**Sequence Diagram-Design Class Diagram**

**Version 2.0**

**Project Management App**

**Team A**

**CSC-355**

**Spring 2016**



03/28/2016

Author: Jennifer Li – Project Leader and Tyler Mariano – System Designer

Submitted To: Dr. Joo Tan

**TABLE OF CONTENTS**

Revision History…………………………………………………………………………………..ii

1.0 Introduction……..….....…………………………….…...…………………………….……....1

2.0 Sequence Diagram……………….……………………………………………………………1

2.1 Sequence Diagram Notation……………....……………………………………..….1-2

2.2 Diagram Drawings………………………………………………………………….3-4

3.0 Design Class Diagram…………………………………………………………………………5

3.1 Design Class Diagram Notation…………………………………………………….5-6

3.2 Diagram Drawings…………………………………………………………………..7-8

**REVISION HISTORY**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Author | Description | Date |
| 1.0 | Jennifer Li | I created the first draft. | 01/31/16 |
|  | Jennifer Li | Added the Introduction in section 1. | 02/01/16 |
|  | Jennifer Li | Added Sequence Diagram section 2. | 02/02/16 |
| 1.1 | Jennifer Li | Added Sequence Diagram section 2.1 and notation chart. | 02/03/16 |
|  | Tyler Mariano | Added the Diagrams in section 2.2.  Added Introduction for section 3 | 02/04/16 |
|  | Tyler Mariano | Added the Diagrams in section 3.2. | 02/05/16 |
|  | Tyler Mariano | Added Introduction for section 3.1 | 02/06/16 |
| 2.0 | Tyler Mariano | Made Dr. Tan’s edits in sections 1.0 - 3.2. | 03/25/16 |
|  | Jennifer Li | Made graphical edits in section 3.1 | 03/25/16 |

**1.0 INTRODUCTION**

This document contains two types of Unified Modeling Language diagrams, or UML diagrams, the sequence diagram and design class diagram. These diagrams are used to describe the characteristics and behavior of objects in a presented scenario. The sequence diagram generally describes an actor’s interaction with the system via a visual diagram representing a scenario specific to the system. The design class diagram defines the objects and relations in the system. Both of these diagrams will be explained and illustrated in detail in the paragraphs below.

**2.0 SEQUENCE DIAGRAM**

Sequence diagram is a visual representation of how groups of objects collaborate. In the basic sense, sequence diagrams captures the behavior of a single scenario. The diagram will map the order of how messages logically interact between the actor and the system with in the specific use case. Order is the key component in this diagram, it maps the use case’s sequence of operation from the beginning to the end.

**2.1 Sequence Diagram Notation**

This section explains the notation used in sequence diagrams. In a sequence diagram, a stick figure is used to represent the actor that has a specific role. A rectangular box containing the word “:Controller” is used to represent the control object of system. This will route the passed in parameters to the appropriate objects. The rectangular box that contains “:View” represents the view or boundary object. This boundary object will collaborate with the database. The other rectangular boxes that contain “:Object 1” and “:Object 2” are problem domain objects that represent classes within the system.

The diagram also contains two types of arrowed lines. The first arrowed line is a solid arrowed line pointing to the system, representing the sent input message. The second arrowed line is a dashed lined arrow that represents the output message.

Last but not least, sequence diagrams have two types of life lines. The first is a dashed vertical line, a life line or a duration that represents the session in which the interactions take place. The second is a long vertical rectangular box, activation life line that represents the life time of a logical function that take place within an object from beginning to end.

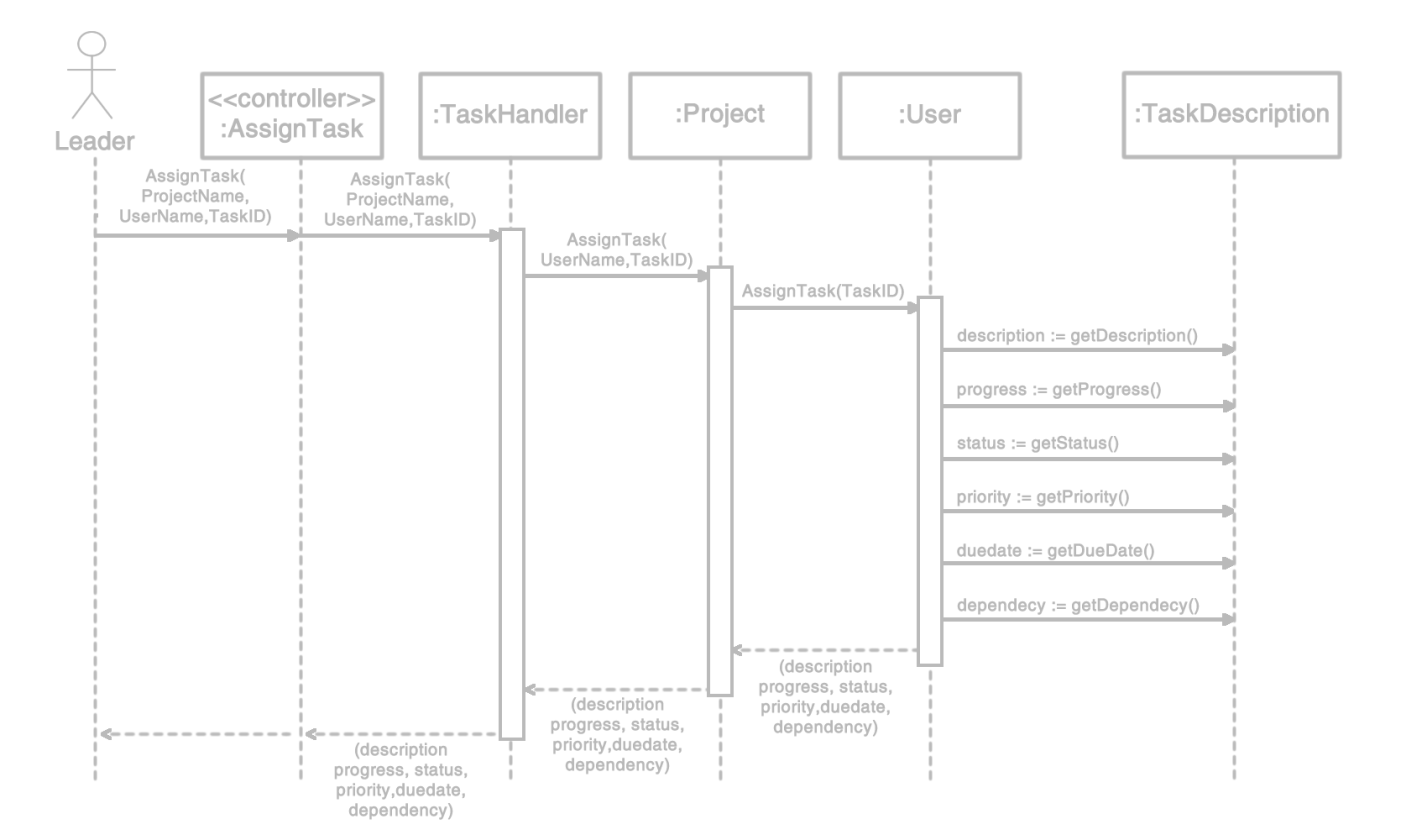
The sequence diagram notation and definitions are displayed in Figure 2.1.

|  |  |
| --- | --- |
| Notation | Definition |
|  | **Actor** |
|  | **Controller Object <<controller>>** |
|  | **View or Boundary Object <<View>>** |
|  | **Problem Domain Objects <<entity>>** |
|  | **Input** |
|  | **Output** |
|  | **Life Line or Duration** |
|  | **Activation Life Line** |

**Figure 2.1** Chart containing sequence diagram notation and definitions.

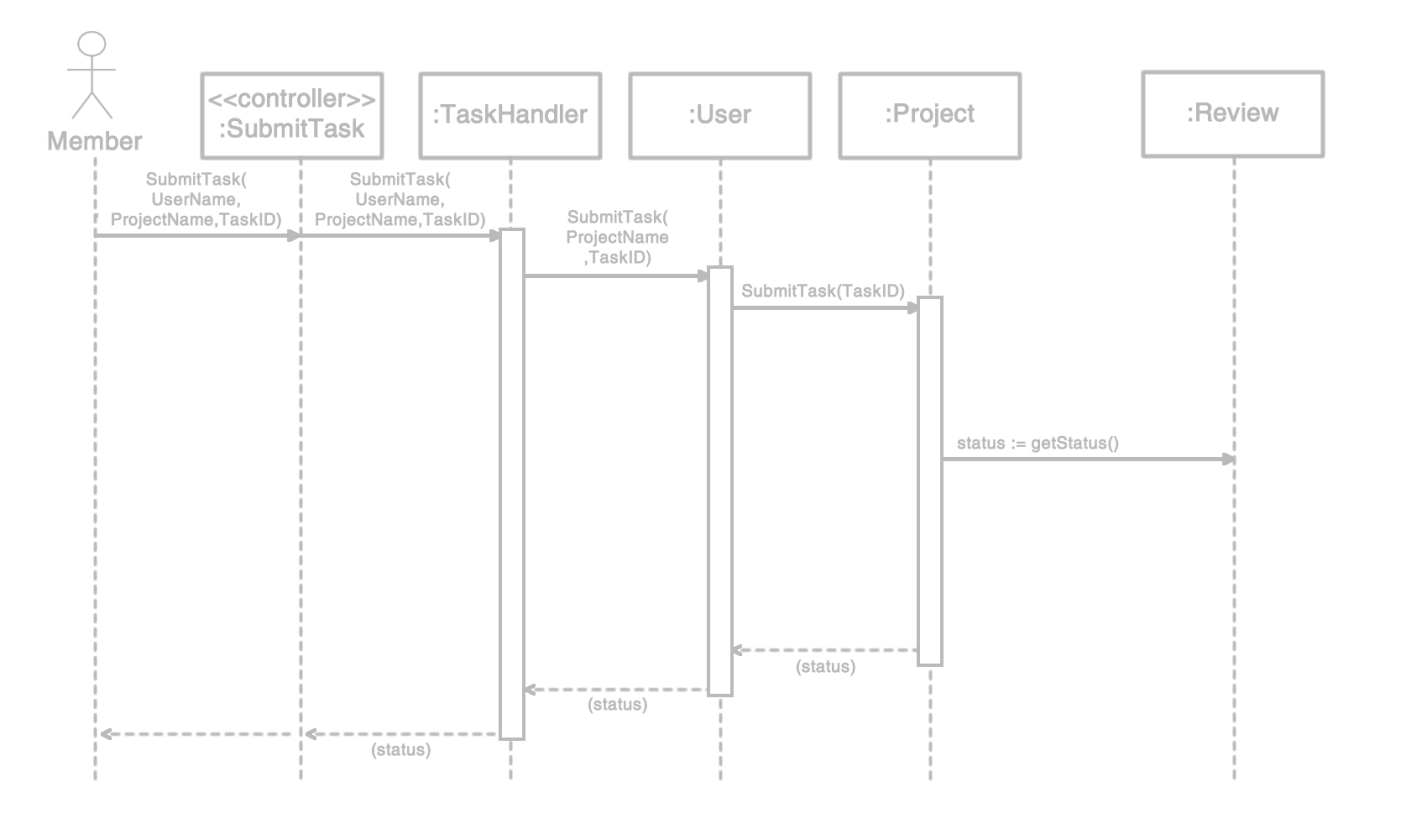
**2.2 Diagram Drawings**

The sequence diagram displayed below is the interaction of how the project leader assigns a task to a member. First, to assign a task a leader must create a task name. Next, the leader must select an assignee from a list of current project members to complete the task. Last, the database will generate an auto incremented identification number or ID.



**Figure 2.2** – Is a sequence diagram displaying the interaction of how the project leader assigns a task to a member using attribute notation.

The sequence diagram displayed below is the interaction of how the project member submits a task to the project leader for review. First, to submit a task a member must login. Next, the member must select a task to complete. Once completed the task will be sent to the leader for review. The leader will review the task and send back a status of complete or will reassign the task to the member to make improvements.



**Figure 2.3 –** Isa sequence diagram displaying the interaction of how the project member submits a task to the project leader for review using attribute notation.

**3.0 DESIGN CLASS DIAGRAM**

Design class diagrams describe the types of objects in the system with various kinds of relationships between objects. The design class diagram also shows the properties and operations of an object with its constraints. Relationships in this diagram are generally static while in sequence diagrams they are dynamic. UML has two ways to illustrate design class diagrams. The first is association notation and the second is attribute notation. Both of these notations will be explained and illustrated in detail in the paragraphs below.

**3.1** **Design Class Diagram Notation**

This section explains the notations used in design class diagrams. In both notations of the design class diagrams, rectangular boxes are used to represent classes. These classes are broken down into three sub-sections. The first section is filled by the class name. The second section is filled by a list of the class’s attributes. The last section is filled with a list of the class’s methods. In the attribute and method list sub-section of the class the attribute and association notation have documentation for visibility.

There are 4 types of documentation for visibility. The first is the “ + ”, used to represent the attribute or method visibility to be public. Second, the “ - ”, is used to represent the visibility of private which cannot be directly accessed by the user. Third, “ # ”, is used to represent the visibility of protected for that attribute or method. When an attribute or method has a visibility of protected this means that only this class and any subclasses may access the method/property. Last, but not least is “ ~ ”, which represent the default setting of the attribute or method.

The design class diagram also contains arrowed lines; these lines shows the connection between classes. Along with the multiplicity notation, viewers will be able to understand the relationship between connected classes. Multiplicity is placed depending on what notation is being used. In attribute notation, the multiplicity is placed in the attribute list section of the class. In the association notation, the multiplicity is placed on the arrowed lines that connects the classes.

Multiplicity’s meaning varies for each relationship; they can be the same or totally different. The use of “ 1 ”, is used to denote exactly one. The “ \* ”, is used to denote a value of many. The usage of “ 0..1 ”, is used to denote the attribute could be optional. Lastly, “ m..n ”, is used to denote the attribute to a specific numerical value.

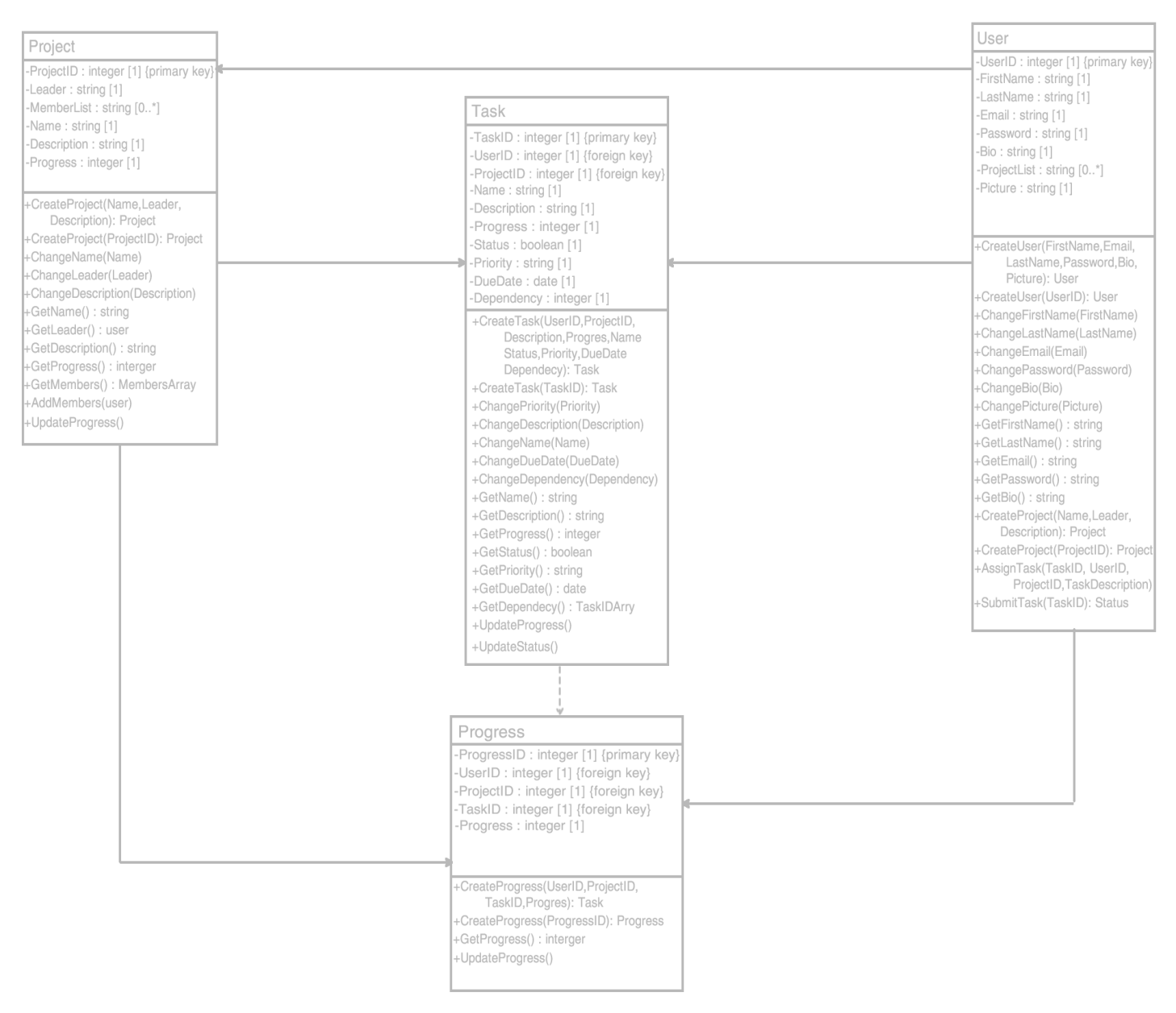
The design class diagram notations and definitions are displayed below in Figure 3.1.

|  |  |
| --- | --- |
| Notation | Definition |
| |  | | --- | | Class Naming | | Attribute Listing | | Method Listing | | **Class** |
|  | **Association Arrow** |
| Visibility | |
| + | **Public** |
| - | **Private** |
| # | **Protected** |
| ~ | **Package (default setting)** |
| Multiplicity | |
| 1 | **Exactly One** |
| \* | **Many (zero or one)** |
| 0..1 | **Optional (zero or one)** |
| m..n | **Numerically Specific** |
| Attribute Notation | |
| |  | | --- | | Class 1 | | - Leader: string [1] | | + GetLeader(): user |  |  | | --- | | Class 2 | | - TaskName: string[1] | | + GetTask() : name | | |
| Association Notation | |
|  | |

**Figure 3.1** Chart containing design class diagram notation and definitions.

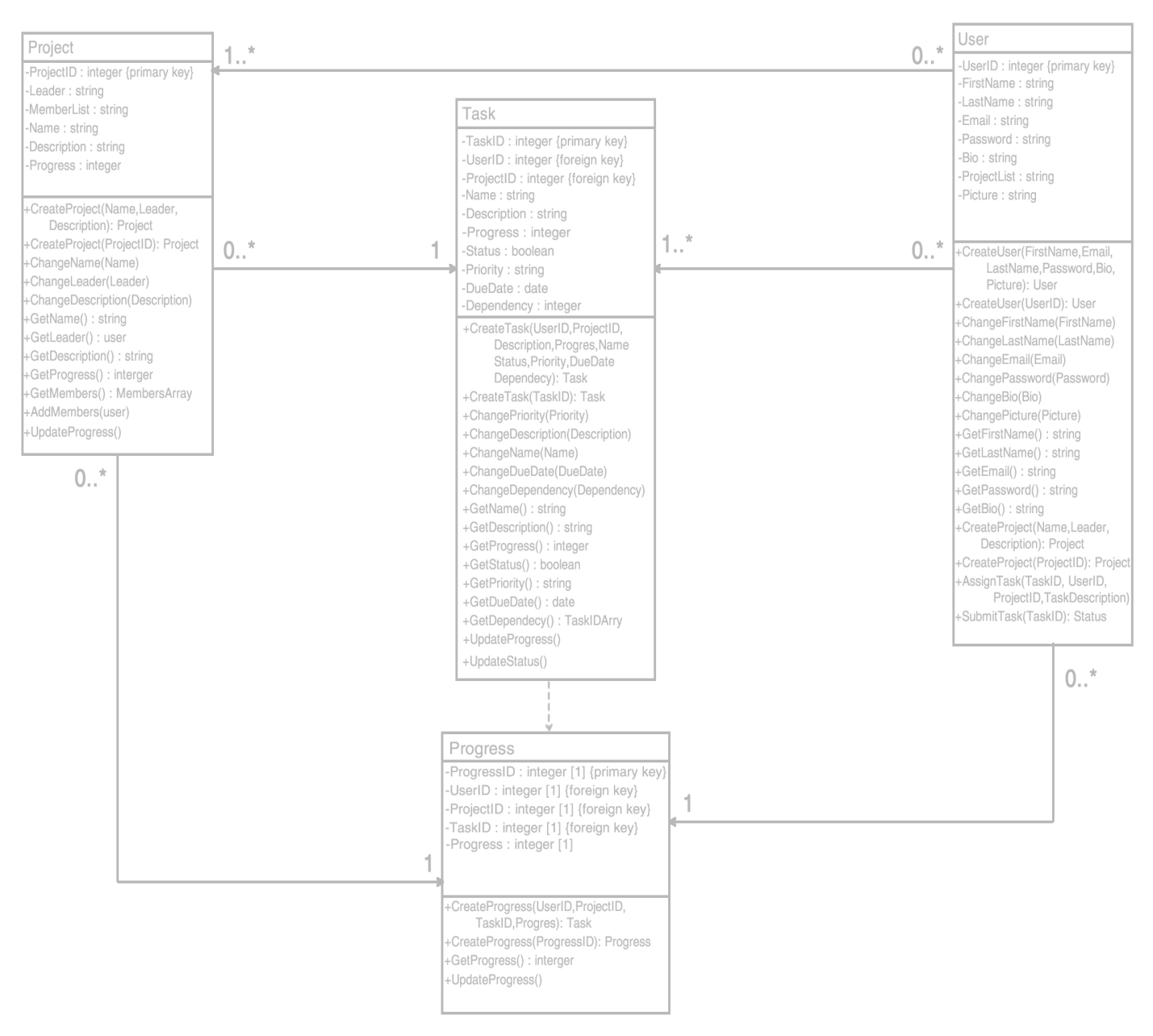
**3.2 Diagram Drawings**

The design class diagram displayed below illustrates the relationships of how the Task class, with its attributes and methods connect to other classes, using attribute notation.



**Figure 3.2** Design Class Diagram using Attribute Notation.

The design class diagram displayed below illustrates the relationships of how the Task class, with its attributes and methods connect to other classes, using association notation.



**Figure 3.3** Design Class Diagram using Association Notation.