1. Order Pizza from Dominos

User can

a. Start an order
b. Update the order
c. Review the order
d. Pay the order
e. Cancel the order

Class: User
Data: name, address, phone, balance, orders
Behavior:

```
startOrder() {
        Order order = new Order
        this.orders.add(order)
        order.status = CREATED
}

updateOrder(order, item, isAdded) {
        if order not exist in this.orders:
                error
        if isAdded:
                order.addItem(item)
        else:
                order.removeItem(item)
}

reviewOrder(order) {
        if order not exist in this.orders:
                error
        order.show()
}

payOrder(order) {
        if order not exist in this.orders:
                error
        balance -=  order.price
        order.status = PAID
}

cancelOrder(order) {
        if order not exist in this.orders:
                error
        balance += order.price
        order.status = CANCELLED
}
```

```
Class: Order
Data: items, price
Behavior:
show() {
        for item in items:
                item.show()
        print(this.price)
}

addItem(item) {
        this.items.add(item)
        this.price += item.price
}

removeItem(item) {
        if item not exist in this.items:
                error
        items.remove(item)
        this.price -= item.price
}

Class: Item
Data: name, price
Behavior:
show() {
        print(name, price)
}
```

2. Design a platform for buying tickets of local events

User can

a. Browse an event
b. Buy a ticket
c. Review tickets
d. Cancel ticket

Class: User
Data: name, phone, address, tickets, balance
Behavior:

```
browseEvent(event) {
    if event exist:
        event.show()
    else:
        error
}
buyTicket(event) {
    if event not exist:
        error
    else:
        ticket = event.createTicket(this)
        this.balance -= ticket.price
        this.tickets.add(ticket)
}

reviewTickets() {
    for (ticket in this.tickets) {
        ticket.show()
    }
}

cancelTicket(ticket) {
if ticket not exist in this.tickets:
    error
else:
    this.balance += ticket.price
    this.tickets.remove(ticket)
}
```

Class: Event
Data: name, location, date, description, price
Behavior:

```
show() {
    print(this.name, this.location, this.date, this.description, this.price)
}
```

```
createTicket(user) {
        Ticket ticket = new Ticket
        ticket.event = this
        ticket.owner = user
        ticket.price = this.price
        return ticket
}
```

Class: Ticket
Data: event, owner, price, isExpired
Behavior:

```
show() {
        print(event, price, isExpired, pwner)
}


updateIsExpired() {
        today = Date.today()
        if today.priorTo(event.date) or today.equals(event.date):
                this.isExpired = false
        else:
                this.isExpired = true
}
```

3. Design a Car Rental System

User can

a. Browse a car
b. Rent a car
c. Return a car

Class: User
Data: name, address, balance, leases, creditScore
Behavior:

```
browseCar(car) {
        if car exist:
                car.show()
        else:
                error
}

rentCar(car, dateToReturn) {
        if car.isOccupied():
                error
        else:
                Lease lease = new lease
                lease.car = car
                lease.user = this
                lease.dateToReturn = dateToReturn
                lease.calculatePrice()
                lease.calculateDeposit()
                this.balance -= lease.price + lease.deposit
                this.leases.add(lease)
                car.lease = lease
}

returnCar(lease) {
        today = Date.today
        if (today.priorOrEqualTo(lease.dateToReturn):
                user.creditScore += 10
                user.balance += lease.deposit
        else:
                user.creditScore -= 20
                user.balance +=  part of lease.deposit
        user.leases.remove(lease)
        car.lease = null

}
```

Class: Car
Data: brand, model, number, description, lease, pricePerDay
Behavior:

```
show() {
        print(brand, model, number, description)
}

isOccupied() {
        if (lease == null):
                return false
        return true
}
```

Class: Lease
Data: car, user, price, deposit, dateToReturn
Behavior:

```
calculatePrice() {
        today = Date.today
        price = (dateToReturn – today) * car.pricePerDay
        this.price = price
}

calculateDeposit() {
        if user.creditScore <= 200:
                this.deposit = price * 20
        else if user.creaditScore <= 500:
                this.deposit = price * 15
        else
                this.deposit = price * 10
}
```

4. Design a Parking Lot

User can

a. Register a car
b. Start parking
c. Find out where the car is parked at
d. Stop parking

Class: User
Data: name, id, balance, cars
Behavior:

```
registerCar(car) {
        if car exist in this.cars:
                error
        this.cars.add(car)
}
parkCar(car, parkingSpace) {
        if (parkingSpace.isOccupied):
                print("The parking space has been occupied, please find another one")
                return false
                parkingSpace.startParking(car)
                car.parkingSpace = parkingSpace
}

findCar(car) {
        return car.getLocation
}

takeCar(car) {
        this.balance -= parkingSpace.getTotalPrice()
        parkingSpace.stopParking()
        car.parkingSpace = null
}
```

Class: Car
Data: number, parkingSpace, price
Behavior:

```
getLocation() {
        return this.parkingSpace.number
}
```

Class: ParkingSpace
Data: number, car, parkStartTime

Behavior:

```
isOccupied() {
        if this.car != null:
                return true
        return false
}

startParking() {
        if this.car != null:
                error
        this.car = car
        parkStartTime  =now()
}

stopParking() {
        this.car = null
        this.parkStartTime = null
}

getTotalPrice() {
        duration = now() – parkStartTime
        return duration * price
}
```

5. Design a Traffic Controller System for a Junction

User can

a. Display the situation of a junction
b. Block (red light) the traffic from a direction for a junction
c. Unblock (green light) the traffic from a direction for a junction

```
Class: User
Data: id
Behavior:
displayTraffic(junction) {
        if junction not exist:
                error
        else:
                junction.show()
}

blockTraffic(junction, direction) {
        if junction not exist:
                error
        else:
                junction.blockTraffic(direction)
}

unblockTraffic(junction, direction) {
        if junction not exist:
                error
        else:
                junction.unblockTraffic(direction)
}

Class: Junction
Data: location, directions
Behavior:
blockTraffic(direction) {
        direction.isBlocked = true
}

unblockTraffic(direction) {
        direction.isBlocked = false
}
show() {
        For direction in this.directions:
                direction.show()
}
```

Class: Direction
Data: location, isBlocked
Behavior:

```
show() {
        print(location, isBlocked)
}
```