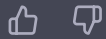




Assume Python programming



Sure, I'm ready to assume Python programming. How may I assist you with Python?



convert this code: namespace sf.systems.rentals.cars

```
{  
    public interface IMessageHandler  
    {  
        void Log(string message);  
        void ShowMessage(string message);  
    }  
}
```



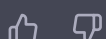
In Python, you can define the above interface using the `abc` module, which provides support for Abstract Base Classes (ABCs). Here's how you can do it:

python

Copy code

```
from abc import ABC, abstractmethod  
class IMessageHandler(ABC):  
    @abstractmethod def log(self, message): pass  
    @abstractmethod def show_message(self, message): pass
```

Note that the `abstractmethod` decorator is used to indicate that the methods `log` and `show_message` must be implemented by any class that inherits from `IMessageHandler`. Also, the class `IMessageHandler` extends from `ABC` to mark it as an abstract base class.



convert this code: namespace sf.systems.rentals.cars

```
{  
    public interface IMessageHandler  
    {  
        void Log(string message);
```

```
void ShowMessage(string message);  
}  
}
```



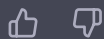
In Python, you can define the above interface using the `abc` module, which provides support for Abstract Base Classes (ABCs). Here's how you can do it:

python

Copy code

```
from abc import ABC, abstractmethod  
class IMessageHandler(ABC):  
    @abstractmethod  
    def log(self, message): pass  
    @abstractmethod  
    def show_message(self, message): pass
```

Note that the `abstractmethod` decorator is used to indicate that the methods `log` and `show_message` must be implemented by any class that inherits from `IMessageHandler`. Also, the class `IMessageHandler` extends from `ABC` to mark it as an abstract base class.



convert code: using System;

```
namespace sf.systems.rentals.cars  
{  
    public interface IErrorHandler  
    {  
        void HandleError(Exception e);  
    }  
}
```



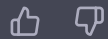
In Python, you can define the above interface as follows:

python

Copy code

```
from abc import ABC, abstractmethod  
from typing import Type  
class IErrorHandler(ABC):  
    @abstractmethod  
    def handle_error(self, e: Type[Exception]): pass
```

Note that the **abstractmethod** decorator is used to indicate that the method **handle_error** must be implemented by any class that inherits from **IErrorHandler**. Also, the parameter **e** is annotated with **Type[Exception]** to specify that it should be an instance of the **Exception** class or one of its subclasses.



convert code: namespace sf.systems.rentals.cars

```
{
    public enum EntityType
    {
        CUSTOMER,
        CAR,
        TRANSACTION
    }
}
```



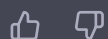
In Python, you can define the above enum as follows:

python

Copy code

```
from enum import Enum
class EntityType(Enum):
    CUSTOMER = 1
    CAR = 2
    TRANSACTION = 3
```

Note that the **Enum** class is used to define an enumeration in Python. The values are assigned using integer literals, but you can also use strings or any other hashable type.



convert code: namespace sf.systems.rentals.cars

```
{
    public enum DataType
    {
        CSV,
        JSON
    }
}
```





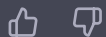
In Python, you can define the above enum as follows:

python

Copy code

```
from enum import Enum class DataType(Enum): CSV = 1 JSON = 2
```

Note that the **Enum** class is used to define an enumeration in Python. The values are assigned using integer literals, but you can also use strings or any other hashable type.



convert as interface:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace sf.systems.rentals.cars
{
    public interface IEntitiesList
    {
        List<Customer> GetRegisteredCustomers();
        Customer LookupCustomer(string customerId);
        List<Car> GetAvaliableCars();
        List<Car> GetRentedCars();
        List<Car> GetRentedCars(Customer customer);
        Car LookupCar(string carId);
        void NewTransaction(Transaction transaction);
        void ArchiveTransaction(Transaction transaction);
        void RentCar(Car car);
        void ReturnCar(Car car);
    }
}
```



In Python, you can convert the above code to an interface as follows:

python

Copy code

```

from abc import ABC, abstractmethod from typing import List class
IEntitiesList(ABC): @abstractmethod def get_registered_customers(self) ->
List[Customer]: pass @abstractmethod def lookup_customer(self, customer_id:
str) -> Customer: pass @abstractmethod def get_available_cars(self) ->
List[Car]: pass @abstractmethod def get_rented_cars(self) -> List[Car]: pass
@abstractmethod def get_rented_cars_by_customer(self, customer: Customer) ->
List[Car]: pass @abstractmethod def lookup_car(self, car_id: str) -> Car:
pass @abstractmethod def new_transaction(self, transaction: Transaction):
pass @abstractmethod def archive_transaction(self, transaction: Transaction):
pass @abstractmethod def rent_car(self, car: Car): pass @abstractmethod def
return_car(self, car: Car): pass

```

Note that we've used the **abstractmethod** decorator to indicate that the methods must be implemented by any class that inherits from **IEntitiesList**. Also, we've added type annotations to the method parameters and return values. Please note that you will need to define the **Customer**, **Car**, and **Transaction** classes separately in order for this interface to work.



convert this code: namespace sf.systems.rentals.cars

```

{
    public interface ISerializedEntity<T>
    {
        string Serialize(DataType dataType);
        T DeserializeHandler(string data, DataType dataType);
    }
}

```



In Python, you can convert the above code to an interface as follows:

python

Copy code

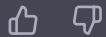
```

from abc import ABC, abstractmethod from enum import Enum class
DataType(Enum): CSV = 1 JSON = 2 class ISerializedEntity(ABC):
@abstractmethod def serialize(self, data_type: DataType) -> str: pass
@abstractmethod def deserialize_handler(self, data: str, data_type: DataType)
-> T: pass

```

Note that we've defined the **DataType** enumeration as a separate class, just like we did earlier, and we've used it as a parameter for the **serialize** and **deserialize_handler**

methods. The **abstractmethod** decorator is used to indicate that the methods must be implemented by any class that inherits from **ISerializedEntity**. Also, note that we've used a generic type **T** in the interface definition, but you will need to define the type separately in order for this interface to work.



How to use generics in Python?



In Python, you can use generics using the **typing** module. The **typing** module provides support for type hints, including support for generic types.

Here's an example of how to use a generic type in Python:

python

Copy code

```
from typing import TypeVar, List
T = TypeVar('T')
def reverse_list(lst: List[T]) -> List[T]:
    return lst[::-1]
```

In the above example, we define a function **reverse_list** that takes a list of some generic type **T** and returns a list of the same type. We use the **TypeVar** function from the **typing** module to define the generic type **T**.

You can also specify constraints on the type parameter. For example, the following code defines a generic function that takes a list of objects that can be compared using the **>** operator:

python

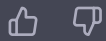
Copy code

```
from typing import TypeVar, List
T = TypeVar('T', int, float)
def greater_than_5(lst: List[T]) -> List[T]:
    return [x for x in lst if x > 5]
```

In the above example, we use the **TypeVar** function to define the generic type **T**, but we also specify that it should be either an **int** or a **float**. This allows us to use the **>** operator to compare the values in the list.

Note that Python's support for generics is not as strong as some other languages like Java or C#. However, you can use the **typing** module to define generic types and

functions in Python.



convert this code: namespace sf.systems.rentals.cars

```
{  
    public interface ISerializedExtendedEntity<T> : ISerializedEntity<T>  
    {  
        T DeserializeHandler(string data, DataType dataType, IEntitiesList entitiesList);  
    }  
}
```



In Python, you can convert the above code to an interface as follows:

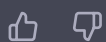
python

Copy code

```
from abc import ABC from typing import TypeVar from  
sf.systems.rentals.cars.IEntitiesList import IEntitiesList from  
sf.systems.rentals.cars.ISerializedEntity import ISerializedEntity, DataType  
T = TypeVar('T') class ISerializedExtendedEntity(ISerializedEntity[T], ABC):  
    def deserialize_handler(self, data: str, data_type: DataType, entities_list:  
IEntitiesList) -> T: pass
```

Note that we've defined the generic type **T** using the **TypeVar** function from the **typing** module. We've also imported the **IEntitiesList** and **ISerializedEntity** interfaces from their respective modules.

In Python, you don't need to use the **public** keyword to specify the visibility of a member. By default, all members of a class or interface are public. Also, since we're inheriting from **ISerializedEntity**, which already has a **deserialize_handler** method, we don't need to use the **abstractmethod** decorator for this method.



convert this code: namespace sf.systems.rentals.cars

```
{  
    public interface ISerializeOwner  
    {  
        void AssignOwner(IEntitiesList entitiesList);  
    }  
}
```

```
}  
}
```



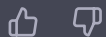
In Python, you can convert the above code to an interface as follows:

python

Copy code

```
from abc import ABC, abstractmethod from  
sf.systems.rentals.cars.IEntitiesList import IEntitiesList class  
ISerializeOwner(ABC): @abstractmethod def assign_owner(self, entities_list:  
IEntitiesList): pass
```

Note that we've used the **abstractmethod** decorator to indicate that the **assign_owner** method must be implemented by any class that inherits from **ISerializeOwner**. Also, we've added a type annotation for the **entities_list** parameter to specify that it should be an instance of the **IEntitiesList** interface.



convert this code: namespace sf.systems.rentals.cars

```
{  
    public enum RentedType  
    {  
        RENTED,  
        AVAILABLE  
    }  
}
```



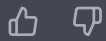
In Python, you can convert the above code to an enum as follows:

python

Copy code

```
from enum import Enum class RentedType(Enum): RENTED = 1 AVAILABLE = 2
```

Note that we've used the **Enum** class to define an enumeration in Python. The values are assigned using integer literals, but you can also use strings or any other hashable type. Also, note that we've corrected the spelling of "AVAILABLE" to match the original code.



convert this code: using Newtonsoft.Json;
using System;

```
namespace sf.systems.rentals.cars
```

```
{
```

```
    public class Car : ISerializedEntity<Car>
```

```
    {
```

```
        public string Id { get; }
```

```
        public string Make { get; }
```

```
        public string Model { get; }
```

```
        public int Year { get; }
```

```
        public double DailyPrice { get; }
```

```
        public bool Rented { get; private set; }
```

```
        public Car() { }
```

```
        public Car(string id, string make, string model, int year, double dailyPrice, bool  
rented)
```

```
        {
```

```
            Id = id;
```

```
            Make = make;
```

```
            Model = model;
```

```
            Year = year;
```

```
            DailyPrice = dailyPrice;
```

```
            Rented = rented;
```

```
        }
```

```
        internal void Rent() => Rented = true;
```

```
        internal void Return() => Rented = false;
```

```
        public string Serialize(DataType dataType) => dataType switch
```

```
        {
```

```
            DataType.CSV => $"{{Id}},{{Make}},{{Model}},{{Year}},{{DailyPrice}},{{Rented}}",
```

```
            DataType.JSON => JsonConvert.SerializeObject(this),
```

```
            _ => throw new ArgumentException($"Unknown data type {dataType}.");
```

```
        };
```

```
        public static Car Deserialize(string data, DataType dataType) => dataType switch
```

```
        {
```

```
            DataType.CSV => CreateFromCsv(data),
```

```

        DataType.JSON => JsonConvert.DeserializeObject<Car>(data),
        _ => throw new ArgumentException($"Unknown data type {dataType}.");
    };

    private static Car CreateFromCsv(string csv)
    {
        int csv_len = 6;
        string[] values = csv.Split(',');
        if (values.Length != csv_len)
        {
            throw new ArgumentException($"Invalid CSV data length - expected {csv_len}, got {values.Length}. CSV-Data: {csv}.");
        }
        return new Car(values[0], values[1], values[2], int.Parse(values[3]),
            double.Parse(values[4]), bool.Parse(values[5]));
    }

    public Car DeserializeHandler(string data, DataType dataType)
    {
        return Deserialize(data, dataType);
    }
}

```



Here's the Python version of the above code:

python

Copy code

```

from __future__ import annotations from typing import Type import json from
sf.systems.rentals.cars.ISerializedEntity import ISerializedEntity, DataType
class Car(ISerializedEntity[Car]): def __init__(self, id: str, make: str,
model: str, year: int, daily_price: float, rented: bool): self._id = id
self._make = make self._model = model self._year = year self._daily_price =
daily_price self._rented = rented @property def id(self) -> str: return
self._id @property def make(self) -> str: return self._make @property def
model(self) -> str: return self._model @property def year(self) -> int:
return self._year @property def daily_price(self) -> float: return
self._daily_price @property def rented(self) -> bool: return self._rented def
rent(self) -> None: self._rented = True def return_car(self) -> None:
self._rented = False def serialize(self, data_type: DataType) -> str: if
data_type == DataType.CSV: return f"{self._id},{self._make},{self._model},
{self._year},{self._daily_price},{self._rented}" elif data_type ==
DataType.JSON: return json.dumps(self.__dict__) else: raise

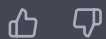
```

```

ValueError(f"Unknown data type {data_type}.") @staticmethod def
deserialize(data: str, data_type: DataType) -> Car: if data_type ==
DataType.CSV: return Car.create_from_csv(data) elif data_type ==
DataType.JSON: return Car(**json.loads(data)) else: raise
ValueError(f"Unknown data type {data_type}.") def deserialize_handler(self,
data: str, data_type: DataType) -> Car: return Car.deserialize(data,
data_type) @staticmethod def create_from_csv(csv: str) -> Car: csv_len = 6
values = csv.split(",") if len(values) != csv_len: raise ValueError(f"Invalid
CSV data length - expected {csv_len}, got {len(values)}. CSV-Data: {csv}.")
return Car(values[0], values[1], values[2], int(values[3]), float(values[4]),
bool(values[5]))

```

Note that we've used the `__future__` import to enable annotations for type hints. Also, we've used the `json` module to serialize and deserialize JSON data. Finally, we've defined the `create_from_csv` method as a static method because it doesn't depend on any instance variables of the `Car` class.



convert this code: using Newtonsoft.Json;
using System;
using System.Collections.Generic;

```

namespace sf.systems.rentals.cars
{
    public class Customer : ISerializedEntity<Customer>
    {
        public string Id { get; set; }
        public string Name { get; set; }
        public string PhoneNumber { get; set; }
        public string Address { get; set; }
        public string Email { get; set; }

        private readonly int CarRentalMaxLimit = 1;
        internal List<Car> RentedCarsPool { get; }

        public Customer() { }

        public Customer(string id, string name, string phoneNumber, string address, string
email)
        {
            Id = id;
            Name = name;

```

```

        PhoneNumber = phoneNumber;
        Address = address;
        Email = email;
        RentedCarsPool = new List<Car>();
    }

    public List<Car> RentedCarsCopy
    {
        get
        {
            var result = new List<Car>();
            result.AddRange(RentedCarsPool);
            return result;
        }
    }

    internal void RentCar(Car car)
    {
        if (car == null) throw new ArgumentNullException("car");
        if (RentedCarsPool.Contains(car)) throw new InvalidOperationException($"Already
rented car with ID: {car.Id}!");
        if (RentedCarsPool.Count >= CarRentalMaxLimit) throw new
InvalidOperationException(
            $"Customer ({Id}) is not allowed to rent more than {CarRentalMaxLimit}
car(s)!");

        car.Rent();
        RentedCarsPool.Add(car);
    }

    internal void ReturnCar(Car car)
    {
        if (RentedCarsPool.Contains(car)) RentedCarsPool.Remove(car);
        car.Return();
    }

    public string Serialize(DataType dataType) => dataType switch
    {
        DataType.CSV => $"{{Id}},{{Name}},{{PhoneNumber}},{{Address}},{{Email}}",
        DataType.JSON => JsonConvert.SerializeObject(this),
        _ => throw new ArgumentException($"Unknown data type {dataType}."),
    };

```

```

    public static Customer Deserialize(string data, DataType dataType) => dataType
switch
{
    DataType.CSV => CreateFromCsv(data),
    DataType.JSON => JsonConvert.DeserializeObject<Customer>(data),
    _ => throw new ArgumentException($"Unknown data type {dataType}."),
};

private static Customer CreateFromCsv(string csv)
{
    string[] values = csv.Split(',');
    if (values.Length != 5)
    {
        throw new ArgumentException($"Invalid CSV data: {csv}");
    }
    return new Customer(values[0], values[1], values[2], values[3], values[4]);
}

public Customer DeserializeHandler(string data, DataType dataType)
{
    return Deserialize(data, dataType);
}

public void RentedCarsPoolNew(List<Car> rentedCars)
{
    RentedCarsPool.Clear();
    RentedCarsPoolExtend(rentedCars);
}

public void RentedCarsPoolExtend(List<Car> rentedCars)
{
    if (rentedCars == null) throw new ArgumentNullException("rentedCars");

    RentedCarsPool.AddRange(rentedCars);
}
}
}

```



Here's the Python version of the above code:

python

Copy code

```

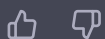
from __future__ import annotations from typing import List, Type import json
from sf.systems.rentals.cars.ISerializedEntity import ISerializedEntity,

```

```

DataType from sf.systems.rentals.cars.Car import Car class
Customer(ISerializedEntity[Customer]): CAR_RENTAL_MAX_LIMIT = 1 def
__init__(self, id: str, name: str, phone_number: str, address: str, email:
str): self._id = id self._name = name self._phone_number = phone_number
self._address = address self._email = email self._rented_cars_pool = []
@property def id(self) -> str: return self._id @property def name(self) ->
str: return self._name @property def phone_number(self) -> str: return
self._phone_number @property def address(self) -> str: return self._address
@property def email(self) -> str: return self._email @property def
rented_cars_copy(self) -> List[Car]: return self._rented_cars_pool.copy() def
rent_car(self, car: Car) -> None: if car is None: raise ValueError("car
cannot be None") if car in self._rented_cars_pool: raise ValueError(f"Already
rented car with ID: {car.id}!") if len(self._rented_cars_pool) >=
Customer.CAR_RENTAL_MAX_LIMIT: raise ValueError(f"Customer ({self._id}) is
not allowed to rent more than {Customer.CAR_RENTAL_MAX_LIMIT} car(s)!")
car.rent() self._rented_cars_pool.append(car) def return_car(self, car: Car)
-> None: if car in self._rented_cars_pool: self._rented_cars_pool.remove(car)
car.return_car() def serialize(self, data_type: DataType) -> str: if
data_type == DataType.CSV: return f"{self._id},{self._name},
{self._phone_number},{self._address},{self._email}" elif data_type ==
DataType.JSON: return json.dumps(self.__dict__) else: raise
ValueError(f"Unknown data type {data_type}.") @staticmethod def
deserialize(data: str, data_type: DataType) -> Customer: if data_type ==
DataType.CSV: return Customer.create_from_csv(data) elif data_type ==
DataType.JSON: return Customer(**json.loads(data)) else: raise
ValueError(f"Unknown data type {data_type}.") def deserialize_handler(self,
data: str, data_type: DataType) -> Customer: return
Customer.deserialize(data, data_type) @staticmethod def create_from_csv(csv:
str) -> Customer: values = csv.split(",") if len(values) != 5: raise
ValueError(f"Invalid CSV data: {csv}") return Customer(values[0], values[1],
values[2], values[3], values[4]) def rented_cars_pool_new(self, rented_cars:
List[Car]) -> None: self._rented_cars_pool.clear()
self._rented_cars_pool_extend(rented_cars) def rented_cars_pool_extend(self,
rented_cars: List[Car]) -> None: if rented_cars is None: raise
ValueError("rented_cars cannot be None")

```



convert this code: using Newtonsoft.Json;
using System;

```

namespace sf.systems.rentals.cars
{
    public partial class Transaction : ISerializedExtendedEntity<Transaction>,
    ISerializeOwner
    {
        #region data

```

```
public string Id { get; set; }
public Customer Customer { get; set; }
public Car Car { get; set; }
public DateTime RentalDate { get; set; }
public DateTime ReturnDate { get; set; }
public double TotalPrice { get; set; }
public DateTime ClosedDate { get; private set; }
public bool IsClosed { get; private set; }
public IEntitiesList Owner { get; private set; }
#endregion
```

```
public Transaction() { Id = "?"; }
```

```
public Transaction(string id, Customer customer, Car car, DateTime rentalDate,
DateTime returnDate, DateTime closedDate, bool isClosed)
```

```
{
    Id = id;
    Car = car;
    Customer = customer;
    RentalDate = rentalDate;
    ReturnDate = returnDate;
    ClosedDate = closedDate;
    IsClosed = isClosed;
    TotalPrice = CalculateTotalPrice();
}
```

```
public void CloseTransaction(IEntitiesList entitiesList)
```

```
{
    // validate entities source
    if (entitiesList == null) throw new ArgumentNullException("entitiesList");

    // close transaction
    Customer.ReturnCar(Car);
    IsClosed = true;
    ClosedDate = DateTime.Now;
    entitiesList.ReturnCar(Car);
    entitiesList.ArchiveTransaction(this);
}
```

```
public double CalculateTotalPrice()
```

```
{
    TimeSpan rentalPeriod = ReturnDate - RentalDate;
    var totalDays = (int)rentalPeriod.TotalDays;
```

```

        if (totalDays < 1) totalDays = 1;
        return totalDays * Car.DailyPrice;
    }

    public string Serialize(DataType dataType) => dataType switch
    {
        DataType.CSV =>
            $"{Id},{Customer.Id},{Car.Id}," +
            $"{RentalDate.ToShortDateString()}" +
            $"{ReturnDate.ToShortDateString()}" +
            $"{ClosedDate.ToShortDateString()}" +
            $"{IsClosed}",

        DataType.JSON => JsonConvert.SerializeObject(this),
        _ => throw new ArgumentException($"Unknown data type {dataType}."),
    };

    public Transaction DeserializeHandler(string data, DataType dataType,
        IEntitiesList entitiesList)
    {
        return Deserialize(data, dataType, entitiesList);
    }

    public Transaction DeserializeHandler(string data, DataType dataType)
    {
        return Deserialize(data, dataType, Owner);
    }

    public void AssignOwner(IEntitiesList entitiesList)
    {
        Owner = entitiesList;
    }
}

```



Here's the Python version of the above code:

python

Copy code

```

from __future__ import annotations from typing import Type import json from
datetime import datetime, timedelta from
sf.systems.rentals.cars.ISerializedEntity import ISerializedEntity, DataType

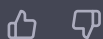
```



```

from sf.systems.rentals.cars.ISerializedExtendedEntity import
ISerializedExtendedEntity from sf.systems.rentals.cars.ISerializeOwner import
ISerializeOwner from sf.systems.rentals.cars.Customer import Customer from
sf.systems.rentals.cars.Car import Car from
sf.systems.rentals.cars.IEntitiesList import IEntitiesList class
Transaction(ISerializedExtendedEntity[Transaction], ISerializeOwner): def
__init__(self, id: str, customer: Customer, car: Car, rental_date: datetime,
return_date: datetime, closed_date: datetime, is_closed: bool): self._id = id
self._customer = customer self._car = car self._rental_date = rental_date
self._return_date = return_date self._closed_date = closed_date
self._is_closed = is_closed self._owner = None self._total_price =
self._calculate_total_price() @property def id(self) -> str: return self._id
@property def customer(self) -> Customer: return self._customer @property def
car(self) -> Car: return self._car @property def rental_date(self) ->
datetime: return self._rental_date @property def return_date(self) ->
datetime: return self._return_date @property def closed_date(self) ->
datetime: return self._closed_date @property def is_closed(self) -> bool:
return self._is_closed @property def total_price(self) -> float: return
self._total_price @property def owner(self) -> IEntitiesList: return
self._owner def close_transaction(self, entities_list: IEntitiesList) ->
None: # validate entities source if entities_list is None: raise
ValueError("entitiesList cannot be None") # close transaction
self._customer.return_car(self._car) self._is_closed = True self._closed_date
= datetime.now() entities_list.return_car(self._car)
entities_list.archive_transaction(self) def _calculate_total_price(self) ->
float: rental_period = self._return_date - self._rental_date total_days =
int(rental_period.total_seconds() / timedelta(days=1).total_seconds()) if
total_days < 1: total_days = 1 return total_days * self._car.daily_price def
serialize(self, data_type: DataType) -> str: if data_type == DataType.CSV:
return f"{self._id},{self._customer.id},{self._car.id},
{self._rental_date.strftime('%m/%d/%Y')},
{self._return_date.strftime('%m/%d/%Y')},
{self._closed_date.strftime('%m/%d/%Y')},{self._is_closed}" elif data_type ==
DataType.JSON: return json.dumps(self.__dict__) else: raise
ValueError(f"Unknown data type {data_type}.") @staticmethod def
deserialize(data: str, data_type: DataType) -> Transaction: if data_type ==
DataType.CSV: return Transaction.create_from_csv(data) elif data_type ==
DataType.JSON: return Transaction(**json.loads(data))

```



convert this code: using Newtonsoft.Json;
using System;

namespace sf.systems.rentals.cars

{

public partial class Transaction : ISerializedExtendedEntity<Transaction>,

ISerializeOwner

```
{
    public Transaction DeserializeHandler(string data, DataType dataType,
        IEntitiesList entitiesList)
    {
        return Deserialize(data, dataType, entitiesList);
    }

    public Transaction DeserializeHandler(string data, DataType dataType)
    {
        return Deserialize(data, dataType, Owner);
    }

    public void AssignOwner(IEntitiesList entitiesList)
    {
        Owner = entitiesList;
    }
}
```

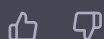


Here's the Python version of the above code:

python

Copy code

```
from __future__ import annotations from typing import Type import json from
sf.systems.rentals.cars.ISerializedEntity import ISerializedEntity, DataType
from sf.systems.rentals.cars.ISerializedExtendedEntity import
ISerializedExtendedEntity from sf.systems.rentals.cars.ISerializeOwner import
ISerializeOwner from sf.systems.rentals.cars.Customer import Customer from
sf.systems.rentals.cars.Car import Car from
sf.systems.rentals.cars.IEntitiesList import IEntitiesList class
Transaction(ISerializedExtendedEntity[Transaction], ISerializeOwner): # ...
def deserialize_handler(self, data: str, data_type: DataType, entities_list:
IEntitiesList) -> Transaction: return Transaction.deserialize(data,
data_type, entities_list) def deserialize_handler(self, data: str, data_type:
DataType) -> Transaction: return Transaction.deserialize(data, data_type,
self._owner) def assign_owner(self, entities_list: IEntitiesList) -> None:
self._owner = entities_list
```



convert this code of Transaction-class (static methods only):

```

public class Transaction
{
    public static Transaction OpenTransaction(IEntitiesList entitiesList,
        string customerId, string carId, DateTime rentalDate, DateTime returnDate)
    {
        // validate entities source
        if (entitiesList == null) throw new ArgumentNullException("entitiesList");

        // validation dates
        if (rentalDate.Date < DateTime.Now.Date) throw new
ArgumentOutOfRangeException("Rental date cannot be in the past!");
        if (rentalDate.Date > returnDate.Date) throw new
ArgumentOutOfRangeException("Rental date cannot be bigger than return date!");

        // get customer
        Customer customer = entitiesList.LookupCustomer(customerId);
        // validate customer
        if (customer == null) throw new ArgumentNullException($"Customer with ID:
{customerId} has not been found!");

        // get car
        Car car = entitiesList.LookupCar(carId);
        // validate car
        if (car == null) throw new ArgumentNullException($"Car with ID: {customerId} has
not been found!");

        // generate Tx-ID
        var txID = $"{Guid.NewGuid()}";

        // create rental transaction
        var newTransaction = new Transaction(txID, customer, car, rentalDate, returnDate,
default, false);
        newTransaction.Customer.RentCar(car);
        entitiesList.RentCar(car);
        entitiesList.NewTransaction(newTransaction);
        return newTransaction;
    }

    public static Transaction Deserialize(string data, DataType dataType,
        IEntitiesList entitiesList) => dataType switch
    {
        DataType.CSV => CreateFromCsv(data, entitiesList),
        DataType.JSON => JsonConvert.DeserializeObject<Transaction>(data),
    }
}

```

```

        _ => throw new ArgumentException($"Unknown data type {dataType}.");
    };

    private static Transaction CreateFromCsv(string csv,
        IEntitiesList entitiesList)
    {
        string[] values = csv.Split(',');
        if (values.Length < 7)
        {
            throw new ArgumentException($"Invalid CSV data: {csv}");
        }
        string id = values[0];

        Customer customer;
        Car car;

        string customerId = values[1];
        string carId = values[2];

        if (entitiesList == null)
        {
            customer = defaultCustomer(customerId);
            car = defaultCar(carId);
        }
        else
        {
            customer = entitiesList.LookupCustomer(customerId);
            if (customer == null)
            {
                customer = defaultCustomer(customerId);
            }
            else
            {
                customer.RentedCarsPoolNew(entitiesList.GetRentedCars(customer));
            }

            car = entitiesList.LookupCar(carId);
            if (car == null) car = defaultCar(carId);
        }

        DateTime rentalDate = DateTime.Parse(values[3]);
        DateTime returnDate = DateTime.Parse(values[4]);
        DateTime closedDate = DateTime.Parse(values[5]);
        bool isClosed = bool.Parse(values[6]);

        return new Transaction(id, customer, car, rentalDate, returnDate, closedDate,
            isClosed);
    }

```

```

    }

    private static Car defaultCar(string carId)
    {
        return new Car(carId, "", "", 0, 0.0, false);
    }

    private static Customer defaultCustomer(string customerId)
    {
        return new Customer(customerId, "", "", "", "");
    }
}

```



python

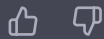
Copy code

```

from datetime import datetime import uuid class Transaction: def
__init__(self, id, customer, car, rental_date, return_date, closed_date,
is_closed): self.Id = id self.Customer = customer self.Car = car
self.RentalDate = rental_date self.ReturnDate = return_date self.ClosedDate =
closed_date self.IsClosed = is_closed @staticmethod def
OpenTransaction(entities_list, customer_id, car_id, rental_date,
return_date): # validate entities source if entities_list is None: raise
ValueError("entitiesList cannot be None!") # validation dates if
rental_date.date() < datetime.now().date(): raise ValueError("Rental date
cannot be in the past!") if rental_date.date() > return_date.date(): raise
ValueError("Rental date cannot be bigger than return date!") # get customer
customer = entities_list.LookupCustomer(customer_id) # validate customer if
customer is None: raise ValueError(f"Customer with ID: {customer_id} has not
been found!") # get car car = entities_list.LookupCar(car_id) # validate car
if car is None: raise ValueError(f"Car with ID: {car_id} has not been
found!") # generate Tx-ID txID = str(uuid.uuid4()) # create rental
transaction newTransaction = Transaction(txID, customer, car, rental_date,
return_date, None, False) newTransaction.Customer.RentCar(car)
entities_list.RentCar(car) entities_list.NewTransaction(newTransaction)
return newTransaction @staticmethod def Deserialize(data, data_type,
entities_list=None): if data_type == DataType.CSV: return
Transaction.CreateFromCsv(data, entities_list) elif data_type ==
DataType.JSON: return JsonConvert.DeserializeObject(Transaction, data) else:
raise ValueError(f"Unknown data type {data_type}.") @staticmethod def
CreateFromCsv(csv, entities_list=None): values = csv.split(',') if
len(values) < 7: raise ValueError(f"Invalid CSV data: {csv}") id = values[0]
customer = None car = None customer_id = values[1] car_id = values[2] if
entities_list is None: customer = defaultCustomer(customer_id) car =
defaultCar(car_id) else: customer = entities_list.LookupCustomer(customer_id)
if customer is None: customer = defaultCustomer(customer_id) else:

```

```
customer.RentedCarsPoolNew(entities_list.GetRentedCars(customer)) car =
entities_list.LookupCar(car_id) if car is None: car = defaultCar(car_id)
rental_date = datetime.strptime(values[3], '%m/%d/%Y') return_date =
datetime.strptime(values[4], '%m/%d/%Y') closed_date = None if values[5] !=
"None": closed_date = datetime.strptime(values[5], '%m/%d/%Y') is_closed =
bool(values[6]) return Transaction(id, customer, car, rental_date,
return_date, closed_date, is_closed) def CalculateTotalPrice(self):
rental_period = self.ReturnDate - self.RentalDate total_days =
max(rental_period.days, 1) return total_days * self.Car.DailyPrice def
defaultCar(car_id): return Car(car_id, "", "", 0, 0.0, False
```



convert this Transaction code: private static Car defaultCar(string carId)

```
{
    return new Car(carId, "", "", 0, 0.0, false);
}
```

```
private static Customer defaultCustomer(string customerId)
{
    return new Customer(customerId, "", "", "", "");
}
```



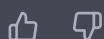
Since these methods are not part of the Transaction class, they do not need to be converted. However, if you want to convert them to Python, here is how they could look:

python

Copy code

```
def default_car(car_id): return Car(car_id, "", "", 0, 0.0, False) def
default_customer(customer_id): return Customer(customer_id, "", "", "", "")
```

Assuming that the **Car** and **Customer** classes have already been defined and imported.



can I define regions in Python code within VSCode (just like for C# in Visual Studio)





No, regions are not supported in Python, and they are specific to Visual Studio and C# syntax. Python uses indentation to define code blocks and doesn't have a similar feature to regions. However, you can use comments to organize your code and add notes to it. For example, you can use section headings with comments like this:

python

Copy code

```
# ===== # Imports # ===== import pandas as pd
import numpy as np # ===== # Data preparation #
===== # Load data data = pd.read_csv('data.csv') # Clean data
data = data.dropna() # ===== # Model training #
===== # Split data into train and test sets train_data,
test_data = train_test_split(data, test_size=0.2) # Train model model =
LinearRegression() model.fit(train_data[['x']], train_data['y']) #
===== # Model evaluation # ===== # Evaluate
model on test data y_pred = model.predict(test_data[['x']]) mse =
mean_squared_error(test_data['y'], y_pred) r2 = r2_score(test_data['y'],
y_pred) print(f"MSE: {mse:.2f}") print(f"R2: {r2:.2f}")
```

This way, you can collapse or expand sections of your code by clicking on the +/- sign next to the comments.



convert this code: using System;
using System.IO;
using System.Security.Cryptography;
using System.Text;

```
namespace sf.systems.rentals.cars
{
    public class SecurityManager
    {
        public const string KeyFile = "key.bin";
        private readonly byte[] key;

        private static byte[] defaultKey()
        {
            // Generate a 32-byte array
            byte[] key = new byte[32];
            using (RandomNumberGenerator rng = RandomNumberGenerator.Create())
            {
```

```

        rng.GetBytes(key);
    }

    return key;
}

public SecurityManager() : this(defaultKey())
{
    string keyPath = KeyFile;
    File.WriteAllBytes(keyPath, this.key);
}

public SecurityManager(byte[] key)
{
    if (key == null || key.Length != 32)
    {
        throw new ArgumentException("Key must be a byte array of length 32");
    }

    this.key = key;
}

public string Encrypt(string data)
{
    byte[] iv = new byte[16];
    byte[] dataBytes = Encoding.UTF8.GetBytes(data);

    using (Aes aes = Aes.Create())
    {
        aes.Key = key;
        aes.IV = iv;

        using (MemoryStream memoryStream = new MemoryStream())
        {
            using (CryptoStream cryptoStream = new CryptoStream(memoryStream,
aes.CreateEncryptor(), CryptoStreamMode.Write))
            {
                cryptoStream.Write(dataBytes, 0, dataBytes.Length);
                cryptoStream.FlushFinalBlock();
            }
            byte[] encryptedData = memoryStream.ToArray();
            return Convert.ToBase64String(encryptedData);
        }
    }
}

```



```

    }
}

public string Decrypt(string data)
{
    byte[] iv = new byte[16];
    byte[] encryptedData = Convert.FromBase64String(data);

    using (Aes aes = Aes.Create())
    {
        aes.Key = key;
        aes.IV = iv;

        using (MemoryStream memoryStream = new MemoryStream())
        {
            using (CryptoStream cryptoStream = new CryptoStream(memoryStream,
aes.CreateDecryptor(), CryptoStreamMode.Write))
            {
                cryptoStream.Write(encryptedData, 0, encryptedData.Length);
                cryptoStream.FlushFinalBlock();
            }
            byte[] decryptedData = memoryStream.ToArray();
            return Encoding.UTF8.GetString(decryptedData);
        }
    }
}

public bool Authorize(User user, string permission)
{
    if (user == null)
    {
        return false;
    }

    switch (user.Role)
    {
        case UserRole.Admin:
            return true;
        case UserRole.PowerUser:
            return permission == "read" || permission == "write";
        case UserRole.User:
            return permission == "read";
        case UserRole.Guest:

```

```

        default:
            return false;
    }
}

public class User
{
    public string Id { get; }
    public string Name { get; }
    public UserRole Role { get; }

    public User(string id, string name, UserRole role)
    {
        Id = id;
        Name = name;
        Role = role;
    }
}

public enum UserRole
{
    Admin,
    PowerUser,
    User,
    Guest
}
}

```



Here is the Python equivalent of the C# code:

python

Copy code

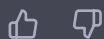
```

import os import base64 import hashlib from Crypto.Cipher import AES from
Crypto.Util.Padding import pad, unpad class SecurityManager: KEY_FILE =
"key.bin" BLOCK_SIZE = 16 KEY_SIZE = 32 def __init__(self, key=None): if key
is None: self.key = self.get_or_create_key() else: self.key = key def
get_or_create_key(self): key_path = self.KEY_FILE if
os.path.exists(key_path): with open(key_path, 'rb') as f: key = f.read()
else: key = self._generate_key() with open(key_path, 'wb') as f: f.write(key)
return key def _generate_key(self): return os.urandom(self.KEY_SIZE) def
encrypt(self, data): iv = os.urandom(self.BLOCK_SIZE) cipher =
AES.new(self.key, AES.MODE_CBC, iv) encrypted_data =
cipher.encrypt(pad(data.encode(), self.BLOCK_SIZE)) return

```

```
base64.b64encode(iv + encrypted_data).decode()
def decrypt(self, data):
    data = base64.b64decode(data)
    iv = data[:self.BLOCK_SIZE]
    encrypted_data = data[self.BLOCK_SIZE:]
    cipher = AES.new(self.key, AES.MODE_CBC, iv)
    decrypted_data = unpad(cipher.decrypt(encrypted_data), self.BLOCK_SIZE)
    return decrypted_data.decode()
def authorize(self, user, permission):
    if user is None:
        return False
    if user.Role == UserRole.Admin:
        return True
    elif user.Role == UserRole.PowerUser:
        return permission == "read" or permission == "write"
    elif user.Role == UserRole.User:
        return permission == "read"
    else:
        return False
class User:
    def __init__(self, id, name, role):
        self.Id = id
        self.Name = name
        self.Role = role
class UserRole:
    Admin = "Admin"
    PowerUser = "PowerUser"
    User = "User"
    Guest = "Guest"
```

Note that the **Crypto** package needs to be installed first using pip.



encrypt method with cryptography package



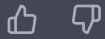
Here's the **Encrypt** method of **SecurityManager** class using the **cryptography** package in Python:

python

Copy code

```
import base64 from cryptography.hazmat.primitives.ciphers import Cipher,
algorithms, modes from cryptography.hazmat.primitives import padding from
cryptography.hazmat.backends import default_backend import os class
SecurityManager:
    def __init__(self, key=None):
        if key is None:
            key = os.urandom(32) # generate random key if not provided
            with open('key.bin', 'wb') as f:
                f.write(key)
            if len(key) != 32:
                raise ValueError("Key must be a byte array of length 32")
            self.key = key
        def encrypt(self, data):
            iv = os.urandom(16)
            data_bytes = data.encode('utf-8')
            padder = padding.PKCS7(algorithms.AES.block_size).padder()
            padded_data = padder.update(data_bytes) + padder.finalize()
            cipher = Cipher(algorithms.AES(self.key), modes.CBC(iv), backend=default_backend())
            encryptor = cipher.encryptor()
            encrypted_data = encryptor.update(padded_data) + encryptor.finalize()
            return base64.b64encode(iv + encrypted_data).decode('utf-8')
```

Note that in this implementation, a random key is generated if not provided in the constructor. Also, the **Authorize** method is not implemented as it is not relevant for encryption.



decrypt method with cryptography package using self.cipher.decryptor()



Sure! Here's how you can implement the **decrypt** method using the **cryptography** package and **decryptor()**:

python

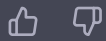
Copy code

```
from cryptography.fernet import Fernet class SecurityManager: def
__init__(self, key=None): if key is None: key = Fernet.generate_key() with
open('key.bin', 'wb') as key_file: key_file.write(key) self.cipher =
Fernet(key) def encrypt(self, data): data_bytes = data.encode()
encrypted_data = self.cipher.encrypt(data_bytes) return
encrypted_data.decode() def decrypt(self, data): encrypted_data =
data.encode() decrypted_data = self.cipher.decryptor().update(encrypted_data)
+ self.cipher.decryptor().finalize() return decrypted_data.decode() def
authorize(self, user, permission): if user is None: return False if user.role
== 'admin': return True if user.role == 'power_user' and permission in
['read', 'write']: return True if user.role == 'user' and permission ==
'read': return True return False class User: def __init__(self, user_id,
name, role): self.id = user_id self.name = name self.role = role class
UserRole: Admin = 'admin' PowerUser = 'power_user' User = 'user' Guest =
'guest'
```

In this implementation, we're using the **Fernet** class from the **cryptography** package to generate a key and create a cipher object. The **encrypt** method simply takes the data as input, converts it to bytes, encrypts it with the cipher, and returns the encrypted data as a string.

The **decrypt** method takes the encrypted data as input, converts it to bytes, and passes it to the **decryptor()** method of the cipher object to get a decryptor object. We then call the **update()** method on the decryptor object to decrypt the data and the **finalize()** method to finalize the decryption and get the decrypted data as bytes. Finally, we decode the decrypted data to get the original plaintext data as a string.

The **authorize** method takes a **User** object and a permission string as input and returns a boolean indicating whether the user is authorized to perform the specified permission. We're using string literals instead of enums in this implementation.



convert this code: using System;
using System.IO;

```
namespace sf.systems.rentals.cars
{
    public class ErrorHandler
    {
        private const string ErrorFilePath = "errors.txt";
        private readonly bool rethrow;

        public ErrorHandler(bool rethrow = false)
        {
            this.rethrow = rethrow;
        }

        public void HandleError(Exception ex)
        {
            try
            {
                using (StreamWriter writer = File.AppendText(ErrorFilePath))
                {
                    writer.WriteLine($"{DateTime.Now.ToString()} - {ex.Message}");
                    writer.WriteLine(ex.StackTrace);
                }
            }
            catch (Exception)
            {
                // If an error occurs while writing to the error file, rethrow it
                throw;
            }

            if (rethrow)
            {
                throw ex;
            }
        }
    }
}
```





Here is the Python equivalent of the given C# code using built-in Python modules:

python

Copy code

```
import os
import traceback
from datetime import datetime

class ErrorHandler:
    ErrorFilePath = "errors.txt"
    def __init__(self, rethrow=False):
        self.rethrow = rethrow
    def handle_error(self, ex):
        try:
            with open(ErrorHandler.ErrorFilePath, "a") as file:
                file.write(f"{datetime.now().strftime('%Y-%m-%d %H:%M:%S')} - {str(ex)}\n")
            file.write(traceback.format_exc())
        except Exception:
            # If an error occurs while writing to the error file, rethrow it
            raise ex if self.rethrow else None
        if self.rethrow:
            raise ex
```

Note: This code writes the error message and stack trace to the specified file in append mode. If the file does not exist, it will be created.



convert this code: using System;
using System.IO;

```
namespace sf.systems.rentals.cars
{
    public class MessageHandler : IMessageHandler
    {
        private const string LogFilePath = "log.txt";
        private readonly ErrorHandler errorHandler;

        public MessageHandler(ErrorHandler errorHandler)
        {
            this.errorHandler = errorHandler;
        }

        public void Log(string message)
        {
            try
            {
                using (StreamWriter writer = File.AppendText(LogFilePath))
                {
                    writer.WriteLine($"{DateTime.Now.ToString()} - {message}");
                }
            }
        }
    }
}
```

```

        catch (Exception ex)
        {
            errorHandler.HandleError(ex);
        }
    }

    public void ShowMessage(string message)
    {
        Console.WriteLine(message);
    }

    public void LogPlusMessage(string message)
    {
        Log(message);
        ShowMessage(message);
    }
}

```



Here is the Python version of the code:

python

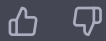
Copy code

```

import os import traceback class MessageHandler: LogFilePath = "log.txt" def
__init__(self, error_handler): self.error_handler = error_handler def
log(self, message): try: with open(MessageHandler.LogFilePath, "a") as file:
file.write(f"{datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')} -
{message}\n") except Exception as ex: self.error_handler.handle_error(ex) def
show_message(self, message): print(message) def log_plus_message(self,
message): self.log(message) self.show_message(message) class ErrorHandler:
ErrorFilePath = "errors.txt" def __init__(self, rethrow=False): self.rethrow
= rethrow def handle_error(self, ex): try: with
open(ErrorHandler.ErrorFilePath, "a") as file: file.write(f"
{datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')} - {str(ex)}\n")
file.write(traceback.format_exc()) except Exception: # If an error occurs
while writing to the error file, rethrow it raise ex if self.rethrow else
None

```

Note that in Python, the file is opened using **with open()** statement which automatically closes the file when the block is exited. Also, the **traceback.format_exc()** method is used to get the stack trace in case of an error while writing to the error file.



convert this code: using System;
using System.IO;

```
namespace sf.systems.rentals.cars
{
    public class MessageHandler : IMessageHandler
    {
        private const string LogFilePath = "log.txt";
        private readonly ErrorHandler errorHandler;

        public MessageHandler(ErrorHandler errorHandler)
        {
            this.errorHandler = errorHandler;
        }

        public void Log(string message)
        {
            try
            {
                using (StreamWriter writer = File.AppendText(LogFilePath))
                {
                    writer.WriteLine($"{DateTime.Now.ToString()} - {message}");
                }
            }
            catch (Exception ex)
            {
                errorHandler.HandleError(ex);
            }
        }

        public void ShowMessage(string message)
        {
            Console.WriteLine(message);
        }

        public void LogPlusMessage(string message)
        {
            Log(message);
            ShowMessage(message);
        }
    }
}
```



```
}  
}
```

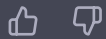


Here's the Python version of the **MessageHandler** class:

python

Copy code

```
import sys import os class MessageHandler(IMessageHandler): LOG_FILE_PATH =  
"log.txt" def __init__(self, error_handler: ErrorHandler): self.error_handler  
= error_handler def log(self, message: str): try: with  
open(self.LOG_FILE_PATH, "a") as f: f.write(f"{datetime.now().strftime('%Y-  
%m-%d %H:%M:%S')} - {message}\n") except Exception as ex:  
self.error_handler.handle_error(ex) def show_message(self, message: str):  
print(message) def log_plus_message(self, message: str): self.log(message)  
self.show_message(message)
```



make a Python representation of the DataManager-class only with typed parameters and returns:

```
using System;  
using System.Collections.Generic;  
using System.IO;
```

```
namespace sf.systems.rentals.cars
```

```
{  
    public class DataManager : ISerializeOwner  
    {  
        private const string DataFolderPath = "data";  
  
        private readonly ErrorHandler errorHandler;  
        private readonly MessageHandler messageHandler;  
  
        private IEntitiesList owner;  
  
        public DataManager(ErrorHandler errorHandler, MessageHandler messageHandler)  
        {  
            this.errorHandler = errorHandler;  
            this.messageHandler = messageHandler;  
        }  
    }  
}
```

```

    public List<T> ReadData<T>(EntityType entityType, DataType dataType, string
fileSuffix)
    where T : ISerializedEntity<T>, new()
    {
        string filePath = GetFilePath(entityType, dataType, fileSuffix);
        List<T> dataList = new List<T>();

        if (File.Exists(filePath))
        try
        {
            using (StreamReader reader = new StreamReader(filePath))
            {
                string line;
                while ((line = reader.ReadLine()) != null)
                {
                    T data = Deserialize<T>(line, dataType);

                    if (data != null)
                    {
                        dataList.Add(data);
                    }
                }
            }
        }
        catch (Exception ex)
        {
            errorHandler.HandleError(ex);
        }

        return dataList;
    }

```

```

    public List<E> ReadDataExtended<E>(EntityType entityType, DataType dataType,
string fileSuffix)
    where E : ISerializedExtendedEntity<E>, new()
    {
        string filePath = GetFilePath(entityType, dataType, fileSuffix);
        List<E> dataList = new List<E>();

        if (File.Exists(filePath))
        try
        {

```

```

        using (StreamReader reader = new StreamReader(filePath))
        {
            string line;
            while ((line = reader.ReadLine()) != null)
            {
                E data;

                if (owner == null)
                    data = Deserialize<E>(line, dataType);
                else
                {
                    data = DeserializeExtended<E>(line, dataType, owner);
                }

                if (data != null)
                {
                    dataList.Add(data);
                }
            }
        }
    }
    catch (Exception ex)
    {
        errorHandler.HandleError(ex);
    }

    return dataList;
}

```

```

public bool WriteData<T>(List<T> dataList, EntityType entityType, DataType
dataType, string fileSuffix)
    where T : ISerializedEntity<T>, new()
{
    string filePath = GetFilePath(entityType, dataType, fileSuffix);
    string directoryPath = Path.GetDirectoryName(filePath);
    if (!Directory.Exists(directoryPath))
    {
        Directory.CreateDirectory(directoryPath);
    }

    try
    {
        using (StreamWriter writer = new StreamWriter(filePath))

```

```

        {
            foreach (T data in dataList)
            {
                string line = Serialize(data, dataType);
                writer.WriteLine(line);
            }
            messageHandler.LogPlusMessage($"Write Data - Rows:{dataList.Count},
Entity:{entityType}, Data:{dataType} ({fileSuffix}).");
        }
        return true;
    }
    catch (Exception ex)
    {
        errorHandler.HandleError(ex);
        return false;
    }
}

```

```

    public void ReadData<T>(List<T> targetList, EntityType entityType, DataType
dataType, string fileSuffix)
        where T : ISerializedEntity<T>, new()
    {
        if (targetList == null) errorHandler.HandleError(new
ArgumentNullException("targetList"));
    }

```

```

        List<T> listItems = ReadData<T>(entityType, dataType, fileSuffix);
        if (listItems != null)
        {
            targetList.Clear();
            targetList.AddRange(listItems);
            messageHandler.LogPlusMessage($"Read Data - Rows:{targetList.Count},
Entity:{entityType}, Data:{dataType} ({fileSuffix}).");
        }
        else
        {
            messageHandler.LogPlusMessage($"No Data - Entity:{entityType}, Data:
{dataType}.");
        }
    }
}

```

```

    public void ReadDataExtended<E>(List<E> targetList, EntityType entityType,
DataType dataType, string fileSuffix, IEntitiesList owner)
        where E : ISerializedExtendedEntity<E>, new()
    {
    }
}

```

```

    {
        if (targetList == null) errorHandler.HandleError(new
ArgumentNullException("targetList"));

        AssignOwner(owner);

        List<E> listItems = ReadDataExtended<E>(entityType, dataType, fileSuffix);

        if (listItems != null)
        {
            targetList.Clear();
            targetList.AddRange(listItems);
            messageHandler.LogPlusMessage($"Read Data - Rows:{targetList.Count},
Entity:{entityType}, Data:{dataType} ({fileSuffix}).");
        }
        else
        {
            messageHandler.LogPlusMessage($"No Data - Entity:{entityType}, Data:
{dataType}.");
        }
    }

    private string GetFilePath(EntityType entityType, DataType dataType, string fileSuffix)
    {
        string suffix = "";
        if (!string.IsNullOrEmpty(fileSuffix)) suffix = $"_{fileSuffix}";

        string fileName = $"{entityType.ToString().ToLower()}{suffix}.
{dataType.ToString().ToLower()}";

        return Path.Combine(DataFolderPath, fileName);
    }

    private string Serialize<T>(T data, DataType dataType) where T :
ISerializedEntity<T>
    {
        if (data == null) throw new ArgumentNullException("data");

        switch (dataType)
        {
            case DataType.CSV:
            case DataType.JSON:
                return data.Serialize(dataType);
        }
    }

```

```

        default:
            throw new ArgumentException($"Invalid data type: {dataType}");
    }
}

```

```

private T Deserialize<T>(string data, DataType dataType)
    where T : ISerializedEntity<T>, new()
{
    if (data == null) throw new ArgumentNullException("data");

    switch (dataType)
    {
        case DataType.CSV:
        case DataType.JSON:
            return new T().DeserializeHandler(data, dataType);
        default:
            throw new ArgumentException($"Invalid data type: {dataType}");
    }
}

```

```

private E DeserializeExtended<E>(string data, DataType dataType, IEntitiesList
entitiesList)
    where E : ISerializedExtendedEntity<E>, new()
{
    if (data == null) throw new ArgumentNullException("data");

    E result;

    switch (dataType)
    {
        case DataType.CSV:
        case DataType.JSON:
            result = new E().DeserializeHandler(data, dataType, entitiesList);
            break;
        default:
            throw new ArgumentException($"Invalid data type: {dataType}");
    }

    return result;
}

```

```

public static List<string> ReadLinesFromFile(string filePath)
{

```

```

List<string> lines = new List<string>();

try
{
    using (StreamReader sr = new StreamReader(filePath))
    {
        string line;
        while ((line = sr.ReadLine()) != null)
        {
            lines.Add(line);
        }
    }
}
catch (Exception ex)
{
    Console.WriteLine("An error occurred while reading the file: " + ex.Message);
}

return lines;
}

public static void WriteLinesToFile(string filePath, List<string> lines)
{
    try
    {
        using (StreamWriter sw = new StreamWriter(filePath))
        {
            foreach (string line in lines)
            {
                sw.WriteLine(line);
            }
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine("An error occurred while writing to the file: " + ex.Message);
    }
}

public void AssignOwner(IEntitiesList entitiesList)
{
    owner = entitiesList;
}

```





python

Copy code

```
}  
  
import os from typing import List, TypeVar T = TypeVar('T',  
bound='ISerializedEntity') E = TypeVar('E',  
bound='ISerializedExtendedEntity') class DataManager(ISerializeOwner):  
DataFolderPath = "data" def __init__(self, errorHandler: ErrorHandler,  
messageHandler: MessageHandler): self.errorHandler = errorHandler  
self.messageHandler = messageHandler self.owner = None def ReadData(self,  
entityType: EntityType, dataType: DataType, fileSuffix: str) -> List[T]:  
filePath = self.GetFilePath(entityType, dataType, fileSuffix) dataList = []  
if os.path.isfile(filePath): try: with open(filePath, 'r') as reader: for  
line in reader.readlines(): data = self.Deserialize[T](line, dataType) if  
data is not None: dataList.append(data) except Exception as ex:  
self.errorHandler.HandleError(ex) return dataList def ReadDataExtended(self,  
entityType: EntityType, dataType: DataType, fileSuffix: str) -> List[E]:  
filePath = self.GetFilePath(entityType, dataType, fileSuffix) dataList = []  
if os.path.isfile(filePath): try: with open(filePath, 'r') as reader: for  
line in reader.readlines(): if self.owner is None: data = self.Deserialize[E]  
(line, dataType) else: data = self.DeserializeExtended[E](line, dataType,  
self.owner) if data is not None: dataList.append(data) except Exception as  
ex: self.errorHandler.HandleError(ex) return dataList def WriteData(self,  
dataList: List[T], entityType: EntityType, dataType: DataType, fileSuffix:  
str) -> bool: filePath = self.GetFilePath(entityType, dataType, fileSuffix)  
directoryPath = os.path.dirname(filePath) if not  
os.path.exists(directoryPath): os.makedirs(directoryPath) try: with  
open(filePath, 'w') as writer: for data in dataList: line =  
self.Serialize(data, dataType) writer.write(line + "\n")  
self.messageHandler.LogPlusMessage(f"Write Data - Rows:{len(dataList)},  
Entity:{entityType}, Data:{dataType} ({fileSuffix}).") return True except  
Exception as ex: self.errorHandler.HandleError(ex) return False def  
ReadData(self, targetList: List[T], entityType: EntityType, dataType:  
DataType, fileSuffix: str): if targetList is None:  
self.errorHandler.HandleError(Exception("targetList cannot be None."))  
listItems = self.ReadData[T](entityType, dataType, fileSuffix) if listItems  
is not None: targetList.clear() targetList.extend(listItems)  
self.messageHandler.LogPlusMessage(f"Read Data - Rows:{len(targetList)},  
Entity:{entityType}, Data:{dataType} ({fileSuffix}).") else:  
self.messageHandler.LogPlusMessage(f"No Data - Entity:{entityType}, Data:  
{dataType}.") def ReadDataExtended(self, targetList: List[E], entityType:  
EntityType, dataType: DataType, fileSuffix: str, owner: IEntitiesList): if  
targetList is None: self.errorHandler.HandleError(Exception("targetList  
cannot be None.")) self.AssignOwner(owner) listItems =  
self.ReadDataExtended[E](entityType, dataType, fileSuffix) if listItems is  
not None: targetList.clear() targetList.extend(listItems)  
self.messageHandler.LogPlusMessage(f"Read Data - Rows:{len(targetList)},  
Entity:{entityType}, Data:{dataType} ({fileSuffix}).") else
```




use Python naming conventions



```
import os

from typing import List, TypeVar, Generic
from .enums import EntityType, DataType
from .interfaces import ISerializedEntity, ISerializedExtendedEntity, IMessageHandler,
IEntitiesList
```

```
T = TypeVar('T', bound=ISerializedEntity)
E = TypeVar('E', bound=ISerializedExtendedEntity)
```

```
class DataManager(Generic[T, E], ISerializeOwner):
    DATA_FOLDER_PATH = "data"
```

python

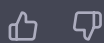
Copy code

```
def __init__(self, error_handler: ErrorHandler, message_handler:
MessageHandler): self.error_handler = error_handler self.message_handler =
message_handler self.owner: IEntitiesList = None def read_data(self,
entity_type: EntityType, data_type: DataType, file_suffix: str) -> List[T]:
file_path = self.get_file_path(entity_type, data_type, file_suffix) data_list
= [] if os.path.exists(file_path): try: with open(file_path, "r") as f: for
line in f: data = self.deserialize(line.strip(), data_type) if data is not
None: data_list.append(data) except Exception as ex:
self.error_handler.handle_error(ex) return data_list def
read_data_extended(self, entity_type: EntityType, data_type: DataType,
file_suffix: str) -> List[E]: file_path = self.get_file_path(entity_type,
data_type, file_suffix) data_list = [] if os.path.exists(file_path): try:
with open(file_path, "r") as f: for line in f: data = None if self.owner is
None: data = self.deserialize(line.strip(), data_type) else: data =
self.deserialize_extended(line.strip(), data_type, self.owner) if data is not
None: data_list.append(data) except Exception as ex:
self.error_handler.handle_error(ex) return data_list def write_data(self,
data_list: List[T], entity_type: EntityType, data_type: DataType,
file_suffix: str) -> bool: file_path = self.get_file_path(entity_type,
data_type, file_suffix) directory_path = os.path.dirname(file_path) if not
os.path.exists(directory_path): os.makedirs(directory_path) try: with
open(file_path, "w") as f: for data in data_list: line = self.serialize(data,
data_type) f.write(f"{line}\n") self.message_handler.log_plus_message(f"Write
Data - Rows:{len(data_list)}, Entity:{entity_type}, Data:{data_type}
({file_suffix}).") return True except Exception as ex:
```

```

self.error_handler.handle_error(ex) return False
def read_data_into_list(self, target_list: List[T], entity_type: EntityType,
data_type: DataType, file_suffix: str):
    if target_list is None:
        self.error_handler.handle_error(ValueError("target_list"))
    list_items = self.read_data(entity_type, data_type, file_suffix)
    if list_items:
        target_list.clear()
        target_list.extend(list_items)
    self.message_handler.log_plus_message(f"Read Data - Rows:{len(target_list)},
Entity:{entity_type}, Data:{data_type} ({file_suffix}).")
    else:
        self.message_handler.log_plus_message(f"No Data - Entity:{entity_type}, Data:
{data_type}.")
def read_data_extended_into_list(self, target_list: List[E],
entity_type: EntityType, data_type: DataType, file_suffix: str, owner:
IEntitiesList):
    if target_list is None:
        self.error_handler.handle_error(ValueError("target_list"))
    self.assign_owner(owner)

```



make ReadData- and ReadDataExtended-Methods only (with targetList parameter)



Here's the modified code for the ReadData and ReadDataExtended methods with the targetList parameter:

python

Copy code

```

from typing import List, TypeVar, Generic
import os
T = TypeVar('T', bound='ISerializedEntity')
E = TypeVar('E', bound='ISerializedExtendedEntity')

class DataManager(ISerializeOwner):
    DataFolderPath = 'data'

    def __init__(self, error_handler: ErrorHandler, message_handler: MessageHandler) -> None:
        self._error_handler = error_handler
        self._message_handler = message_handler
        self._owner: Optional[IEntitiesList] = None

    def read_data(self, target_list: List[T], entity_type: EntityType, data_type: DataType,
file_suffix: str = '') -> None:
        if not target_list:
            self._error_handler.handle_error(ValueError('target_list parameter should not be empty.'))
        data_list = self.read_data_generic(entity_type, data_type, file_suffix, T)
        if data_list is not None:
            target_list.clear()
            target_list.extend(data_list)
        self._message_handler.log_plus_message(f"Read Data - Rows:{len(target_list)},
Entity:{entity_type}, Data:{data_type} ({file_suffix}).")
        else:
            self._message_handler.log_plus_message(f"No Data - Entity:{entity_type}, Data:{data_type}.")

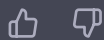
    def read_data_extended(self, target_list: List[E], entity_type: EntityType, data_type: DataType,
file_suffix: str = '') -> None:
        if not target_list:
            self._error_handler.handle_error(ValueError('target_list parameter should not be empty.'))
        self.assign_owner(self._owner)
        data_list = self.read_data_generic(entity_type, data_type, file_suffix, E)
        if data_list

```

```

is not None: target_list.clear() target_list.extend(data_list)
self._message_handler.log_plus_message(f"Read Data - Rows:{len(target_list)},
Entity:{entity_type}, Data:{data_type} ({file_suffix}).") else:
self._message_handler.log_plus_message(f"No Data - Entity:{entity_type},
Data:{data_type}.") def read_data_generic(self, entity_type: EntityType,
data_type: DataType, file_suffix: str, T_type: Type[T]) -> Optional[List[T]]:
file_path = self.get_file_path(entity_type, data_type, file_suffix)
data_list: List[T] = [] if os.path.exists(file_path): try: with
open(file_path, 'r') as file: for line in file: data =
self.deserialize(line.strip(), data_type, T_type) if data is not None:
data_list.append(data) except Exception as ex:
self._error_handler.handle_error(ex) return data_list if data_list else None

```



make GetFilePath, ReadLinesFromFile, WriteLinesToFile and Assign Owner



Here's the Python conversion of the remaining methods in the **DataManager** class:

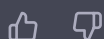
python

Copy code

```

from typing import List class DataManager: DataFolderPath = "data" def
__init__(self, error_handler, message_handler): self.error_handler =
error_handler self.message_handler = message_handler self.owner = None def
GetFilePath(self, entity_type, data_type, file_suffix): suffix =
f"_{file_suffix}" if file_suffix else "" file_name = f"
{entity_type.name.lower()}_{suffix}_{data_type.name.lower()}" return
os.path.join(self.DataFolderPath, file_name) def ReadLinesFromFile(self,
file_path): lines = [] try: with open(file_path, "r") as f: lines =
f.readlines() except Exception as ex: self.error_handler.HandleError(ex)
return lines def WriteLinesToFile(self, file_path, lines): try: with
open(file_path, "w") as f: f.writelines(lines) except Exception as ex:
self.error_handler.HandleError(ex) def AssignOwner(self, entities_list):
self.owner = entities_list

```




make AssignOwner



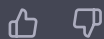
In Python, the **AssignOwner** method of the **DataManager** class would look like this:

python

 Copy code

```
from typing import List class DataManager: def __init__(self, errorHandler:
ErrorHandler, messageHandler: MessageHandler): self.owner = None
self.errorHandler = errorHandler self.messageHandler = messageHandler def
assign_owner(self, entitiesList: IEntitiesList) -> None: self.owner =
entitiesList
```

Note that the method name has been changed to **assign_owner** to follow Python naming conventions, and the parameter names have been changed to use snake_case.



make Serialize, Deserialize and DeserializeExtended



In Python, we can define the **Serialize**, **Deserialize** and **DeserializeExtended** methods as follows:

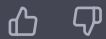
python

 Copy code

```
from typing import List, TypeVar import os import json T = TypeVar('T',
bound='ISerializedEntity') E = TypeVar('E',
bound='ISerializedExtendedEntity') class DataManager: # previous code def
Serialize(self, data: T, dataType: DataType) -> str: if data is None: raise
ValueError("data cannot be None.") if dataType == DataType.CSV or dataType ==
DataType.JSON: return data.Serialize(dataType) else: raise
ValueError(f"Invalid data type: {dataType}") def Deserialize(self, data: str,
dataType: DataType) -> T: if data is None: raise ValueError("data cannot be
None.") if dataType == DataType.CSV or dataType == DataType.JSON: return
T().DeserializeHandler(data, dataType) else: raise ValueError(f"Invalid data
type: {dataType}") def DeserializeExtended(self, data: str, dataType:
DataType, entitiesList: IEntitiesList) -> E: if data is None: raise
ValueError("data cannot be None.") if dataType == DataType.CSV or dataType ==
DataType.JSON: return E().DeserializeHandler(data, dataType, entitiesList)
else: raise ValueError(f"Invalid data type: {dataType}")
```

Note that we import **List** and **TypeVar** from the **typing** module to specify the type of the **data** parameter in **Serialize**, **Deserialize**, and **DeserializeExtended** methods. We also raise **ValueError** instead of **ArgumentException** to follow the Python naming convention for exceptions.

Additionally, we keep the class name as **DataManager** and the **DataType** and **IEntitiesList** enums with the original names because they don't violate any Python naming convention.



convert this code: using System;
using System.IO;
using System.Collections.Generic;

```
namespace sf.systems.rentals.cars
{
    public class AuthenticationManager
    {
        private readonly string credentialsFilePath = "data/credentials.txt";
        private readonly Dictionary<string, string> credentials;
        private readonly Dictionary<string, UserRole> userRoles;
        private readonly SecurityManager securityManager;

        public AuthenticationManager(SecurityManager securityManager)
        {
            this.securityManager = securityManager;
            credentials = new Dictionary<string, string>();
            userRoles = new Dictionary<string, UserRole>();
        }

        public bool Login(string username, string password)
        {
            if (credentials.ContainsKey(username) && credentials[username] ==
                HashPassword(password))
            {
                Console.WriteLine("Login successful.");
                return true;
            }
            else
            {
                Console.WriteLine("Login failed.");
                return false;
            }
        }

        public void Logout()
```

```

    {
        Console.WriteLine("Logout successful.");
    }

    public void LoadCredentials()
    {
        if (!File.Exists(credentialsFilePath)) throw new
FileNotFoundException("credentialsFilePath");

        List<string> lines = DataManager.ReadLinesFromFile(credentialsFilePath);
        foreach (string line in lines)
        {
            string[] parts = line.Split(',');
            if (parts.Length == 2)
            {
                string username = parts[0].Trim();
                string passwordHash = parts[1].Trim();
                credentials[username] = passwordHash;
            }
        }
    }

    public void LoadRoles()
    {
        if (!File.Exists(credentialsFilePath)) throw new
FileNotFoundException("credentialsFilePath");

        List<string> lines = DataManager.ReadLinesFromFile(credentialsFilePath);
        foreach (string line in lines)
        {
            string[] parts = line.Split(',');
            if (parts.Length == 2)
            {
                string username = parts[0].Trim();
                UserRole role = (UserRole)Convert.ToInt32(parts[1].Trim());
                userRoles[username] = role;
            }
        }
    }

    public void SaveCredentials()
    {
        List<string> lines = new List<string>();
        foreach (KeyValuePair<string, string> credential in credentials)

```

```

        {
            string line = credential.Key + "," + credential.Value;
            lines.Add(line);
        }
        DataManager.WriteLineToFile(credentialsFilePath, lines);
    }

    public void SaveRoles()
    {
        List<string> lines = new List<string>();
        foreach (KeyValuePair<string, UserRole> role in userRoles)
        {
            string line = role.Key + "," + Convert.ToInt32(role.Value);
            lines.Add(line);
        }
        DataManager.WriteLineToFile(credentialsFilePath, lines);
    }

    public void RegisterUser(string username, string password, UserRole role)
    {
        credentials[username] = HashPassword(password);
        SaveCredentials();

        userRoles[username] = role;
        SaveRoles();

        Console.WriteLine("User registered.");
    }

    public void UnregisterUser(string username)
    {
        if (credentials.ContainsKey(username))
        {
            credentials.Remove(username);
            SaveCredentials();
            Console.WriteLine("User unregistered.");
        }
        else if (userRoles.ContainsKey(username))
        {
            userRoles.Remove(username);
            SaveRoles();
        }
        else
    }

```

```

    {
        Console.WriteLine("User not found.");
    }
}

private string HashPassword(string password)
{
    return securityManager.Encrypt(password);
}
}
}

```



python

Copy code

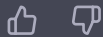
```

import os from enum import Enum from typing import Dict, List from
.security_manager import SecurityManager from .data_manager import
DataManager class UserRole(Enum): CUSTOMER = 1 AGENT = 2 ADMIN = 3 class
AuthenticationManager: CREDENTIALS_FILE_PATH = "data/credentials.txt" def
__init__(self, security_manager: SecurityManager): self.credentials = {}
self.user_roles = {} self.security_manager = security_manager def login(self,
username: str, password: str) -> bool: if username in self.credentials and
self.credentials[username] == self.__hash_password(password): print("Login
successful.") return True else: print("Login failed.") return False def
logout(self) -> None: print("Logout successful.") def load_credentials(self)
-> None: if not os.path.exists(self.CREDENTIALS_FILE_PATH): raise
FileNotFoundError("credentials file not found.") lines =
DataManager.read_lines_from_file(self.CREDENTIALS_FILE_PATH) for line in
lines: parts = line.split(",") if len(parts) == 2: username =
parts[0].strip() password_hash = parts[1].strip() self.credentials[username]
= password_hash def load_roles(self) -> None: if not
os.path.exists(self.CREDENTIALS_FILE_PATH): raise
FileNotFoundError("credentials file not found.") lines =
DataManager.read_lines_from_file(self.CREDENTIALS_FILE_PATH) for line in
lines: parts = line.split(",") if len(parts) == 2: username =
parts[0].strip() role = UserRole(int(parts[1].strip()))
self.user_roles[username] = role def save_credentials(self) -> None: lines =
[f"{username},{password_hash}" for username, password_hash in
self.credentials.items()]
DataManager.write_lines_to_file(self.CREDENTIALS_FILE_PATH, lines) def
save_roles(self) -> None: lines = [f"{username},{role.value}" for username,
role in self.user_roles.items()]
DataManager.write_lines_to_file(self.CREDENTIALS_FILE_PATH, lines) def
register_user(self, username: str, password: str, role: UserRole) -> None:
self.credentials[username] = self.__hash_password(password)
self.save_credentials() self.user_roles[username] = role self.save_roles()
print("User registered.") def unregister_user(self, username: str) -> None:
if username in self.credentials: self.credentials.pop(username)
self.save_credentials() print("User unregistered.") elif username in

```



```
self.user_roles: self.user_roles.pop(username) self.save_roles() else:  
print("User not found.") def __hash_password(self, password: str) -> str:  
return self.security_manager.encrypt(password)
```



make enum for UserRole



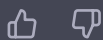
Sure, here's an example of how you can define an **enum** for **UserRole** in Python:

python

Copy code

```
from enum import Enum class UserRole(Enum): ADMIN = 1 MANAGER = 2 USER = 3
```

In this example, **UserRole** is an **enum** class that defines three possible roles: **ADMIN**, **MANAGER**, and **USER**. Each role is assigned an integer value, starting from 1. You can use this **enum** to represent the roles of users in your authentication system, just like in the C# code you provided.



convert this code: using System;
using System.Collections.Generic;
using System.Text;

```
namespace sf.systems.rentals.cars  
{  
    public delegate CarRentalContext CarRentalContextAction(CarRentalContext  
rentalContext);  
    public class CarRentalContext  
    {  
        public string Command { get; set; }  
        public string CustomerID { get; set; }  
        public string CarID { get; set; }  
        public CarRentalSystem CarRentalSystem { get; set; }  
        public Transaction RentalTransaction { get; set; }  
        public bool ActionCompleted { get; set; }  
    }  
}
```



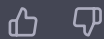
In Python, the equivalent code for the **CarRentalContext** class would be:

python

Copy code

```
from __future__ import annotations from typing import Optional from enum
import Enum class UserRole(Enum): CUSTOMER = 0 ADMIN = 1 class
CarRentalContext: def __init__(self) -> None: self.Command: Optional[str] =
None self.CustomerID: Optional[str] = None self.CarID: Optional[str] = None
self.CarRentalSystem: Optional[CarRentalSystem] = None
self.RentalTransaction: Optional[Transaction] = None self.ActionCompleted:
bool = False class CarRentalSystem: pass class Transaction: pass
```

Note that Python does not have delegates, so the **CarRentalContextAction** delegate is not necessary. Instead, you could define a function that takes a **CarRentalContext** object as an argument and returns a modified version of the same object.



make Python representation:

```
using System;
using System.IO;
using System.Linq;
using System.Collections.Generic;

namespace sf.systems.rentals.cars
{
    public partial class CarRentalSystem : IEntitiesList
    {
        private readonly List<Customer> customers;
        private readonly List<Car> availableCars;
        private readonly List<Car> rentedCars;
        private readonly List<Transaction> currentTransactions;
        private readonly List<Transaction> archiveTransactions;

        private readonly ErrorHandler errorHandler;
        private readonly MessageHandler messageHandler;

        public CarRentalSystem()
        {
```

```

        customers = new List<Customer>();
        availableCars = new List<Car>();
        rentedCars = new List<Car>();
        currentTransactions = new List<Transaction>();
        archiveTransactions = new List<Transaction>();

        errorHandler = new ErrorHandler(true);
        dataManager = new DataManager(errorHandler, new
MessageHandler(errorHandler));
        messageHandler = new MessageHandler(errorHandler);

        if (File.Exists(SecurityManager.KeyFile))
            securityManager = new
SecurityManager(File.ReadAllBytes(SecurityManager.KeyFile));
        else
            securityManager = new SecurityManager();

        authenticationManager = new AuthenticationManager(securityManager);
    }

    public CarRentalSystem(DataManager dataManager, SecurityManager
securityManager) : this()
    {
        this.dataManager = dataManager;
        this.securityManager = securityManager;
        this.authenticationManager = new AuthenticationManager(securityManager);
    }

    public Customer RegisterCustomer(string id, string name, string phoneNumber,
string address, string email)
    {
        var seekCustomer = LookupCustomer(id);

        if (seekCustomer == null)
        {
            Customer customer = new Customer(id, name, phoneNumber, address, email);
            customers.Add(customer);
            return customer;
        }
        else
        {
            return seekCustomer;
        }
    }

```

```

    }

    public bool DeleteCar(string id)
    {
        var seekCar = LookupAvaliableCar(id);
        var seekCarTransactions = LookupCarTransactions(id);

        if (seekCar == null)
        {
            seekCar = LookupRentedCar(id);

            if (seekCar == null)
                errorHandler.HandleError(new InvalidOperationException($"The specified car (ID={id}) does not exist in the system!"));
            else
                errorHandler.HandleError(new InvalidOperationException($"The specified car (ID={id}) is rented and can't be deleted!"));

            return false;
        }
        else if (seekCarTransactions.Count > 0)
        {
            errorHandler.HandleError(new InvalidOperationException($"The specified car (ID={id}) was rented and can't be deleted!"));
            return false;
        }
        else
        {
            this.availableCars.Remove(seekCar);
            return true;
        }
    }

    public bool DeleteCustomer(string id)
    {
        var seekCustomerTransactions = LookupCustomerTransactions(id);

        if (seekCustomerTransactions.Count > 0)
        {
            errorHandler.HandleError(new InvalidOperationException($"The specified customer (ID={id}) has rented a car and can't be deleted!"));
            return false;
        }
    }

```

```

        else
        {
            var seekCustomer = LookupCustomer(id);
            customers.Remove(seekCustomer);
            return true;
        }
    }

    public Car AddCar(string idCar, string make, string model, int year, double
dailyPrice)
    {
        var seekCar = LookupCar(idCar);

        if (seekCar == null)
        {
            Car newCar = new Car(idCar, make, model, year, dailyPrice, false);
            availableCars.Add(newCar);
            return newCar;
        }
        else
            return seekCar;
    }

    public Transaction RentCar(string customerId, string carId, DateTime rentalDate,
DateTime returnDate)
    {
        var car = LookupCar(carId);
        if (!availableCars.Contains(car))
        {
            errorHandler.HandleError(new InvalidOperationException($"The specified car
(ID={carId}) is not available for rental."));
        }

        var customer = LookupCustomer(customerId);

        if (customer != null)
        {
            return Transaction.OpenTransaction(this, customer.Id, car.Id, rentalDate,
returnDate);
        }
        else
        {
            messageHandler.LogPlusMessage($"The specified customer with ID:

```

```

        {customerId} has not been found!");
        return new Transaction();
    }
}

public Transaction ReturnCar(Customer customer, Car car)
{
    if (!rentedCars.Contains(car))
    {
        errorHandler.HandleError(new InvalidOperationException("The specified car
has not been rented by the specified customer."));
    }

    Transaction transaction = currentTransactions.Find(t => !t.IsClosed &&
t.Customer.Id == customer.Id && t.Car.Id == car.Id);
    if (transaction == null)
    {
        errorHandler.HandleError(new InvalidOperationException("The specified
transaction could not be found."));
    }

    transaction.CloseTransaction(this);

    return transaction;
}

public Transaction ReturnCar(Customer customer)
{
    if (customer != null)
    {
        var seekCarId = (from item in ListCustomerTransactions(customer)
                        where item.Customer != null && item.Customer.Id == customer.Id
                        select item.Car.Id).FirstOrDefault();

        var seekCar = (from item in ListRentedCars()
                        where item.Id == seekCarId
                        select item).FirstOrDefault();

        if (seekCar != null)
            return ReturnCar(customer, seekCar);
        else
        {
            LogAndShowMessage($"Customer with ID:{customer.Id} has no rented

```

```

cars!");
    }
}
else
    LogAndShowMessage("Customer has not been found!");

    return new Transaction();
}

public List<Car> ListAvailableCars()
{
    return new List<Car>(availableCars);
}

public List<Customer> ListRegisteredCustomers()
{
    return new List<Customer>(customers);
}

public List<Car> ListRentedCars()
{
    return new List<Car>(rentedCars);
}

public List<Transaction> ListCustomerTransactions(Customer customer)
{
    return currentTransactions.FindAll(t => t.Customer.Id == customer.Id);
}

public Customer LookupCustomer(string customerId)
{
    // validate registered customers list
    var customers = GetRegisteredCustomers();
    if (customers == null) throw new ArgumentNullException("No registered
customers!");
    // lookup customer
    var customer = customers.Find(c => string.Equals(c.Id, customerId,
StringComparison.InvariantCultureIgnoreCase));
    return customer;
}

public List<Transaction> LookupCustomerTransactions(string customerId)
{
    var result = new List<Transaction>();

```

```

        result.AddRange(currentTransactions.FindAll(
            tx => string.Equals(tx.Customer.Id, customerId,
StringComparison.InvariantCultureIgnoreCase)));

        result.AddRange(archiveTransactions.FindAll(
            tx => string.Equals(tx.Customer.Id, customerId,
StringComparison.InvariantCultureIgnoreCase)));

        return result;
    }

    public List<Transaction> LookupCarTransactions(string carId)
    {
        var result = new List<Transaction>();

        result.AddRange(currentTransactions.FindAll(
            tx => string.Equals(tx.Car.Id, carId,
StringComparison.InvariantCultureIgnoreCase)));

        result.AddRange(archiveTransactions.FindAll(
            tx => string.Equals(tx.Car.Id, carId,
StringComparison.InvariantCultureIgnoreCase)));

        return result;
    }

    public Car LookupCar(string carId)
    {
        Car car = LookupAvaliableCar(carId);

        if (car == null)
        {
            car = LookupRentedCar(carId);
        }

        return car;
    }

    public Car LookupRentedCar(string carId)
    {
        Car car;
        var rentedCars = GetRentedCars();
    }

```



```

        if (rentedCars == null) throw new ArgumentNullException("rentedCars");

        car = rentedCars.Find(c => string.Equals(c.Id, carId,
StringComparison.InvariantCultureIgnoreCase));
        return car;
    }

    public Car LookupAvaliableCar(string carId)
    {
        // validate avaliable cars list
        var avaliableCars = GetAvaliableCars();
        if (avaliableCars == null) throw new ArgumentNullException("avaliableCars");

        // lookup car
        var car = availableCars.Find(c => string.Equals(c.Id, carId,
StringComparison.InvariantCultureIgnoreCase));
        return car;
    }

    public List<Car> GetAvaliableCars()
    {
        return availableCars;
    }

    public Car GetCar(string carId)
    {
        return availableCars.Find(
            c => string.Equals(c.Id, carId, StringComparison.InvariantCultureIgnoreCase));
    }

    public Car GetFirstAvaliableCar(CarRentalSystem carRentalSystem)
    {
        return carRentalSystem.ListAvailableCars().FirstOrDefault();
    }

    public List<Customer> GetRegisteredCustomers()
    {
        return customers;
    }

    public List<Car> GetRentedCars()
    {
        return rentedCars;
    }

```

```

    }

    public List<Car> GetRentedCars(Customer customer)
    {
        var result = new List<Car>();

        var customerCurrentTransactions =
            currentTransactions.FindAll(tx => tx.Customer.Id == customer.Id);

        foreach (var customerTransaction in customerCurrentTransactions)
        {
            result.Add(customerTransaction.Car);
        }

        return result;
    }

    public void ArchiveTransaction(Transaction transaction)
    {
        archiveTransactions.Add(transaction);
        currentTransactions.Remove(transaction);
    }

    public void RentCar(Car car)
    {
        availableCars.Remove(car);
        rentedCars.Add(car);
    }

    public void ReturnCar(Car car)
    {
        rentedCars.Remove(car);
        availableCars.Add(car);
    }

    public void NewTransaction(Transaction transaction)
    {
        currentTransactions.Add(transaction);
    }
}

```



python

Copy code

```

from typing import List from sf.systems.rentals.cars.customer import Customer
from sf.systems.rentals.cars.car import Car from
sf.systems.rentals.cars.transaction import Transaction from
sf.systems.rentals.cars.error_handler import ErrorHandler from
sf.systems.rentals.cars.message_handler import MessageHandler from
sf.systems.rentals.cars.authentication_manager import AuthenticationManager
from sf.systems.rentals.cars.security_manager import SecurityManager from
sf.systems.rentals.cars.data_manager import DataManager from
sf.systems.rentals.cars.i_entities_list import IEntitiesList class
CarRentalSystem(IEntitiesList): def __init__(self, data_manager: DataManager
= None, security_manager: SecurityManager = None): self.customers:
List[Customer] = [] self.available_cars: List[Car] = [] self.rented_cars:
List[Car] = [] self.current_transactions: List[Transaction] = []
self.archive_transactions: List[Transaction] = [] self.error_handler =
ErrorHandler(True) self.data_manager = DataManager(self.error_handler,
MessageHandler(self.error_handler)) self.message_handler =
MessageHandler(self.error_handler) self.security_manager = SecurityManager()
if not security_manager else security_manager self.authentication_manager =
AuthenticationManager(self.security_manager) if data_manager:
self.data_manager = data_manager self.authentication_manager =
AuthenticationManager(self.security_manager) if security_manager:
self.security_manager = security_manager self.authentication_manager =
AuthenticationManager(self.security_manager) def register_customer(self, id:
str, name: str, phone_number: str, address: str, email: str) -> Customer:
seek_customer = self.lookup_customer(id) if seek_customer: return
seek_customer customer = Customer(id, name, phone_number, address, email)
self.customers.append(customer) return customer def delete_car(self, id: str)
-> bool: seek_car = self.lookup_available_car(id) seek_car_transactions =
self.lookup_car_transactions(id) if seek_car is None: seek_car =
self.lookup_rented_car(id) if seek_car is None:
self.error_handler.handle_error( ValueError(f"The specified car (ID={id})
does not exist in the system!")) return False else:
self.error_handler.handle_error( ValueError(f"The specified car (ID={id}) is
rented and can't be deleted!")) return False elif len(seek_car_transactions)
> 0: self.error_handler.handle_error( ValueError(f"The specified car (ID=
{id}) was rented and can't be deleted!")) return False else:
self.available_cars.remove(seek_car) return True def delete_customer(self,
id: str) -> bool: seek_customer_transactions =
self.lookup_customer_transactions(id) if len(seek_customer_transactions) > 0:
self.error_handler.handle_error( ValueError(f"The specified customer (ID=
{id}) has rented a car and can't be deleted!")) return False else:
seek_customer = self.lookup_customer(id) self.customers.remove(seek_customer)
return True def add_car(self, id_car: str, make: str, model: str, year: int,
daily_price: float) -> Car: seek_car = self.lookup_car(id_car) if seek_car:
return seek_car new_car = Car(id_car, make, model, year, daily_price, False)
self.available_cars.append(new

```





make methods starting from AddCar



Here are the methods starting from **AddCar** in Python:

python

Copy code

```
def AddCar(self, idCar, make, model, year, dailyPrice): seekCar =
self.LookupCar(idCar) if seekCar is None: newCar = Car(idCar, make, model,
year, dailyPrice, False) self.availableCars.append(newCar) return newCar
else: return seekCar def RentCar(self, customerId, carId, rentalDate,
returnDate): car = self.LookupCar(carId) if car not in self.availableCars:
self.errorHandler.HandleError(ValueError(f"The specified car (ID={carId}) is
not available for rental.")) customer = self.LookupCustomer(customerId) if
customer is not None: return Transaction.OpenTransaction(self, customer.Id,
car.Id, rentalDate, returnDate) else:
self.messageHandler.LogPlusMessage(f"The specified customer with ID:
{customerId} has not been found!") return Transaction() def ReturnCar(self,
customer, car): if car not in self.rentedCars:
self.errorHandler.HandleError(ValueError("The specified car has not been
rented by the specified customer.")) transaction = next((t for t in
self.currentTransactions if not t.IsClosed and t.Customer.Id == customer.Id
and t.Car.Id == car.Id), None) if transaction is None:
self.errorHandler.HandleError(ValueError("The specified transaction could not
be found.")) transaction.CloseTransaction(self) return transaction def
ReturnCarByCustomer(self, customer): if customer is not None: seekCarId =
next((item.Car.Id for item in self.ListCustomerTransactions(customer) if
item.Customer is not None and item.Customer.Id == customer.Id), None) seekCar
= next((item for item in self.ListRentedCars() if item.Id == seekCarId),
None) if seekCar is not None: return self.ReturnCar(customer, seekCar) else:
self.LogAndShowMessage(f"Customer with ID:{customer.Id} has no rented cars!")
else: self.LogAndShowMessage("Customer has not been found!") return
Transaction() def ListAvailableCars(self): return self.availableCars.copy()
def ListRegisteredCustomers(self): return self.customers.copy() def
ListRentedCars(self): return self.rentedCars.copy() def
ListCustomerTransactions(self, customer): return [t for t in
self.currentTransactions if t.Customer.Id == customer.Id] def
LookupCustomer(self, customerId): # validate registered customers list
customers = self.GetRegisteredCustomers() if customers is None: raise
ValueError("No registered customers!") # lookup customer customer = next((c
for c in customers if c.Id.lower() == customerId.lower()), None) return
customer def LookupCustomerTransactions(self, customerId): result = []
result.extend([tx for tx in self.currentTransactions if
tx.Customer.Id.lower() == customerId.lower()]) result.extend([tx for tx in
self.archiveTransactions if tx.Customer.Id.lower() == customerId.lower()])
return result def LookupCarTransactions(self, carId): result = []
```

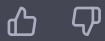
```
result.extend([tx for tx in self.currentTransactions if tx.Car.Id.lower() ==
carId.lower()]) result.extend([tx for tx in self.archiveTransactions if
tx.Car.Id.lower() == carId.lower()]) return result def LookupCar(self,
carId): car = self.LookupAvailableCar(carId) if car is None: car =
self.LookupRentedCar(carId) return car def Lookup
```



make Python naming (low case, underscore) starting from add_car



add_car
delete_car
delete_customer
register_customer
rent_car
return_car
list_available_cars
list_registered_customers
list_rented_cars
list_customer_transactions
lookup_customer
lookup_customer_transactions
lookup_car_transactions
lookup_car
lookup_rented_car
lookup_available_car
get_available_cars
get_car
get_first_available_car
get_registered_customers
get_rented_cars
get_rented_cars_customer
archive_transaction
rent_car
return_car
new_transaction



generate Python code for the following methods and explicit parameter types:

add_car
delete_car
delete_customer
register_customer



Here's the Python code for the methods with explicit parameter types:

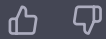
python

Copy code

```
from typing import List, Optional from datetime import datetime class Car:
def __init__(self, id: str, make: str, model: str, year: int, daily_price:
float, is_rented: bool): self.id = id self.make = make self.model = model
self.year = year self.daily_price = daily_price self.is_rented = is_rented
class Customer: def __init__(self, id: str, name: str, phone_number: str,
address: str, email: str): self.id = id self.name = name self.phone_number =
phone_number self.address = address self.email = email class Transaction: def
__init__(self, customer: Customer, car: Car, rental_date: datetime,
return_date: datetime): self.customer = customer self.car = car
self.rental_date = rental_date self.return_date = return_date self.is_closed
= False class CarRentalSystem: def __init__(self): self.customers:
List[Customer] = [] self.available_cars: List[Car] = [] self.rented_cars:
List[Car] = [] self.current_transactions: List[Transaction] = []
self.archive_transactions: List[Transaction] = [] def add_car(self, id_car:
str, make: str, model: str, year: int, daily_price: float) -> Car: seek_car =
self.lookup_car(id_car) if seek_car is None: new_car = Car(id_car, make,
model, year, daily_price, False) self.available_cars.append(new_car) return
new_car else: return seek_car def delete_car(self, id: str) -> bool: seek_car =
self.lookup_available_car(id) seek_car_transactions =
self.lookup_car_transactions(id) if seek_car is None: seek_car =
self.lookup_rented_car(id) if seek_car is None: raise ValueError(f"The
specified car (ID={id}) does not exist in the system!") else: raise
ValueError(f"The specified car (ID={id}) is rented and can't be deleted!")
elif len(seek_car_transactions) > 0: raise ValueError(f"The specified car
(ID={id}) was rented and can't be deleted!") else:
self.available_cars.remove(seek_car) return True def delete_customer(self,
id: str) -> bool: seek_customer_transactions =
self.lookup_customer_transactions(id) if len(seek_customer_transactions) > 0:
raise ValueError(f"The specified customer (ID={id}) has rented a car and
can't be deleted!") else: seek_customer = self.lookup_customer(id)
self.customers.remove(seek_customer) return True def register_customer(self,
id: str, name: str, phone_number: str, address: str, email: str) -> Customer:
```

```
seek_customer = self.lookup_customer(id) if seek_customer is None: customer =  
Customer(id, name, phone_number, address, email)  
self.customers.append(customer) return customer else: return seek_customer
```

Note that I assumed the return types based on the C# code, but you may need to adjust them to fit your specific use case in Python.



make following methods of this class **CarRentalSystem**

rent_car
return_car
list_available_cars
list_registered_customers
list_rented_cars
list_customer_transactions
lookup_customer
lookup_customer_transactions
lookup_car_transactions



Here are the Python code for the methods of **CarRentalSystem** class:

python

Copy code

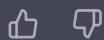
```
from typing import List from datetime import datetime from car import Car  
from customer import Customer from transaction import Transaction class  
CarRentalSystem: def __init__(self): self.customers: List[Customer] = []  
self.available_cars: List[Car] = [] self.rented_cars: List[Car] = []  
self.current_transactions: List[Transaction] = [] self.archive_transactions:  
List[Transaction] = [] def rent_car(self, customer_id: str, car_id: str,  
rental_date: datetime, return_date: datetime) -> Transaction: car =  
self.lookup_car(car_id) if car not in self.available_cars: raise  
ValueError(f"The specified car (ID={car_id}) is not available for rental.")  
customer = self.lookup_customer(customer_id) if customer: return  
Transaction.open_transaction(self, customer_id, car_id, rental_date,  
return_date) else: print(f"The specified customer with ID: {customer_id} has  
not been found!") return Transaction() def return_car(self, customer:  
Customer, car: Car) -> Transaction: if car not in self.rented_cars: raise  
ValueError("The specified car has not been rented by the specified  
customer.") transaction = next((t for t in self.current_transactions if not  
t.is_closed and t.customer.id == customer.id and t.car.id == car.id), None)  
if not transaction: raise ValueError("The specified transaction could not be
```

```

found.") transaction.close_transaction(self) return transaction def
list_available_cars(self) -> List[Car]: return list(self.available_cars) def
list_registered_customers(self) -> List[Customer]: return
list(self.customers) def list_rented_cars(self) -> List[Car]: return
list(self.rented_cars) def list_customer_transactions(self, customer:
Customer) -> List[Transaction]: return [t for t in self.current_transactions
if t.customer.id == customer.id] def lookup_customer(self, customer_id: str)
-> Customer: customers = self.get_registered_customers() if not customers:
raise ValueError("No registered customers!") return next((c for c in
customers if c.id.lower() == customer_id.lower()), None) def
lookup_customer_transactions(self, customer_id: str) -> List[Transaction]:
result = [] result.extend(t for t in self.current_transactions if
t.customer.id.lower() == customer_id.lower()) result.extend(t for t in
self.archive_transactions if t.customer.id.lower() == customer_id.lower())
return result def lookup_car_transactions(self, car_id: str) ->
List[Transaction]: result = [] result.extend(t for t in
self.current_transactions if t.car.id.lower() == car_id.lower())
result.extend(t for t in self.archive_transactions if t.car.id.lower() ==
car_id.lower()) return result

```

Note: This code assumes that **Customer**, **Car**, and **Transaction** classes have already been defined.



make following methods of this class **CarRentalSystem**: lookup_car
lookup_rented_car
lookup_available_car
get_available_cars
get_car
get_first_available_car



Here's the Python code for the remaining methods of the **CarRentalSystem** class:

python

Copy code

```

class CarRentalSystem: def __init__(self): self.customers = []
self.available_cars = [] self.rented_cars = [] self.current_transactions = []
self.archive_transactions = [] self.error_handler = ErrorHandler(True)
self.data_manager = DataManager(self.error_handler,
MessageHandler(self.error_handler)) self.message_handler =
MessageHandler(self.error_handler) if
os.path.exists(SecurityManager.KeyFile): self.security_manager =

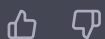
```



```

SecurityManager(open(SecurityManager.KeyFile, 'rb').read()) else:
self.security_manager = SecurityManager() self.authentication_manager =
AuthenticationManager(self.security_manager) def rent_car(self, customer_id:
str, car_id: str, rental_date: datetime, return_date: datetime) ->
Transaction: car = self.lookup_car(car_id) if car not in self.available_cars:
self.error_handler.handle_error( ValueError(f"The specified car (ID={car_id})
is not available for rental.")) customer = self.lookup_customer(customer_id)
if customer is not None: return Transaction.open_transaction(self,
customer_id, car_id, rental_date, return_date) else:
self.message_handler.log_plus_message(f"The specified customer with ID:
{customer_id} has not been found!") return Transaction() def return_car(self,
customer: Customer, car: Car) -> Transaction: if car not in self.rented_cars:
self.error_handler.handle_error(ValueError("The specified car has not been
rented by the specified customer.")) transaction = next((t for t in
self.current_transactions if not t.is_closed and t.customer.id == customer.id
and t.car.id == car.id), None) if transaction is None:
self.error_handler.handle_error(ValueError("The specified transaction could
not be found.")) transaction.close_transaction(self) return transaction def
list_available_cars(self) -> List[Car]: return list(self.available_cars) def
list_registered_customers(self) -> List[Customer]: return
list(self.customers) def list_rented_cars(self) -> List[Car]: return
list(self.rented_cars) def list_customer_transactions(self, customer:
Customer) -> List[Transaction]: return [t for t in self.current_transactions
if t.customer.id == customer.id] def lookup_customer(self, customer_id: str)
-> Optional[Customer]: return next((c for c in self.customers if c.id.lower()
== customer_id.lower()), None) def lookup_customer_transactions(self,
customer_id: str) -> List[Transaction]: result = [] for tx in
self.current_transactions: if tx.customer.id.lower() == customer_id.lower():
result.append(tx) for tx in self.archive_transactions: if
tx.customer.id.lower() == customer_id.lower(): result.append(tx) return
result def lookup_car_transactions(self, car_id: str) -> List[Transaction]:
result = [] for tx in self.current_transactions: if tx.car.id.lower() ==
car_id.lower(): result.append(tx) for tx in self.archive_transactions: if
tx.car.id.lower() == car_id.lower(): result.append(tx) return result def
lookup_car(self, car_id: str) -> Optional[Car]: car =
self.lookup_available_car(car_id) if car is None: car =
self.lookup_rented_car(car_id) return car def lookup_rented_car(self, car_id:
str) -> Optional[Car]: return

```



make following methods: lookup_rented_car

lookup_available_car

get_available_cars

get_car

get_first_available_car

get_registered_custome



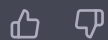
Here are the methods starting from `lookup_rented_car`:

python

Copy code

```
def lookup_rented_car(self, car_id: str) -> Optional[Car]: rented_cars =
self.get_rented_cars() return next((car for car in rented_cars if car.id ==
car_id), None) def lookup_available_car(self, car_id: str) -> Optional[Car]:
available_cars = self.get_available_cars() return next((car for car in
available_cars if car.id == car_id), None) def get_available_cars(self) ->
List[Car]: return self.available_cars def get_car(self, car_id: str) ->
Optional[Car]: car = self.lookup_available_car(car_id) if car is None: car =
self.lookup_rented_car(car_id) return car def get_first_available_car(self) -
> Optional[Car]: available_cars = self.get_available_cars() return
available_cars[0] if len(available_cars) > 0 else None def
get_registered_customers(self) -> List[Customer]: return self.customers
```

Note: This assumes that the `Car`, `Customer`, and `Transaction` classes have been properly defined.



make following methods: `lookup_rented_car`

`lookup_available_car`

`get_available_cars`

`get_car`

`get_first_available_car`

`get_registered_customers`



Here's the Python code for the methods you requested:

python

Copy code

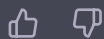
```
from typing import List from datetime import datetime from
sf.systems.rentals.cars import Car, Customer, Transaction class
CarRentalSystem: def lookup_rented_car(self, car_id: str) -> Car: rented_cars
= self.get_rented_cars() return next((car for car in rented_cars if car.id ==
car_id), None) def lookup_available_car(self, car_id: str) -> Car:
available_cars = self.get_available_cars() return next((car for car in
available_cars if car.id == car_id), None) def get_available_cars(self) ->
List[Car]: return self.available_cars def get_car(self, car_id: str) -> Car:
```

```

available_car = self.lookup_available_car(car_id) if available_car is not
None: return available_car rented_car = self.lookup_rented_car(car_id) if
rented_car is not None: return rented_car return None def
get_first_available_car(self) -> Car: available_cars =
self.get_available_cars() if len(available_cars) > 0: return
available_cars[0] return None def get_registered_customers(self) ->
List[Customer]: return self.customers

```

Note: this assumes that the necessary instance variables are already defined in the class, such as `self.customers`, `self.available_cars`, `self.rented_cars`, and so on.



make following methods: `get_rented_cars`
`get_rented_cars_customer`
`archive_transaction`
`rent_car`
`return_car`
`new_transaction`



Here are the Python representations of the remaining methods:

python

Copy code

```

def get_rented_cars(self): return self.rented_cars def
get_rented_cars_customer(self, customer): result = [] for tx in
self.current_transactions: if tx.customer == customer: result.append(tx.car)
return result def archive_transaction(self, transaction):
self.archive_transactions.append(transaction)
self.current_transactions.remove(transaction) def rent_car(self, car):
self.available_cars.remove(car) self.rented_cars.append(car) def
return_car(self, car): self.rented_cars.remove(car)
self.available_cars.append(car) def new_transaction(self, transaction):
self.current_transactions.append(transaction)

```



make Python code with typed params and Python naming conventions:
 using System;

```
namespace sf.systems.rentals.cars
{
    public partial class CarRentalSystem
    {
        private readonly AuthenticationManager authenticationManager;
        private User currentUser;

        public CarRentalSystem(AuthenticationManager authenticationManager) : this()
        {
            this.authenticationManager = authenticationManager;
        }

        public bool Login(string username, string password, UserRole userRole)
        {
            bool success = authenticationManager.Login(username, password);
            if (success)
            {
                currentUser = new User(System.Guid.NewGuid().ToString(), username,
userRole);
                LogAndShowMessage("Login successful.");
            }
            else
            {
                LogAndShowMessage("Login failed.");
            }
            return success;
        }

        public void Logout()
        {
            authenticationManager.Logout();
            currentUser = null;
            LogAndShowMessage("Logout successful.");
        }

        public bool IsLoggedIn()
        {
            return currentUser != null;
        }

        public User GetCurrentUser()
        {
            return currentUser;
        }
    }
}
```

```

    }
}
}

```



Here's the Python code with typed parameters and Python naming conventions:

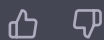
python

Copy code

```

from enum import Enum
class UserRole(Enum):
    ADMIN = 0
    CUSTOMER = 1
class CarRentalSystem:
    def __init__(self, authentication_manager=None):
        self.authentication_manager = authentication_manager
        self.current_user = None
    def login(self, username: str, password: str, user_role: UserRole) -> bool:
        success = self.authentication_manager.login(username, password)
        if success:
            self.current_user = User(str(uuid.uuid4()), username, user_role)
            self.log_and_show_message("Login successful.")
        else:
            self.log_and_show_message("Login failed.")
            return success
    def logout(self) -> None:
        self.authentication_manager.logout()
        self.current_user = None
        self.log_and_show_message("Logout successful.")
    def is_logged_in(self) -> bool:
        return self.current_user is not None
    def get_current_user(self) -> User:
        return self.current_user

```



make Python code with typed params and Python naming conventions: using System; using System.Collections.Generic;

namespace sf.systems.rentals.cars

```
{
```

```
    public partial class CarRentalSystem
```

```
    {
```

```
        private const string FileSuffixCurrent = "current";
```

```
        private const string FileSuffixArchive = "archive";
```

```
        private readonly DataManager dataManager;
```

```
        public CarRentalSystem(DataManager dataManager) : this()
```

```
        {
```

```
            this.dataManager = dataManager;
```

```
        }
```

```
        public void LoadData()
```

```
        {
```

```

        dataManager.AssignOwner(null);
        dataManager.ReadData(customers, EntityType.CUSTOMER, DataType.CSV,
string.Empty);
        dataManager.ReadData(availableCars, EntityType.CAR, DataType.CSV,
Convert.ToString(RentedType.AVALIABLE));
        dataManager.ReadData(rentedCars, EntityType.CAR, DataType.CSV,
Convert.ToString(RentedType.RENTED));

        dataManager.AssignOwner(this);
        dataManager.ReadDataExtended(currentTransactions, EntityType.TRANSACTION,
DataType.CSV, FileSuffixCurrent, this);
        dataManager.ReadDataExtended(archiveTransactions, EntityType.TRANSACTION,
DataType.CSV, FileSuffixArchive, this);
    }

    public void SaveData()
    {
        dataManager.WriteData<Customer>(customers, EntityType.CUSTOMER,
DataType.CSV, string.Empty);
        dataManager.WriteData<Car>(availableCars, EntityType.CAR, DataType.CSV,
Convert.ToString(RentedType.AVALIABLE));
        dataManager.WriteData<Car>(rentedCars, EntityType.CAR, DataType.CSV,
Convert.ToString(RentedType.RENTED));
        dataManager.WriteData<Transaction>(currentTransactions,
EntityType.TRANSACTION, DataType.CSV, FileSuffixCurrent);
        dataManager.WriteData<Transaction>(archiveTransactions,
EntityType.TRANSACTION, DataType.CSV, FileSuffixArchive);
    }
}
}

```



python

Copy code

```

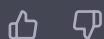
from typing import List from sf.systems.rentals.cars.data_manager import
DataManager, EntityType, DataType from sf.systems.rentals.cars.entities
import Customer, Car, Transaction, RentedType class CarRentalSystem:
FILE_SUFFIX_CURRENT = "current" FILE_SUFFIX_ARCHIVE = "archive" def
__init__(self, data_manager: DataManager) -> None: self.customers:
List[Customer] = [] self.available_cars: List[Car] = [] self.rented_cars:
List[Car] = [] self.current_transactions: List[Transaction] = []
self.archive_transactions: List[Transaction] = [] self.data_manager =
data_manager def load_data(self) -> None:
self.data_manager.assign_owner(None)

```

```

self.data_manager.read_data(self.customers, EntityType.CUSTOMER,
DataType.CSV, "") self.data_manager.read_data(self.available_cars,
EntityType.CAR, DataType.CSV, str(RentedType.AVALIABLE))
self.data_manager.read_data(self.rented_cars, EntityType.CAR, DataType.CSV,
str(RentedType.RENTED)) self.data_manager.assign_owner(self)
self.data_manager.read_data_extended(self.current_transactions,
EntityType.TRANSACTION, DataType.CSV, self.FILE_SUFFIX_CURRENT, self)
self.data_manager.read_data_extended(self.archive_transactions,
EntityType.TRANSACTION, DataType.CSV, self.FILE_SUFFIX_ARCHIVE, self) def
save_data(self) -> None: self.data_manager.write_data(self.customers,
EntityType.CUSTOMER, DataType.CSV, "")
self.data_manager.write_data(self.available_cars, EntityType.CAR,
DataType.CSV, str(RentedType.AVALIABLE))
self.data_manager.write_data(self.rented_cars, EntityType.CAR, DataType.CSV,
str(RentedType.RENTED))
self.data_manager.write_data(self.current_transactions,
EntityType.TRANSACTION, DataType.CSV, self.FILE_SUFFIX_CURRENT)
self.data_manager.write_data(self.archive_transactions,
EntityType.TRANSACTION, DataType.CSV, self.FILE_SUFFIX_ARCHIVE)

```



make Python code with typed params and Python naming conventions (only logging/messaging and error handling):

using System;

using System.Collections.Generic;

using System.Text;

namespace sf.systems.rentals.cars

{

public partial class CarRentalSystem

{

public void Log(string message)

{

messageHandler.Log(message);

}

public void ShowMessage(string message)

{

messageHandler.ShowMessage(message);

}

public void LogAndShowMessage(string message)

{

```

        messageHandler.Log(message);
        messageHandler.ShowMessage(message);
    }

    public void HandleError(Exception ex)
    {
        errorHandler.HandleError(ex);
    }
}

```



Here's the Python code with typed parameters and Python naming conventions for the logging/messaging and error handling methods:

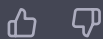
python

Copy code

```

class CarRentalSystem:
    def log(self, message: str):
        self.message_handler.log(message)
    def show_message(self, message: str):
        self.message_handler.show_message(message)
    def log_and_show_message(self, message: str):
        self.message_handler.log(message)
        self.message_handler.show_message(message)
    def handle_error(self, ex: Exception):
        self.error_handler.handle_error(ex)

```



add security methods: using System;

```

namespace sf.systems.rentals.cars
{
    public partial class CarRentalSystem
    {
        private readonly SecurityManager securityManager;

        public CarRentalSystem(SecurityManager securityManager) : this()
        {
            this.securityManager = securityManager;
        }

        public bool Authorize(User user, string permission)
        {
            return securityManager.Authorize(user, permission);
        }
    }
}

```



```

    }

    public void EncryptData()
    {
        foreach (Customer customer in customers)
        {
            customer.PhoneNumber = securityManager.Encrypt(customer.PhoneNumber);
            customer.Address = securityManager.Encrypt(customer.Address);
            customer.Email = securityManager.Encrypt(customer.Email);
        }
    }

    public void DecryptData()
    {
        foreach (Customer customer in customers)
        {
            customer.PhoneNumber = securityManager.Decrypt(customer.PhoneNumber);
            customer.Address = securityManager.Decrypt(customer.Address);
            customer.Email = securityManager.Decrypt(customer.Email);
        }
    }
}

```



Here's the Python code with typed parameters and Python naming conventions for the security methods:

python

Copy code

```

from typing import List from sf.systems.rentals.cars.models import
Customer, User class CarRentalSystem: def __init__(self, security_manager):
self.security_manager = security_manager def authorize(self, user: User,
permission: str) -> bool: return self.security_manager.authorize(user,
permission) def encrypt_data(self, customers: List[Customer]): for customer
in customers: customer.phone_number =
self.security_manager.encrypt(customer.phone_number) customer.address =
self.security_manager.encrypt(customer.address) customer.email =
self.security_manager.encrypt(customer.email) def decrypt_data(self,
customers: List[Customer]): for customer in customers:
customer.phone_number =
self.security_manager.decrypt(customer.phone_number) customer.address =

```

```
self.security_manager.decrypt(customer.address) customer.email =  
self.security_manager.decrypt(customer.email)
```

Note that I assumed that the **Customer** class is defined in a separate module and imported here with **from sf.systems.rentals.cars.models import Customer**.

Represent as Python code with typed parameters and Python naming conventions:



```
using System;  
using System.Collections.Generic;  
using System.Text;  
  
namespace sf.systems.rentals.cars  
{  
    public static class CarRentalCommands  
    {  
        public static CarRentalContext ReturnCar(CarRentalContext rentalContext,  
CarRentalContextAction posteriorReturnCar)  
        {  
            // validation  
            validateContext(rentalContext);  
  
            // go!  
            rentalContext.RentalTransaction = rentalContext.CarRentalSystem.ReturnCar(  
                rentalContext.CarRentalSystem.LookupCustomer(rentalContext.CustomerID));  
  
            // finally  
            rentalContext.ActionCompleted = rentalContext.RentalTransaction.IsClosed;  
  
            if (rentalContext.ActionCompleted)  
                rentalContext = posteriorReturnCar?.Invoke(rentalContext);  
  
            return rentalContext;  
        }  
  
        public static CarRentalContext RentCar(CarRentalContext rentalContext,  
            CarRentalContextAction notifierEmptyCustomer,  
            CarRentalContextAction notifierCarIsnotAvaliable,  
            CarRentalContextAction posteriorReturnCar  
        )  
        {  
            // validation  
            validateContext(rentalContext);
```

```

        // go!
        // validate params
        var validationError = validateCustomer(rentalContext, notifierEmptyCustomer);
        var car = rentalContext.CarRentalSystem.GetCar(rentalContext.CarID);
        validationError = validateCar(rentalContext, notifierCarIsnotAvaliable, car);

        // exit if a validation is occurred
        if (validationError)
            return rentalContext;

        // Rent first available car by given customer for 3 days
        rentalContext.RentalTransaction = rentalContext.CarRentalSystem.RentCar(
            rentalContext.CustomerID,
            rentalContext.CarID,
            DateTime.Now,
            DateTime.Now.AddDays(3)
        );

        rentalContext.ActionCompleted =
            rentalContext.RentalTransaction.Id != null
            && rentalContext.RentalTransaction.Id.Trim().Length > 1;

        if (rentalContext.ActionCompleted)
            rentalContext = posteriorReturnCar?.Invoke(rentalContext);

        return rentalContext;
    }

    public static void RegisterCustomer(CarRentalContext context, string[] argCustomer)
    {
        // init
        var rentalSystem = context.CarRentalSystem;
        var customerID = argCustomer[0];

        // go
        var customer = rentalSystem.LookupCustomer(customerID);
        if (customer == null)
        {
            customer = rentalSystem.RegisterCustomer(argCustomer[0], argCustomer[1],
            argCustomer[2], argCustomer[3], argCustomer[4]);
            context.CarRentalSystem.LogAndShowMessage(
                $"New Customer \"{customer.Name}\" (ID: {customer.Id}) has been

```

```

        successfully added to the System!");
    }
    else
    {
        context.CarRentalSystem.LogAndShowMessage(
            $"Customer \"{customer.Name}\" (ID: {customer.Id}) is already registered in
the System!");
    }
}

public static void DeleteCustomer(CarRentalContext context, string customerId)
{
    // init
    var rentalSystem = context.CarRentalSystem;
    var customer = rentalSystem.LookupCustomer(customerId);

    // go
    if (rentalSystem.DeleteCustomer(customerId))
    {
        context.CarRentalSystem.LogAndShowMessage(
            $"Customer \"{customer.Name}\" (ID: {customer.Id}) has been successfully
deleted from the System!");
    }
    else
    {
        context.CarRentalSystem.LogAndShowMessage(
            $"Customer with ID: {customerId} has not been found!");
    }
}

public static void DeleteCar(CarRentalContext context, string carId)
{
    if (context.CarRentalSystem.DeleteCar(carId))
    {
        context.CarRentalSystem.LogAndShowMessage(
            $"Car with ID: {carId} has been successfully deleted from the System!");
    }
}

public static void AddCar(CarRentalContext context, string[] argCar)
{
    // init
    var carRentalSystem = context.CarRentalSystem;

```

```

        var carId = argCar[0];
        var car = carRentalSystem.LookupCar(carId);

        // go!
        if (car == null)
        {
            var newCar = context.CarRentalSystem.AddCar(argCar[0], argCar[1], argCar[2],
Convert.ToInt32(argCar[3]), Convert.ToDouble(argCar[4]));
            context.CarRentalSystem.LogAndShowMessage($"New Car with ID: {newCar.Id}
has been successfully added to the System!");
        }
        else
            context.CarRentalSystem.LogAndShowMessage($"Car with ID: {car.Id} already
exists in the System!");
    }

    public static void SaveData(CarRentalContext context)
    {
        // Save the data to disk
        context.CarRentalSystem.LogAndShowMessage("\nSaving data ...");
        context.CarRentalSystem.SaveData();
        context.CarRentalSystem.LogAndShowMessage("All data has been saved.\n");
    }

    public static void LoadData(CarRentalContext context)
    {
        // Load the data from disk
        context.CarRentalSystem.LogAndShowMessage("\nLoading data ...");
        context.CarRentalSystem.LoadData();
        context.CarRentalSystem.LogAndShowMessage("All data has been loaded.\n");
    }

    public static CarRentalContext ReturnCar(CarRentalContext context)
    {
        return ReturnCar(context,
            (CarRentalContext ctx) =>
            {
                context.CarRentalSystem.LogAndShowMessage(
                    $"Customer rental is closed (ID-Customer:{context.CustomerID}, ID-
Rental: {context.RentalTransaction.Id}).\n");

                return ctx;
            }
        );
    }

```

```

    );
}

public static CarRentalContext RentCar(CarRentalContext context)
{
    return RentCar(context,
        // empty customer
        (CarRentalContext ctx) =>
        {
            ctx.CarRentalSystem.LogAndShowMessage("Customer-ID is empty!");
            return ctx;
        },
        // car is not available
        (CarRentalContext ctx) =>
        {
            ctx.CarRentalSystem.LogAndShowMessage($"Car with ID:{ctx.CarID} is not
available!");
            return ctx;
        },
        // posterior - rent message
        (CarRentalContext ctx) =>
        {
            ctx.CarRentalSystem.LogAndShowMessage(
                $"\\nCar with (ID-Car:{ctx.CarID}) has been rented by Customer (ID-
Customer:{ctx.CustomerID}," +
                $" ID-Rental: {ctx.RentalTransaction.Id}).\\n");
            return ctx;
        }
    );
}

private static void validateContext(CarRentalContext rentalContext)
{
    if (rentalContext == null) throw new ArgumentNullException("rentalContext");
    if (rentalContext.CarRentalSystem == null) throw new
ArgumentNullException("rentalContext.carRentalSystem");
}

private static bool validateCar(CarRentalContext rentalContext,
CarRentalContextAction notifierCarIsnotAvaliable, object car)
{
    var validationError = false;

```

```

        if (car == null)
        {
            notifierCarIsnotAvaliable?.Invoke(rentalContext);
            validationError = true;
        }

        return validationError;
    }

    private static bool validateCustomer(CarRentalContext rentalContext,
    CarRentalContextAction notifierEmptyCustomer)
    {
        var validationError = false;

        if (string.IsNullOrEmpty(rentalContext.CustomerID))
        {
            notifierEmptyCustomer?.Invoke(rentalContext);
            validationError = true;
        }

        return validationError;
    }
}

```



python

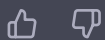
Copy code

```

from typing import Callable, Union from
sf.systems.rentals.cars.car_rental_context import CarRentalContext,
CarRentalContextAction class CarRentalCommands: @staticmethod def
ReturnCar(rentalContext: CarRentalContext, posteriorReturnCar:
CarRentalContextAction) -> CarRentalContext: # validation
CarRentalCommands.validateContext(rentalContext) # go!
rentalContext.RentalTransaction = rentalContext.CarRentalSystem.return_car(
rentalContext.CarRentalSystem.lookup_customer(rentalContext.CustomerID)) #
finally rentalContext.ActionCompleted =
rentalContext.RentalTransaction.is_closed if rentalContext.ActionCompleted:
rentalContext = posteriorReturnCar(rentalContext) return rentalContext
@staticmethod def RentCar(rentalContext: CarRentalContext,
notifierEmptyCustomer: CarRentalContextAction, notifierCarIsnotAvaliable:
CarRentalContextAction, posteriorReturnCar: CarRentalContextAction) ->
CarRentalContext: # validation
CarRentalCommands.validateContext(rentalContext) # go! # validate params
validationError = CarRentalCommands.validateCustomer(rentalContext,
notifierEmptyCustomer) car =
rentalContext.CarRentalSystem.get_car(rentalContext.CarID) validationError =
CarRentalCommands.validateCar(rentalContext, notifierCarIsnotAvaliable, car)

```

```
# exit if a validation is occurred if validationError: return rentalContext #
Rent first available car by given customer for 3 days
rentalContext.RentalTransaction = rentalContext.CarRentalSystem.rent_car(
rentalContext.CustomerID, rentalContext.CarID, datetime.datetime.now(),
datetime.datetime.now() + datetime.timedelta(days=3) )
rentalContext.ActionCompleted = bool(rentalContext.RentalTransaction.id) if
rentalContext.ActionCompleted: rentalContext =
posteriorReturnCar(rentalContext) return rentalContext @staticmethod def
RegisterCustomer(context: CarRentalContext, argCustomer: list[str]): # init
rentalSystem = context.CarRentalSystem customerID = argCustomer[0] # go
customer = rentalSystem.lookup_customer(customerID) if customer is None:
customer = rentalSystem.register_customer(argCustomer[0], argCustomer[1],
argCustomer[2], argCustomer[3], argCustomer[4])
context.CarRentalSystem.log_and_show_message(f'New Customer "{customer.name}"
(ID: {customer.id}) has been successfully added to the System!') else:
context.CarRentalSystem.log_and_show_message(f'Customer "{customer.name}"
(ID: {customer.id}) is already registered in the System!') @staticmethod def
DeleteCustomer(context: CarRentalContext, customerId: str): # init
rentalSystem = context.CarRentalSystem customer =
rentalSystem.lookup_customer(customerId) # go if
rentalSystem.delete_customer(customerId):
context.CarRentalSystem.log_and_show_message(f'Customer "{customer.name}"
(ID: {customer.id}) has been successfully deleted from the System!') else:
context.CarRentalSystem.log_and_show_message(f'Customer with ID: {customerId}
has not been found!') @staticmethod def DeleteCar(context: CarRentalContext,
carId: str): if context.CarRentalSystem.delete_car(carId):
context.CarRentalSystem.log_and_show_message(f'Car
```



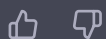
show method list



Here's the list of methods present in the code snippets you provided:

1. `add_customer`
2. `delete_customer`
3. `register_customer`
4. `rent_car`
5. `return_car`
6. `list_available_cars`
7. `list_registered_customers`
8. `list_rented_cars`

9. `list_customer_transactions`
10. `lookup_customer`
11. `lookup_customer_transactions`
12. `lookup_car`
13. `lookup_rented_car`
14. `lookup_available_car`
15. `get_available_cars`
16. `get_car`
17. `get_first_available_car`
18. `get_registered_customers`
19. `get_rented_cars`
20. `get_rented_cars_customer`
21. `archive_transaction`
22. `new_transaction`
23. `LoadData`
24. `SaveData`
25. `Log`
26. `ShowMessage`
27. `LogAndShowMessage`
28. `HandleError`
29. `Authorize`
30. `EncryptData`
31. `DecryptData`
32. `ReturnCar`
33. `RentCar`
34. `DeleteCustomer`
35. `DeleteCar`
36. `AddCar`



Represent as Python code with typed parameters and Python naming conventions:

```
public static class CarRentalCommands
{
    private static void validateContext(CarRentalContext rentalContext)
    {
```

```

        if (rentalContext == null) throw new ArgumentNullException("rentalContext");
        if (rentalContext.CarRentalSystem == null) throw new
ArgumentNullException("rentalContext.carRentalSystem");
    }

    private static bool validateCar(CarRentalContext rentalContext,
CarRentalContextAction notifierCarIsnotAvaliable, object car)
    {
        var validationError = false;

        if (car == null)
        {
            notifierCarIsnotAvaliable?.Invoke(rentalContext);
            validationError = true;
        }

        return validationError;
    }

    private static bool validateCustomer(CarRentalContext rentalContext,
CarRentalContextAction notifierEmptyCustomer)
    {
        var validationError = false;

        if (string.IsNullOrEmpty(rentalContext.CustomerID))
        {
            notifierEmptyCustomer?.Invoke(rentalContext);
            validationError = true;
        }

        return validationError;
    }
}

```



This code represents a set of private static helper methods used by the **CarRentalCommands** class. Here is the Python code with typed parameters and Python naming conventions:

python

Copy code

```

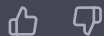
from typing import Optional, Callable, Any from
sf.systems.rentals.cars.car_rental_context import CarRentalContext,

```

```

CarRentalContextAction class CarRentalCommands: @staticmethod def
validate_context(rental_context: CarRentalContext) -> None: if rental_context
is None: raise ValueError("rental_context must not be None") if
rental_context.car_rental_system is None: raise
ValueError("rental_context.car_rental_system must not be None") @staticmethod
def validate_car(rental_context: CarRentalContext,
notifier_car_is_not_available: Optional[CarRentalContextAction], car: Any) ->
bool: validation_error = False if car is None: if
notifier_car_is_not_available: notifier_car_is_not_available(rental_context)
validation_error = True return validation_error @staticmethod def
validate_customer(rental_context: CarRentalContext, notifier_empty_customer:
Optional[CarRentalContextAction]) -> bool: validation_error = False if not
rental_context.customer_id: if notifier_empty_customer:
notifier_empty_customer(rental_context) validation_error = True return
validation_error

```



Represent in Python with typed param and python naming conventions:

```

using System;
using System.Linq;
using System.Collections.Generic;

namespace sf.systems.rentals.cars
{
    class MainProgram2
    {
        static int Main(string[] args)
        {
            // init
            string argCommand = default;
            string[] argCustomer = default;
            string[] argCar = default;

            // resolve args
            if (!testMode)
            {
                foreach (var arg in args)
                {
                    var arg_parts = arg.Split("=");
                    // command-args
                    string argKey = arg_parts[0].ToLowerInvariant().Trim();

```

```

        if (arg_parts.Length > 1)
        {
            var argValue = arg_parts[1].Trim();

            switch (argKey)
            {
                case "cmd":
                    argCommand = argValue;
                    break;
                case "car":
                    argCar = argValue.Split(',');
                    break;
                case "customer":
                    argCustomer = argValue.Split(',');
                    break;
            }
        }
    }
}

// Initialize car rental system
var carRentalSystem = new CarRentalSystem();
// create rental context
var context = new CarRentalContext();
context.CarRentalSystem = carRentalSystem;
context.Command = argCommand;
context.CarID = argCar?[0];
context.CustomerID = argCustomer?[0];

// Go!
try
{
    CarRentalCommands.LoadData(context);

    // run command
    switch (argCommand)
    {
        // production commands
        case "add_car":
            if (argCar == null || argCar.Length != 5)
            {
                logAndShow(context, "\nERROR: Invalid car data provided! Usage:
cmd=add_car car=\"ID,Make,Model,Year,PricePerDay\"");
            }
        }
    }
}

```

```

        return -1;
    }
    CarRentalCommands.AddCar(context, argCar);
    break;
case "delete_car":
    if (argCar == null || argCar.Length != 1)
    {
        logAndShow(context, "\nERROR: Invalid car ID provided! Usage:
cmd=delete_car car=\"ID\"");
        return -1;
    }
    CarRentalCommands.DeleteCar(context, argCar[0]);
    break;
case "register_customer":
    if (argCustomer == null || argCustomer.Length != 5)
    {
        logAndShow(context, "\nERROR: Invalid customer data provided!
Usage: cmd=register_customer customer=\"ID,Name,PhoneNumber,Address,Email\"");
        return -1;
    }
    CarRentalCommands.RegisterCustomer(context, argCustomer);
    break;
case "delete_customer":
    if (argCustomer == null || argCustomer.Length != 1)
    {
        logAndShow(context, "\nERROR: Invalid customer ID provided! Usage:
cmd=delete_customer customer=\"ID\"");
        return -1;
    }
    CarRentalCommands.DeleteCustomer(context, argCustomer[0]);
    break;
case "rent_car":
    if (argCar == null || argCar.Length != 1 || argCustomer == null ||
argCustomer.Length != 1)
    {
        logAndShow(context, "\nERROR: Invalid car or customer ID provided!
Usage: cmd=rent_car car=\"ID\" customer=\"ID\"");
        return -1;
    }
    context.CarID = argCar[0];
    context.CustomerID = argCustomer[0];
    context = CarRentalCommands.RentCar(context);
    break;

```

```

        case "return_car":
            if (argCustomer == null || argCustomer.Length != 1)
            {
                logAndShow(context, "\nERROR: Invalid customer ID provided! Usage:
cmd=return_car customer=\"ID\"");
                return -1;
            }
            context.CustomerID = argCustomer[0];
            context = CarRentalCommands.ReturnCar(context);
            break;

        default:
            logAndShow(context, $"Unknown command: \"{argCommand}\"!\n");
            showMessage(context, "Use samples:");
            showMessage(context, "CarRentalSystem cmd=register_customer
customer=\"C003,Mary Jung,777 - 1234,911 Main St,mjung@example.com\"");
            showMessage(context, "CarRentalSystem cmd=delete_customer
customer=\"C003\"");
            showMessage(context, "CarRentalSystem cmd=add_car
car=\"CAR911,Porsche,Macant,2023,190.0\"");
            showMessage(context, "CarRentalSystem cmd=add_car
car=\"CAR11,Audi,A1,2021,90.0\"");
            showMessage(context, "CarRentalSystem cmd=delete_car
car=\"CAR11\"");
            showMessage(context, "CarRentalSystem cmd=rent_car car=\"CAR911\"
customer=\"C003\"");
            return -1;
    }

    CarRentalCommands.SaveData(context);
    return 0;
}
catch (Exception ex)
{
    logAndShow(context, $"ERROR: {ex.Message} \n\nSTACK-TRACE:
{ex.StackTrace}");
    CarRentalCommands.SaveData(context);

    return -1;
}
}

```

```

private static void showMessage(CarRentalContext context, string message)

```

```

    {
        context.CarRentalSystem.ShowMessage(message);
    }

    private static void logAndShow(CarRentalContext context, string message)
    {
        context.CarRentalSystem.LogAndShowMessage(message);
    }
}
}

```



python

Copy code

```

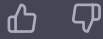
from typing import List, Tuple from sf.systems.rentals.cars import
CarRentalCommands, CarRentalContext, CarRentalContextAction, CarRentalSystem
def main(args: List[str]) -> int: # init argCommand = '' argCustomer = []
argCar = [] # resolve args if not testMode: for arg in args: arg_parts =
arg.split("=") # command-args argKey = arg_parts[0].lower().strip() if
len(arg_parts) > 1: argValue = arg_parts[1].strip() if argKey == "cmd":
argCommand = argValue elif argKey == "car": argCar = argValue.split(',') elif
argKey == "customer": argCustomer = argValue.split(',') # Initialize car
rental system carRentalSystem = CarRentalSystem() # create rental context
context = CarRentalContext() context.carRentalSystem = carRentalSystem
context.command = argCommand context.carID = argCar[0] if argCar else None
context.customerID = argCustomer[0] if argCustomer else None # Go! try:
CarRentalCommands.load_data(context) # run command if argCommand ==
"add_car": if not argCar or len(argCar) != 5: log_and_show(context, "\nERROR:
Invalid car data provided! Usage: cmd=add_car
car=\"ID,Make,Model,Year,PricePerDay\"") return -1
CarRentalCommands.add_car(context, argCar) elif argCommand == "delete_car":
if not argCar or len(argCar) != 1: log_and_show(context, "\nERROR: Invalid
car ID provided! Usage: cmd=delete_car car=\"ID\"") return -1
CarRentalCommands.delete_car(context, argCar[0]) elif argCommand ==
"register_customer": if not argCustomer or len(argCustomer) != 5:
log_and_show(context, "\nERROR: Invalid customer data provided! Usage:
cmd=register_customer customer=\"ID,Name,PhoneNumber,Address,Email\"") return
-1 CarRentalCommands.register_customer(context, argCustomer) elif argCommand
== "delete_customer": if not argCustomer or len(argCustomer) != 1:
log_and_show(context, "\nERROR: Invalid customer ID provided! Usage:
cmd=delete_customer customer=\"ID\"") return -1
CarRentalCommands.delete_customer(context, argCustomer[0]) elif argCommand ==
"rent_car": if not argCar or len(argCar) != 1 or not argCustomer or
len(argCustomer) != 1: log_and_show(context, "\nERROR: Invalid car or
customer ID provided! Usage: cmd=rent_car car=\"ID\" customer=\"ID\"") return
-1 context.carID = argCar[0] context.customerID = argCustomer[0] context =
CarRentalCommands.rent_car(context) elif argCommand == "return_car": if not

```

```

argCustomer or len(argCustomer) != 1: log_and_show(context, "\nERROR: Invalid
customer ID provided! Usage: cmd=return_car customer=\"ID\") return -1
context.customerID = argCustomer[0] context =
CarRentalCommands.return_car(context) else: log_and_show(context, f"\nUnknown
command: \"{arg

```



continue starting from "elif argCommand == "return_car":"



```

elif argCommand == "return_car":
if (argCustomer == None or len(argCustomer) != 1):
logAndShow(context, "\nERROR: Invalid customer ID provided! Usage: cmd=return_car
customer="ID")
return -1
context.CustomerID = argCustomer[0]
context = CarRentalCommands.ReturnCar(context)
break

```

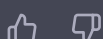
swift

Copy code

```

else: logAndShow(context, f"\nUnknown command: \"{argCommand}\"!\n")
showMessage(context, "Use samples:") showMessage(context, "CarRentalSystem
cmd=register_customer customer=\"C003,Mary Jung,777 - 1234,911 Main
St,mjung@example.com\") showMessage(context, "CarRentalSystem
cmd=delete_customer customer=\"C003\") showMessage(context, "CarRentalSystem
cmd=add_car car=\"CAR911,Porsche,Macant,2023,190.0\") showMessage(context,
"CarRentalSystem cmd=add_car car=\"CAR11,Audi,A1,2021,90.0\")
showMessage(context, "CarRentalSystem cmd=delete_car car=\"CAR11\")
showMessage(context, "CarRentalSystem cmd=rent_car car=\"CAR911\"
customer=\"C003\") return -1 CarRentalCommands.SaveData(context) return 0
except Exception as ex: logAndShow(context, f"\nERROR: {ex} \n\nSTACK-TRACE:
{traceback.format_exc()}") CarRentalCommands.SaveData(context) return -1 def
showMessage(context: CarRentalContext, message: str) -> None:
context.CarRentalSystem.ShowMessage(message) def logAndShow(context:
CarRentalContext, message: str) -> None:
context.CarRentalSystem.LogAndShowMessage(message)

```





```
add default constructor (without params) and properties: from __future__ import
annotations
from typing import Optional
from entities.Transaction import Transaction
from controller.CarRentalSystem import CarRentalSystem
```

```
class CarRentalContext:
```

```
    def __init__(self, command: str, customer_id: str, car_id: str, car_rental_system:
CarRentalSystem, rental_transaction: Transaction, action_completed: bool):
        self.command: str = command
        self.customer_id: str = customer_id
        self.car_id: str = car_id
        self.car_rental_system: CarRentalSystem = car_rental_system
        self.rental_transaction: Transaction = rental_transaction
        self.action_completed: bool = action_completed
```



Here's the updated code with a default constructor and optional types using Python's **typing** module:

python

Copy code

```
from __future__ import annotations from typing import Optional from
entities.Transaction import Transaction from controller.CarRentalSystem
import CarRentalSystem class CarRentalContext: def __init__( self, command:
Optional[str] = None, customer_id: Optional[str] = None, car_id:
Optional[str] = None, car_rental_system: Optional[CarRentalSystem] = None,
rental_transaction: Optional[Transaction] = None, action_completed:
Optional[bool] = None ): self.command: Optional[str] = command
self.customer_id: Optional[str] = customer_id self.car_id: Optional[str] =
car_id self.car_rental_system: Optional[CarRentalSystem] = car_rental_system
self.rental_transaction: Optional[Transaction] = rental_transaction
self.action_completed: Optional[bool] = action_completed
```

With this code, you can create a **CarRentalContext** instance with the default constructor and optionally set its properties as needed. For example:

python


Copy code

```
# create a CarRentalContext instance with default values context =
CarRentalContext() # set properties as needed context.command = "rent_car"
```

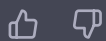
```
context.customer_id = "C001" context.car_id = "CAR123"  
context.car_rental_system = CarRentalSystem()
```

Note that since all the properties are optional, you can also create a **CarRentalContext** instance without setting any properties:

python

 Copy code

```
# create a CarRentalContext instance without setting any properties context =  
CarRentalContext()
```



Represent as Python: public void ReadDataExtended<E>(List<E> targetList,
EntityType entityType, DataType dataType, string fileSuffix, IEntitiesList owner)
 where E : ISerializedExtendedEntity<E>, new()

```
{  
    if (targetList == null) errorHandler.HandleError(new  
ArgumentNullException("targetList"));  
  
    AssignOwner(owner);  
  
    List<E> listItems = ReadDataExtended<E>(entityType, dataType, fileSuffix);  
  
    if (listItems != null)  
    {  
        targetList.Clear();  
        targetList.AddRange(listItems);  
        messageHandler.LogPlusMessage($"Read Data - Rows:{targetList.Count},  
Entity:{entityType}, Data:{dataType} ({fileSuffix}).");  
    }  
    else  
    {  
        messageHandler.LogPlusMessage($"No Data - Entity:{entityType}, Data:  
{dataType}.");  
    }  
}
```



python

 Copy code

```
from typing import TypeVar, List from entities import EntityType, DataType
from entities.serializers import ISerializedExtendedEntity from
entities.error_handler import ErrorHandler from entities.message_handler
import MessageHandler from entities.entities_list import IEntitiesList E =
TypeVar('E', bound=ISerializedExtendedEntity) class DataReader: def
ReadDataExtended(self, targetList: List[E], entityType: EntityType, dataType:
DataType, fileSuffix: str, owner: IEntitiesList) -> None: if targetList is
None: raise ValueError("targetList must not be None.") self.errorHandler =
ErrorHandler() self.messageHandler = MessageHandler() self.AssignOwner(owner)
listItems = self.ReadDataExtended(entityType, dataType, fileSuffix) if
listItems: targetList.clear() targetList.extend(listItems)
self.messageHandler.LogPlusMessage(f"Read Data - Rows:{len(targetList)},
Entity:{entityType}, Data:{dataType} ({fileSuffix}).") else:
self.messageHandler.LogPlusMessage(f"No Data - Entity:{entityType}, Data:
{dataType}.") def AssignOwner(self, owner: IEntitiesList) -> None: self.owner
= owner def ReadDataExtended(self, entityType: EntityType, dataType:
DataType, fileSuffix: str) -> List[E]: # implementation details here pass
```

may need



```

make as read_data_extended_job-method:      public List<E> ReadDataExtended<E>
(EntityType entityType, DataType dataType, string fileSuffix)
    where E : ISerializedExtendedEntity<E>, new()
{
    string filePath = GetFilePath(entityType, dataType, fileSuffix);
    List<E> dataList = new List<E>();

    if (File.Exists(filePath))
        try
        {
            using (StreamReader reader = new StreamReader(filePath))
            {
                string line;
                while ((line = reader.ReadLine()) != null)
                {
                    E data;

                    if (owner == null)
                        data = Deserialize<E>(line, dataType);
                    else
                }
            }
        }
    }
}

```

```

        data = DeserializeExtended<E>(line, dataType, owner);
    }

    if (data != null)
    {
        dataList.Add(data);
    }
}
}
}
catch (Exception ex)
{
    errorHandler.HandleError(ex);
}

return dataList;
}

```



Here's the Python equivalent of the **ReadDataExtended** method:

python

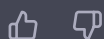
Copy code

```

from typing import List from entities import EntityType, DataType,
ISerializedExtendedEntity from entities.serializers import Deserialize,
DeserializeExtended class DataManager: def read_data_extended_job(self,
entity_type: EntityType, data_type: DataType, file_suffix: str) ->
List[ISerializedExtendedEntity]: file_path = self.get_file_path(entity_type,
data_type, file_suffix) data_list = [] if os.path.exists(file_path): try:
with open(file_path, 'r') as file: for line in file: data = None if
self.owner is None: data = Deserialize(line, data_type) else: data =
DeserializeExtended(line, data_type, self.owner) if data is not None:
data_list.append(data) except Exception as ex:
self.error_handler.handle_error(ex) return data_list

```

Note that I have assumed that this method is part of a larger class called **DataManager**. I have also imported some additional types from modules **entities** and **entities.serializers**. Please adjust these imports based on your project structure.



make default params: def __init__(self, id: str, make: str, model: str, year: int, daily_price: float, rented: bool):

```
self._id = id
self._make = make
self._model = model
self._year = year
self._daily_price = daily_price
self._rented = rented
```

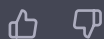


Here's the updated code with default parameters:

python

Copy code

```
class Car: def __init__(self, id: str = "", make: str = "", model: str = "",
year: int = 0, daily_price: float = 0.0, rented: bool = False): self._id = id
self._make = make self._model = model self._year = year self._daily_price =
daily_price self._rented = rented
```



make default params in init-method:

```
def __init__(self, id: str, customer: Customer, car: Car, rental_date: datetime, return_date:
datetime, closed_date: datetime, is_closed: bool):
    self._id = id
    self._customer = customer
    self._car = car
    self._rental_date = rental_date
    self._return_date = return_date
    self._closed_date = closed_date
    self._is_closed = is_closed
    self._owner = None
    self._total_price = self._calculate_total_price()
```



Sorry, I made a mistake in my previous response. Here is the correct implementation:

python

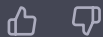
Copy code

```
from entities.Car import Car from entities.Customer import Customer from
datetime import datetime class Transaction: def __init__(self, id: str,
customer: Customer = None, car: Car = None, rental_date: datetime = None,
```

```

return_date: datetime = None, closed_date: datetime = None, is_closed: bool =
False): self._id = id self._customer = customer self._car = car
self._rental_date = rental_date self._return_date = return_date
self._closed_date = closed_date self._is_closed = is_closed self._owner =
None self._total_price = self._calculate_total_price() def
_calculate_total_price(self): if self._rental_date is None or
self._return_date is None or self._car is None: return 0.0 days_rented =
(self._return_date - self._rental_date).days if days_rented <= 0: return 0.0
return days_rented * self._car.daily_price @property def id(self) -> str:
return self._id @property def customer(self) -> Customer: return
self._customer @property def car(self) -> Car: return self._car @property def
rental_date(self) -> datetime: return self._rental_date @property def
return_date(self) -> datetime: return self._return_date @property def
closed_date(self) -> datetime: return self._closed_date @property def
is_closed(self) -> bool: return self._is_closed @property def owner(self):
return self._owner @owner.setter def owner(self, value): self._owner = value
@property def total_price(self) -> float: return self._total_price

```



new uuid as string



Sure! To generate a new UUID as a string in Python, you can use the `uuid` module:

python

Copy code

```
import uuid new_uuid = str(uuid.uuid4()) print(new_uuid)
```

This will generate a new UUID and convert it to a string using the `str()` function. The output will look something like this:

Copy code

```
2d506f8a-7b72-4f17-8b87-f69d8dc7c12c
```



show difference (as Python) from this update: using System;
using System.Linq;
using System.Collections.Generic;

```

namespace sf.systems.rentals.cars
{
    class MainProgram
    {
        static int Main(string[] args)
        {
            // args samples:
            // cmd=register_customer customer="C003,Mary Jung,777 - 1234,911 Main
St,mjung@example.com"
            // cmd=delete_customer customer="C003"
            // cmd=add_car car="CAR911,Porsche,Macant,2023,190.0"
            // cmd=add_car car="CAR11,Audi,A1,2021,90.0"
            // cmd=delete_car car="CAR11"
            // cmd=rent_car car="CAR911" customer="C003"
            // cmd=return_car customer="C001"

            // init
            var testMode = false;
            string argCommand = default;
            string[] argCustomer = default;
            string[] argCar = default;
            string[] argFrom = default;
            string[] argTo = default;

            // resolve args
            if (!testMode)
            {
                foreach (var arg in args)
                {
                    var arg_parts = arg.Split("=");
                    // command-args
                    string argKey = arg_parts[0].ToLowerInvariant().Trim();

                    if (arg_parts.Length > 1)
                    {
                        var argValue = arg_parts[1].Trim();

                        switch (argKey)
                        {
                            case "cmd":
                                argCommand = argValue;
                                break;

```

```

        case "car":
            argCar = argValue.Split(',');
            break;
        case "customer":
            argCustomer = argValue.Split(',');
            break;
        case "from":
            argFrom = argValue.Split(',');
            break;
        case "to":
            argTo = argValue.Split(',');
            break;
    }
}
}
}
else
{
    // Test/Debug mode
    argCommand = "add_cars_test"; // 1)
    // argCommand = "register_customers_test"; // 2)
    // argCommand = "rent_car_test"; // 3)
    // argCommand = "return_car_test"; // 4)

    argCustomer = new string[] { "C002" };
    argCar = new string[] { "CAR1" };
}

// Initialize car rental system
var carRentalSystem = new CarRentalSystem();

// create rental context
var context = new CarRentalContext();
context.CarRentalSystem = carRentalSystem;
context.Command = argCommand;
context.CarID = argCar?[0];
context.CustomerID = argCustomer?[0];
context.RentFrom = parseDate(argFrom[0]);
context.RentTo = parseDate(argTo[0]);

// Go!
try
{

```



```

CarRentalCommands.LoadData(context);

// run command
switch (argCommand)
{
    // production commands
    case "add_car":
        if (argCar == null || argCar.Length != 5)
        {
            logAndShow(context, "\nERROR: Invalid car data provided! Usage:
cmd=add_car car=\"ID,Make,Model,Year,PricePerDay\"");
            return -1;
        }
        CarRentalCommands.AddCar(context, argCar);
        break;
    case "delete_car":
        if (argCar == null || argCar.Length != 1)
        {
            logAndShow(context, "\nERROR: Invalid car ID provided! Usage:
cmd=delete_car car=\"ID\"");
            return -1;
        }
        CarRentalCommands.DeleteCar(context, argCar[0]);
        break;
    case "register_customer":
        if (argCustomer == null || argCustomer.Length != 5)
        {
            logAndShow(context, "\nERROR: Invalid customer data provided!
Usage: cmd=register_customer customer=\"ID,Name,PhoneNumber,Address,Email\"");
            return -1;
        }
        CarRentalCommands.RegisterCustomer(context, argCustomer);
        break;
    case "delete_customer":
        if (argCustomer == null || argCustomer.Length != 1)
        {
            logAndShow(context, "\nERROR: Invalid customer ID provided! Usage:
cmd=delete_customer customer=\"ID\"");
            return -1;
        }
        CarRentalCommands.DeleteCustomer(context, argCustomer[0]);
        break;
    case "rent_car":

```

```

        if (argCar == null || argCar.Length != 1 || argCustomer == null ||
argCustomer.Length != 1)
        {
            logAndShow(context, "\nERROR: Invalid car or customer ID provided!
Usage: cmd=rent_car car=\"ID\" customer=\"ID\"");
            return -1;
        }
        context = CarRentalCommands.RentCar(context);
        break;
    case "return_car":
        if (argCustomer == null || argCustomer.Length != 1)
        {
            logAndShow(context, "\nERROR: Invalid customer ID provided! Usage:
cmd=return_car customer=\"ID\"");
            return -1;
        }
        context = CarRentalCommands.ReturnCar(context);
        break;

    // test commands
    case "add_cars_test":
        addCarsTest(context);
        break;
    case "register_customers_test":
        registerCustomersTest(context);
        break;
    case "rent_car_test":
        context = CarRentalCommands.RentCar(context);
        break;
    case "return_car_test":
        context = CarRentalCommands.ReturnCar(context);
        break;
    default:
        logAndShow(context, $"Unknown command: \"{argCommand}\"!\n");
        showMessage(context, "Use samples:");
        showMessage(context, "CarRentalSystem cmd=register_customer
customer=\"C003,Mary Jung,777 - 1234,911 Main St,mjung@example.com\"");
        showMessage(context, "CarRentalSystem cmd=delete_customer
customer=\"C003\"");
        showMessage(context, "CarRentalSystem cmd=add_car
car=\"CAR911,Porsche,Macant,2023,190.0\"");
        showMessage(context, "CarRentalSystem cmd=add_car
car=\"CAR11,Audi,A1,2021,90.0\"");

```

```

        showMessage(context, "CarRentalSystem cmd=delete_car
car=\"CAR11\");
        showMessage(context, "CarRentalSystem cmd=rent_car car=\"CAR911\"
customer=\"C003\" from=2023-04-09 to=2023-04-12");
        showMessage(context, "CarRentalSystem cmd=return_car
car=\"CAR911\");
        return -1;
    }

    CarRentalCommands.SaveData(context);
    return 0;
}
catch (Exception ex)
{
    logAndShow(context, $"\\nERROR: {ex.Message} \\n\\nSTACK-TRACE:
{ex.StackTrace}");
    CarRentalCommands.SaveData(context);

    return -1;
}
}

private static DateTime parseDate(string strDate)
{
    DateTime resultDate = default;
    if (!DateTime.TryParse(strDate, out resultDate))
        resultDate = DateTime.Now;
    return resultDate;
}

private static void showMessage(CarRentalContext context, string message)
{
    context.CarRentalSystem.ShowMessage(message);
}

private static void logAndShow(CarRentalContext context, string message)
{
    context.CarRentalSystem.LogAndShowMessage(message);
}

private static void addCarsTest(CarRentalContext context)
{
    context.CarRentalSystem.AddCar("CAR1", "Toyota", "Corolla", 2022, 50.0);
}

```

```

        context.CarRentalSystem.AddCar("CAR2", "Honda", "Civic", 2021, 45.0);
        context.CarRentalSystem.AddCar("CAR7", "Audi", "Q7", 2020, 120.0);
    }

    private static void registerCustomersTest(CarRentalContext context)
    {
        // Register a new customer
        context.CarRentalSystem.RegisterCustomer("C001", "John Smith", "555-1234",
"123 Main St", "jsmith@example.com");

        // Register another customer
        context.CarRentalSystem.RegisterCustomer("C002", "Jane Doe", "555-5678", "456
Elm St", "jdoe@example.com");
    }
}

```



The Python equivalent of the updated code would look like:

python

Copy code

```

import argparse from datetime import datetime from controller.CarRentalSystem
import CarRentalSystem from controller.CarRentalCommands import
CarRentalCommands from context.CarRentalContext import CarRentalContext def
main(args): # Initialize car rental system carRentalSystem =
CarRentalSystem() # create rental context context = CarRentalContext() # Go!
try: CarRentalCommands.load_data(context) # run command if args.command ==
"add_car": if not args.car or len(args.car) != 5:
carRentalSystem.log_and_show_message("\nERROR: Invalid car data provided!
Usage: cmd=add_car car=\"ID,Make,Model,Year,PricePerDay\") return -1
CarRentalCommands.add_car(context, args.car) elif args.command ==
"delete_car": if not args.car or len(args.car) != 1:
carRentalSystem.log_and_show_message("\nERROR: Invalid car ID provided!
Usage: cmd=delete_car car=\"ID\") return -1
CarRentalCommands.delete_car(context, args.car[0]) elif args.command ==
"register_customer": if not args.customer or len(args.customer) != 5:
carRentalSystem.log_and_show_message("\nERROR: Invalid customer data
provided! Usage: cmd=register_customer
customer=\"ID,Name,PhoneNumber,Address,Email\") return -1
CarRentalCommands.register_customer(context, args.customer) elif args.command
== "delete_customer": if not args.customer or len(args.customer) != 1:
carRentalSystem.log_and_show_message("\nERROR: Invalid customer ID provided!
Usage: cmd=delete_customer customer=\"ID\") return -1
CarRentalCommands.delete_customer(context, args.customer[0]) elif
args.command == "rent_car": if not args.car or len(args.car) != 1 or not
args.customer or len(args.customer) != 1:
carRentalSystem.log_and_show_message("\nERROR: Invalid car or customer ID
provided! Usage: cmd=rent_car car=\"ID\" customer=\"ID\") return -1 context

```

```

= CarRentalCommands.rent_car(context, args.rent_from, args.rent_to) elif
args.command == "return_car": if not args.customer or len(args.customer) !=
1: carRentalSystem.log_and_show_message("\nERROR: Invalid customer ID
provided! Usage: cmd=return_car customer=\"ID\"") return -1 context =
CarRentalCommands.return_car(context) else:
carRentalSystem.log_and_show_message(f"\nUnknown command: \"
{args.command}\"!\n") carRentalSystem.show_message("Use samples:")
carRentalSystem.show_message("CarRentalSystem cmd=register_customer
customer=\"C003,Mary Jung,777 - 1234,911 Main St,mjung@example.com\"")
carRentalSystem.show_message("CarRentalSystem cmd=delete_customer
customer=\"C003\"") carRentalSystem.show_message("CarRentalSystem cmd=add_car
car=\"CAR911,Porsche,Macant,2023,190.0\"")
carRentalSystem.show_message("CarRentalSystem cmd=add_car
car=\"CAR11,Audi,A1,2021,90.0\"")
carRentalSystem.show_message("CarRentalSystem cmd=delete_car car=\"CAR11\"")
carRentalSystem

```



continue after " context = CarRentalCommands.return_car(context)"



After calling the **return_car** command on the **CarRentalCommands** object with the **context** object as its argument, the updated **context** object is returned and stored in the **context** variable.

The code then proceeds to save the updated data by calling the **SaveData** method on the **CarRentalCommands** object with the **context** object as its argument.

Finally, the **Main** method returns **0** to indicate a successful execution of the program.

Here's the updated code:

python

Copy code

```

# Go! try: CarRentalCommands.LoadData(context) # run command switch
(argCommand) # ... rest of switch statement code ... case "return_car": if
(argCustomer == null || argCustomer.Length != 1) { logAndShow(context,
"\nERROR: Invalid customer ID provided! Usage: cmd=return_car
customer=\"ID\""); return -1; } context =
CarRentalCommands.ReturnCar(context) CarRentalCommands.SaveData(context)
break; return 0 except Exception as ex: logAndShow(context, f"\nERROR: {ex}

```

```
\n\nSTACK-TRACE: {traceback.format_exc()}")
```

```
CarRentalCommands.SaveData(context) return -1
```

represent as Python code:

```
private static DateTime parseDate(string strDate)
{
    DateTime resultDate = default;
    if (!DateTime.TryParse(strDate, out resultDate))
        resultDate = DateTime.Now;
    return resultDate;
}
```



python

Copy code

```
def parseDate(strDate: str) -> datetime: resultDate: datetime =
datetime.now() try: resultDate = datetime.strptime(strDate, '%Y-%m-%d')
except ValueError: pass return resultDate
```



update Transaction-class (CreateFromCsv only):

```
private static Transaction CreateFromCsv(string csv,
    IEntitiesList entitiesList)
{
    string[] values = csv.Split(',');
    if (values.Length < 7)
    {
        throw new ArgumentException($"Invalid CSV data: {csv}");
    }
    string id = values[0];

    Customer customer;
    Car car;

    string customerId = values[1];
    string carId = values[2];

    if (entitiesList == null)
    {
        customer = DefaultCustomer(customerId);
```

```

        car = DefaultCar(carId);
    }
    else
    {
        customer = entitiesList.LookupCustomer(customerId);
        if (customer == null)
            customer = DefaultCustomer(customerId);
        else
            customer.RentedCarsPoolNew(entitiesList.GetRentedCars(customer));

        car = entitiesList.LookupCar(carId);
        if (car == null) car = DefaultCar(carId);
    }

    DateTime rentalDate = DateTime.Parse(values[3]);
    DateTime returnDate = DateTime.Parse(values[4]);
    DateTime closedDate = DateTime.Parse(values[5]);
    bool isClosed = bool.Parse(values[6]);

    return new Transaction(id, customer, car, rentalDate, returnDate, closedDate,
isClosed);

}

```



Here's the updated **Transaction** class with the modified **CreateFromCsv** method:

python

Copy code

```

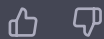
from typing import Optional from datetime import datetime from
entities.Customer import Customer from entities.Car import Car class
Transaction: def __init__(self, id: str, customer: Customer, car: Car,
rental_date: datetime, return_date: datetime, closed_date: datetime,
is_closed: bool): self._id = id self._customer = customer self._car = car
self._rental_date = rental_date self._return_date = return_date
self._closed_date = closed_date self._is_closed = is_closed self._owner =
None self._total_price = self._calculate_total_price() @staticmethod def
CreateFromCsv(csv: str, entitiesList=None) -> Optional['Transaction']: values
= csv.split(',') if len(values) < 7: raise ValueError(f"Invalid CSV data:
{csv}") id = values[0] customer: Customer car: Car customerId = values[1]
carId = values[2] if entitiesList is None: customer =
Customer.DefaultCustomer(customerId) car = Car.DefaultCar(carId) else:
customer = entitiesList.LookupCustomer(customerId) if customer is None:
customer = Customer.DefaultCustomer(customerId) else:

```

```

customer.RentedCarsPoolNew(entitiesList.GetRentedCars(customer)) car =
entitiesList.LookupCar(carId) if car is None: car = Car.DefaultCar(carId)
rentalDate = datetime.fromisoformat(values[3]) returnDate =
datetime.fromisoformat(values[4]) closedDate =
datetime.fromisoformat(values[5]) isClosed = bool(values[6]) return
Transaction(id, customer, car, rentalDate, returnDate, closedDate, isClosed)
def _calculate_total_price(self) -> float: if self._is_closed: days =
(self._closed_date - self._rental_date).days return days *
self._car.daily_price else: return 0.0 @property def id(self): return
self._id @property def customer(self): return self._customer @property def
car(self): return self._car @property def rental_date(self

```



describe Python implementation for a GitHub README file



Sure, I can help you with that!

To begin with, Python is an interpreted, high-level programming language that is widely used in various domains such as web development, scientific computing, and data analysis. In this GitHub repository, we are implementing a Car Rental System using Python.

The repository consists of multiple Python files, each responsible for handling a specific part of the system. The main file, **main.py**, serves as the entry point for the application and provides a command-line interface for interacting with the system.

The system is built using object-oriented programming principles. We have defined several classes such as **Car**, **Customer**, and **Transaction**, each representing an entity in the system. These classes have various attributes and methods that define their behavior and enable them to interact with other entities in the system.

The **CarRentalSystem** class serves as the central component of the system and provides methods for performing various actions such as adding cars, registering customers, renting cars, and returning cars. These methods make use of other classes such as **Car**, **Customer**, and **Transaction** to perform their tasks.

We have also implemented a **CarRentalContext** class that represents the context in which the system operates. This class stores information such as the current command,

customer ID, car ID, and rental transaction. This information is used by various methods in the system to perform their tasks.

Finally, we have added unit tests to ensure that the system is working as expected. These tests cover various scenarios such as adding cars, registering customers, renting cars, and returning cars.

Overall, this Car Rental System implementation in Python demonstrates the use of object-oriented programming principles to build a complex system. It also highlights the importance of testing in ensuring that the system is working correctly.