Model-View-Controller and corresponding UML class diagram

Model-View-Controller

- Serves as a basis of interaction management in many web-based systems.
- Decouples three major interconnected components:
    - The model is the central component of the pattern that directly manages the data, logic and rules of the application. It is the application's dynamic data structure, independent of the user interface.
    - A view can be any output representation of information, such as a chart or a diagram. Multiple views of the same information are possible.
    - The controller accepts input and converts it to commands for the model or view.
- Supported by most language frameworks.

Controller: Maps user actions to model updates. Selects View (Controller --> Model :State Change, Controller --> View : View Selection)

View: Renders model, Requests model updates, Sends user events to controller (View --> Controller : User events, View --> Model : State query)

Model: Encapsulates application state, Notifies view of state changes. (Model --> View : Change Notification)

Overview of Model-View-Controller (MVC) pattern:

Description: Separates presentation and interaction from the system data. The system is structured into three logical components that interact with each other. The Model component manages the system data and associated operations on that data. The View component defines and manages how the data is presented to the user. The Controller component manages user interaction (e.g., key presses, mouse clicks, etc.) and passes these interactions to the View and the Model.

Problem description: The display presented to the user frequently changes over time in response to input or computation. Different users have different needs for how they want to view the program.s information. The system needs to reflect data changes to all users in the way that they want to view them, while making it easy to make changes to the user interface.

Solution description: This involves separating the data being manipulated from the manipulation logic and the details of display using three components: Model (a problem-domain component with data and operations, independent of the user interface), View (a data display component), and Controller (a component that receives and acts on user input).

Consequences: Advantages: views and controllers can be easily be added, removed, or changed; views can be added or changed during execution; user interface components can be changed, even at runtime. Disadvantages: views and controller are often hard to separate; frequent updates may slow data display and degrade user interface performance; the MVC style makes user interface components (views, controllers) highly dependent on model components.

To further clarify “Model-View-Controller” architecture, let's explore a specific example.

<u>MVC (Model-View-Controller) architecture example:</u>
Problem Description:
A car rental company wants to develop a software system that will enable customers to rent cars online. Customers should be able to browse available cars, select a car to rent, and make a reservation. The system should keep track of the availability of cars, as well as the reservations made by customers.

Identify Use Cases

- *Browse available cars*
- *Select a car to rent*
- *Make a reservation*

UML class diagram for above problem with MVC architecture:

```
 @startuml
  class CarRentalView {
    +displayCar()
    +displayReservationDetails()
  }

class CarRentalController {
    -model: CarRentalModel
    -view: CarRentalView
    +searchCars()
    +reserveCars()
  }

class CarRentalModel {
    -customers: List<>
    -cars: List<>
    -reservations: List<>
    +fetchCars()
    +updateReservationDetails()
  }
CarRentalView -- CarRentalController: Request/Response
CarRentalController --> CarRentalModel : Request Data
CarRentalModel --> Database : Connection
@enduml
```

*The **Model** includes the **Customer**, **Car**, and **Reservation** entities, along with methods for fetching for cars and making reservations.*
*The **View** is represented by the CarRentalView, which defines methods for displaying cars and reservation details.*
*The **Controller** is represented by the CarRentalController class, which interacts with the Model and View to handle user actions and update the system state.*