**Implementation:**

While designing, we have to bear in mind about potential extensions. For instance, multi-level parking, dynamic pricing changes, changes to park hours, etc.

**Entities of the System**:

The system will be designed using the following entities, starting with smaller ones.

1. **Vehicle** :

class Vehicle {

constructor(vehicleNumber, vehicleColor) {  
 this.vehicleNumber = vehicleNumber;  
 this.vehicleColor = vehicleColor;  
}  
  
getVehicleNumber() {  
 return vehicleNumber;  
}  
  
setVehicleNumber(vehicleNumber) {  
 this.vehicleNumber = vehicleNumber;  
}  
  
getVehicleColor() {  
 return vehicleColor;  
}  
  
setVehicleColor(vehicleColor) {  
 this.vehicleColor = vehicleColor;

}

1. **ParkingSlot**

class Slot {  
   
   
 constructor(id, number) {  
 this.id = id;  
 this.number = number;  
 }  
  
 getParkVehicle() {  
 return parkVehicle;  
 }  
  
 setParkVehicle(parkVehicle) {  
 this.parkVehicle = parkVehicle;  
 }  
  
 removeVehicle() {  
 parkVehicle = null;  
 this.isEmpty = false;  
 }  
  
 placeVehicle(parkVehicle) {  
 this.parkVehicle = parkVehicle;  
 this.isEmpty = true;  
 }  
  
 getSlotNumber() {  
 return number;  
 }  
  
 setNumber(number) {  
 this.number = number;  
 }  
  
  
 isEmpty() {  
 return isEmpty;  
 }  
  
 setEmpty(boolean empty) {  
 isEmpty = empty;  
 }  
  
 getId() {  
 return id;  
 }  
  
 setId(id) {  
 this.id = id;  
 }  
}

3. **Parking**

class Parking extends ParkingFloor {  
   
Parking(floorNumber) {  
 this.floorNumber = floorNumber;  
 this.slots = []  
}  
  
getParkingFloor(floorNumber) {  
 if (*parkingLot* == null)  
 *parkingLot* = new Parking(floorNumber);  
 return *parkingLot*;  
}  
  
clearAll() {  
 *parkingLot* = null;  
}  
  
Slot getNextEmptySlotOnFloor() throws NoEmptySlotAvailable {  
 for (Slot slot : slots) {  
 if (!slot.isEmpty()) {  
 return slot;  
 }  
 }  
 throw new NoEmptySlotAvailable("For floorNumber " + floorNumber + " No Empty Slot available");  
}  
  
createParkingSLot(numberOfSlots) {  
 if (inputValidator(InputValidator.*isValidSlotNumber*(numberOfSlots)) || slots.size() > 0)  
 return false;  
  
 for (int i = 1; i <= numberOfSlots; i++) {  
 slots.add(new Slot(UUID.*randomUUID*().toString(), i));  
 }  
 console.log("Created a parking lot with %s slots %n", numberOfSlots);  
 return true;  
}  
  
inputValidator(validSlotNumber) {  
 if (!validSlotNumber) {  
 return true;  
 }  
 return false;  
}  
  
parkVehicle(vehicle) throws NoEmptySlotAvailable {  
 Slot nextEmptySlotOnFloor = getNextEmptySlotOnFloor();  
 nextEmptySlotOnFloor.placeVehicle(vehicle);  
 console.log("Allocated slot number: %d \n", nextEmptySlotOnFloor.getSlotNumber());  
 return true;  
}  
  
public int unParkVehicle(int slotNumber) {  
 if (slotNumber <= 0) {  
 throw new InvalidSlotNumberException(String.*format*("%d is invalid slot number,slotNumber > 1", slotNumber));  
 }  
 Slot slot = slots.get(slotNumber - 1);  
 if (slot != null) {  
 slot.removeVehicle();  
 } else {  
 throw new InvalidSlotNumberException(String.*format*("%d is invalid slot number", slotNumber));  
 }  
 return slotNumber;  
}  
  
printStatus() {  
 console.log("Slot No. Registration No Color");  
 slots.forEach(slot->{  
 if (!slots.isEmpty()) {  
 Vehicle parkVehicle = slot.getParkVehicle();  
 if (parkVehicle != null)  
 System.*out*.printf("%d %s %s\n", slot.getSlotNumber(), parkVehicle.getVehicleNumber(), parkVehicle.getVehicleColor());  
 }  
 });  
}  
  
getVehicleNumbersByColor(color) {  
 vehicleNumbers = []  
 slots.forEach(slot->{  
 if (slot.isEmpty() && slot.getParkVehicle().getVehicleColor().equalsIgnoreCase(color)) {  
 vehicleNumbers.push(slot.getParkVehicle().getVehicleNumber());  
 }  
 });  
 if (vehicleNumbers.isEmpty()) {  
 throw new VehicleNotFoundException(String.*format*("Vehicle not found for color %s", color));  
 }  
 return vehicleNumbers;  
}  
  
getSlotNumbersByColor(color) {  
 slotNumbers = this.slots.stream()  
 .filter(slot->slot.isEmpty() && slot.getParkVehicle().getVehicleColor().equalsIgnoreCase(color)).map(Slot::getSlotNumber)  
 .collect(Collectors.*toList*());  
 if (slotNumbers.isEmpty()) {  
 throw new VehicleNotFoundException(String.*format*("Vehicle not found for color %s", color));  
 }  
 return slotNumbers;  
}  
  
getSlotNumberByVehicleNumber(vehicleNumber) {  
 slotOptional = slots.stream()  
 .filter(slot->slot.getParkVehicle().getVehicleNumber().equalsIgnoreCase(vehicleNumber)).map(Slot::getSlotNumber)  
 .findAny();  
 return slotOptional.orElseThrow(()->new VehicleNotFoundException(String.*format*("Provided vehicle number %s is not present", vehicleNumber)));  
}  
}

We can pass ParkingCost from outside the aforesaid class as we are aware that its implementation may change in the future. Although we may also put it in the ParkingLot class.

As of right now, we have finished building the parking system, and I'm using enum to interface with it. Each command will implement my enum's extension of Interface CommandI and take the proper action.