CLASS DIAGRAM FOR CAR RENTAL BUSINESS SIMULATOR

<<interface>>
Observable

+ registerObserver(Observer): void
+ removeObserver(Observer):void
+ notifyObservers(Object): void

<<interface>>
Observer

+ update(Object): void

Summarizer

+ update(Object): void
+ generateDailyReport(Recorder): void
+ generateOverallStatus(Recorder): void
+ close(): void

Store

- recorder: Recorder

getNewCar(Car.Type): Car
decorateCar(): Car
+ registerObserver(Observer): void
+ removeObserver(Observer):void
+ notifyObservers(Object): void
+ addNewRental(Transaction): boolean
+ completeRental(Transaction): boolean

Recorder

- transaction: Transaction
- store: Store
- overallStatusReport: Report
- rule: Rule
- lplCarMap: Map<String,Car>

+ addNewRental(Transaction): boolean
+ completeRental(Transaction): boolean

<<interface>>
Rule

+ validate(Object): boolean

BusinessRule

+ validate(Object): boolean

Report

CarRental

getNewCar(Car.Type): Car

Simulator

Transaction

Customer

- customerID: String

Regular

Business

Car

- licensePlateNumber: String
- pricePerDay: Double

getRentalCost(Integer): Double

Luxury

getRentalCost(Integer): Double

Economy

getRentalCost(Integer): Double

CarOption

- car: Car
- optionPrice: Double

getRentalCost(Integer): Double

ChildSeat

getRentalCost(Integer): Double

GPSModule

getRentalCost(Integer): Double

UniqueIDGenerator

- instance: UniqueIDGenerator

+ static getInstance(): UniqueIDGenerator
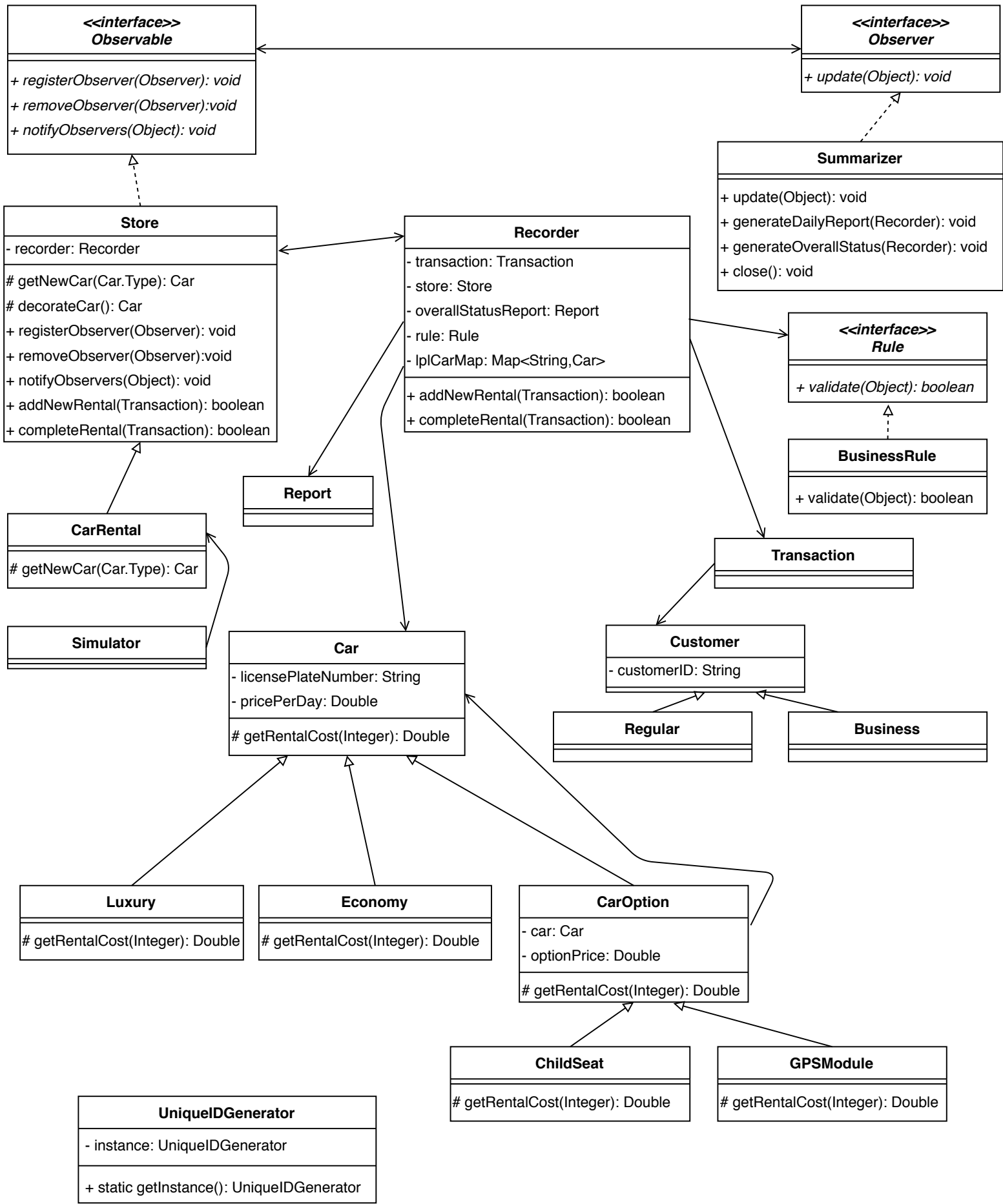
Note: Since the problem is huge there are a lot more utility classes, methods and attributes that has been used to complete the whole program but has been deliberately excluded from this diagram. The classes which constitute the main framework and which have a role in implementation of design pattern has been presented here. Adding utility classes and methods would only increase the complexity of the diagrams without revealing any important aspects of the implemented Car Rental Business Simulator.