuition associated to Student Record

19. Method: setVeteran(veteran)
    Return Type: void
    Parameters: veteran
    Return value: no return value
    Pre-condition: None
    Post-condition: veteran is associated to the Student Record
    Attributes read/updated: veteran
    Processing logic: set/update the veteran associated to Student Record

20. Method: setActiveDuty(activeDuty)
    Return Type: void
    Parameters: activeDuty
    Return value: no return value
    Pre-condition:
    Post-condition: activeDuty is associated to the Student Record
    Attributes read/updated: activeDuty
    Processing logic: set/update the activeDuty associated to Student Record

21. Method: setScholarship(scholarship)
    Return Type: void
    Parameters: scholarship
    Return value: no return value
    Pre-condition: None
    Post-condition: scholarship is associated to the Student Record
    Attributes read/updated: scholarship
    Processing logic: set/update the scholarship associated to Student Record

22. Method: setStudyAbroadStatus(studyAbroadStatus)
    Return Type: void
    Parameters: studyAbroadStatus
    Return value: no return value
    Pre-condition: None
    Post-condition: studyAbroadStatus is associated to the Student Record
    Attributes read/updated: studyAbroadStatus
    Processing logic: set/update the studyAbroadStatus associated to Student Record

23. Method: setNationalStudentExchange(nationalStudentExchange)
    Return Type: void
    Parameters: nationalStudentExchange
    Return value: no return value
    Pre-condition: None
    Post-condition: nationalStudentExchange is associated to the Student Record
    Attributes read/updated: nationalStudentExchange
    Processing logic: set/update the nationalStudentExchange associated to Student Record

24. Method: setOutsideInsurance (outsideInsurance)
    Return Type: void
    Parameters: outsideInsurance
    Return value: no return value
    Pre-condition: None
    Post-condition: outsideInsurance is associated to the Student Record
    Attributes read/updated: outsideInsurance
    Processing logic: set/update the outsideInsurance associated to Student Record

25. Method: setCourses(Course[*])
    Return Type: void
    Parameters: Array of Courses
    Return value: no return value
    Pre-condition: None
    Post-condition: Course Array is associated to the Student Record
    Attributes read/updated: Course [1..*]
    Processing logic: set/update the Courses associated to Student Record

26. Method: setTransaction(Transaction[*])
    Return Type: void
    Parameters: Array of Transactions
    Return value: no return value
    Pre-condition: None
    Post-condition: Transaction Array is associated to the Student Record
    Attributes read/updated: Transaction [1..*]
    Processing logic: set/update the Transaction Array associated to Student Record

27. Method: getId()
    Return Type: String
    Parameters: no parameter
    Return value: id
    Pre-condition: None
    Post-condition: None
    Attributes read/updated: id
    Processing logic: return the id associated to Student Record

28. Method: getFirstName()
    Return Type: String
    Parameters: no parameter
    Return value: firstName
    Pre-condition: None
    Post-condition: None
    Attributes read/updated: firstName
    Processing logic: return the firstName associated to Student Record

29. Method: getLastName()
    Return Type: String
    Parameters: no parameter
    Return value: lastName
    Pre-condition: None
    Post-condition: None
    Attributes read/updated: lastName

Processing logic: return the lastName associated to Student Record

30. Method: getPhone()
Return Type: String
Parameters: no parameter
Return value: phone
Pre-condition: None
Post-condition: None
Attributes read/updated: phone
Processing logic: return the phone associated to Student Record

31. Method: getEmailAddress()
Return Type: String
Parameters: no parameter
Return value: emailAddress
Pre-condition: None
Post-condition: None
Attributes read/updated: emailAddress
Processing logic: return the emailAddress associated to Student Record

32. Method: getAddressStreet()
Return Type: String
Parameters: no parameter
Return value: addressStreet
Pre-condition: None
Post-condition: None
Attributes read/updated: addressStreet
Processing logic: return the addressStreet associated to Student Record

33. Method: getAddressState()
Return Type: String
Parameters: no parameter
Return value: addressState
Pre-condition: None
Post-condition: None
Attributes read/updated: addressState
Processing logic: return the addressState associated to Student Record

34. Method: getAddressCity()
Return Type: String
Parameters: no parameter
Return value: addressCity
Pre-condition: None
Post-condition: None
Attributes read/updated: addressCity
Processing logic: return the addressCity associated to Student Record

35. Method: getAddressPostalCode()
Return Type: String
Parameters: no parameter
Return value: addressPostalCode
Pre-condition: None
Post-condition: a None

Attributes read/updated: addressPostalCode
Processing logic: return the addressPostalCode associated to Student Record

36. Method: getCollege()
    Return Type: String
    Parameters:  no parameter
    Return value: college
    Pre-condition: None
    Post-condition: None
    Attributes read/updated: college
    Processing logic: return the college associated to Student Record

37. Method: getTermBegan()
    Return Type: String
    Parameters:  no parameter
    Return value: termBegan
    Pre-condition: None
    Post-condition: None
    Attributes read/updated: termBegan
    Processing logic: return the termBegan associated to Student Record

38. Method: getCapstoneEnrolled()
    Return Type: String
    Parameters:  no parameter
    Return value: capstoneEnrolled
    Pre-condition: None
    Post-condition: capstoneEnrolled is associated to the Student Record
    Attributes read/updated: capstoneEnrolled
    Processing logic: return the capstoneEnrolled associated to Student Record

39. Method: getClassStatus()
    Return Type: String
    Parameters:  no parameter
    Return value: classStatus
    Pre-condition: None
    Post-condition: None
    Attributes read/updated: classStatus
    Processing logic: return the classStatus associated to Student Record

40. Method: getGraduateAssistant()
    Return Type: Boolean
    Parameters:  no parameter
    Return value: graduateAssistant
    Pre-condition: None
    Post-condition: None
    Attributes read/updated: graduateAssistant
    Processing logic: return the graduateAssistant associated to Student Record

41. Method: getInternational()
    Return Type: Boolean
    Parameters:  no parameter
    Return value: international
    Pre-condition: None

Post-condition: None
Attributes read/updated: international
Processing logic: return the international associated to Student Record

42. Method: getInternationalStatus()
Return Type: InternationalStatus
Parameters:  no parameter
Return value: internationalStatus
Pre-condition: None
Post-condition: None
Attributes read/updated: internationalStatus
Processing logic: return the internationalStatus associated to Student Record

43. Method: getResident()
Return Type: Boolean
Parameters:  no parameter
Return value: resident
Pre-condition: None
Post-condition: None
Attributes read/updated: resident
Processing logic: return the resident associated to Student Record

44. Method: getFreeTuition()
Return Type: Boolean
Parameters:  no parameter
Return value: no return value
Pre-condition: None
Post-condition: None
Attributes read/updated: freeTuition
Processing logic: return the freeTuition associated to Student Record

45. Method: getVeteran()
Return Type: Boolean
Parameters:  no parameter
Return value: veteran
Pre-condition: None
Post-condition: None
Attributes read/updated: veteran
Processing logic: return the veteran associated to Student Record

46. Method: getActiveDuty()
Return Type: Boolean
Parameters:  no parameter
Return value: activeDuty
Pre-condition:
Post-condition: None
Attributes read/updated: activeDuty
Processing logic: return the activeDuty associated to Student Record

47. Method: getScholarship()
Return Type: String
Parameters:  no parameter
Return value: scholarship

Pre-condition: None
Post-condition: None
Attributes read/updated: scholarship
Processing logic: return the scholarship associated to Student Record

48. Method: getStudyAbroad()
Return Type: String
Parameters: no parameter
Return value: studyAbroad
Pre-condition: None
Post-condition: None
Attributes read/updated: studyAbroadStatus
Processing logic: return the studyAbroadStatus associated to Student Record

49. Method: getNationalStudentExchange()
Return Type: Boolean
Parameters: no parameter
Return value: nationalStudentExchange
Pre-condition: None
Post-condition: None
Attributes read/updated: nationalStudentExchange
Processing logic: return the nationalStudentExchange associated to Student Record

50. Method: getOutsideInsurance ()
Return Type: Boolean
Parameters: no parameter
Return value: outsideInsurance
Pre-condition: None
Post-condition: None
Attributes read/updated: outsideInsurance
Processing logic: return the outsideInsurance associated to Student Record

51. Method: getCourses()
Return Type: Array of Courses
Parameters: no parameter
Return value: courses
Pre-condition: None
Post-condition: None
Attributes read/updated: Course [1..*]
Processing logic: return the Courses associated to Student Record

52. Method: getTransaction()
Return Type: Array of Transactions
Parameters: no parameter
Return value: transaction
Pre-condition: None
Post-condition: None
Attributes read/updated: Transaction [1..*]
Processing logic: return the Transaction Array associated to Student Record

#### 4.3.2.4 Class: Transaction

- Purpose: Like all DTO objects, the Transaction exists to perform data transformation. In this case from data which represents payments and charges against a student's account.
- Constraints: None.
- Persistent: No, Transaction is a data transformation object

##### 4.3.2.4.1 Attribute Descriptions

1. Attribute: type
   Type: String
   Description: Categorization as to whether the transaction was a payment or charge
   Constraint: Valid value of the enumerated type TransactionType
2. Attribute: amount
   Type: Double
   Description: The dollar amount of the transaction
   Constraint: None
3. Attribute: transactionDto
   Type: TransactionDto
   Description: transactionDto representing a month, day, & year
   Constraint: Standard date constraints, e.g. Feb 30 disallowed, no month outside of 1-12, etc.
4. Attribute: Note
   Type: String
   Description: Ancillary text which may be associated with a transaction
   Constraint: None

##### 4.3.2.4.2 Method Descriptions

1. Method: set Type (type)
   Return Type: void
   Parameters: type
   Return value: no return value
   Pre-condition: none
   Post-condition: type is associated to the Transaction
   Attributes read/updated: type
   Processing logic: set/update the type associated to Transaction

2. Method: setAmount(amount)
   Return Type: void
   Parameters: amount
   Return value: no return value
   Pre-condition: none
   Post-condition: Amount is associated to the Transaction
   Attributes read/updated: amount
   Processing logic: set/update the amount associated to Transaction

3. Method: setTransactionDate (transactionDate)
   Return Type: void
   Parameters: transactionDate
   Return value: no return value
   Pre-condition: none
   Post-condition: transactionDate is associated to the Transaction
   Attributes read/updated: transactionDate

Processing logic: set/update the transactionDate associated to Transaction

4. Method: setNote(note)
   Return Type: void
   Parameters: note
   Return value: no return value
   Pre-condition: none
   Post-condition: reason is associated to the Transaction
   Attributes read/updated: reason
   Processing logic: set/update the reason associated to Transaction

5. Method: getType()
   Return Type: type
   Parameters: no parameter
   Return value: user
   Pre-condition: None
   Post-condition: None
   Attributes read/updated: type
   Processing logic: return the type associated with the Transaction

6. Method: getAmount()
   Return Type: String
   Parameters: no parameter
   Return value: amount
   Pre-condition: None
   Post-condition: None
   Attributes read/updated: amount
   Processing logic: return the amount associated to Transaction

7. Method: getTransactionDate()
   Return Type: String
   Parameters: no parameter
   Return value: transactionDate
   Pre-condition: None
   Post-condition: None
   Attributes read/updated: transactionDate
   Processing logic: return the transactionDate associated to Transaction

8. Method: getNote()
   Return Type: String
   Parameters: no parameter
   Return value: note
   Pre-condition: None
   Post-condition: None
   Attributes read/updated: note
   Processing logic: return the note associated to Transaction

### 4.3.3 Classes and Interfaces in the Repository package

#### 4.3.3.1 Interface: UserRepository

- Purpose: *To handle CRUD operations on user-related data*
- Constraints: *None*

- Persistent: *Not applicable*

#### 4.3.3.2 Attribute Descriptions

No attribute existed for this interface.

#### 4.3.3.3 Methods Descriptions

1. Method: *findOneById (id)*
   Return Type: *User*
   Parameters: *id*
   Return value: *null or user object/s associated with requested id*
   Pre-condition: *the length of id should be 8 characters consists of both letter and number*
   Post-condition: *null or user object/s associated with requested id*
   Attributes read/updated: *no attribute.*
   Processing logic: *not applicable*

2. Method: *findByUserRoles_Role (role)*
   Return Type: *User[*]*
   Parameters: *role*
   Return value: *0 or more of user object/s associated with requested role*
   Pre-condition: *no pre-condition*
   Post-condition: *0 or more of user object/s associated with requested role*
   Attributes read/updated: *no attribute*
   Processing logic: not applicable

3. Method: findByUserRoles_College(college)
   Return Type: *User*
   Parameters: *id*
   Return value: *user object associated with requested college.*
   Pre-condition: *no pre-condition*
   Post-condition:
   Attributes read/updated: *no attribute.*
   Processing logic: *not applicable*

4. Method: insert( user )
   Return Type: *void*
   Parameters: user
   Return value: persisted user.
   Pre-condition: *no pre-condition*
   Post-condition: user is persisted in datasource.
   Attributes read/updated: *no attribute.*
   Processing logic: *not applicable*

5. Method: update( user )
   Return Type: *void*
   Parameters: user
   Return value: persisted user.
   Pre-condition: *no pre-condition*
   Post-condition: user with the same id will be updated and persisted in datasource.
   Attributes read/updated: *no attribute.*
   Processing logic: *not applicable*

#### 4.3.3.4 Interface: StudentProfileRepository

- Purpose: *To handle CRUD operations on student profile-related data*
- Constraints: *None*
- Persistent: *Not applicable*

##### 4.3.3.4.1 Attribute Descriptions

No attribute existed for this interface.

##### 4.3.3.4.2 Methods Descriptions

1. Method: findOneByUser_UserId(id)
   Return Type: *User*
   Parameters: *id*
   Return value: user object associated with requested id.
   Pre-condition: *the length of id should be 8 characters consists of both letter and number*
   Post-condition:
   Attributes read/updated: *no attribute.*
   Processing logic: *not applicable*

2. Method: findByClassStatus (role)
   Return Type: *User[*]*
   Parameters: role
   Return value: user object/s associated with requested id
   Pre-condition: *no pre-condition*
   Post-condition: role is returned
   Attributes read/updated: *no attribute.*
   Processing logic: *not applicable*

3. Method: insert( studentProfile )
   Return Type: *StudentProfile*
   Parameters: studentProfile
   Return value: persisted studentProfile.
   Pre-condition: *object shouldn't be null*
   Post-condition: studentProfile is persisted in data source.
   Attributes read/updated: *no attribute.*
   Processing logic: *not applicable*

4. Method: *update( studentProfile )*
   Return Type: *StudentProfile*
   Parameters: studentProfile
   Return value: persisted studentProfile.
   Pre-condition: *object shouldn't be null*
   Post-condition: *studentProfile with the same user id will be updated and persisted in datasource.*
   Attributes read/updated: *no attribute.*
   Processing logic: *not applicable*

#### 4.3.3.5 Interface: PaymentHistoryRepository

- Purpose: *To handle CRUD operations on* PaymentHistory-*related data*
- Constraints: *None*
- Persistent: *Not applicable*

#### 4.3.3.5.1 Attribute Descriptions

No attribute existed for this interface.

#### 4.3.3.5.2 Methods Descriptions

1. Method: insert( paymentHistory )
   Return Type: PaymentHistory
   Parameters: paymentHistory
   Return value: persisted paymentHistory.
   Pre-condition: *object shouldn't be null*
   Post-condition: paymentHistory is persisted in data source.
   Attributes read/updated: *no attribute.*
   Processing logic: *not applicable*

2. Method: update( paymentHistory)
   Return Type: PaymentHistory
   Parameters: paymentHistory
   Return value: persisted paymentHistory.
   Pre-condition: *object shouldn't be null*
   Post-condition: paymentHistory with the same id will be updated and persisted in data source.
   Attributes read/updated: *no attribute.*
   Processing logic: *not applicable*

### 4.3.3.6 Interface: ChargeHistoryRepository

- Purpose: *To handle CRUD operations on* ChargeHistory*-related data*
- Constraints: *None*
- Persistent: *Not applicable*

#### 4.3.3.6.1 Attribute Descriptions

No attribute existed for this interface.

#### 4.3.3.6.2 Methods Descriptions

1. Method: findByUser_UserId ( userId )
   Return Type: *ChargeHistory*
   Parameters: course
   Return value: 0 or more of chargeHistory objects.
   Pre-condition: *the length of id should be 8 characters consists of both letter and number*
   Post-condition: 0 or more of ChargeHistory objects.
   Attributes read/updated: *no attribute.*
   Processing logic: *not applicable*

2. Method: findByUserIdAndDateBetween ( userId, years, semesters)
   Return Type: *Course*
   Parameters: userId, years, semesters
   Return value: 0 or more of chargeHistory objects
   Pre-condition: *the length of id should be 8 characters consists of both letter and number, each of years should consists of 4 integer number and validated, and each of semesters should not be null.*
   Post-condition: 0 or more of ChargeHistory objects.
   Attributes read/updated: *no attribute.*
   Processing logic: *not applicable*

3. Method: insert( chargeHistory )
   Return Type: *ChargeHistory*
   Parameters: chargeHistory
   Return value: persisted chargeHistory.
   Pre-condition: *object shouldn't be null*
   Post-condition: chargeHistory *is persisted in data source.*
   Attributes read/updated: *no attribute.*
   Processing logic: *not applicable*

4. Method: update( chargeHistory )
   Return Type: ChargeHistory
   Parameters: chargeHistory
   Return value: persisted chargeHistory.
   Pre-condition: *object shouldn't be null*
   Post-condition: chargeHistory with the same id will be updated and persisted in data source.
   Attributes read/updated: *no attribute.*
   Processing logic: *not applicable*

### 4.3.4 Classes and interfaces in the Service package

#### 4.3.4.1 Interface: UserManagementService

- Purpose: *To handle business operations that concerns collectively user and user role data.*
- Constraints: *None*
- Persistent: *Not applicable*

##### 4.3.4.1.1 Attribute Descriptions

No attribute existed for this interface.

##### 4.3.4.1.2 Methods Descriptions

1. Method: *loadUsers(filename)*
   Return Type: *void*
   Parameters: *filename*
   Return value: *no return value*
   Pre-condition: *no pre-condition*
   Post-condition: *JSON file associated with filename that is existed in the system's resources will be reloaded into datasource*
   Attributes read/updated: *not attribute*
   Processing logic: *not applicable*

2. Method: *login ( userId )*
   Return Type: *void*
   Parameters: *userId*
   Return value: *no return value*
   Pre-condition: *no pre-condition*
   Post-condition: *User associated with userId will be logged-in*
   Attributes read/updated: *not applicable*
   Processing logic: *not applicable*

3. Method: *logout ()*
   Return Type: *void*

Parameters: *no parameter*
Return value: *no return value*
Pre-condition: *no pre-condition*
Post-condition: *the temporarily logged-in session of the user will be deleted*
Attributes read/updated: *not applicable*
Processing logic: *not applicable*

4. Method: *getUserIdsUnderAdminPrivilidge ()*
   Return Type: *String[*]*
   Parameters: *userId*
   Return value: 0 or more of *userIds that is accessible by the logged-in admin user*
   Pre-condition: *no pre-condition*
   Post-condition: 0 or more of *userIds that is accessible by the logged-in admin user*
   Attributes read/updated: *userRepository*
   Processing logic: *not applicable*

5. Method: *getLoggedInUserId()*
   Return Type: *User*
   Parameters: *no parameter*
   Return value: *loggedInUser*
   Pre-condition: *an existing logged-in user, loggedInUser shouldn't be null*
   Post-condition: *userId of loggedInUser object*
   Attributes read/updated:
   Processing logic: *not applicable*

#### 4.3.4.2 Class: UserManagementServiceImpl

- Purpose: *To handle business operations that concerns collectively user and user role data.*
- Constraints: *it should implement UserManagementService Interface*
- Persistent: *No persistent*

##### 4.3.4.2.1 Attribute Descriptions

1. Attribute: *loggedInUser*
   Type: *User*
   Description: *object of user who is logged-in*
   Constraints: *no constraint*

2. Attribute: *userRepository*
   Type: *UserRepository*
   Description: *instance of UserRepository*
   Constraints: *should be annotated as @autowired in order the object to be initiated*

3. Attribute: *studentManagementService*
   Type: *StudentManagementService*
   Description: *instance of StudentManagementService*
   Constraints: *should be annotated as @autowired in order the object to be initiated*

##### 4.3.4.2.2 Methods Descriptions

1. Method: *loadUsers(filename)*
   Return Type: *void*
   Parameters: *filename*
   Return value: *no return value*

Pre-condition: *no pre-condition*
Post-condition: *JSON file associated with filename that is existed in the system's resources will be reloaded into datasource through UserRepository.*
Attributes read/updated: *userRepository*
Processing logic:

- *Check the resource folder of system, and retrieve the JSON file associated with the filename*
- *Map it to user object/s, and persisting them sequentially through UserRepository.*
- *If no JSON file associated with requested filename existed, FileNotfoundException will be thrown*

2. Method: *login ( userId )*
Return Type: *void*
Parameters: *userId*
Return value: *no return value*
Pre-condition: *no pre-condition*
Post-condition: *User associated with userId will be logged-in*
Attributes read/updated: *loggedInUser, userRepository*
Processing logic:
    - *Retrieve the user associated with userId through UserRepository*
    - *Set/update the loggedInUser. In case user not returned, UserNotFoundException will be thrown*

3. Method: *logout ()*
Return Type: *void*
Parameters: *no parameter*
Return value: *no return value*
Pre-condition: *no pre-condition*
Post-condition: *the temporarily logged-in session of the user will be deleted*
Attributes read/updated: *loggedInUser, studentManagementService*
Processing logic: *set/update loggedInUser object to null and temporaryStudentProfile to null through StudentManagmentService.*

4. Method: *getUserIdsUnderAdminPrivilidge ()*
Return Type: *String[*]*
Parameters: *userId*
Return value: 0 or more of *userIds that is accessible by the logged-in admin user*
Pre-condition: *no pre-condition*
Post-condition: 0 or more of *userIds that is accessible by the logged-in admin user* Attributes read/updated: *userRepository*
Processing logic:
    - Validate the role of loggedInUser as an admin, and retrieving all the users whose role is student through userRepository
    - Compare classStatus and college attributes associated to student with the college attribute associated to loggedInUser.
    - In case of logged-in user's is not admin, no users will be returned.
    - In case of no users existed under logged-in user privilege, also no users will be returned

5. Method: *getLoggedInUserId()*
Return Type: *User*
Parameters: *no parameter*

Return value: *loggedInUser*
Pre-condition: *an existing logged-in user, loggedInUser shouldn't be null*
Post-condition: *userId of loggedInUser object*
Attributes read/updated: *loggedInUser*
Processing logic: *returning the userId of loggedInUser object*. In case *loggedInUser* is equal to null, UserNotFoundException will be thrown.

6. Method: *getUserRoles (userId)*
   Return Type: *UserRole[1..*]*
   Parameters: *userId*
   Return value: 1 or more of *userRoles will be returned*
   Pre-condition: *no pre-condition*
   Post-condition: 1 or more of *userRoles will be returned*
   Attributes read/updated: *userRepository*
   Processing logic:
   - *Retrieve 1 or more roles of requested userId through UserRepository.*
   - In case no role is returned, UserNotFoundException will be thrown.

7. Method: *checkAccess (userId)*
   Return Type: *boolean*
   Parameters: loggedInUser, *userId*
   Return value: *true or false*
   Pre-condition: *the length of userId should be 8 characters, and consists of both letter and number*
   Post-condition: *true or false*
   Attributes read/updated: *userRepository*
   Processing logic:
   - Check whether if id of loggedInUser is equal to userId. If they are not equal, checking the role of loggedInUser.
   - If the role is equal to ADMIN, retrieve the StudentProfile object associated with userId through userRepository
   - Compare classStatus and college attributes associated to student with the college attribute associated to admin user
   - In case of either loggedInUser equal to null or logged-in user has no privilege to access student record or there is no student profile with userId, returning false. Otherwise, returning true

#### 4.3.4.3 Class: StudentManagementService

- Purpose: *To handle business operations that concerns student profile data.*
- Constraints: *no constraint*
- Persistent: *No persistent*

##### 4.3.4.3.1 Attribute Descriptions

*No attribute*

##### 4.3.4.3.2 Methods Descriptions

1. Method: loadStudentProfiles *(filename)*
   Return Type: *void*
   Parameters: *filename*
   Return value: *no return value*
   Pre-condition: *no pre-condition*

Post-condition: *JSON file associated with filename that is existed in the system's resources will be reloaded into datasource through StudentProfileRepository.*
Attributes read/updated: *no attribute*
Processing logic: *not applicable*

2. Method: getStudentProfile *(userId)*
   Return Type: *StudentRecord*
   Parameters: *userId*
   Return value: *studentRecord associated with the userId*
   Pre-condition: *no pre-condition*
   Post-condition: *studentRecord associated with the userId returned*
   Attributes read/updated: *no attribute*
   Processing logic: *not applicable*

3. Method: updateStudentProfile *(userId, studentRecord, permanent)*
   Return Type: *StudentRecord*
   Parameters: *userId, studentRecord, permenant*
   Return value: not return value
   Pre-condition: *no pre-condition*
   Post-condition: studentRecord associated with userId is updated
   Attributes read/updated: *no attribute*
   Processing logic: *not applicable*

#### 4.3.4.4 Class: StudentManagementServiceImpl

- Purpose: *To handle business operations that concerns student profile data.*
- Constraints: *it should implement StudentProfileService Interface and should be annotated as @Component in order class to be autowired by other classes*
- Persistent: *No persistent*

##### 4.3.4.4.1 Attribute Descriptions

1. Attribute: *tempStudentProfile*
   Type: *User*
   Description: *student profile that is not persisted in datasource, rather, temporarily in memory during the session of logged-in user*
   Constraints: *no constraint*
2. Attribute: *studentProfileRepository*
   Type: *StudentProfileRepository*
   Description: instance of *StudentProfileRepository*
   Constraints: should be annotated as @autowired in order the object to be initiated
3. Attribute: userManagementService
   Type: UserManagementService
   Description: instance of UserManagementService
   Constraints: *should be annotated as @autowired in order the object to be initiated*
4. Attribute: *billService*
   Type: *BillService*
   Description: instance of *BillService*
   Constraints: should be annotated as @autowired in order the object to be initiated

##### 4.3.4.4.2 Methods Descriptions

1. Method: *loadStudentProfiles (filename)*
   Return Type: *void*
   Parameters: *filename*

Return value: *no return value*
Pre-condition: *no pre-condition*
Post-condition: *JSON file associated with filename that is existed in the system's resources will be reloaded into datasource*
Attributes read/updated: *userRepository*
Processing logic:

- *Declare list of student profile*
- *Check the resource folder of system, and retrieve the JSON file associated with the filename*
- *Map jsonArray to StudentProfile object/s through statically DataMapper*
- *Persist them sequentially through StudentProfileRepository*
- *If no JSON file associated with requested file name, FileNotfoundException will be thrown*

2. Method: getStudentProfile *(userId)*
Return Type: *StudentRecord*
Parameters: *userId*
Return value: *studentRecord associated with the userId*
Pre-condition: *no pre-condition*
Post-condition: *studentRecord associated with the userId returned*
Attributes read/updated: *studentProfileRepository, userManagementService,* billService
Processing logic:
    - Validate the logged-in user has the privilege to access student profile associated with requested user id through *userManagementService*
    - Retrieve the student profile associated with userId through studentProfileRepository
    - Retrieve all currentCharges of student through billService
    - Retrieve chargeHistory of student through billService
    - Retrieve all paymentHistory of student through billService
    - Get the balance through billService
    - Map the studentProfile, currentCharges, chargeHistory, paymentHistory through statically DataMapper, and return the studentRecord
    - In case of the logged-in user has no privilege to access student record, UserPermissionException will be thrown
    - In case of no student profile existed associated with requested userId, NoStudentProfileFoundException will be thrown

3. Method: *updateStudentProfile (userId, studentRecord, permanent)*
Return Type: *StudentRecord*
Parameters: *userId, studentRecord, permenant*
Return value: not return value
Pre-condition: *no pre-condition*
Post-condition: studentRecord associated with userId is updated
Read/updated: *studentProfileRepository, userManagementService, tempStudentProfile*
Processing logic:
    - Validate that userId match with the id of studentRecords
    - Validate the logged-in user has the privilege to access student profile associated with requested user id through userManagementService
    - Retrieve the student profile associated with userId through studentProfileRepository
    - Validate UserRoles's Roles of studentProfile match with roles of studentRecords

- Map the studentRecord through statically DataMapper, and returning the studentProfile
- Check permanent Boolean value, if true, persist the studentProfile record through StudentProfileRepository, otherwise, updating the local variable tempStudentProfile
- In case of the id and roles attribute doesn't match in both studentRecord and studentProfile, UserPermissionException will be thrown
- In case of the logged-in user has no privilege to access student record, UserPermissionException will be thrown
- In case of no student profile existed associated with requested userId, NoStudentProfileFoundException will be thrown

4. Method: setTempStudentPofile ( studentProfile )
   Return Type: *void*
   Parameters: studentProfile
   Return value: no return value
   Pre-condition: no pre-condition
   Post-condition: tempStudentProfile is updated
   Attributes read/updated: tempStudentProfile
   Processing logic: *set/update* tempStudentProfile

5. Method: getTempStudentProfile ()
   Return Type: *StudentProfile*
   Parameters: no parameter
   Return value: StudentProfile object
   Pre-condition: *middleName shouldn't be empty and be consisted of only letters.*
   Post-condition: tempStudentProfile returned.
   Attributes read/updated: tempStudentProfile
   Processing logic: *return* tempStudentProfile

#### 4.3.4.5 Interface: BillService

- Purpose: *To handle business operations that concerns generating charges and applying payment for student.*
- Constraints: *no constraint*
- Persistent: *No persistent*

##### 4.3.4.5.1 Attribute Descriptions

*No attribute*

##### 4.3.4.5.2 Methods Descriptions

1. Method: loadTransactions (filename)
   Return Type: Transactions[*]
   Parameters: *jsonArray*
   Return value: list of Transaction objects
   Pre-condition: *no pre-condition*
   Post-condition: list of Transactions objects
   Attributes read/updated: *no attribute*
   Processing logic: *no attribute*

2. Method: getBill (userId)
   Return Type: Bill
   Parameters: userId

Return value: bill of student associated with userId
Pre-condition: no pre-condition
Post-condition: bill of student associated with userId will be returned
Attributes read/updated: *no attribute*
Processing logic: *Not applicable*

*3.* Method*: getCharges (userId, startDate, endDate)*
*Return Type: Bill*
*Parameters: userId*
*Return value: bill of student associated with userId*
*Pre-condition: no pre-condition*
*Post-condition: bill of student associated with userId will be returned*
*Attributes read/updated: no attribute*
*Processing logic: not applicable*

4. Method: geCurrentCharges (userId)
Return Type: Map<String, double>
Parameters: userId
Return value: map of currentCharges which consists of note and the amount
Pre-condition: no pre-condition
Post-condition: map of currentCharges which consists of note and the amount will be returned
Attributes read/updated: *no attribute*
Processing logic: *Not applicable*

5. Method: *geChargesHistory (userId, startDate, endDate)*
Return Type: *ChargesHistory[*]*
Parameters: *userId*
Return value: *0 or more of ChargesHistory objects*
Pre-condition: *no pre-condition*
Post-condition: *studentRecord associated with the userId*
Attributes read/updated: *no attribute*
Processing logic: *Not applicable*

6. Method: *getChargesHistory (userId, startDate, endDate)*
Return Type: *ChargesHistory[*]*
Parameters: *userId*
Return value: *0 or more of ChargesHistory objects*
Pre-condition: *no pre-condition*
Post-condition: *studentRecord associated with the userId*
Attributes read/updated: *no attribute*
Processing logic: *Not applicable*

7. Method: *getPaymentHistory (userId, startDate, endDate)*
Return Type: *PaymentHistory*
Parameters: *userId, startDate, endDate*
Return value: *0 or more of PaymentHistory objects*
Pre-condition: *no pre-condition*
Post-condition: *userId*
Attributes read/updated: *no attribute*
Processing logic: *Not applicable*

8. Method: *getPaymentHistory (userId, startDate, endDate)*

Return Type: *PaymentHistory*
Parameters: *userId, startDate, endDate*
Return value: *0 or more of PaymentHistory objects*
Pre-condition: *no pre-condition*
Post-condition: *userId*
*Attributes read/updated: no attribute*
*Processing logic: Not applicable*

9. Method: *getBalance (currentCharges, chargeHistory, paymentHistory)*
   Return Type: *double*
   Parameters: *currentCharges, chargeHistory, paymentHistory*
   Return value: *balance*
   Pre-condition: *no pre-condition*
   Post-condition: *balance amount will be returned*
   Attributes read/updated: *no attribute*
   Processing logic: *Not applicable*

10. Method: *getBalance (currentCharges, paymentHistory)*
    Return Type: *double*
    Parameters: *currentCharges, paymentHistory*
    Return value: *balance*
    Pre-condition: *no pre-condition*
    Post-condition: *balance amount will be returned*
    *Attributes read/updated: no attribute*
    *Processing logic: Not applicable*

11. Method: addPayment (userId, amount, note)
    *Return Type: void*
    *Parameters: userId, amount, note*
    *Return value: no return value*
    *Pre-condition: no pre-condition*
    *Post-condition: payment with requested amount and note along with date will be persisted*
    *Attributes read/updated: no attribute*
    *Processing logic: Not applicable*

#### 4.3.4.6 Class: BillServiceImpl

- Purpose: *To handle business operations that concerns generating charges and applying payment for student.*
- Constraints: *it should implement BillService Interface and should be annotated as @Component in order for the class to be autowired by other classes*
- Persistent: *No persistent*

##### 4.3.4.6.1 Attribute Descriptions

1. Attribute: *userManagmentService*
   Type: *UserManagmentService*
   Description: *instance of* UserManagementService
   Constraints: *should be annotated as @autowired in order the object to be initiated.*
2. Attribute: *studentManagmentService*
   Type: *StudentManagmentService*
   Description: *instance of StudentManagmentSerivce*
   Constraints: *should be annotated as @autowired in order the object to be initiated.*
5. Attribute: *chargeHistoryRepository*

Type: *ChargeHistoryRepository*
Description: *instance of ChargeHistoryRepository*
Constraints: should be annotated as @autowired in order the object to be initiated

6. Attribute: paymentHistoryRepository
Type: PaymentHistoryRepository
Description: instance of PaymentHistoryRepository
Constraints: *should be annotated as @autowired in order the object to be initiated*

### 4.3.4.6.2 Methods Descriptions

1. Method: loadTransactions (filename)
Return Type: void
Parameters: *filename*
Return value: no return value
Pre-condition: *no pre-condition*
Post-condition: list of Transactions objects loaded and persisted into the datasource
Attributes read/updated: *no attribute*
Processing logic:
    - *Declare* list of ChargeHistory and PaymentHistory object
    - *Check the resource folder of system, and retrieve the JSON file associated with the filename*
    - *Map ChargeHistory and PaymentHistory through statically DataMapper which in turn return list of ChargeHistory and list of PaymentHistory*
    - *Persist List of ChargeHistory sequentially through chargeHistoryRepository*
    - *Persist List of* PaymentHistory *sequentially through* paymentHistory *Repository*

2. Method: getBill (userId)
Return Type: Bill
Parameters: userId
Return value: bill of student associated with userId
Pre-condition: no pre-condition
Post-condition: bill of student associated with userId will be returned
Attributes read/updated: studentProfileRepository, userManagementService
Processing logic:
    - Validate the logged-in user has the privilege to access student profile associated with requested userId through userManagementService
    - Retrieve student profile associated with userId through studentManagementService Check if the classStatus of the studentProfile is not "Graduated", and call the ChargeCalculator to calculate current tuition and fees and finally return the map of charges
    - Retrieve the start date and end date that falls in the range of current date from the file properties that contains list of semester's start date and end date
    - Retrieve the payment history through locally getPaymentHistory that falls within startDate and EndDate
    - Map the current charges and paymentHistory along with student profile through statically DataMapper which in turn return Bill
    - Return Bill
    - In case of the logged-in user has no privilege to access student record, UserPermissionException will be thrown.
    - In case of both no payment history record existed nor current charge, NoBillExistedException will be thrown

3. Method: getCharges (userId, startDate, endDate)

Return Type: Bill
Parameters: userId
Return value: bill of student associated with userId
Pre-condition: no pre-condition
Post-condition: bill of student associated with userId will be returned
Attributes read/updated: userManagementService, studentManagementService
Processing logic:

- *Validate the logged-in user has the privilege to access student profile associated with requested userId through userManagementService*
- *Check if the current date falls in the range between start date and end date, if true, retrieve student profile associated with userId through studentManagementService and check if the classStatus of the studentProfile is not "Graduated"*
- *AND, retrieve the current charges through locally getCurrentCharges*
- *Retrieve the payment history through locally getPaymentHistory that falls within startDate and endDate*
- *Retrieve the charge history through locally getChargeHistory that falls within startDate and endDate*
- *Map the current charges, chargeHistory and paymentHistory along with student profile through statically DataMapper which in turn return Bill*
- *Return Bill*
- *In case of the logged-in user has no privilege to access student record, UserPermissionException will be thrown.*
- *In case of both no payment history record existed and no current charge, NoBillExistedException will be thrown*

4. Method: geCurrentCharges (userId)
Return Type: Map<String, double>
Parameters: userId
Return value: map of currentCharges which consists of note and the amount
Pre-condition: no pre-condition
Post-condition: map of currentCharges which consists of note and the amount will be returned
Attributes read/updated: studentProfileRepository, userManagementService
Processing logic:
    - Validate the logged-in user has the privilege to access student profile associated with requested userId through userManagementService
    - Retrieve student profile associated with userId, check if the classStatus of the studentProfile is not "Graduated"
    - Call the ChargeCalculator to calculate current tuition and fees and finally return the map of charges
    - In case of the logged-in user has no privilege to access student record, UserPermissionException will be thrown.

5. Method: *geChargesHistory (userId, startDate, endDate)*
Return Type: *ChargesHistory[*]*
Parameters: *userId*
Return value: *0 or more of ChargesHistory objects*
Pre-condition: *no pre-condition*
Post-condition: *studentRecord associated with the userId*
Attributes read/updated: *userManagementService,* chargeHistoryRepository
Processing logic:

- Validate the logged-in user has the privilege to access student profile associated with requested userId through userManagementService
- Retrieve all chargeHistory objects associated with userId through chargeHistoryRepository and return them
- In case of the logged-in user has no privilege to access student record, UserPermissionException will be thrown

6. Method: *getChargesHistory (userId, startDate, endDate)*
   Return Type: *ChargesHistory[*]*
   Parameters: *userId*
   Return value: *0 or more of ChargesHistory objects*
   Pre-condition: *no pre-condition*
   Post-condition: *studentRecord associated with the userId*
   Attributes read/updated: *userManagementService,* chargeHistoryRepository
   Processing logic:
   - Validate the logged-in user has the privilege to access student profile associated with requested userId through userManagementService
   - Retrieve all chargeHistory objects associated with userId through chargeHistoryRepository that falls within startDate and endDate and and return them
   - In case of the logged-in user has no privilege to access student record, UserPermissionException will be thrown

7. Method: *getPaymentHistory (userId, startDate, endDate)*
   Return Type: *PaymentHistory*
   Parameters: *userId, startDate, endDate*
   Return value: *0 or more of PaymentHistory objects*
   Pre-condition: *no pre-condition*
   Post-condition: *userId*
   Attributes read/updated: paymentHistoryRepository*, userManagementService*
   Processing logic:
   - Validate the logged-in user has the privilege to access student profile associated with requested userId through userManagementService
   - Retrieve all paymentHistory objects through paymentHistoryRepository that falls within startDate and endDate and return them.
   - In case of the logged-in user has no privilege to access student record, UserPermissionException will be thrown

8. Method: *getPaymentHistory (userId, startDate, endDate)*
   Return Type: *PaymentHistory*
   Parameters: *userId, startDate, endDate*
   Return value: *0 or more of PaymentHistory objects*
   Pre-condition: *no pre-condition*
   Post-condition: *userId*
   Attributes read/updated: paymentHistoryRepository*, userManagementService*
   Processing logic:
   - Validate the logged-in user has the privilege to access student profile associated with requested userId through userManagementService
   - Retrieve all paymentHistory objects through paymentHistoryRepository and return them.
   - In case of the logged-in user has no privilege to access student record, UserPermissionException will be thrown.

9. Method: *getBalance (currentCharges, chargeHistory, paymentHistory)*
   Return Type: *double*
   Parameters: *currentCharges, chargeHistory, paymentHistory*
   Return value: *balance*
   Pre-condition: *no pre-condition*
   Post-condition: *balance amount will be returned*
   Attributes read/updated: *no attribute*
   Processing logic: *get the balance through statically ChargeCalculator and return it*

10. Method: *getBalance (currentCharges, paymentHistory)*
    Return Type: *double*
    Parameters: *currentCharges, paymentHistory*
    Return value: *balance*
    Pre-condition: *no pre-condition*
    Post-condition: *balance amount will be returned*
    Attributes read/updated: *no attribute*
    Processing logic: *get the balance through statically ChargeCalculator and return it*

11. Method: *addPayment (userId, amount, note)*
    Return Type: *void*
    Parameters: *userId, amount, note*
    Return value: *no return value*
    Pre-condition: *no pre-condition*
    Post-condition: *payment with requested amount and note along with date will be persisted*
    Attributes read/updated: *studentProfileRepository, userManagementService, paymentHistoryRepository*
    Processing logic:
    - Validate the logged-in user has the privilege to access student profile associated with requested userId through userManagementService
    - Validate whether there is an existing student profile for the given userId through *studentProfileRepository*
    - Persist payment with amount and note associated with userId through paymentHistoryRepository
    - In case of the logged-in user has no privilege to access student record, UserPermissionException will be thrown.
    - In case of no student profile existed associated with requested userId, NoStudentProfileFoundException will be thrown

### 4.3.5 Classes and interfaces in the API package

#### 4.3.5.1 Class: BILLIntf

- Purpose: *To handle the validation operations, including requests and responses that concerns the overall user, student and bill data.*
- Constraints: *it should implement BillService*
- Persistent: *no persistent*

##### 4.3.5.1.1 Attribute Descriptions

*No Attribute*

##### 4.3.5.1.2 Methods Descriptions

1. Method: *login ( userId )*
   Return Type: *void*
   Parameters: *userId*
   Return value: *no return value*
   Pre-condition: *the length of userId should be 8 characters, and consists of both letter and number*
   Post-condition: no-post Condition
   Attributes read/updated: *no attribute*
   Processing logic: *Not applicable*

2. Method: *logout ()*
   Return Type: *void*
   Parameters: *no parameter*
   Return value: *no return value*
   Pre-condition: *no pre-condition*
   Post-condition: no post-condition
   Attributes read/updated:
   Processing logic: *Not applicable*

3. Method: *getUserIdsUnderAdminPrivilidge ()*
   Return Type: *String[*]*
   Parameters: *userId*
   Return value: 0 or more of *userIds that is accessible by the logged-in admin user*
   Pre-condition: *no pre-condition*
   Post-condition: 0 or more of *userIds that is accessible by the logged-in user*
   Attributes read/updated: *no attribute*
   Processing logic: *Not applicable*

4. Method: *getUser ()*
   Return Type: *User*
   Parameters:
   Return value: *logged-in userId*
   Pre-condition: *logged-in user*
   Post-condition: *logged-in user id will be returned*
   Attributes read/updated: *no attribute*
   Processing logic: *Not applicable*

5. Method: *loadUsers(filename)*
   Return Type: *void*
   Parameters: *filename*
   Return value: *no return value*
   Pre-condition: *filename shouldn't be null and consists of single or more characters*
   Post-condition: *JSON file associated with filename that is existed in the system's resources will be reloaded into datasource.*
   Attributes read/updated: *no attribute*
   Processing logic: *Not applicable*

6. Method: *loadRecords(filename)*
   Return Type: *void*
   Parameters: *filename*
   Return value: *no return value*
   Pre-condition: *filename shouldn't be null and consists of single or more characters*

Post-condition: *JSON file associated with filename that is existed in the system's resources will be reloaded into datasource.*
Attributes read/updated: *no attribute*
Processing logic: *Not applicable*

7. Method: *getRecord (userId)*
   Return Type: *StudentRecord*
   Parameters: *userId*
   Return value: *StudentRecord object associated with requested userId*
   Pre-condition: *the length of userId should be 8 characters, and consists of both letter and number*
   Post-condition: *StudentRecord object associated with requested userId returned.*
   Attributes read/updated: *no attribute*
   Processing logic: *Not applicable*

8. Method: *updateRecord (userId, studentRecord, permenant)*
   Return Type: *User*
   Parameters: *userId*
   Return value: *no return value*
   Pre-condition: *the length of userId should be 8 characters, and consists of both letter and number*
   Post-condition: *StudentRecord object associated with requested userId updated.*
   Attributes read/updated: *no attribute*
   Processing logic: *Not applicable*

9. Method: *generateBill (userId)*
   Return Type: *Bill*
   Parameters: *userId*
   Return value: *Bill object associated with requested userId*
   Pre-condition: *the length of userId should be 8 characters, and consists of both letter and number*
   Post-condition: *Bill object associated with requested userId returned*
   Attributes read/updated: *no attribute*
   Processing logic: *Not applicable*

10. Method: *viewCharges (userId, startMonth, startDay, startYear, endMonth, endDay, endYear)*
    Return Type: *Bill*
    Parameters: *userId, startMonth, startDay, startYear, endMonth, endDay, endYear*
    Return value: *Bill object associated with requested userId*
    Pre-condition: *the length of userId should be 8 characters, and consists of both letter and number, the startMonth and endMonth should in the range 1-12, startDay and endDay should in the range 1-31, startYear and endYear should be consisted of 4-digit number..*
    Post-condition: *Bill object associated with requested userId returned*
    Attributes read/updated: *no attribute*
    Processing logic: *Not applicable*

11. Method: *applyPayment (userId, amount, note)*
    Return Type: *void*
    Parameters: *userId, amount, note*
    Return value: *no return value*
    Pre-condition: *the length of userId should be 8 characters, and consists of both letter and number, amount should be more than 0, and note should consist of 1 or more characters*

Post-condition: *Bill object associated with requested userId returned*
Attributes read/updated: *no attribute*
Processing logic: *Not applicable*

#### 4.3.5.2 Class: BILLIntfImpl

- Purpose: *To handle the validation operations, including requests and responses that concerns the overall user, student and bill data.*
- Constraints: *it should implement BillService*
- Persistent: *no persistent*

##### 4.3.5.2.1 Attribute Descriptions

1. Attribute: *userManagmentService*
   Type: *UserManagmentService*
   Description: instance of *UserManagmentService*
   Constraints: *should be annotated as @autowired in order the object to be initiated.*

2. Attribute: *studentManagmentService*
   Type: *StudentManagmentService*
   Description: instance of *StudentManagmentService*
   Constraints: *should be annotated as @autowired in order the object to be initiated.*

3. Attribute: *billService*
   Type: *BillService*
   Description: instance of BillService
   Constraints: *should be annotated as @autowired in order the object to be initiated.*

##### 4.3.5.2.2 Methods Descriptions

1. Method: *login ( userId )*
   Return Type: *void*
   Parameters: *userId*
   Return value: *no return value*
   Pre-condition: *the length of userId should be 8 characters, and consists of both letter and number*
   Post-condition: no-post Condition
   Attributes read/updated: *userManagementService*
   Processing logic:
   - *Validate userId as a valid input through statically DataValidation*
   - *Set/update the user associated with userId as logged-in user through UserManagmentService.*
   - *In case the userId is not validated, IllegalArgumentException will be thrown instantaneously*

2. Method: *logout ()*
   Return Type: *void*
   Parameters: *no parameter*
   Return value: *no return value*
   Pre-condition: *no pre-condition*
   Post-condition: no post-condition
   Attributes read/updated: *userManagementService*
   Processing logic: *trigger userManagementService in order for the user to be logged-out*

3. Method: *getUserIdsUnderAdminPrivilidge ()*

Return Type: *String[*]*
Parameters: *userId*
Return value: 0 or more of *userIds that is accessible by the logged-in admin user*
Pre-condition: *no pre-condition*
Post-condition: 0 or more of *userIds that is accessible by the logged-in user*
Attributes read/updated: *userManagementService*
Processing logic: *retrieve list of users that is accessiable by logged-in user through userManagementService*

4. Method: *getUser ()*
   Return Type: *User*
   Parameters:
   Return value: *logged-in userId*
   Pre-condition: *logged-in user*
   Post-condition: *logged-in user id will be returned*
   Attributes read/updated: *userRepository*
   Processing logic:
   - *Retrieve the userId of logged-in user through userManagmentService*
   - *In case no user id is returned from UserManagementService, UserNotFoundException will be thrown instantaneously.*

5. Method: *loadUsers(filename)*
   Return Type: *void*
   Parameters: *filename*
   Return value: *no return value*
   Pre-condition: *filename shouldn't be null and consists of single or more characters*
   Post-condition: *JSON file associated with filename that is existed in the system's resources will be reloaded into datasource.*
   Attributes read/updated: *userManagementService*
   Processing logic:
   - *Validate that filename is not empty and consists of more than 0 characters*
   - *Retrieve the JSON file associated with filename and persist the data through UserManagmentService. In case the filename is not validated, IllegalArgumentException will be thrown instantaneously*

6. Method: *loadRecords(filename)*
   Return Type: *void*
   Parameters: *filename*
   Return value: *no return value*
   Pre-condition: *filename shouldn't be null and consists of single or more characters*
   Post-condition: *JSON file associated with filename that is existed in the system's resources will be reloaded into datasource.*
   Attributes read/updated: *studentManagmentService*
   Processing logic:
   - *Validate that filename is not null and consists of more than 0 characters*
   - *Retrieve the JSON file associated with filename and persist the data through studentManagmentService and billService. In case the filename is not validated, IllegalArgumentException will be thrown instantaneously*

7. Method: *getRecord (userId)*
   Return Type: *StudentRecord*
   Parameters: *userId*

Return value: *StudentRecord object associated with requested userId*
Pre-condition: *the length of userId should be 8 characters, and consists of both letter and number*
Post-condition: *StudentRecord object associated with requested userId returned.*
Attributes read/updated: *studentProfileManagmentService*
Processing logic:

- *validate userId as a valid input through statically DataValidation*
- *Retrieve StudentRecord associated with userId through studentProfileManagmentService.*
- *In case the userId is not validated, IllegalArgumentException will be thrown instantaneously.*
- *In case no student record returned from UserManagementService, NoStudentProfileFoundException will be thrown instantaneously*

8. Method: *updateRecord (userId, studentRecord, permenant)*
   Return Type: *User*
   Parameters: *userId*
   Return value: *no return value*
   Pre-condition: *the length of userId should be 8 characters, and consists of both letter and number*
   Post-condition: *StudentRecord object associated with requested userId updated.*
   Attributes read/updated: *studentManagmentService*
   Processing logic:
   - *Validate userId and studentRecord as a valid input through statically InputValidator*
   - *Retrieve StudentRecord associated with userId through studentManagmentService*
   - *In case the either userId or studentRecord is not validated, IllegalArgumentException will be thrown instantaneously*

9. Method: *generateBill (userId)*
   Return Type: *Bill*
   Parameters: *userId*
   Return value: *Bill object associated with requested userId*
   Pre-condition: *the length of userId should be 8 characters, and consists of both letter and number*
   Post-condition: *Bill object associated with requested userId returned*
   Attributes read/updated: *billService*
   Processing logic:
   - *validate userId as a valid input through statically InputValidator*
   - *Retrieve bill through billService.*
   - *In case the userId is not validated, IllegalArgumentException will be thrown instantaneously.*
   - *In case no bill record returned from billService, NoBillExistedException will be thrown instantaneously*

10. Method: *viewCharges (userId, startMonth, startDay, startYear, endMonth, endDay, endYear)*
    Return Type: *Bill*
    Parameters: *userId, startMonth, startDay, startYear, endMonth, endDay, endYear*
    Return value: *Bill object associated with requested userId*
    Pre-condition: *the length of userId should be 8 characters, and consists of both letter and number, the startMonth and endMonth should in the range 1-12, startDay and endDay should in the range 1-31, startYear and endYear should be consisted of 4-digit number..*

Post-condition: *Bill object associated with requested userId returned*
Attributes read/updated: *billService*
Processing logic:

- *Validate userId as valid inputs through statically InputValidator*
- *Validate startMonth, startDay, startYear, endMonth, endDay, endYear as a valid input through statically InputValidator*
- *Convert those date-related attributes into Date objects as startDate and endDate and further validate that both are valid date and startDate less or equal endDate through statically InputValidator*
- *Retrieve bill of charges associated with requested userId through billService.*
- *In case of either userId or, is not validated, IllegalArgumentException will be thrown instantaneously.*
- *In case startMonth, startDay, startYear, endMonth, endDay, and endYear as both not valid input and dates, IllegalArgumentException will be thrown instantaneously*
- *In case startDate is not prior/less than endDate, IllegalArgumentException will be thrown instantaneously*
- *In case no bill record returned from billService, NoBillExistedException will be thrown instantaneously*

11. Method: *applyPayment (userId, amount, note)*
Return Type: *void*
Parameters: *userId, amount, note*
Return value: *no return value*
Pre-condition: *the length of userId should be 8 characters, and consists of both letter and number, amount should be more than 0, and note should consist of 1 or more characters*
Post-condition: *Bill object associated with requested userId returned*
Attributes read/updated: *billService*
Processing logic:
    - *Validate userId, as valid inputs through statically InputValidator*
    - *Validate that note is not null and consists of more than 0 characters*
    - *Validate that amount more than 0*
    - *Apply payment associated to the userId with amount specified and note through billService.*
    - *In case of either userId or amount or note not validated, IllegalArgumentException will be thrown instantaneously*

### 4.3.6 Classes and Interfaces in the Utility package

#### 4.3.6.1 Class: DataMapper

- Purpose: *To handle data mapping and transformation between different type of objects*
- Constraints: *no constraint*
- Persistent: *No persistent*

##### 4.3.6.1.1 Attribute Descriptions

No attribute

##### 4.3.6.1.2 Methods Descriptions

1. Method: *getBill (studentProfile, currentCharges, chargeHistory, paymentHistory, balance)*
Return Type: Bill
Parameters: *studentProfile , currentCharges, chargeHistory, paymentHistory, balance*

Return value: Bill object
Pre-condition: *no pre-condition*
Post-condition: Bill object returned
Attributes read/updated: *no attribute*
Processing logic:

- *Declare Bill* object *and list of Transaction object variable*
- *Map balance and some of student profile attributes which are user's id, firstName, lastName, and phone, emailAddress, addressStreet, addressState, addressCity, addressPostalCode, college, classStatus, and list of courses to the counterparts in the bill attributes.*
- *Map currentCharge, chargeHistory paymentHistory through locally getTransaction the list of transactions and assign it to their counterpart in the Bill object*
- Return the Bill object

2. Method: *getBill (studentProfile, currentCharges, paymentHistory, balance)*
   Return Type: Bill
   Parameters: *studentProfile, currentCharges, paymentHistory, balance*
   Return value: Bill object
   Pre-condition: *no pre-condition*
   Post-condition: Bill object returned
   Attributes read/updated: *no attribute*
   Processing logic:
   - *Declare Bill object and list of Transaction object variable*
   - *Map balance and some of student profile attributes which are user's id, firstName, lastName, and phone, emailAddress, addressStreet, addressState, addressCity, addressPostalCode, college, classStatus, and list of courses to their counterparts in the bill attributes.*
   - *Map currentCharge and paymentHistory through locally getTransaction, retrieve the list of transactions and assign it to its counterpart in the Bill object*
   - Return the Bill object

3. Method: *getStudentRecord (studentProfile, currentCharges, chargeHistory, paymentHistory, balance)*
   Return Type: Bill
   Parameters: *studentProfile, currentCharges, paymentHistory, balance*
   Return value: *StudentRecord object*
   Pre-condition: *no pre-condition*
   Post-condition: *StudentRecord object* returned
   Attributes read/updated: *no attribute*
   Processing logic:
   - *Declare StudentRecord object*
   - *Map balance and most of student profile attributes which are user's id, firstName, lastName, and phone, emailAddress, addressStreet, addressState, addressCity, addressPostalCode, college, classStatus, list of courses, termBegan, gradAssistant, international, internationalStatus, capstoneEnrolled, scholarship, veteran, studyAbroad, outsideInsuranc, nationalStudentExchange to their counterparts in the StudentRecord attributes.*
   - *Map currentCharge, chargeHistory, and paymentHistory through locally getTransaction, retrieve the list of transactions and assign it to its counterpart in the StudentRecord object*
   - Return the *StudentRecord* object

4. Method: getStudentProfile(studentRecord)

Return Type: *StudentProfile*
Parameters: *studentProfile*
Return value: *StudentRecord object*
Pre-condition: *no pre-condition*
Post-condition: *StudentProfile object returned*
Attributes read/updated: *no attribute*
Processing logic:

- *Declare StudentProfile, User, UserRoles object*
- *Map attribute concerning UserRole which are college, and roles to the User object*
- *Map attributes concerning User in StudentRecord which are user's id, firstName, lastName*
- *Map phone, emailAddress, addressStreet, addressState, addressCity, addressPostalCode, classStatus, list of courses, termBegan, gradAssistant, international, internationalStatus, capstoneEnrolled, scholarship, veteran, studyAbroad, outsideInsuranc, nationalStudentExchange to their counterparts in the StudentProfile*
- Return the *StudentProfile* object

5. Method: getTransactions(currentCharges, chargeHistory, paymentHistory)
   Return Type: Transaction[*]
   Parameters: *currentCharges, chargeHistory, paymentHistory*
   Return value: list of Transaction object
   Pre-condition: *no pre-condition*
   Post-condition: list of Transaction object returned
   Attributes read/updated: *no attribute*
   Processing logic:
   - *Declare list of Transaction object variable*
   - *Iterate over currentCharges and chargeHistory and map amount and note in each object with Transaction's attribute and assign type as CHARGE from Type enumeration and current month, day, year to its counterpart in the transaction object. Add each transaction to the declared list*
   - *Iterate over currentCharges and chargeHistory and map amount and note in each object with Transaction's attribute and assign type as PAYMENT from Type enumeration and current month, day, year to its counterpart in the transaction object. Add each transaction to the declared list*
   - *Return the list of transactions*

6. Method: getTransactions(currentCharges, paymentHistory)
   Return Type: Transaction[*]
   Parameters: *currentCharges, chargeHistory, paymentHistory*
   Return value: list of Transaction object returned
   Pre-condition: *no pre-condition*
   Post-condition: list of Transaction object returned
   Attributes read/updated: *no attribute*
   Processing logic:
   - *Declare list of Transaction object variable*
   - *Iterate over currentCharges and map amount and note in each object with Transaction's attribute and assign type as CHARGE from Type enumeration and current month, day, year to its counterpart in the transaction object.*
   - *Iterate over currentCharges and chargeHistory and map amount and note in each object with Transaction's attribute and assign type as PAYMENT from Type*

*enumeration and current month, day, year (through locally getTransactionDate) to its counterpart in the transaction object. Add each transaction to the declared list*
- *Return the list of transactions*

7. Method: getTransactionDate(date)
   Return Type: TransactionDate
   Parameters: *date*
   Return value: TransactionDate object
   Pre-condition: *no pre-condition*
   Post-condition: list of Transaction object returned
   Attributes read/updated: *no attribute*
   Processing logic:
   - *Declare TrnasactionDate object*
   - *Extract the month, day and year from the date object and assign them to their counterparts in TransactionDate object*
   - *Return TransactionDate object*

8. Method: getDate(TransactionDate)
   Return Type: Date
   Parameters: transactionDate
   Return value: Date object
   Pre-condition: *no pre-condition*
   Post-condition: list of Transaction object returned
   Attributes read/updated: *no attribute*
   Processing logic:
   - *Declare* TransactionDate *object*
   - *Extract the month, day and year from the data using Calender or SimpleDateFormatter APIs and assign each to their counterparts in TransactionDate object*
   - *Return TransactionDate object*

9. Method: *getTerm (date)*
   Return Type: Date
   Parameters: date
   Return value: Term object
   Pre-condition: *no pre-condition*
   Post-condition: Term object returned
   Attributes read/updated: *no attribute*
   Processing logic:
   *Declare* Term *object*
   - *Retrieve file properties that contains list semester with start date and end date*
   - *Compare which semester's start date and end date that data falls in, and assign semester, year to their counterpart in* Term object
   - *Return* Term *object*

10. Method: getStartDateAndEndDate (term)
    Return Type: Date[1..*]
    Parameters: term
    Return value: dates[2]
    Pre-condition: *no pre-condition*
    Post-condition: list of Transaction object returned
    Attributes read/updated: *no attribute*

Processing logic:

- *Declare* array of date *object with size 2*
- *Retrieve file properties that contains list semester with start date and end date*
- *Compare which semester and year term's semester and year match, and assign start Date to the first index and end Date to the second index if declared object*
- *Return* Term *object*

11. Method: getAllUsers (jsonArray)
    Return Type: User[*]
    Parameters: *jsonArray*
    Return value: list of User objects
    Pre-condition: *no pre-condition*
    Post-condition: list of User Object
    Attributes read/updated: *no attribute*
    Processing logic:
    - *Declare* list of User object
    - *Iterate over jsonArray and assign id, firstname, lastname, college, roles to its counterparts in User object. Validate the user object through statically InputValidator. add each user object to the list*
    - *Return* list of User object

12. Method: getAlStudentProfile (jsonArray)
    Return Type: StudentProfile[*]
    Parameters: *jsonArray*
    Return value: list of StudentProfile objects
    Pre-condition: *no pre-condition*
    Post-condition: list of StudentProfile Object
    Attributes read/updated: *no attribute*
    Processing logic:
    *Declare* list of StudentProfile object
    - *Iterate over jsonArray map most of student profile attributes which are user's id, firstName, lastName, and phone, emailAddress, addressStreet, addressState, addressCity, addressPostalCode, college, classStatus, list of courses, termBegan, gradAssistant, international, internationalStatus, capstoneEnrolled, scholarship, veteran, studyAbroad, outsideInsuranc, nationalStudentExchange to their counterparts in StudentProfile object. Validate each StudentProfile object profile through InputValidator, and add each to the list.*
    - *Return* list of StudentProfile object

13. Method: getAllChargeHistory (jsonArray)
    Return Type: ChargeHistory[*]
    Parameters: *jsonArray*
    Return value: list of User objects
    Pre-condition: *jsonArray doesn't equal to null and should consists of more than 0 objects*
    Post-condition: list of StudentProfile objects returned
    Attributes read/updated: *no attribute*
    Processing logic:
    - *Declare* list of ChargeHistory object
    - *Iterate over jsonArray and* Validate each transaction through locally InputValidator
    - *Check the type of Transaction as a "CHARGE"*
    - *Map user's id to its counterpart in* ChargeHistory object

- *Convert month, day, year to Date object through locally getDate and map it along amount and note to its counterparts in* ChargeHistory *object.*
- *Validate each* ChargeHistory object *through InputValidator. Add each to the list*
- *Return* list of ChargeHistory object

14. Method: getAllPaymentHistory (jsonArray)
    Return Type: PaymentHistory [*]
    Parameters: *jsonArray*
    Return value: list of PaymentHistory objects
    Pre-condition: *no pre-condition*
    Post-condition: list of PaymentHistory objects returned
    Attributes read/updated: *no attribute*
    Processing logic:
    - *Declare* list of PaymentHistory object
    - *Iterate over jsonArray and* Validate each transaction through locally InputValidator
    - *Check the type of Transaction as a "CHARGE"*
    - *Map user's id to its counterpart in* PaymentHistory object
    - *Convert month, day, year to Date object through locally getDate and map it along amount and note to its counterparts in* PaymentHistory *object.*
    - *Validate each* PaymentHistory object *through InputValidator. Add each to the list*
    - *Return* list of PaymentHistory object

#### 4.3.6.2 Class: ChargeCalculator

- Purpose: *To handle the calculation of tuition and fee*
- Constraints: *no constraint*
- Persistent: *No persistent*

##### 4.3.6.2.1 Attribute Descriptions

1. Attribute: UndergradResOrVetTuition
Type: double
Description: Tuition amount per credit hour for undergraduate resident or veteran
Constraint: Read only

2. Attribute: UndergradNonResTuition
Type: double
Description: Tuition amount per credit hour for undergradaute non-resident or tuition
Constraint: Read only

3. Attribute: UndergradGenOrAthlOrWodrSchTuition
Type: double
Description: Tuition amount per credit hour for undergraduate with general or athletic or woodrow scholarship
Constraint: Read only

4. Attribute: UndergradSimsSchTuition
Type: double
Description: Tuition amount per credit hour for undergraduate with Sims scholarship
Constraint: Read only

5. Attribute: UndergradActDutyTuition
Type: double
Description: Tuition amount per credit hour for undergradaute on active duty
Constraint: Read only

6. Attribute: UndergradStAbroadTuition
Type: double
Description: Tuition amount per credit hour for undergraduate studying abroad
Constraint: Read only

7. Attribute: gradResidentTuition
Type: double
Description: Tuition amount per credit hour for graduate resident
Constraint: Read only

8. Attribute: gradNonResidentTuition
Type: double
Description: Tuition amount per credit hour for graduate non-resident
Constraint: Read only

9. Attribute: gradAssistantTuition
Type: double
Description: Tuition amount per credit hour for graduate assistant
Constraint: Read only

10. Attribute: gradOnlineTuition
Type: double
Description: Tuition amount per credit hour for graduate who is taking course online
Constraint: Read only

11. Attribute: undergradResSeventeenCreditOrAbvFee
Type: double
Description: Undergraduate resident with seventeen credit hours or above fee amount
Constraint: Read only

12. Attribute: undergradSchOrActDutySeventeenOrAbvCreditFee
Type: double
Description: Undergraduate with scholarship or active duty and at the same time with seventeen credit hours or above fee amount
Constraint: Read only

13. Attribute: undergradNonSeventeenOrAbvCreditFee
Type: double
Description: Undergraduate non-resident with seventeen credit hours or above fee amount
Constraint: Read only

14. Attribute: gradResSeventeenOrAbvCreditFee
Type: double
Description: Graduate resident with seventeen credit hours or above fee amount
Constraint: Read only

15. Attribute: outsideInsurance
Type: double
Description: Third-party health insurance amount for graduate and undergraduate
Constraint: Read only

16. Attribute: outsideStAbroadInsurance
Type: double
Description: Third-party health insurance amount for graduate and undergraduate studying abroad
Constraint: Read only

17. Attribute: undergradHealthSixToElevenCreditFee
Type: double

Description: Undergraduate with six to eleven credit hours fee amount
Constraint: Read only

18. Attribute: gradHealthSixToEightCreditFee
Type: double
Description: Undergraduate with six to eight credit hours fee amount
Constraint: Read only

19. Attribute: gradHealthNineToTwelveCreditFee
Type: double
Description: Undergraduate with nine to twelve credit hours fee amount
Constraint: Read only

20. Attribute: technologyFeePerHour
Type: double
Description: technology fee amount per credit hour
Constraint: Read only

21. Attribute: enrollmentFee
Type: double
Description: Enrollment fee amount
Constraint: Read only

22. Attribute: internationalShortTermFee
Type: double
Description: International with short-term status fee amount
Constraint: Read only

23. Attribute: internationalSponseredFee
Type: double
Description: International with sponsored status fee amount
Constraint: Read only

24. Attribute: nationalStudentExchangeFee
Type: double
Description: National student exchange fee amount
Constraint: Read only

25. Attribute: stAbroadExProgFee
Type: double
Description: Undergraduate studying abroad exchange program fee amount
Constraint: Read only

26. Attribute: stAbroadRegFee
Type: double
Description: Undergraduate studying abroad with regular status fee amount
Constraint: Read only

27. Attribute: stAbroadCohFee
Type: double
Description: Undergraduate studying abroad with cohort status fee amount
Constraint: Read only

28. Attribute: marticulationFee
Type: double
Description: Marticulation fee amount
Constraint: Read only

29. Attribute: capstoneFee
Type: double

Description: Student enrolled in capstone fee amount
Constraint: Read only

#### 4.3.6.2.2 Methods Descriptions

1. Method: calculateFinalCharges (studentProfile)
   Return Type: double
   Parameters: *studentProfile*
   Return value: finalCharges
   Pre-condition: *studentProfile not equal-to null*
   Post-condition: *final charges is returned*
   Attributes read/updated: *no attribute*
   Processing logic:
   - *Declare finalCharges variable as a double*
   - *Calculate the charges through locally calculateCharges which in turn return a map of charges associated specific amount*
   - *Iterate over the map of charges and add each to final charges*
   - *Return finalCharges*

2. Method: calculateCharges (studentProfile)
   Return Type: Map<String, double>
   Parameters: *studentProfile*
   Return value: map of charges associated with amount of each charge
   Pre-condition: *studentProfile not equal-to null*
   Post-condition: map of charges associated with amount of each charge is returned
   Attributes read/updated: *no attribute*
   Processing logic:
   - *Declare chargesMap variable as a Map<String, double>*
   - *Calculate the tuitions through locally calculateTuitions which in turn return a map of tuition associated with specific amount*
   - *Calculate the fees through locally calculateFees which in turn return a map of fee associated with specific amount*
   - *Merge two maps into chargesMap and return it*

3. Method: calculateTuitions (studentProfile)
   Return Type: Map<String, double>
   Parameters: *studentProfile*
   Return value: map of tuitions associated with amount of each tuition
   Pre-condition: *studentProfile not equal-to null*
   Post-condition: map of tuitions associated with amount of each tuition is returned
   Attributes read/updated: *no attribute*
   Processing logic:
   - *Declare tuitionsMap variable as a Map<String, double>*
   - *Calculate tuitions based on the steps specified thoroughly in Requirement document – section* ***3.6 View Charges***
   - *Add each applicable amount with short description to tuitionsMap*
   - *Return tuitionsMap*

4. Method: calculateFees (studentProfile)
   Return Type: Map<String, double>
   Parameters: *studentProfile*
   Return value: map of fees associated with amount of each fee
   Pre-condition: *studentProfile not equal-to null*

Post-condition: map of fees associated with amount of each fee is returned
Attributes read/updated: *no attribute*
Processing logic:

- *Declare feesMap variable as a Map<String, double>*
- *Calculate tuitions based on the steps specified thoroughly in Requirement document – section **3.6 View Charges***
- *Add each applicable amount with short description to feesMap*
- *Return feesMap*

5. Method: calculateBalance (currentCharges, chargeHistory, paymentHistory)
   Return Type: double
   Parameters: *currentCharges, chargeHistory, paymentHistory*
   Return value: balance
   Pre-condition: *studentProfile not equal-to null*
   Post-condition: balance is returned
   Attributes read/updated: *no attribute*
   Processing logic:
   - *Declare totalChargesAmount and totalPaymentsAmount variable as a double*
   - *Iterating over currentCharges and chargeHistory, and sum the amount of each object to the totalChargesAmount amount*
   - *Iterating over paymentHistory, and sum the amount of each object to the totalPaymentsAmount*
   - *Subtract the totalChargesAmount against totalPaymentsAmount and return the result*

6. Method: calculateBalance (chargeHistory, paymentHistory)
   Return Type: double
   Parameters: *currentCharges, chargeHistory, paymentHistory*
   Return value: balance
   Pre-condition: *studentProfile not equal-to null*
   Post-condition: balance is returned
   Attributes read/updated: *no attribute*
   Processing logic:
   - *Declare totalChargesAmount and totalPaymentsAmount variable as a double*
   - *Iterating over chargeHistory, and sum the amount of each object to the totalChargesAmount amount*
   - *Iterating over paymentHistory, and sum the amount of each object to the totalPaymentsAmount*
   - *Subtract the totalChargesAmount against totalPaymentsAmount and return the result*

#### 4.3.6.3 Class: InputValidator

- Purpose: *To handle the validation of different types of input*
- Constraints: *no constraint*
- Persistent: *No persistent*

##### 4.3.6.3.1 Attribute Descriptions

No attribute

##### 4.3.6.3.2 Methods Descriptions

1. Method: validateUserId (userId)

Return Type: void
Parameters: *userId*
Return value: no return value
Pre-condition: *the length of userId should be 8 characters, and consists of both letter and number*
Post-condition: no post-condition
Attributes read/updated: *no attribute*
Processing logic:

- *Validate that the userId is not empty, and the length of String should be 8 characters, and consists of both letter and number*
- *In case either of the above not validated, IllegalArgumentException will be thrown*

2. Method: validateUser (user)
   Return Type: void
   Parameters: *userId*
   Return value: no return value
   Pre-condition: *the length of userId should be 8 characters, and consists of both letter and number*
   Post-condition: no post-condition
   Attributes read/updated: *no attribute*
   Processing logic:
   - Validate that user is not null
   - Validate that the User's id is a valid through locally validateUserId
   - Validate firstName and lastName are not empty
   - Validate User's UserRole through locally validateUserRole
   - *In case either of the above not validated, IllegalArgumentException will be thrown*

3. Method: validateUserRole *(userRole[1..*])*
   Return Type: void
   Parameters: *userRole[1..*]*
   Return value: no return value
   Pre-condition: list of UserRole objects is not null and consist of more than 0 userRole
   Post-condition: no post-condition
   Attributes read/updated: *no attribute*
   Processing logic:
   - Validate that list of UserRole objects is not null and consist of more than 0 userRole
   - Validate that the UserRole's role and college is a valid through Role and college enumeration
   - *In case either of the above not validated, IllegalArgumentException will be thrown*

4. Method: validateStudentRecord*(*studentRecord*)*
   Return Type: void
   Parameters: studentRecord
   Return value: no return value
   Pre-condition: student record is not null
   Post-condition: no post-condition
   Attributes read/updated: *no attribute*
   Processing logic:
   - Validate that student record is not null
   - Validate id of studentRecord through locally validateUserId
   - Validate the attributes that concerning tuition and fee calculation based on the rules specified in test document – USER_PROFILE_RESTRICTION

- Validate each course in studentRecord through locally validateCourse
- Validate each transaction in studentRecord through locally validateTransaction
- Validate firstName, lastName, and phone, emailAddress, addressStreet, addressState, addressCity, addressPostalCode, college, classStatus, termBegan, internationalStatus, capstoneEnrolled, studyAbroad, and scholarship are not empty
- Validate addressPostalCode consists of 5-digit number
- Validate emailAddress is a valid email address
- Validate phone has a format of xxx-xxx-xxxx and in which x represent integer format
- Validate classStatus, scholarshop, college and internationalStatus as a valid value through checking ClassStatus, College, scholarship, and InternationalStatus enumerations
- Validate termBegin's year as a valid value, and semester as a valid through checking semester enumeration
- *In case either of the above not validated, IllegalArgumentException will be thrown*

5. Method: validateCourse *(course)*
   Return Type: void
   Parameters: *course*
   Return value: no return value
   Pre-condition: *course's* shouldn't be empty and should consist of 4-letter followed by 3-number, and name shouldn't be empty and numCredits bigger than 0
   Post-condition: no post-condition
   Attributes read/updated: *no attribute*
   Processing logic:
   - Validate the id attribute is not empty and should consist of 4-letter followed by 3-number
   - *Validate the name attribute is not empty, and numCredits* bigger *than 0*
   - *In case either of the above not validated, IllegalArgumentException will be thrown*

6. Method: validateTransaction *(transaction)*
   Return Type: void
   Parameters: *course*
   Return value: no return value
   Pre-condition: *transaction's type either "CHARGE" or "PAYMENT", startMonth, startDay, startYear collectively forming a valid date, amount should be bigger than 0, and note shouldn't be empty*
   Post-condition: no post-condition
   Attributes read/updated: *no attribute*
   Processing logic:
   - *Validate transaction's type either "CHARGE" or "PAYMENT"*
   - *Validate startMonth, startDay, startYear as a valid date through locally validateDate*
   - *Validate amount attribute bigger than 0, and note attribute shouldn't be emoty*
   - *In case either of the above not validated, IllegalArgumentException will be thrown*

7. Method: validateDate *(month, day, year)*
   Return Type: void
   Parameters: *month, day, year*
   Return value: no return value

   Pre-condition: *month, day, and year are collectively forming a valid Gregorian date.*

Post-condition: no post-condition
Attributes read/updated: *no attribute*
Processing logic:

- Validate that the *month should be in the range 1-12, day should be in the range 1-31, year should be 4 digit number*
- *Validate that month, day, and year are collectively forming a valid date through SimpleDateFormat or Calendar APIs*
- *In case either of the above not validated, IllegalArgumentException will be thrown*

8. Method: validateDateRange *(*startDate, endDate*)*
Return Type: void
Parameters: startDate*, endDate*
Return value: no return value
Pre-condition: *startDate is prior/ less than endDate*
Post-condition: no post-condition
Attributes read/updated: *no attribute*
Processing logic:
    - Validate *startDate is prior/less than* endDate
    - *In case is not validated, IllegalArgumentException will be thrown*

## 4.3.7 Classes and Interfaces in the Exception package

### 4.3.7.1 Class: UserPermissionException

- Purpose: *To handle anomalous or exceptional conditions related to accessibility or privilege when requesting on behalf of user.*
- Constraints: *it should extend from Exception class*
- Persistent: *No persistent*

#### 4.3.7.1.1 Attribute Descriptions

No attribute

#### 4.3.7.1.2 Methods Descriptions

1. Method: UserPermissionException *(message)*
Return Type: UserPermissionException
Parameters: *message*
Return value: instance of class
Pre-condition: *message states why this occurrence is exceptional*
Post-condition: UserPermissionException thrown
Attributes read/updated: *no attribute*
Processing logic: calling and passing requested message to the parent constructor

### 4.3.7.2 Class: UserNotFoundException

- Purpose: *To handle anomalous or exceptional conditions related to the existence of user records*
- Constraints: *it should extend from Exception class*
- Persistent: *No persistent*

#### 4.3.7.2.1 Attribute Descriptions

No attribute

#### 4.3.7.2.2 Methods Descriptions

1. Method: UserNotFoundException *(message)*
   Return Type: UserNotFoundException
   Parameters: *message*
   Return value: instance of class
   Pre-condition: *message states why this occurrence is exceptional*
   Post-condition: UserNotFoundException thrown
   Attributes read/updated: *no attribute*
   Processing logic: calling and passing requested message to the parent constructor

#### 4.3.7.3 Class: NoStudentProfileFoundException

- Purpose: *To handle anomalous or exceptional conditions related to the existence of student profile record*
- Constraints: *it should extend from Exception class*
- Persistent: *No persistent*

##### 4.3.7.3.1 Attribute Descriptions

No attribute

##### 4.3.7.3.2 Methods Descriptions

1. Method: NoStudentProfileFoundException *(message)*
   Return Type: NoStudentProfileFoundException
   Parameters: *message*
   Return value: instance of class
   Pre-condition: *message states why this occurrence is exceptional during the runtime*
   Post-condition: NoStudentProfileFoundException thrown
   Attributes read/updated: *no attribute*
   Processing logic: calling and passing the message to the parent constructor

#### 4.3.7.4 Class: NoBillExistedException

- Purpose: *To handle anomalous or exceptional conditions related to existence of bill when requesting*
- Constraints: *it should extend from Exception class*
- Persistent: *No persistent*

##### 4.3.7.4.1 Attribute Descriptions

No attribute

##### 4.3.7.4.2 Methods Descriptions

1. Method: NoBillExistedException *(message)*
   Return Type: NoBillExistedException
   Parameters: *message*
   Return value: instance of class
   Pre-condition: *message states why this occurrence is exceptional during the runtime*
   Post-condition: NoBillExistedException thrown
   Attributes read/updated: *no attribute*
   Processing logic: calling and passing the message to the parent constructor