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: CMT202 Coursework
Assessment Number: 1
Date Set: Thursday 7 March 2024.
Submission Date and Time: by Thursday 25 April 2024 at 9:30am.
Feedback return date: Thursday 23 May 2024.

If you have been granted an extension for Extenuating Circumstances, then the submission deadline and return date will be later than that stated above. You will be advised of your revised submission deadline when/if your extension is approved.

If you defer an Autumn or Spring semester assessment, you may fail a module and have to resit the failed or deferred components.

If you have been granted a deferral for Extenuating Circumstances, then you will be assessed in the next scheduled assessment period in which assessment for this module is carried out.

If you have deferred an Autumn or Spring assessment and are eligible to undertake summer resits, you will complete the deferred assessment in the summer resit period.

If you are required to repeat the year or have deferred an assessment in the resit period, you will complete the assessment in the next academic year.

As a general rule, students can only resit 60 failed credits in the summer assessment period (see section 3.4 of the academic regulations). Those with more than 60 failed credits (and no more than 100 credits for undergraduate programmes and 105 credits for postgraduate programmes) will be required to repeat the year. There are some exceptions to this rule and they are applied on a case-by-case basis at the exam board.

If you are an MSc student, please note that deferring assessments may impact the start date of your dissertation. This is because you must pass all taught modules before you can begin your dissertation. If you are an overseas student, any delay may have consequences for your visa, especially if it is your intention to apply for a post study work visa after the completion of your programme.

NOTE: The summer resit period is short and support from staff will be minimal. Therefore, if the number of assessments is high, this can be an intense period of work.

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.

Extensions to the coursework submission date can ***only*** be requested using the Extenuating Circumstances procedure. Only students with ***approved*** extenuating circumstances may use the extenuating circumstances submission deadline. Any coursework submitted after the initial submission deadline without *approved* extenuating circumstances will be treated as late.

More information on the extenuating circumstances procedure and academic regulations can be found on the Student Intranet:

https://intranet.cardiff.ac.uk/students/study/exams-and-assessment/extenuating-circumstances
https://intranet.cardiff.ac.uk/students/study/your-rights-and-responsibilities/academic-regulations

[1] https://intranet.cardiff.ac.uk/students/study/exams-and-assessment/academic-integrity/cheating-and-academic-misconduct

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. This is a string data type.
2. Manufacturer country which refers to the country in which the manufacturer is based (e.g. BMW is based in Germany). This is a string data type.

Each rental car has the following two associated pieces of information which should be stored in the system:
1. Car manufacturer name. This is a string data type.
2. Car model. This is a string data type.
Note, multiple cars with the same manufacturer and model can exist in the system.

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

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("Conor Reilly", "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 collection of users currently stored in the system (i.e. a collection 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 and understood by a human reader:
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”)

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

Task 4
Return all associated pieces of information relating to the collection of manufacturers currently stored in the system (i.e. a collection 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”)

You can assume that the manufacturer in question has previously been added using the add_manufacturer method.

When a car is first added to the system it is initially not rented to any user. Multiple cars with the same manufacturer and model may be added to the system.

Task 6
Return all associated pieces of information relating to the collection of rental cars currently **not rented** (i.e. a collection 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)

You can assume that the user in question has previously been added using the add_user method. You can also assume that no two manufacturers will have a car with the same model name. Therefore, there will never exist any ambiguity regarding which car is being referred to.

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 year, month, day parameters are integer values greater than zero, integers in the interval (0, 12] and integers in the interval (0, 31] respectively.

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 collection of cars **currently rented** (i.e. a collection 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 to the rental company, a rented car of a specified model by a specified user on a specified date; that is, set the status of the rental 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 the user has previously rented the car in question and this method call corresponds to them returning that car. You can also assume that a user will only rent a single car of each model at any given time. Therefore, there will never exist any ambiguity regarding which car is being returned.

The year, month, day parameters are integer values greater than zero, integers in the interval (0, 12] and integers in the interval (0, 31] respectively.

Task 10
Delete from the system all rental cars of a specified model which are currently not rented. Rental cars that 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 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")

You can assume that the user in question has previously been added using the add_user method and has not previously been deleted.

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

Task 12
Return all rental cars (i.e. a collection of car manufacturers and models) a user previously has 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 rental cars returned may contain duplicates if the user rented the car in question more than once.

You can assume that the user in question has previously been added using the add_user method.

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 solutions to the above 12 tasks, 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.

A Python file different to rental_test.py will be used to evaluate your solution. You can assume that all method calls in this file will have valid parameter values.

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.

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.

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 where the maximum possible mark is 50:

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 with only very minor errors. This will be demonstrated by returning the correct solutions for a set of test cases. Where human-interpretable output is required, its quality is excellent.

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 (0-49%) if they fail to correctly implement any tasks without major errors.

Feedback and suggestion for future learning

Feedback on your coursework will address the above criteria. Feedback and marks will be returned on Thursday 23 May 2024 via via LearningCentral.

Submission Instructions

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

Submission should be made via LearningCentral.

Any code submitted will be run on a system equivalent to a University provided Windows laptop and must be submitted as stipulated in the instructions above.

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.

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

---

Support for assessment

Questions about the assessment can be asked on the LearningCentral discussion board, during the lecturer office hours, or at the beginning of the lectures.

Support for the programming elements of the assessment will be available *in the daily drop-in lab sessions.*