**Class Diagram:** The class diagram is the main building block of object-oriented modeling. It is used both for general conceptual modeling of the systematic of the application, and for detailed modeling translating the models into programming code. Class diagrams can also be used for data modeling and the classes in a class diagram represent both the main elements, interactions in the application, and the classes to be programmed.

In the diagram, classes are represented with boxes that contain three compartments:

* The top compartment contains the name of the class. It is printed in bold and centered, and the first letter is capitalized.
* The middle compartment contains the attributes of the class. They are left-aligned and the first letter is lowercase.
* The bottom compartment contains the operations the class can execute. They are also left-aligned and the first letter is lowercase.

In the design of a system, a number of classes are identified and grouped together in a class diagram that helps to determine the static relations between them. With detailed modeling, the classes of the conceptual design are often split into a number of subclasses.

Members of the class Diagram for Real Estate System are as follows:

**Visibility:** To specify the visibility of a class member (i.e. any attribute or method), these notations must be placed before the member's name.

|  |  |
| --- | --- |
| + | Public |
| - | Private |
|  |  |
|  |  |
|  |  |

**Relationships:** A relationship is a general term covering the specific types of logical connections found between the classes. It shows the following relationships:

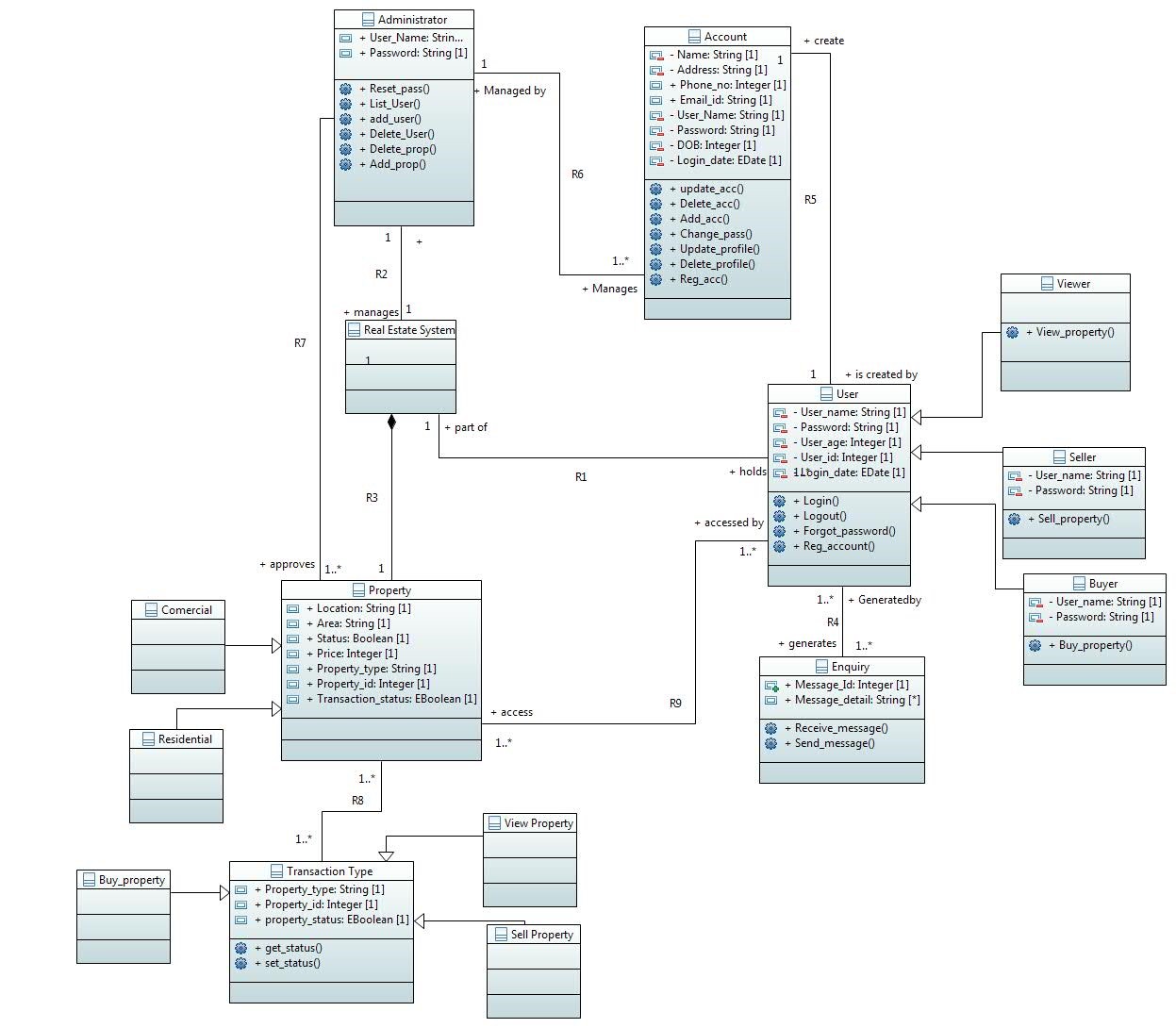
* **Association:** An *association* represents a family of links. An association can link any number of classes. A binary association (with two ends) is normally represented as a line. An association can be named, and the ends of an association can be adorned with role names, ownership indicators, multiplicity, visibility, and other properties.
* **Aggregation:** Aggregationis a variant of the "has a" association relationship, it is more specific than association. It is an association that represents a part-whole or part-of relationship. However, an aggregation may not involve more than two classes; it must be a binary association. Furthermore, there is hardly a difference between aggregations and associations during implementation, and the diagram may skip aggregation relations altogether.
* **Generalization:** It indicates that one of the two related classes (the *subclass*) is considered to be a specialized form of the other (the *super type*) and the super class is considered a Generalization of the subclass. In practice, this means that any instance of the subtype is also an instance of the super class. The generalization relationship is also known as the *inheritance* or *"is a"* relationship.

**Multiplicity:** This association relationship indicates that (at least) one of the two related classes makes reference to the other. This relationship is usually described as "A has a B"

1 TO 1-> Exactly one instance

1..\* TO 1..\*-. Many to Many instances

1 TO 1...\* One to Many instances



OCL :

**2. Constraints**

The Object Constraint Language (OCL) is a declarative language for describing rules that apply to Unified Modeling Language. Constraints are the imposed conditions that must be satisfied while performing certain function to get the desired output. It is not always possible to express all the constraints in graphical language form so we use Object Constraint Language (OCL) that provides a formal language for specifying constraints which supplement the models created in terms of UML diagrams.

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

2. User must login once in 6 months in order keep his account active.

**Context** User: get\_login(a: date)

**Pre:** self.logindate <= 6

**Post:** login date = logindate@pre +1

3. Admin deactivate user’s account if the user is not logged in for 6 months.

**Context** Admin

**Inv :** self .R6-> Reject (login date >=6)