**Unified Modeling Language**

UML is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems.

UML was created by Object Management Group (OMG) and UML 1.0 specification draft was proposed to the OMG in January 1997.

* UML stands for **U**nified **M**odeling **L**anguage.
* UML is different from the other common programming languages like C++, Java, COBOL etc.
* UML is a pictorial language used to make software blue prints.

UML can be described as a general purpose visual modeling language to visualize, specify, construct and document software system. Although UML is generally used to model software systems but it is not limited within this boundary. It is also used to model non software systems as well like process flow in a manufacturing unit etc.

UML is not a programming language but tools can be used to generate code in various languages using UML diagrams. UML has a direct relation with object oriented analysis and design. After some standardization UML is become an OMG (Object Management Group) standard.

**Diagrams:**

**Structural Diagrams:**

The *structural diagrams* represent the static aspect of the system. These static aspects represent those parts of a diagram which forms the main structure and therefore stable.

These static parts are represents by classes, interfaces, objects, components and nodes. The four structural diagrams are:

* Class diagram
* Object diagram
* Component diagram
* Deployment diagram

**Behavioral Diagrams:**

Behavioral diagrams basically capture the dynamic aspect of a system. Dynamic aspect can be further described as the changing/moving parts of a system.

UML has the following five types of behavioral diagrams:

* Use case diagram
* Sequence diagram
* Collaboration diagram
* Statechart diagram
* Activity diagram

**Use Case Diagram**

**Introduction:**

* A Usecase diagram is a diagram that shows a set of use cases actors and their relationships.
* Usecase diagrams are central to modeling the requirements of the behavior of a system, a subsystem, or a class.
* Usecase diagrams are important for visualizing, specifying and documenting the behavior of an element.

**Purpose:**

The purpose of use case diagram is to capture the dynamic aspect of a system. Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. So when a system is analyzed to gather its functionalities use cases are prepared and actors are identified. when the initial task is complete use case diagrams are modeled to present the outside view.

So in brief, the purposes of use case diagrams can be as follows:

* Used to gather requirements of a system.
* Used to get an outside view of a system.
* Identify external and internal factors influencing the system.
* Show the interacting among the requirements are actors.

**When to Use: Use Cases Diagrams**

* Use cases are used in almost every project.
* They are helpful in exposing requirements and planning the project.
* During the initial stage of a project most use cases should be defined.

**Usecase diagram contents**

|  |  |  |
| --- | --- | --- |
| **Name** | **Representation** | **Description** |
| Usecase |  | A use case represents a user goal that can be achieved by accessing the system or software application. |
| Actor |  | Actors are the entities that interact with a system. Although in most cases, actors are used to represent the users of system, actors can actually be anything  that needs to exchange information with the system.  So an actor may be people, computer hardware, other  systems, etc. Note that actor represent a role that a user can play, but not a specific user. |
| Package |  | Package is defined as collection of classes. Classes are unified together using a package. |
| Note |  | Note is generally used to write comment in use-case diagram. |
| Association |  | Actor and use case can be  associated to indicate that the actor participates in that use case. Therefore, an association corresponds to a sequence of actions between the actor and use case in achieving the use |
| Generalization |  | A generalization relationship is used to represent inheritance relationship between model elements of same type. |
| Dependency | <<include>>  -----------------------> | An include relationship specifies how the behavior for the inclusion use case is inserted into the behavior defined for the base use case. |
| Extend | <<extend>>  -----------------------> | An extend relationship specifies how the behavior of the extension use case can be inserted into the behavior defined for the base use case. |
| Constraint | {condition}  -----------------------> | Show condition exists between actors an activity. |
| Anchor | ……………………… | Anchor is used to connect a note the use case in use case diagram |
| System boundary | System | The scope of a system can be represented by a system boundary. The use cases of the system are placed inside the system boundary, while the actors who interact with the system are put outside the system. The use cases in the system make up the total requirements of the system. |
| Interface |  | Interface is used to connect package and use-case. Head is linked with package and tail linked with use- case. |

**Usecase Diagram for ATM System**

****

**Usecase Diagram for Library Management System**



**Class Diagram**

**Introduction**

* Class diagram is a diagram that shows a set of classes, interfaces and collaborations and their relationships.
* Class diagrams are used to model the static design view of a system.
* Class diagrams are foundation for a couple of related diagrams: component diagrams and deployment diagrams.

**Purpose**

* Analysis and design of the static view of an application.
* Describe responsibilities of a system.
* Base for component and deployment diagrams.

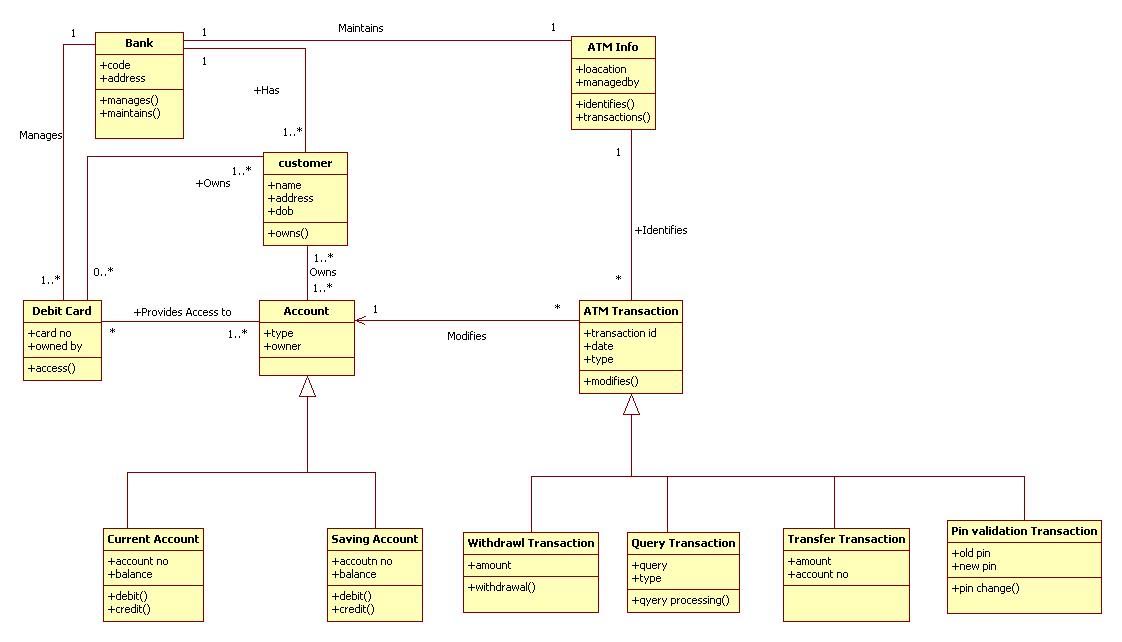
**When to use : Class Diagram**

* Useful for Forward and Reverse engineering.
* Class diagrams are useful both for abstract modeling and for designing actual programs.
* Developer uses class diagram for implementation decision.
* Business analysts can use class diagrams to model systems from the business perspective.

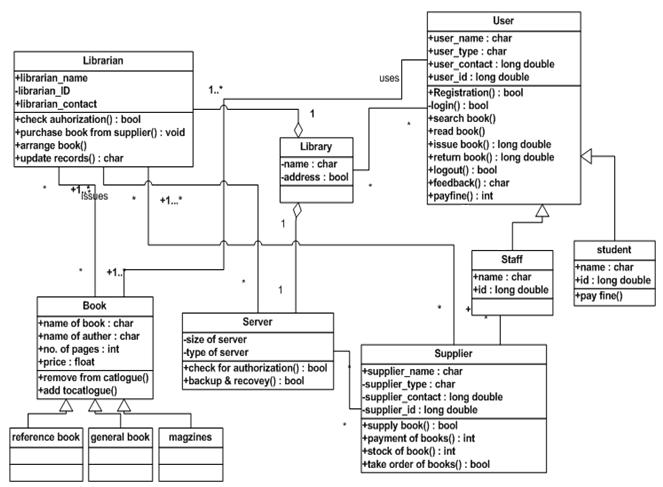
**Class diagram contents**

|  |  |  |
| --- | --- | --- |
| **Name** | **Representation** | **Description** |
| Class |  | Class is an entity of class diagram. It describes group of objects with same properties and behavior. |
| Object | Object name : class | An object is instance or occurrence of a class. |
| Link | Object2  Object1 | A link is a physical or conceptual connection among objects. |
| Association |  | An association is a description of links with common structure and common semantics |
| Multiplicity | One to one(1..1)  One to many(1..\*)  Many to many(\*..\*)  Many to one(\*..1) | Multiplicity specifies the number of instances of one class that may relate to a single instance of an associated class. It is a constraint on the cardinality of a set |
| Association class |  | It is an association that is  a class that describes the association with attributes. |
| Cardinality |  | It describes the count of elements from collection |
| Ordering |  | It is used to indicate an ordered set of objects with no  Duplication allowed. |
| Sequence |  | A sequence is an ordered collection of elements with duplicates allowed. |
| Qualified Association | Image result for images for qualified association | Qualification increases the precision of a model. It is used to avoid many to many multiplicities and it converts into one to one multiplicity. |
| Generalization |  | Generalization organizes classes by their super class and sub class relationships. |
| Enumeration |  | An enumeration is a data type that has finite set of values. |
| Aggregation |  | It is a strong form of association in which an aggregation object is made of constituent parts. |
| Abstract class |  | It is a class that has no direct instances. |
| Package |  | A package is a group of elements with common theme. |
| Concrete class |  | It is a class that is intangible. It can have direct instances. Class2 is example of concrete class. |

**Class Diagram for ATM System**

****

**Class Diagram for Library Management System**



**Sequence Diagram**

**Introduction**

* A sequence diagram is an interaction diagram that emphasizes the time ordering of messages.
* Graphically a sequence diagram is a table that shows objects arranged along X axis and messages, ordered in increasing time, along Y axis.
* Sequence diagrams model the dynamic aspects of a software system.

**Purpose**

* To capture the behavior in a system.
* To describe the structural organization of objects.
* To describe interaction among objects.
* To describe the message flow in a system.

**When to use : Sequence Diagram**

* Sequence diagram can be a helpful modeling tool when the dynamic behavior of objects needs to be observed in a particular use case or when there is a need for visualizing the “big picture of message flow”.
* A company’s technical staff could utilize sequence diagrams in order to document the behavior of a future system.
* It is during the design period that developers and architects utilize the diagram to showcase the system’s object interactions, thereby putting out a more fleshed out overall system design.

**Sequence diagram contents**

|  |  |  |
| --- | --- | --- |
| **Name** | **Representation** | **Description** |
| Object | [instance] : class | It represents the existence of an object of a particular time. |
| Lifeline |  | Lifeline represents the duration during which an object is alive and interacting with other objects in the system. It is represented by dashed lines. |
| Scope |  | It shows the time period during which an object or actor is performing an action. |
| Message Transition | message  **>** | To send message from one object to another. |
| Message with Attribute | {attributes}  **>** | To send message with some particular attribute. |
| Message with constraint | {constraint}  **>** | To send message from one object to another with some constraint. |
| Acknowledgement | {acknowledgement}  **<- - - - - - - - - - - -** | It represents communication between objects conveying acknowledgement. |
| Self Message |  | Self message occurs when an object sends message to itself. |
| Recursive Message |  | Self message occurs when an object sends message to itself within recursive scope. |

**Sequence Diagram for ATM System**

****

**Sequence Diagram for Library Management System**

****

**Collaboration Diagram**

**Introduction**

* A collaboration diagram is an interaction diagram that emphasizes the structural organization of the objects that send and receive messages.
* Graphically a collaboration diagram is a collection of vertices and arcs.
* Collaboration diagrams model the dynamic aspects of a software system.

**Purpose**

* To capture the behavior in a system.
* To describe the structural organization of objects.
* To describe interaction among objects.

**When to use : Collaboration Diagram**

* A company’s technical staff could utilize collaboration diagrams in order to document the behavior of a future system.
* It is during the design period that developers and architects utilize the diagram to showcase the system’s object interactions, thereby putting out a more fleshed out overall system design.

**Collaboration diagram contents**

|  |  |  |
| --- | --- | --- |
| **Name** | **Representation** | **Description** |
| Object | [Instance] : Class | It represents the existence of an object of a particular time. |
| Link |  | Creates a link between two objects. |
| Note |  | Note is a symbol to render constraints and comments related to an element or a group of elements. |
| Link to self |  | Creates a link to self |
| Link message |  | Creates a message on a link |
| Reverse link message |  | Creates a message on the link but in the reverse direction |

**Collaboration Diagram for ATM System**

****

**Collaboration Diagram for Library Management System**

****