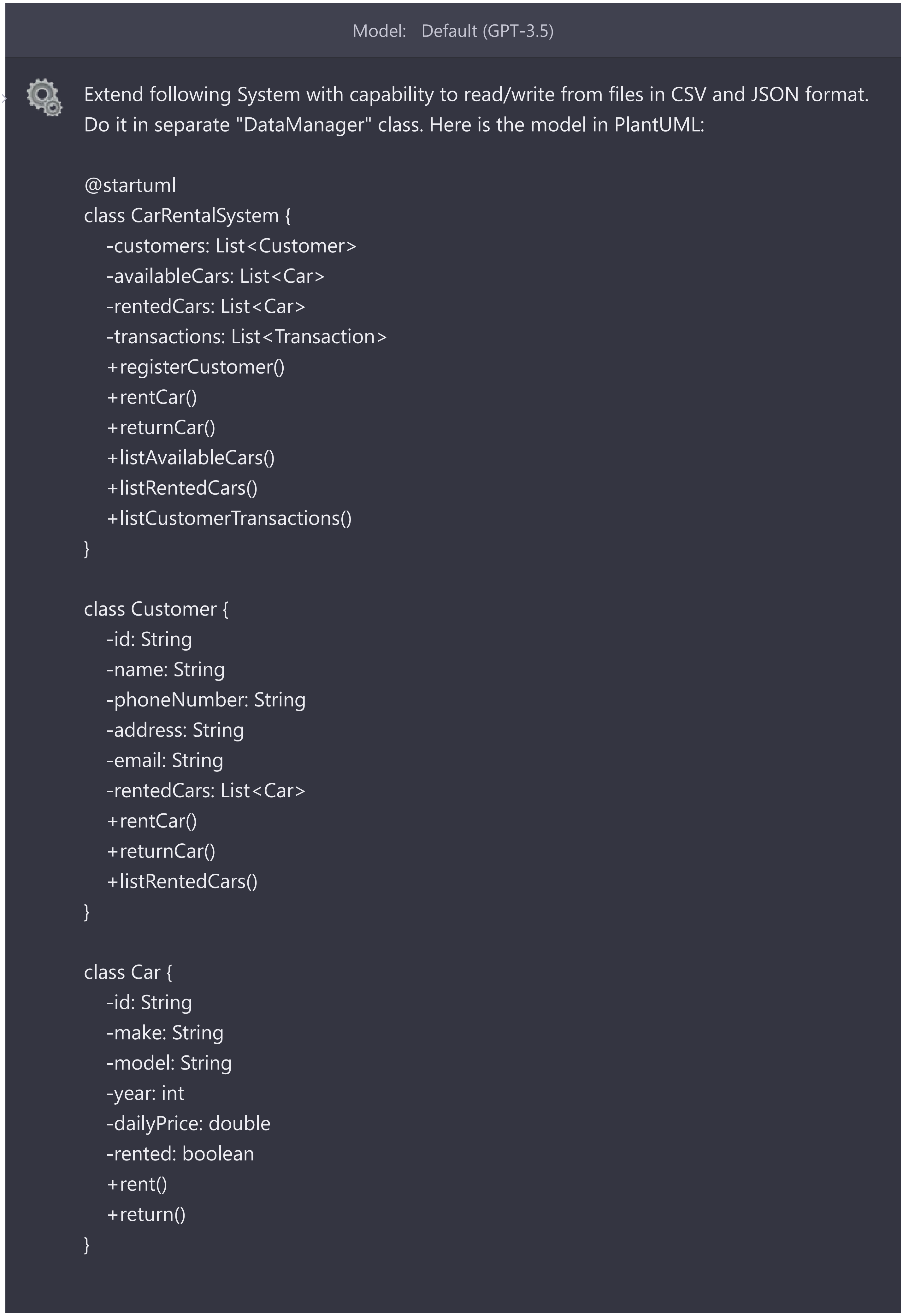
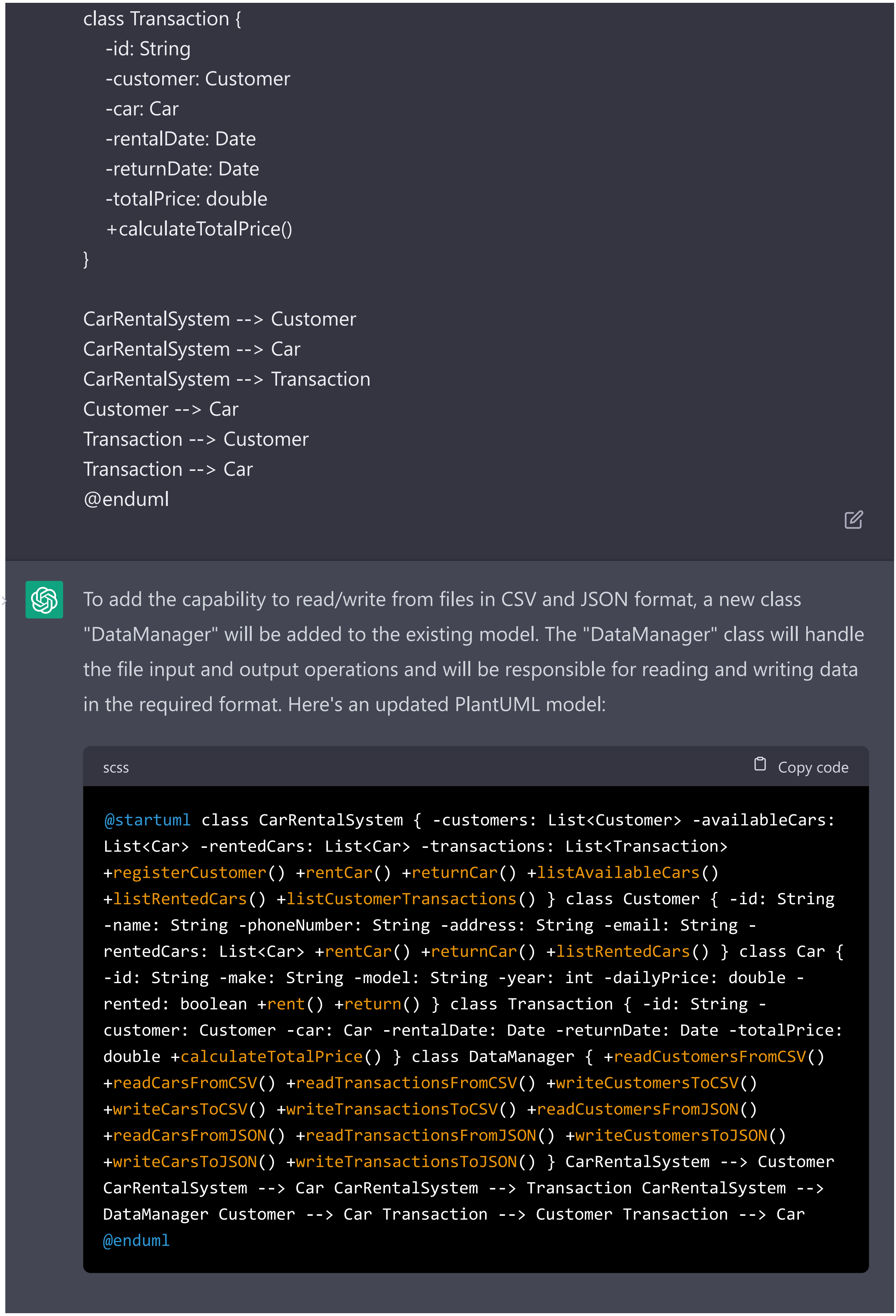


Extend following System with capability to read/write from files in CSV and JSON format. Do it in separate "DataManager" class. Here is the model in PlantUML:

```
@startuml
class CarRentalSystem {
    -customers: List<Customer>
    -availableCars: List<Car>
    -rentedCars: List<Car>
    -transactions: List<Transaction>
    +registerCustomer()
    +rentCar()
    +returnCar()
    +listAvailableCars()
    +listRentedCars()
    +listCustomerTransactions()
}

class Customer {
    -id: String
    -name: String
    -phoneNumber: String
    -address: String
    -email: String
    -rentedCars: List<Car>
    +rentCar()
    +returnCar()
    +listRentedCars()
}

class Car {
    -id: String
    -make: String
    -model: String
    -year: int
    -dailyPrice: double
    -rented: boolean
    +rent()
    +return()
}
```



```
class Transaction {
    -id: String
    -customer: Customer
    -car: Car
    -rentalDate: Date
    -returnDate: Date
    -totalPrice: double
    +calculateTotalPrice()
}

CarRentalSystem --> Customer
CarRentalSystem --> Car
CarRentalSystem --> Transaction
Customer --> Car
Transaction --> Customer
Transaction --> Car
@enduml
```

To add the capability to read/write from files in CSV and JSON format, a new class "DataManager" will be added to the existing model. The "DataManager" class will handle the file input and output operations and will be responsible for reading and writing data in the required format. Here's an updated PlantUML model:

```
@startuml class CarRentalSystem { -customers: List<Customer> -availableCars:
List<Car> -rentedCars: List<Car> -transactions: List<Transaction>
+registerCustomer() +rentCar() +returnCar() +listAvailableCars()
+listRentedCars() +listCustomerTransactions() } class Customer { -id: String
-name: String -phoneNumber: String -address: String -email: String -
rentedCars: List<Car> +rentCar() +returnCar() +listRentedCars() } class Car {
-id: String -make: String -model: String -year: int -dailyPrice: double -
rented: boolean +rent() +return() } class Transaction { -id: String -
customer: Customer -car: Car -rentalDate: Date -returnDate: Date -totalPrice:
double +calculateTotalPrice() } class DataManager { +readCustomersFromCSV()
+readCarsFromCSV() +readTransactionsFromCSV() +writeCustomersToCSV()
+writeCarsToCSV() +writeTransactionsToCSV() +readCustomersFromJSON()
+readCarsFromJSON() +readTransactionsFromJSON() +writeCustomersToJSON()
+writeCarsToJSON() +writeTransactionsToJSON() } CarRentalSystem --> Customer
CarRentalSystem --> Car CarRentalSystem --> Transaction CarRentalSystem -->
DataManager Customer --> Car Transaction --> Customer Transaction --> Car
@enduml
```

The new "DataManager" class has methods to read and write data in CSV and JSON format for customers, cars, and transactions. These methods can be called from the "CarRentalSystem" class whenever data needs to be saved or retrieved from files.

Note that the implementation of the file input and output operations in the "DataManager" class will depend on the programming language used for implementation.

Extend "DataManager" methods with the entity type and data type enumerations.

To further extend the "DataManager" class with entity type and data type enumerations, we can add two new enumerations: "EntityType" and "DataType". The "EntityType" enumeration will define the types of entities that can be managed, and the "DataType" enumeration will define the type of data that can be read or written.

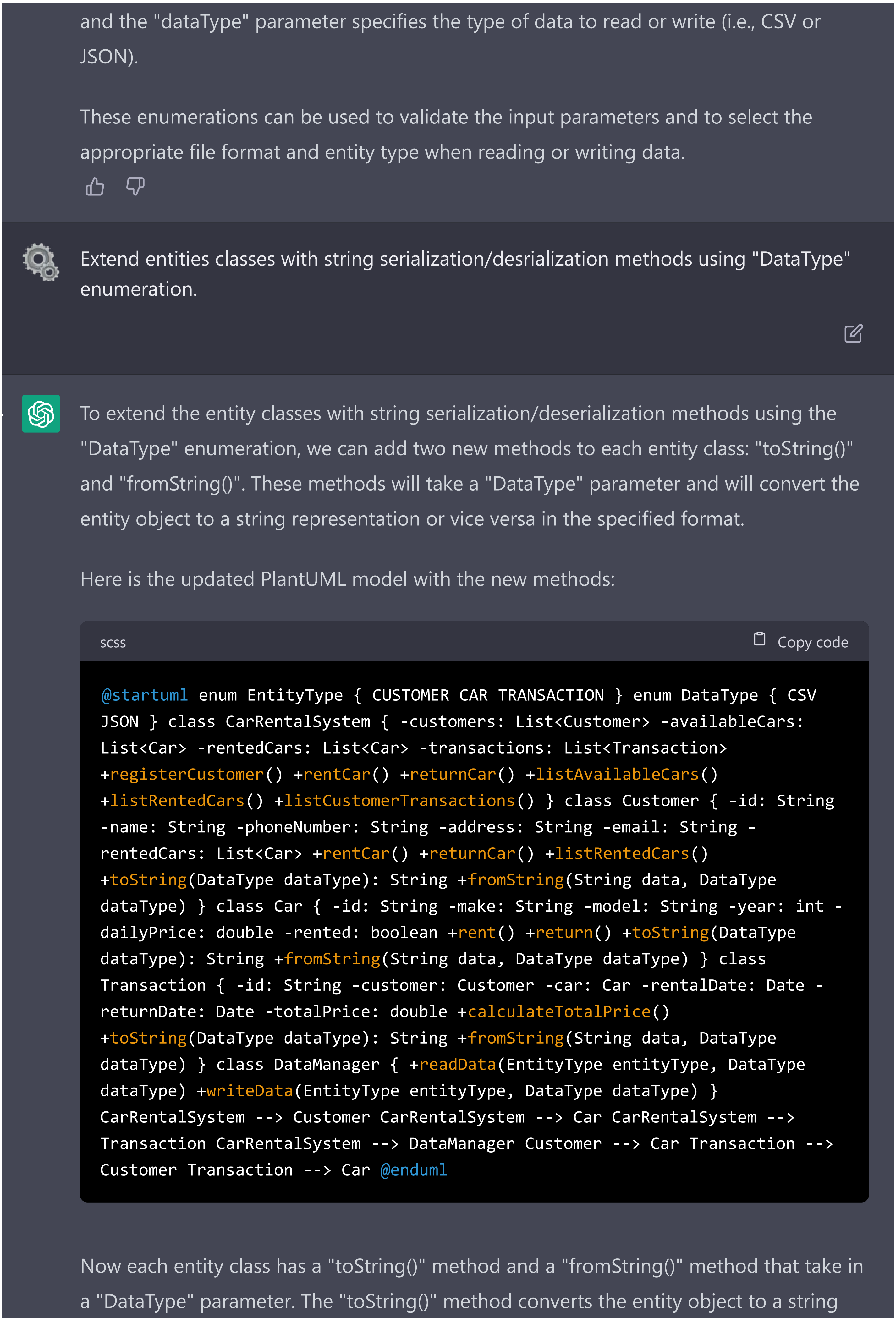
Here is the updated PlantUML model with the new enumerations:

typescript

Copy code

```
@startuml enum EntityType { CUSTOMER CAR TRANSACTION } enum DataType { CSV
JSON } class CarRentalSystem { -customers: List<Customer> -availableCars:
List<Car> -rentedCars: List<Car> -transactions: List<Transaction>
+registerCustomer() +rentCar() +returnCar() +listAvailableCars()
+listRentedCars() +listCustomerTransactions() } class Customer { -id: String
-name: String -phoneNumber: String -address: String -email: String -
rentedCars: List<Car> +rentCar() +returnCar() +listRentedCars() } class Car {
-id: String -make: String -model: String -year: int -dailyPrice: double -
rented: boolean +rent() +return() } class Transaction { -id: String -
customer: Customer -car: Car -rentalDate: Date -returnDate: Date -totalPrice:
double +calculateTotalPrice() } class DataManager { +readData(EntityType
entityType, DataType dataType) +writeData(EntityType entityType, DataType
dataType) } CarRentalSystem --> Customer CarRentalSystem --> Car
CarRentalSystem --> Transaction CarRentalSystem --> DataManager Customer -->
Car Transaction --> Customer Transaction --> Car @enduml
```

The "DataManager" class now has two methods, "readData()" and "writeData()", which take in two parameters each: "entityType" and "dataType". The "entityType" parameter specifies the type of entity that the data pertains to (i.e., customer, car, or transaction),



and the "dataType" parameter specifies the type of data to read or write (i.e., CSV or JSON).

These enumerations can be used to validate the input parameters and to select the appropriate file format and entity type when reading or writing data.

Extend entities classes with string serialization/desrialization methods using "DataType" enumeration.

To extend the entity classes with string serialization/deserialization methods using the "DataType" enumeration, we can add two new methods to each entity class: "toString()" and "fromString()". These methods will take a "DataType" parameter and will convert the entity object to a string representation or vice versa in the specified format.

Here is the updated PlantUML model with the new methods:

```
@startuml enum EntityType { CUSTOMER CAR TRANSACTION } enum DataType { CSV
JSON } class CarRentalSystem { -customers: List<Customer> -availableCars:
List<Car> -rentedCars: List<Car> -transactions: List<Transaction>
+registerCustomer() +rentCar() +returnCar() +listAvailableCars()
+listRentedCars() +listCustomerTransactions() } class Customer { -id: String
-name: String -phoneNumber: String -address: String -email: String -
rentedCars: List<Car> +rentCar() +returnCar() +listRentedCars()
+toString(DataType dataType): String +fromString(String data, DataType
dataType) } class Car { -id: String -make: String -model: String -year: int -
dailyPrice: double -rented: boolean +rent() +return() +toString(DataType
dataType): String +fromString(String data, DataType dataType) } class
Transaction { -id: String -customer: Customer -car: Car -rentalDate: Date -
returnDate: Date -totalPrice: double +calculateTotalPrice()
+toString(DataType dataType): String +fromString(String data, DataType
dataType) } class DataManager { +readData(EntityType entityType, DataType
dataType) +writeData(EntityType entityType, DataType dataType) }
CarRentalSystem --> Customer CarRentalSystem --> Car CarRentalSystem -->
Transaction CarRentalSystem --> DataManager Customer --> Car Transaction -->
Customer Transaction --> Car @enduml
```

Now each entity class has a "toString()" method and a "fromString()" method that take in a "DataType" parameter. The "toString()" method converts the entity object to a string representation in the specified format, and the "fromString()" method converts the string representation back to an entity object.

These methods can be used by the "DataManager" class to serialize and deserialize entity objects to and from file formats. The "DataType" parameter can be passed to the appropriate method in each entity class to ensure that the data is converted to the correct format.

Make realtions to enumeration types

To make the relationships between the new enumeration types and the other classes in the model more explicit, we can add multiplicity and association labels to the existing relationships in the diagram.

Here is the updated PlantUML model:

scss Copy code

```
@startuml enum EntityType { CUSTOMER CAR TRANSACTION } enum DataType { CSV
JSON } class CarRentalSystem { -customers: List<Customer> -availableCars:
List<Car> -rentedCars: List<Car> -transactions: List<Transaction>
+registerCustomer() +rentCar() +returnCar() +listAvailableCars()
+listRentedCars() +listCustomerTransactions() } class Customer { -id: String
-name: String -phoneNumber: String -address: String -email: String -
rentedCars: List<Car> +rentCar() +returnCar() +listRentedCars()
+toString(DataType dataType): String +fromString(String data, DataType
dataType) } class Car { -id: String -make: String -model: String -year: int -
dailyPrice: double -rented: boolean +rent() +return() +toString(DataType
dataType): String +fromString(String data, DataType dataType) } class
Transaction { -id: String -customer: Customer -car: Car -rentalDate: Date -
returnDate: Date -totalPrice: double +calculateTotalPrice()
+toString(DataType dataType): String +fromString(String data, DataType
dataType) } class DataManager { +readData(EntityType entityType, DataType
dataType) +writeData(EntityType entityType, DataType dataType) }
CarRentalSystem --> Customer CarRentalSystem --> Car CarRentalSystem -->
Transaction CarRentalSystem --> DataManager Customer --> Car : has
Transaction --> Customer : has Transaction --> Car : has DataManager -->
EntityType : manages DataManager --> DataType : uses @enduml
```

In the updated diagram, we have added association labels to the relationships between the "DataManager" class and the "EntityType" and "DataType" enumerations. The "DataManager" class "manages" the entity types and "uses" the data types. This makes it clear that the "EntityType" and "DataType" enumerations are used to specify the types of data that are managed by the "DataManager" class.

We have also added multiplicity labels to the relationships between the entity classes and the "DataType" enumeration. Each entity class can have a "toString()" and "fromString()" method that takes a "DataType" parameter, indicating that these methods are specific to the data format.

Make annotations of "CarRentalSystem" class relations

To further clarify the relationships between the "CarRentalSystem" class and the other classes in the model, we can add annotations to the existing relationships in the diagram.

Here is the updated PlantUML model:

scss Copy code

```scss
@startuml enum EntityType { CUSTOMER CAR TRANSACTION } enum DataType { CSV
JSON } class CarRentalSystem { -customers: List<Customer> -availableCars:
List<Car> -rentedCars: List<Car> -transactions: List<Transaction>
+registerCustomer() +rentCar() +returnCar() +listAvailableCars()
+listRentedCars() +listCustomerTransactions() } class Customer { -id:
String -name: String -phoneNumber: String -address: String -email: String -
rentedCars: List<Car> +rentCar() +returnCar() +listRentedCars()
+toString(DataType dataType): String +fromString(String data, DataType
dataType) } class Car { -id: String -make: String -model: String -year: int
-dailyPrice: double -rented: boolean +rent() +return() +toString(DataType
dataType): String +fromString(String data, DataType dataType) } class
Transaction { -id: String -customer: Customer -car: Car -rentalDate: Date -
returnDate: Date -totalPrice: double +calculateTotalPrice()
+toString(DataType dataType): String +fromString(String data, DataType
dataType) } class DataManager { +readData(EntityType entityType, DataType
dataType) +writeData(EntityType entityType, DataType dataType) }
CarRentalSystem -down-> Customer : manages > CarRentalSystem -down-> Car :
manages > CarRentalSystem -down-> Transaction : manages > CarRentalSystem -
up-> DataManager : uses > Customer --> Car : has Transaction --> Customer :
has Transaction --> Car : has DataManager --> EntityType : manages
DataManager --> DataType : uses @enduml
```

In the updated diagram, we have added annotations to the relationships between the "CarRentalSystem" class and the other classes in the model. We have used the "manages" label to indicate that the "CarRentalSystem" class manages the "Customer", "Car", and "Transaction" classes, and the "uses" label to indicate that it uses the "DataManager" class.

We have also added the "down" and "up" symbols to the relationships to indicate the direction of the relationships. The "CarRentalSystem" class manages the "Customer", "Car", and "Transaction" classes, so the relationships point down from the "CarRentalSystem" class to these classes. The "CarRentalSystem" class uses the "DataManager" class, so the relationship points up from the "CarRentalSystem" class to the "DataManager" class.

These annotations make it clear that the "CarRentalSystem" class is the central class in the model and manages the other classes, while the "DataManager" class is responsible for data input and output operations.