**Assignment 5**

**Class Model using OCL**

AIM - Design and implementation of the design model from the analysis model.

PROBLEM STATEMENT -

Prepare a Design Model from Analysis Model.

Identify Design classes and evolve the Analysis Model.

Draw Design class Model using OCL and UML2.0 Notations.

OBJECTIVE -

To Identify Design level Classes.

To Draw a Design level class Model using an analysis model.

To Implement Design Model-class diagram

THEORY -

The analysis model is refined and formalized to get a design model. Design modeling means refinement to analysis level models to adapt to the actual implementation environment. In the design space, yet another new dimension has been added to the analysis space to include the implementation environment. This means adopting the analysis model to fit in the implementation model at the same time as it is refined.

1) Creating Design level Class Diagrams :

Class diagrams model the static structure of a package or of a complete system. As the blueprints of the system, class diagrams model the objects that make up the system, allowing to display the relationships among those objects and to describe what the objects can do and the services they can provide.

2) OCL :

The Object Constraint Language (OCL) is adeclarative language describing rules applying toUnified Modeling Language (UML) models developed atIBM and is now part of the UML standard. Initially, OCL was merely a formal specification language extension for UML.

OCL supplements UML by providing expressions that have neither the ambiguities of natural language nor the inherent difficulty of using complex mathematics. OCL is also a navigation language for graph-based models.

Object Constraint Language (OCL), is a formal language to express side-effect-free constraints. Users of the Unified Modeling Language and other languages can use OCL to specify constraints and other expressions attached to their models.

The disadvantage of traditional formal languages is that they are useable to persons with a strong mathematical background, but difficult for the average business or system modeler to use. OCL has been developed to fill this gap. It is a formal language that remains easy to read and write.

OCL is a pure expression language. Therefore, an OCL expression is guaranteed to be without

side effects it cannot change anything in the model. This means that the state of the system will

never change because of an OCL expression, even though an OCL expression can be used to

specify a state change, e.g. in a post-condition. All values for all objects, including all links, will

not change. Whenever an OCL expression is evaluated, it simply delivers a value.

OCL statements are constructed in four parts:

1. a context that defines the limited situation in which the statement is valid
2. a property that represents some characteristics of the context (e.g., if the context is a class, a property might be an attribute)
3. an operation (e.g., arithmetic, set-oriented) that manipulates or qualifies a property, and
4. keywords (e.g., if, then, else, and, or, not, implies) that are used to specify conditional expressions.

Examples :

Constraint - The age of a person is not negative

OCL Equivalent - context Person inv: self.age >=0

Constraint - A person is younger than its parents

OCL Equivalent - Constraint - Person inv: self.parents->forAll(p|p.age>self.age)

Applications of OCL :

OCL can be used for a number of different purposes:

**•** To specify invariants on classes and types in the class model.

**•** To specify type invariant for Stereotypes.

**•** To describe pre-and post-conditions on Operations and Methods.

**•** To describe Guards.

**•** As a navigation language.

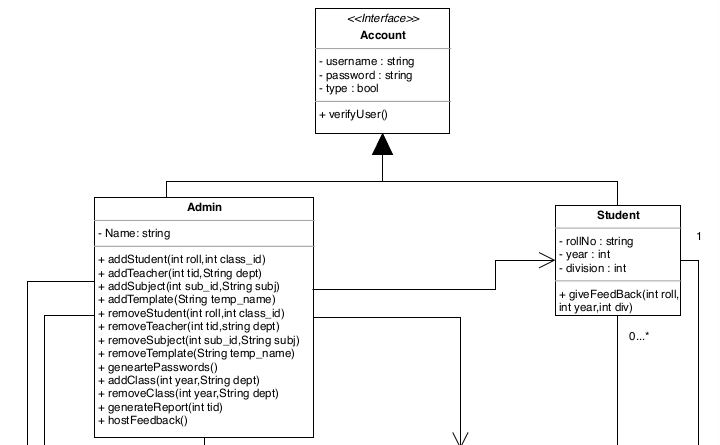
**•** To specify constraints on operations.

For the Feedback management system,

Classes included being - Account, Admin, Class, Subject, Student, Teacher, Question, Template, and Feedback.

Abstraction relationships :

An abstraction relationship is a dependency between model elements that represent the same concept at different levels of abstraction or from different viewpoints. The account class is further inherited into Admin and Student classes.

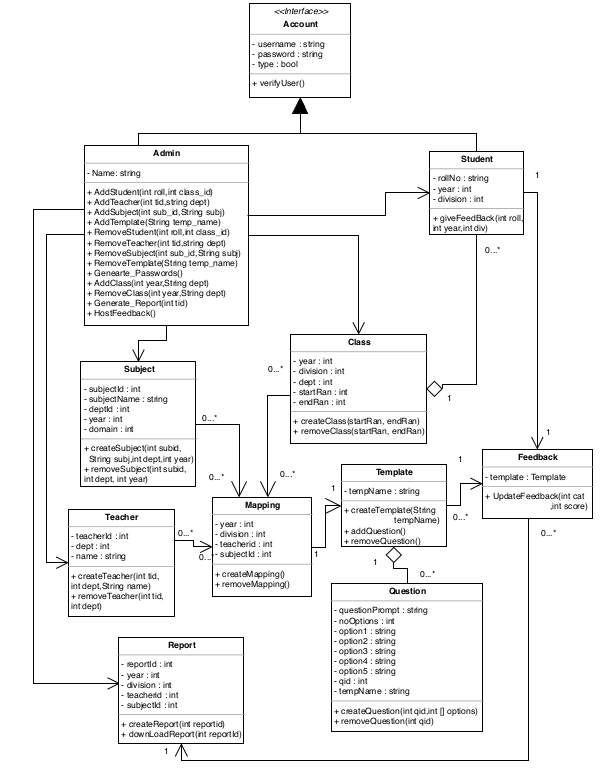


Aggregation relationships :

In UML models, an aggregation relationship shows a class as a part of or subordinate to

another class. An aggregation is a special type of association in which objects are assembled or configured together to create a more complex object. Aggregation relationship is established between classes Question and Template.

In the Feedback management system, we are using OCL to give constraints to all of the objects. The whole system is controlled and objects are created by Admin. So, the majority of the OCL and its constraints are given to the Admin object. For example, the Admin object login should be true. Then for all the subsequent operations performed by the Admin, like AddStudent, AddSubject, Addteacher, etc. we need to give the constraints for all the operations.



**Object Constraint Language for the aforementioned Class Diagram**

Context Admin::

inv: Self.login == true

Context Admin:: AddStudent(roll:int, class\_id:int): void

pre : length(roll) == 5 and class\_id > 0

post : Student->exists()

Context Admin:: AddTeacher(tid:int,dept:String): void

pre : dept == “AS” or dept == “FE” or dept == “SE” or dept == “TE” or dept == “BE”

post : Teacher->exists()

Context Admin:: AddSubject(sub\_id:int,subj:String): void

pre : length(sub\_id) == 5 and subj->notEmpty()

post : Subject->exists()

Context Admin:: AddTemplate(temp\_name:String): void

pre : temp\_name->notEmpty()

post : Template->exists()

Context Admin:: AddClass(year:int,dept:String): void

pre : dept == “AS” or dept == “FE” or dept == “SE” or dept == “TE” or dept == “BE” and year>0

post : Class->exists()

Context Admin:: RemoveStudent(roll:int, class\_id:int): void

pre : length(roll) == 5 and class\_id > 0

post : Student->delete()

Context Admin:: RemoveTeacher(tid:int,dept:String): void

pre : dept == “AS” or dept == “FE” or dept == “SE” or dept == “TE” or dept == “BE”

post : Teacher->teacher()

Context Admin:: RemoveSubject(sub\_id:int,subj:String): void

pre : length(sub\_id) == 5 and subj->notEmpty()

post : Subject->delete()

Context Admin:: RemoveTemplate(temp\_name:String): void

pre : temp\_name->notEmpty()

post : Template->delete()

Context Admin:: RemoveClass(year:int,dept:String): void

pre : dept == “AS” or dept == “FE” or dept == “SE” or dept == “TE” or dept == “BE” and year>0

post : Class->delete()

Context Admin:: Genearte\_Passwords(): void

pre : forAll(Mappings)->exists()

post : Self->displayPasswords()

Context Admin:: Generate\_Report(tid:int): void

pre : tid>0 and forAny(Student)->feedbackGiven()

post : Self->displayReport()

Context Admin:: HostFeedback(): void

pre : forAll(Student)->passwordsExist()

post : Self->createFeedbackTables()

Context Subject::

inv: Admin->exists()

Context Student::

inv: Admin->exists()

Context Student::giveFeedBack(roll:int,year:int,div:int)::void

pre : length(Self.password)>0

post : Feedback->exists()

Context Class::

inv: Admin->exists()

Context Template::

inv: Admin->exists()

Context Template::addQuestion()::void

pre : length(Self.tempName)>0

Context Teacher::

inv: Admin->exists()

Context Question::

inv: Admin->exists()

Context Report::

inv: Admin->exists()

Context Class::

inv: Admin->exists()

CONCLUSION -

Thus we have prepared a Design Model from Analysis Model and studied in detail the working of the system/project. Identified Design classes/ Evolve Analysis Model and used advanced relationships. We drew a Design class Model using OCL and UML2.0 Notations and implemented the design model with a suitable object-oriented language.