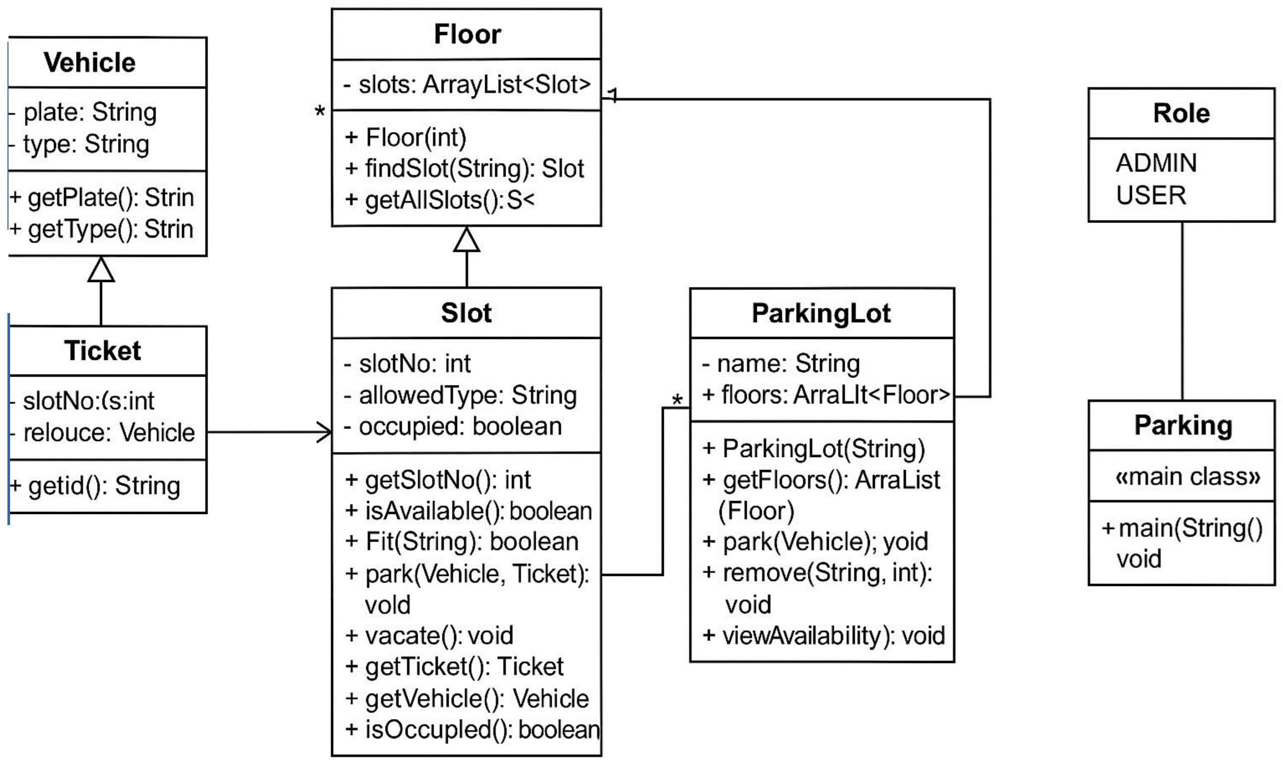
PARKINGSLOT

CLASS DIAGRAM



OOP CONCEPTS

Class/Object:
new Vehicle(...), new ParkingLot(...)

Encapsulation:
private fields, public getters/setters in Slot, Vehicle, etc.

Abstraction:
lot.park(), lot.remove() hide complexity from the main program

Enum:
enum Role { ADMIN, USER }

Aggregation:
ParkingLot → Floor → Slot → Vehicle, Ticket

Principles

SOLID

1. SRP (Single Responsibility Principle)

Each class in the code has a single, well-defined responsibility:

Vehicle: Holds vehicle details.
Ticket: Manages ticket information.
Slot: Manages parking slot state and vehicle assignments.
Floor: Manages a group of slots.
Payment: Calculates and prints payment.
ParkingLot: Handles overall parking operations.

2. LSP (Liskov Substitution Principle)

While not explicitly using inheritance, the Vehicle class can be extended (e.g., Car, Bike, Truck), and such subclasses would work with existing Slot, Ticket, and Payment classes without breaking behavior.

3. ISP (Interface Segregation Principle)

If you introduce an interface like ParkingService with only necessary methods (parkVehicle, removeVehicle, checkAvailability), classes are not forced to implement unused methods.

2. Don’t Repeat Yourself (DRY)

Your code avoids duplication by using centralized logic:

- Slot type-checking (`Fit()`), parking (`park()`), and vacating (`vacate()`) are reused consistently.

- ❿Ticket generation logic is reused and systematic.

3. Keep It Simple, Stupid (KISS)

Design is straightforward and readable:

- ❿ Simple control flow in `main()` and `ParkingLot`.
- ❿Intuitive class designs without complex inheritance or patterns.