**Munster Technological University – Kerry Campus**

**Ord/Hons BSc. in Computing with Specialism - Year 2**

**Continuous Assessment #2**

**Date: 12/11/24**

**Time: 4 – 6 p.m.**

**Object Oriented Programming**

**Instructions:** You must close all OOP module materials that you might have access to, before attempting the following question. You may refer to the file **JavaSpellChecker.docx** during the CA though.

Launch IntelliJ and create a new project called **OOPCA2\_*FirstNameLastName***. When you are finished coding, use **Git** to version control your project and then zip it and upload it to the **exam drop-box** within Canvas. Make a copy of your project to OneDrive and leave it unaltered in case I need to access it later.

Please do not leave the lab until I have let you know that I have received your code successfully.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Q1. (100 marks)**

Create a package called **assessment** within your project to store all the Java files you will create.

Consider the UML VOPC diagram below and use it to write the code for the **Owner**, **Pet**, **Appointment** , **VetShop** and **TestVetShop** classes. To speed things up for yourself, be sure to use the **code generation tool** within IntelliJ and aim to **maximise software reuse** in your solution (your constructors should call the mutators and your toString() methods should call the relevant accessors). Also note that **you will have to modify the toString() methods** also to achieve the **desired output format** shown in the screenshots.

A screenshot of a computer

Description automatically generated

A screenshot of a computer code

Description automatically generated

The **Owner** class is a “standard” instantiable class with no additional functionality.

You should modify the **setName**() method of the Owner class so that if an attempt is made to set the name of the owner to null or empty string, then it will be set to the value “No name specified” instead.

The **Pet** class is another “standard” instantiable class with no additional functionality. Note that its constructor **only takes 5 arguments**, as the ID attribute will be set internally, using the static attribute count. So the first pet will end up with the ID value 100001, the second with ID value 100002 etc.

You should modify the **setOwner**() method of the Pet class to ensure that the Owner object it owns cannot be tampered with from outside the class.

The **Appointment** class is yet another “standard” instantiable class with no additional functionality. Note that its constructor **only takes 4 arguments**, as the counter attribute is to be used simply to keep track of the total number of Appointment objects created and the status attribute will be given a default value of ‘A’ here to indicate that, to begin with at least, all Appointment objects have an “active” status. If an appointment were to be cancelled then the status would be changed to ‘C’ to indicate this, but I won’t get you to do that for this CA, so the setStatus() method will actually go unused here.

The **VetShop** constructor will only take 3 arguments in this case. Its three array attributes will be given values directly within the constructor body here. They should be given sizes of 20, 500 and 2000 respectively for the vets, pets and appointments arrays.

The **addVet**() method of the **VetShop** class will allow a vet to be added to the vets array. In this case, you simply want to add this vet’s name in the next available slot within the array i.e. the next slot of the vets array that has the value **null**. You will be **looping** to find this slot. The method will return nothing.

The **addPet**() method of the **VetShop** class will allow a Pet object to be added to the pets array. In this case, you can simply use the pet’s ID value to ensure that the Pet object gets added to the correct slot of the pets array. The method will return nothing.

Note that the **toString**() method of the VetShop class can be written to use the **Arrays.toString()** method for its vets attribute. However, you will need **loops** and **null testing** to display the state of the pets and appointments attributes.

A method named **getBookedVets**() should be written which will act as a helper method to the method makeAppointment() to be written later. This method will take as arguments a String representing the date and an integer representing the hour of an appointment. The goal is to check the appointments array to discover what vets are booked for that particular date and time. Any vets that are booked will be added to a String array called **bookedVets** that you will create within the method. This array will have the same size as the vets array. The method will return a reference to the bookedVets array.

A method named **getAvailableVets**() should be written **after** getBookedVets() and will act as a helper method to the method makeAppointment() to be written later. This method will take as an argument the bookedVets array returned by the previous method. The goal is to loop through the vets array to discover what vets are available (for the date and time an appointment is being scheduled for). A **linearSearch**() method can be used within this loop to see whether a particular vet is found in the bookedVets array. If the vet is found, we know they are booked, but if they are not found then they are available for an appointment and so can be added to a String array called **availableVets** that you will create within the method. This array will have the same size as the vets array. The method will return a reference to the availableVets array.

Note that there is a certain amount of similarity between getBookedVets() and getAvailableVets() so you should be able to use getBookedVets() as a basis for the coding of getAvailableVets().

The **linearSearch**() method used by getAvailableVets() will take two arguments, the bookedVets array and a String argument representing a single vet. It will determine whether the vet exists in the bookedVets array. If it does, it will return true, otherwise it will return false.

To **test out** the three methods above, you can **make them public temporarily**, then call the methods on the VetShop object you will have created in the driver, passing in a set of test values for the arguments and displaying the values returned to the console using **Arrays.toString()**. As no Appointment objects currently exist, then the bookedVets array returned by getBookedVets() should just contain all null values and when this is passed to the getAvailableVets() method, it should return an array containing the vets "Justin Flaherty" and "Bridget Mulcahy" and the rest of the slots should be null. Of course, you should have called the addVet() method twice **before** these tests in the driver to add the two vets to the vets array.

A method named **makeAppointment**() should be written which will allow an appointment to be made for a particular date and time, for a particular pet. This method will take as arguments a String representing the particular date, an integer representing the hour of the appointment, and a Pet object representing the pet the appointment is being made for. The method will operate as follows:

* Call **getBookedVets**() to get an array of all the vets that are booked for the particular date/time of the requested appointment
* Call **getAvailableVets**() to get an array of all the vets that are available for the particular date/time of the requested appointment – this method will take the array of bookedVets as an argument and use the **linearSearch**() method to determine the available vets.
* Loop through the availableVets array to get a count of the total number of vets available at the date/time requested.
* If it turns out that the count is greater than zero, then it means there is at least one vet available. Now generate a random number based on this count value to select a random vet from the group of available vets. Then create the **Appointment** object using the randomly selected vet and add it to the appointments array. If it turns out that the count is zero, then it means that no vet is available for that date/time and simply issue a “Sorry, no vets are available at this time” message to the user.

The TestVetShop class will act as a **driver** for the VetShop, Appointment , Pet and Owner classes and will do the following:

* Create two Owner objects that have the following data (**copy/paste** into IntelliJ to save time):

"Timmy Clifford","72 Spa Rd., Tralee, Co Kerry","085-2891234"

"Mary O Connor","32 Park Rd., Killarney, Co Kerry","083-1231256"

* Create three Pet objects that use the above objects and have the following data (again copy/paste to save time):

"Fido","Dog","10-01-2020","16-08-2019", owner1

"Rascal","Cat","21-03-2017","01-03-2016", owner2

"Sammy","Snake","02-11-2022","14-05-2022", owner2

note that the last piece of data in each line above is the name of the reference I used

myself for the two Owner objects

* Create a VetShop object now with the following data

"McGrath's Vet Centre","Rock St., Tralee, Co. Kerry", "087-5342571"

* Display the initial state of the VetShop object – it should produce the following output – you need to code the toString() method to achieve this format. Nothing is displayed for the pets or appointments as both arrays just contain null values at this point.

A close-up of a computer screen

Description automatically generated

* Now call the **addPet**() method on the VetShop object for each of the three Pet objects created earlier.
* Next call the **addVet**() method on the VetShop object twice to add two vets to the vets array and you can use the names "Justin Flaherty" and "Bridget Mulcahy"
* Next call the **makeAppointment**() method on the VetShop object twice, using each of the sets of data below:

"24-02-2020",15,pet1

"24-02-2020",15,pet2

This makes appointments for two different pets for the same date and time. As there are two vets in the vets array at this point, it means that the Appointment objects will be created successfully and a random vet will be assigned to each appointment.

* Next the updated state of the VetShop object should be displayed – and it should produce the following output.

A close up of a text

Description automatically generated

Note that, as the **vet is** **assigned randomly** for each appointment, you may see Justin Flaherty assigned for the first appointment in the list and Bridget Mulcahy for the second appointment.

* Finally, another call should be made to **makeAppointment**() to attempt to make another appointment based on the following data:

"24-02-2020",15,pet3

Note that we have only two vets in the vets array and both are already booked for appointments at this exact date and time. Therefore, there will be no vet available to take this additional appointment and the user will be given an appropriate message as follows:

