Car Rental System

**System Design Document**

Group no : TY-25

|  |  |  |
| --- | --- | --- |
| **Roll. No.** | **Gr. No.** | **Name** |
| **22** | **11810820** | **Shreya Hirapurkar** |
| **58** | **11810499** | **Vinayak Kokane** |
| **60** | **11810970** | **Dhanraj Kore** |
| **63** | **11920122** | **Vaishnavi Kulkarni** |

**Approvals Signature Block**

|  |  |  |
| --- | --- | --- |
| **Project Responsibility** | **Signature** | **Date** |
| *Project Guide (Internal)* |  |  |
| *Project Guide (External)* |  |  |
| *Documentation Leader* |  |  |

*The document in this file is for UML Class Diagram Documentation Specifications, which conforms to the requirements of UML 2.0.*

*Items that are intended to stay in as part of your document are in* **bold***; blue italic text is used for explanatory information that should be removed when the template is used.*

**Table of Contents**

**[1.](#_gjdgxs)****GENERAL INFORMATION 4**

[1.1](#_30j0zll) Version Control 4

[1.2](#_1fob9te) Information Details 4

**[2.](#_3znysh7)****CLASS DIAGRAM DESCRIPTION 4**

[2.1](#_2et92p0) CRC Template 4

[2.2](#_tyjcwt) CRC Cards Overview 5

[Responsibility 5](#_3dy6vkm)

[Collaborator 5](#_1t3h5sf)

[Domain Users (3-5 people) 5](#_4d34og8)

[OO Design Analyst (1-2 people) 5](#_2s8eyo1)

[Facilitator (1 person) 6](#_17dp8vu)

[Scribe (1-2 people) 6](#_3rdcrjn)

[Observers (0-N people) 6](#_26in1rg)

[The CRC Technique 6](#_lnxbz9)

[Selecting a Scenario 6](#_35nkun2)

[Creating Initial CRC Card(s) 7](#_1ksv4uv)

[Arranging the CRC Cards 7](#_44sinio)

[The CRC Interview 7](#_2jxsxqh)

[Finding Classes 7](#_z337ya)

[Finding Collaborators 7](#_3j2qqm3)

[2.2.1](#_1y810tw) General Guidelines 8

[2.2.2](#_4i7ojhp) Class Style Guidelines 9

[2.2.3](#_2xcytpi) Interfaces 10

[2.2.4](#_1ci93xb) Relationship Guidelines 10

[2.2.5](#_3whwml4) Inheritance Guidelines 11

[2.2.6](#_2bn6wsx) Aggregation and Composition Guidelines 12

[2.3](#_qsh70q) Class Diagram 12

# **General Information**

*Fill in the following details to keep trace of Project Phases and Iterations.*

|  |  |
| --- | --- |
| **Informational Item** | **Information** |
| Document Title | Class Diagram of Car Rental System |
| Version | 1.0 |
| Author | Shreya Hirapurkar, Vinayak Kokane, Dhanraj Kore, Vaishnavi Kulkarni |
| Project Name | Car Rental System |
| Project Phase | Phase 1 |
| Project Iteration | 1 |

## **Version Control**

*Fill in the following details to keep trace of Versioning Details.*

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| May-18-2021 | 1.0 | Created | Shreya Hirapurkar, Vinayak Kokane, Dhanraj Kore, Vaishnavi Kulkarni |

## **Information Details**

*Fill in the following details to keep trace of information that you are generating.*

|  |  |
| --- | --- |
| **Informational Item** | **Information** |
| Filename | TY\_25\_Class\_Diagram.docx |
| Last Saved On | Monday, May 18, 2021, 4:28:00 PM |
| Last Saved By | Shreya Hirapurkar, Vinayak Kokane, Dhanraj Kore, Vaishnavi Kulkarni |
| Number of Pages | 14 |

# **Class diagram description**

## **CRC Cards Overview**

*CRC (Class-Responsibility-Collaborator) Card Modeling is a simple yet powerful object-oriented analysis technique. CRC modeling often includes the users, analysts, and developers in a modeling and design process, bringing together the entire development team to form a common understanding of an OO development project. It is one of many tools that should be used in the collaborative design of a system.*

A CRC Model is a collection of cards (usually standard index cards or larger) that are divided into three sections.

1. Class
2. Responsibility
3. Collaborator

Class

*A Class represents a collection of similar objects. Objects are things of interest in the system being modeled. They can be a person, place, thing, or any other concept important to the system at hand. The Class name appears across the top of the CRC card.*

Responsibility

*A Responsibility is anything that the class knows or does. These responsibilities are things that the class has knowledge about itself, or things the class can do with the knowledge it has.*

*For example, a person class might have knowledge (and responsibility) for its name, address, and phone number. In another example an automobile class might have knowledge of its size, its number of doors, or it might be able to do things like stop and go. The Responsibilities of a class appear along the left side of the CRC card.*

Collaborator

*A Collaborator is another class that is used to get information for, or perform actions for the class at hand. It often works with a particular class to complete a step (or steps) in a scenario. The Collaborators of a class appear along the right side of the CRC card. The CRC team is comprised of different types of people involved in the project. The overall size of the CRC team is important, and active participants should be limited to 6.*

*Interaction between active participants grows experientially with the number candidates.*

There are generally three types of skill sets for active participants in the CRC session.

1. Domain Users
2. OO Design Analyst
3. Facilitator

*Additional non-active participants can include a scribe and observers.*

Domain Users (3-5 people)

*These members of the CRC team are the users of the system being designed. They should have a good amount of business domain knowledge for which the system is being modeled.*

Good domain users also have the following characteristics:

* Know the business being modeled.
* Think logically.
* Good communication skills.
* Are willing to invest the time in systems designs.

OO Design Analyst (1-2 people)

These members of the CRC team are analysts or developers familiar with OO methodologies and techniques. Generally the design analysts have responsibilities on the project well beyond the CRC modeling sessions.

*These may include fleshing out the CRC cards to create a formal object model, and interpreting the CRC collaborators to document the OID’s (Object Interaction Diagrams).*

Good Design analyst have the following characteristics:

* Understand the CRC modeling process and methodology.
* Understand the OO modeling process and methodology.
* Experience developing OO systems.

Facilitator (1 person)

This is the member who runs the CRC session. This person is perhaps the most important member of the team. It is the facilitator’s responsibility to keep the CRC session progressing forward.

*This may include acting as an intermediary when collaboration debates occur, and generally making sure that the session is on track with regards to the agenda.*

Good facilitators have the following characteristics:

* Good meeting skills.
* Understand the CRC modeling process and methodology.
* Understand the OO modeling process and methodology.

Scribe (1-2 people)

This is the member responsible for documenting any business logic and discussion that isn’t captured on the CRC Cards themselves. Often analysis logic on why a process is the way it is discovered within a CRC session, and should be captured and documented.

*This documentation is often rolled back into the requirements and business case documents, as well as used by the design analysts to further the systems object model.*

Good scribes have the following characteristics:

* Listen extremely well.
* Good written communications skills
* Can identify business logic

Observers (0-N people)

*These are members who aren’t directly participating in the CRC session. They are generally other users of the system, or other project team members. It is important that the facilitator ensure that these people are not active during the CRC session.*

Good observers have the following characteristics:

* Know how to contain opinions during the modeling session, i.e. shut-up.

The CRC Technique

The facilitator should begin by explaining the purpose of the CRC session. Remember that scenario execution is at the heart of the CRC session. An explanation of the CRC cards, as well as some sample CRC cards themselves can be created.

The concept of role playing techniques should be covered. Role-playing involves acting out what a class does in order to facilitate the scenario execution. A person who owns a particular card should 'become' that class when control of the scenario is passed to them.

*This is often done by holding the card in the air while the class has control of the execution. This way, a card is an object while being held in the air, and a card is a class while resting on the table.*

Selecting a Scenario

One scenario should be selected to begin the CRC session. Often an initial brainstorming time can be set aside at the beginning of the session to discuss and identify an appropriate scenario.

The scenario chosen should be well documented and very specific. Any related scenario's should be modeled separately. Often it is desirable to choose an easy scenario to begin with.

*If there are already use-case’s diagrams or descriptions for the problem domain, a good scenario is any 'normal coarse' through one of these 'use cases'. Using just the ‘normal coarse’ ensures that the session focuses on one particular scenario, and not the entire use case.*

Creating Initial CRC Card(s)

Initial CRC cards can be created for any class identified during the scenario selection. CRC cards are also created for existing classes in the system. Often these classes are the result of an existing legacy system.

Further brainstorming is an option depending on the experience level of the CRC session participants.).

Arranging the CRC Cards

As CRC cards are created, they are often placed on a centralized table for all participants to see. Once collaborations are established between classes, the arrangement of the classes can be moved so that collaborating classes are in close proximity to each other.

The more that two classes collaborate throughout the session, the closer they should be placed to each other.

In the initial stages of the CRC session, you can expect that the location of the classes on the table move quite frequently. This is typical of card placement, and will settle down after collaboration become more established.

It is often helpful to put the busy cards towards the center of the table, and those less busy cards around the outside perimeter. While the importance of card placement is subtle, beware of just choosing a card location because there is empty space on the table.

*If the table fills up quickly during a scenario execution, the quite possibly the scenario chosen is overly complicated, or is not the primary coarse through a 'use case'.*

The CRC Interview

It is often useful to start the session using the interview technique, or new employee technique. The interview technique involves identifying a scenario, then identifying a user that asks a ‘starting’ class to perform the scenario. The class in turns asks questions (interviews) the requester (user) to get enough information to perform the task.

The user can respond with the information, or respond that the class should know that information. The class can create a responsibility, or collaborate with other classes to get the information it requires.

Finding Classes

A Class represents a collection of similar objects. Objects are things of interest in the system being modeled. They can be a person, place, thing, or any other concept important to the system at hand. There are many ways to identify classes. One of the easiest to start with is noun extraction. Noun extraction identifies all of the Nouns in a problem statement and/or use-case scenario. The nouns extracted make excellent candidate classes the ways to identify classes are to look for items that interact with the system, or things that are part of the system. Ask if there is a customer of the system, and identify what the customer interacts with.

Finding Responsibilities

A Responsibility is anything that the class knows or does. These responsibilities are things that the class has knowledge about itself, or things the class can do with the knowledge it has.

To compliment the 'Noun Extraction' technique above is verb extraction. Verb extraction identifies all of the verbs in a problem statement and/or use-case scenario. These are usually good indicators of actions that must be performed by the classes of the system.

Other techniques included asking what the class knows, and what information must be stored about the class to make it unique. Ask what the class does, and then flesh out how the class might do it.

Finding Collaborators

Collaboration occurs when a class needs information that it doesn’t have. Classes know specific things about themselves. Very often to perform a task a class needs information that it doesn't have.

Often it's necessary to get this information from another class, in the form of collaboration.

Collaboration can also occur if a class needs to modify information that it doesn’t have. One property of information that a class knows about itself is the ability to update the information. Often a class will want to update information that it doesn't have. When this happens, the class will often ask another class, in the form of a collaboration, to update the information for it.

Generally for a collaboration to occur, one class is the initiator. In other words, there has to be a starting point for collaboration. Often times the initiating class is doing very little work beyond initialing the collaboration itself.

|  |  |
| --- | --- |
| **Class Description** | **Type** |
| A class that is never instantiated into an object. | Abstract Class |
| An interesting property of a class. Usually in the form of a value. | Attribute |
| A representation of a collection of similar objects. Classes are often objects, representing a person, place, or thing. | Class |
| A class that provides information or performs an action for another class. | Collaborator |
| An instance of a particular class. While a class may represent people, an object represents one particular person. | Object |
| Anything a class knows or does. | Responsibility |
| A class that may become a permanent class in the system after further analysis. | Candidate Class |

### **General Guidelines**

Because class diagrams are used for a variety of purposes – from understanding requirements to describing your detailed design – you will need to apply a different style in each circumstance. This section describes style guidelines pertaining to different types of class diagrams.

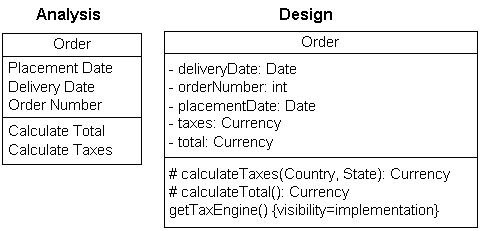


Figure 1. Analysis and design versions of a class.

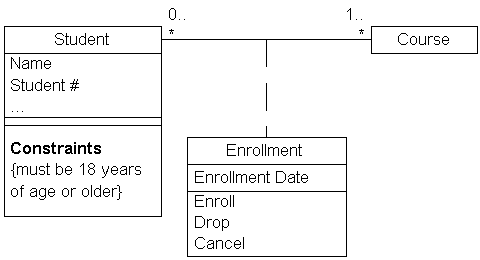
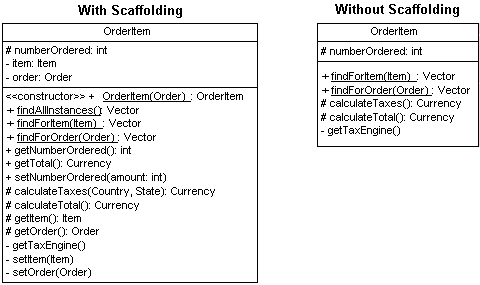


Figure 2. Modeling association classes.

1. Identify Responsibilities on Domain Class Diagrams.
2. Indicate Visibility Only On Design Models.
3. Indicate Language-Dependent Visibility With Property Strings.
4. Indicate Types Only On Design Models.
5. Indicate Types On Analysis Models Only When The Type is an Actual Requirement.
6. Design Class Diagrams Should Reflect Language Naming Conventions.
7. Model Association Classes On Analysis Diagrams.
8. Do Not Name Associations That Have Association Classes.
9. Center The Dashed Line of an Association Class.

### **Class Style Guidelines**

A class is effectively a template from which objects are created (instantiated). Although in the real world Doug, Wayne, John, and Bill are all student objects we would model the class *Student* instead. Classes define attributes, information that is pertinent to their instances, and operations, functionality that the objects support. Classes will also realize interfaces (more on this later).  Note that you may need to soften some of the naming guidelines to reflect your implementation language or software purchased from a third-party vendor.

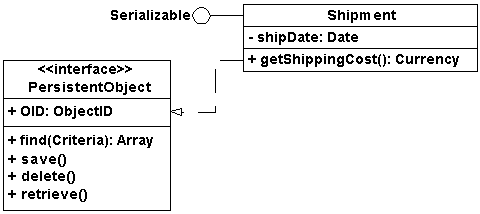


**Figure 3. The OrderItem class with and without scaffolding code.**

1. Use Common Terminology for Names
2. Prefer Complete Singular Nouns for Class Names
3. Name Operations with a Strong Verb
4. Name Attributes With a Domain-Based Noun
5. Do Not Model Scaffolding Code.  Scaffolding code refers to the attributes and operations required to implement basic functionality within your classes, such as the code required to implement relationships with other classes.
6. Never Show Classes With Just Two Compartments
7. Label Uncommon Class Compartments
8. Include an Ellipsis ( … ) At The End of Incomplete Lists
9. List Static Operations/Attributes Before Instance Operations/Attributes
10. List Operations/Attributes in Decreasing Visibility
11. For Parameters That Are Objects, Only List Their Type
12. Develop Consistent Method Signatures
13. Avoid Stereotypes Implied By Language Naming Conventions
14. Indicate Exceptions In An Operation’s Property String.

### **Interfaces**

An interface is a collection of operation signature and/or attribute definitions that ideally defines a cohesive set of behaviors. Interfaces are implemented, “realized” in UML parlance, by classes and components – to realize an interface a class or component must implement the operations and attributes defined by the interface. Any given class or component may implement zero or more interfaces and one or more classes or components can implement the same interface.

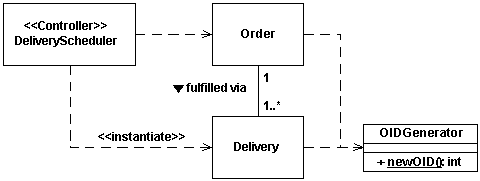


**Figure 5. Interfaces on UML class diagrams.**

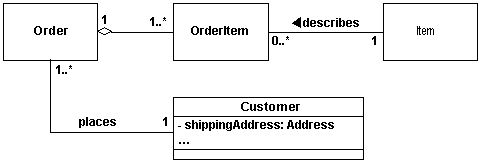
1. Interface Definitions Must Reflect Implementation Language Constraints.
2. Name Interfaces According To Language Naming Conventions
3. Apply “Lollipop” Notation To Indicate That A Class Realizes an Interface
4. Define Interfaces Separately From Your Classes
5. Do Not Model the Operations and Attributes of an Interface in Your Classes.
6. Consider an Interface to Be a Contract

### **Relationship Guidelines**

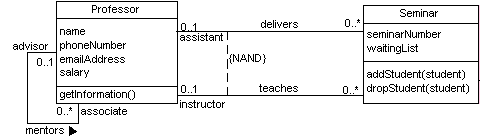
For ease of discussion the term relationships shall include all UML concepts such as associations, aggregation, composition, dependencies, inheritance, and realizations – in other words, if it’s a line on a UML class diagram we’ll consider it a relationship.



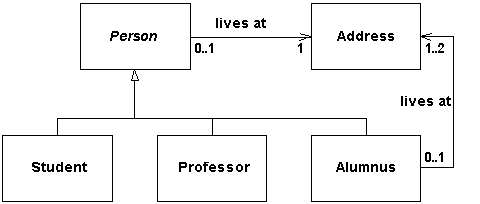
**Figure 6. Shipping an order.**



**Figure 7. Modeling an order.**



**Figure 8. Professors and seminars.**



**Figure 9. Modeling people at a university.**

1. Model Relationships Horizontally
2. Collaboration Indicates Need for a Relationship
3. Model a Dependency When The Relationship is Transitory
4. Depict Similar Relationships Involving A Common Class As A Tree.
5. Always Indicate the Multiplicity
6. Avoid a Multiplicity of “\*”
7. Replace Relationships By Indicating Attribute Types.
8. Do Not Model Implied Relationships
9. Do Not Model Every Single Dependency
10. Center Names on Associations
11. Write Concise Association Names In Active Voice
12. Indicate Directionality To Clarify An Association Name
13. Name Unidirectional Associations In The Same Direction
14. Word Association Names Left-To-Right
15. Indicate Role Names When Multiple Associations Between Two Classes Exist
16. Indicate Role Names on Recursive Associations
17. Make Associations Bi-Directional Only When Collaboration Occurs In Both Directions.
18. Redraw Inherited Associations Only When Something Changes
19. Question Multiplicities Involving Minimums And Maximums

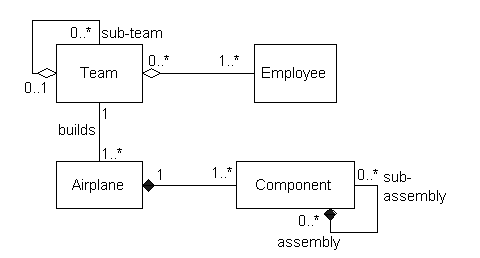
### **Inheritance Guidelines**

Inheritance models “is a” and “is like” relationships, enabling you to easily reuse existing data and code. When “A” inherits from “B” we say that “A” is the subclass of “B” and that “B” is the superclass of “A.” Furthermore, we say that we have “pure inheritance” when “A” inherits all of the attributes and methods of “B.” The UML modeling notation for inheritance is a line with a closed arrowhead pointing from the subclass to the superclass.

1. Apply the Sentence Rule For Inheritance
2. Place Subclasses Below Superclasses
3. Beware of Data-Based Inheritance
4. A Subclass Should Inherit Everything

### **Aggregation and Composition Guidelines**

Sometimes an object is made up of other objects. For example, an airplane is made up of a fuselage, wings, engines, landing gear, flaps, and so on. A delivery shipment contains one or more packages. A team consists of two or more employees. These are all examples of the concept of aggregation, which represents “is part of” relationships. Aggregation is a specialization of association, specifying a whole-part relationship between two objects. Composition is a stronger form of aggregation where the whole and parts have coincident lifetimes, and it is very common for the whole to manage the lifecycle of its parts.

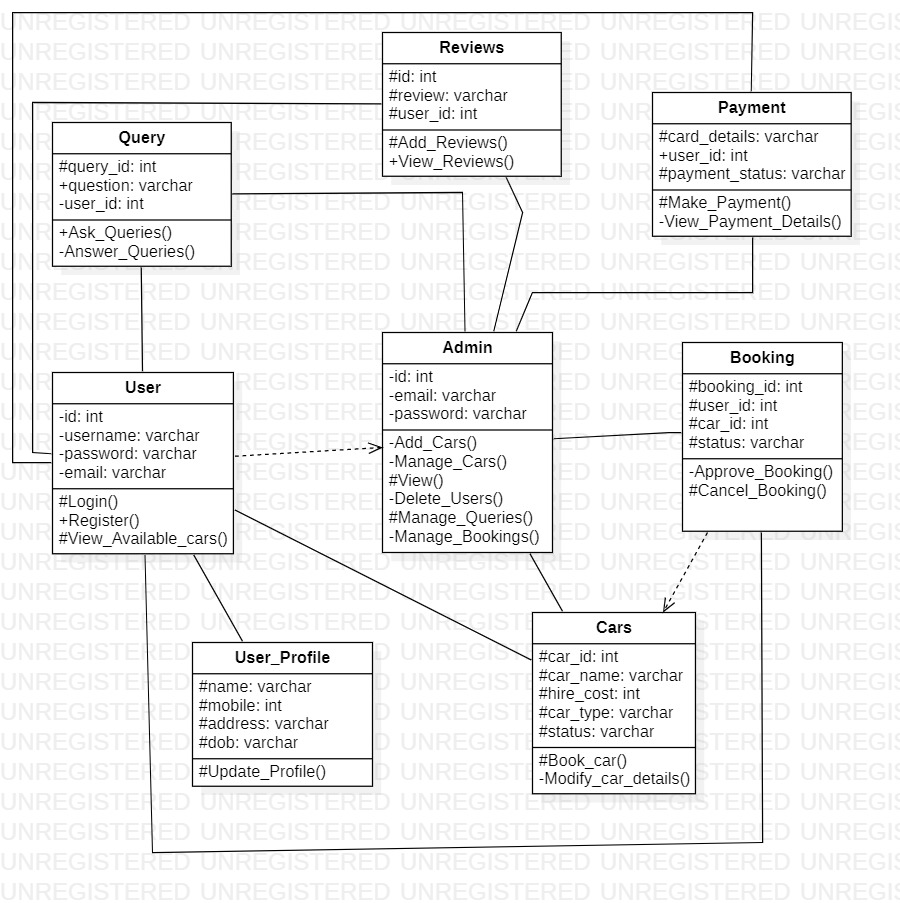


**Figure 10. Examples of aggregation and composition.**

1. Apply the Sentence Rule for Aggregation
2. You Should Be Interested In Both The Whole And The Part
3. Depict the Whole to the Left of the Part
4. Apply Composition to Aggregates of Physical Items
5. Apply Composition When the Parts Share The Persistence Lifecycle With the Whole
6. Don’t Worry About Getting the Diamonds Right

## **Class Diagram**

*Associated Class Diagram (both Conceptual and Software Class Diagram) need be placed here.*

**