Cardiff School of Computer Science and Informatics

Coursework Assessment Pro-forma

Module Code: CMT202
Module Title: Distributed and Cloud Computing
Lecturer: Padraig Corcoran
Assessment Title: CMT 202 Coursework
Date Set: Monday 7 March 2022
Submission Date and Time: Monday 25 April 2022 at 9:30AM
Return Date: by Monday 23 May 2022

This assignment is worth 50% of the total marks available for this module. If coursework is submitted late (and where there are no extenuating circumstances):

1. If the assessment is submitted no later than 24 hours after the deadline, the mark for the assessment will be capped at the minimum pass mark;
2. If the assessment is submitted more than 24 hours after the deadline, a mark of 0 will be given for the assessment.

Your submission must include the official Coursework Submission Cover sheet, which can be found here:

https://docs.cs.cf.ac.uk/downloads/coursework/Coversheet.pdf

Submission Instructions

All submission should be via Learning Central unless agreed in advance with the Director of Teaching.

Description		Type	Name
Cover sheet	**Compulsory**	One PDF (.pdf) file	[student number].pdf
Solutions	**Compulsory**	One zip (.zip) file containing the Python code developed.	[student number].zip

The only additional software libraries which can be used are pyro5 and those in the Python standard library. For example, you can use datetime but not pandas and other database systems.

Any deviation from the submission instructions above (including the number and types of files submitted) may result in a mark of zero for the assessment or question part.

Staff reserve the right to invite students to a meeting to discuss coursework submissions.

Assignment

You have been hired by a car rental company to build a distributed data storage system using a remote object paradigm that will allow one to store and access information relating to rental cars, manufacturers of cars and users who rent cars.

Each manufacturer has the following two associated pieces of information which should be stored in the system:
1. Manufacturer name.
2. Manufacturer country which refers to the country in which the manufacturer is based (e.g. BMW is based in Germany).

Each rental car has the following two associated pieces of information which should be stored in the system:
1. Car manufacturer name.
2. Car model.

Note, multiple cars with the same manufacturer and model can exist in the system. You can assume that no two manufacturers will have a car model with the same name.

Each user has the following two associated pieces of information which should be stored in the system:
1. User name.
2. User contact phone number.

Design and implement the above distributed data storage system using a remote object paradigm which allows employees of the car rental company to perform the following twelve tasks:

Task 1
Add a user to the system. Implement this using a method with the following header:
def add_user(self, user_name, user_number)

An example of calling this method is:
rental_object.add_user("Allen Hatcher", 123456)

You can assume that each user added to the system has a unique user name.

Task 2
Return all associated pieces of information relating to the set of users currently stored in the system (i.e. a set of user names and contact numbers). Implement this using a method with the following header:
def return_users(self)

An example of calling this method is:
rental_object.return_users()

The information returned by this method should have the property that it can be easily interpreted when displayed using the Python print function. That is, the output from the following print function should be easily interpreted:
print(rental_object.return_users())

Task 3
Add a car manufacturer to the system. Implement this using a method with the following header:
def add_manufacturer(self, manufacturer_name, manufacturer_country)

An example of calling this method is:
rental_object.add_manufacturer(“BMW”, “Germany”)

Task 4
Return all associated pieces of information relating to the set of manufacturers currently stored in the system (i.e. a set of manufacturer names and countries). Implement this using a method with the following header:
def return_manufacturers(self)

An example of calling this method is:
rental_object.return_manufacturers()

The information returned by this method should have the property that it can be easily interpreted when displayed using the Python print function.

Task 5
Add a rental car to the system. Implement this using a method with the following header:
def add_rental_car(self, manufacturer_name, car_model)

An example of calling this method is:
rental_object.add_rental_car(“BMW”, “3 Series”)

When a rental is first added to the system it is initially not rented by any user. Multiple cars with the same manufacturer and model may be added to the system. You can assume that no two manufacturers will have a car model with the same name.

Task 6
Return all associated pieces of information relating to the set of rental cars currently **not rented** (i.e. a set of car manufacturers and models). Implement this using a method with the following header:
def return_cars_not_rented(self)

An example of calling this method is:
rental_object.return_cars_not_rented()

The information returned by this method should have the property that it can be easily interpreted when displayed using the Python print function.

Task 7
Rent a car of a specified model to a specified user on a specified date. Implement this using a method with the following header:
def rent_car(self, user_name, car_model, year, month, day)

An example of calling this method is:
rental_object.rent_car("Conor Reilly", "3 Series", 2019, 1, 3)

Each rental car can only be rented to a single user at a time. For example, consider the case where there are two rental cars in the system with model equal to "3 Series" and both are currently rented. In this case another car with model equal to "3 Series" cannot be rented until one of the two above cars is returned or an additional car with model equal to "3 Series" is added to the system.

The method rent_car should return a value of 1 if the car in question was successfully rented. Otherwise, the method should return a value of 0.

Task 8
Return all associated pieces of information relating to the set of cars **currently rented** (i.e. a set of car manufacturers and models). Implement this using a method with the following header:
def return_cars_rented(self)

An example of calling this method is:
rental_object.return_cars_rented()

The information returned by this method should have the property that it can be easily interpreted when displayed using the Python print function.

Task 9
Return a rented car of a specified model by a specified user on a specified date; that is, change the status of the car in question from rented to not rented. Implement this using a method with the following header:
def end_rental(self, user_name, car_model, year, month, day)

An example of calling this method is:
rental_object.end_rental("Conor Reilly", "3 Series", 2019, 2, 4)

You can assume that each user will only rent a single car of each model at any given time. Therefore, when a user returns a car of a specified model there is no confusion with respect to which car they are returning.

Task 10

Delete from the system all rental cars of a specified model which are currently not rented. Rental cars which are currently rented should not be deleted. Implement this using a method with the following header:
def delete_car(self, car_model)

An example of calling this method is:
rental_object.delete_car(“3 Series”)

Task 11
Delete from the system a specified user. A user should only be deleted if they currently do not rent and have never rented a car. Implement this using a method with the following header:
def delete_user(self, user_name)

An example of calling this method is:
rental_object.delete_user(“Conor Reilly”)

The method delete_user should return a value of 1 if the user in question was successfully deleted. Otherwise, the method should return a value of 0.

Task 12
Return all car models a user previously rented where the corresponding rental and return dates both lie between a specified start and end date inclusive.
Implement this using a method with the following header:
def user_rental_date(self, user_name, start_year, start_month, start_day, end_year, end_month, end_day)

An example of calling this method is:
rental_object.user_rental_date(“Conor Reilly”, 2010, 1, 1, 2029, 2, 1)

Note, the car models returned may contain duplicates if the user rented the model in question more than once.

The information returned by this method should have the property that it can be easily interpreted when displayed using the Python print function.

In your solution to the above assignment, the class in question should be called rental. That is, when defining the class use the expression “class rental(object):”. Also, the class must be contained in a file entitled rental.py.

In the file rental.py you should create an object of type rental and register this object with the name server using the name example.rental. That is, the file rental.py should contain the following code snippet:

```
daemon = Daemon()
serve({rental: "example.rental"}, daemon=daemon, use_ns=True)
```

I have provided a Python file entitled rental_test.py to help you test your code and ensure all your methods have correct headers. To run this file, first run the name server in one command prompt, the file rental.py in a second command prompt and the file rental_test.py in a third command prompt.

Note that, to successfully complete all tasks, you are only required to store data while your code is running; there is no requirement to write data to an external database or file.

Learning Outcomes Assessed

The following learning outcomes from the module description are specifically being assessed in this assignment.
Demonstrate and apply knowledge about the state-of-the-art in distributed-systems architectures.
Appreciate the difference between various distributed computing middleware and their communication mechanisms.

Criteria for assessment

Marks will be awarded based on successful system implementation as follows:

Successfully implement task 1 specified above.	[4 marks]
Successfully implement task 2 specified above.	[4 marks]
Successfully implement task 3 specified above.	[4 marks]
Successfully implement task 4 specified above.	[4 marks]
Successfully implement task 5 specified above.	[4 marks]
Successfully implement task 6 specified above.	[4 marks]
Successfully implement task 7 specified above.	[4 marks]
Successfully implement task 8 specified above.	[4 marks]
Successfully implement task 9 specified above.	[4 marks]
Successfully implement task 10 specified above.	[4 marks]
Successfully implement task 11 specified above.	[5 marks]
Successfully implement task 12 specified above.	[5 marks]

Feedback on your performance will address each of these criteria.

A student can expect to receive a distinction (70-100%) if they correctly implement all tasks.
A student can expect to receive a merit (60-69%) if they correctly implement most tasks without major errors.
A student can expect to receive a pass (50-59%) if they correctly implement some tasks without major errors.
A student can expect to receive a fail (0-50%) if they fail to correctly implement some tasks without major errors.

IMPORTANT – All code submitted must be written in Python 3 and use the pyro5 library to implement remote objects. The only additional software libraries which can be used are pyro5 and those in the Python standard library. For example, you can use datetime but not pandas and other database systems.

Feedback and suggestion for future learning

Feedback on your coursework will address the above criteria. Feedback and marks will be returned on Monday 23 May 2022 via Learning Central. Where requested, this will be supplemented with oral feedback.