CSC8014 Assessed Coursework
Shelter Management System

The deadline for this coursework is 14:30 on **22nd Feb 2024 and must be submitted on NESS.**

The coursework is marked out of **100** and accounts for **100%** of the module mark for CSC8014. All work will be checked for plagiarism. For further information, please see: **https://www.ncl.ac.uk/academic-skills-kit/good-academic-practice/plagiarism/**

---

1. Aim

The aim of this coursework is for you to practice the design and good practice principles covered in lectures. You will develop interfaces and classes to demonstrate that you have learned and understood the module material, including:

- appropriate overriding of `Object` class methods, including overriding `toString` and providing a static `valueOf` method when appropriate
- design of interface-based hierarchies, programming through interfaces, casting and late binding
- the use of factories to control instantiation of objects, including how to guarantee the instantiation of unique instances
- the use of defensive programming, including use of immutability, appropriate error handling, validation of input and using appropriate accessors and modifiers on instance variables, methods, and classes.
- the use of appropriate interfaces and classes from the Collections framework
- appropriate use of Javadocs to document your interfaces and classes
- the use of testing

The coursework is **not** algorithmically challenging. The focus is on good design and good practice.

The coursework is **not** about development of an end-user application. You are developing interfaces and classes that could be used for the development of an application. You should **not** develop a graphical user interface or a command line interface. They are not necessary, and you will be given no credit for doing so.

Note the Shelter management system specified below is a deliberate simplification. It is not an accurate model of real-world systems. Your solution should correspond to the simplicity of the specification. You risk losing marks if you attempt to provide a more realistic model or provide a solution that is more complicated than necessary.

---

2. System overview

A shelter provides temporary homes for stray or unwanted pet animals which can be offered later for adoption. This shelter needs a set of interfaces and classes to manage pets' adoption. This shelter can accommodate two types of pets: **Cats** and **Dogs**.

The services offered by the shelter are:

- Adding a new pet to the shelter (the shelter needs to be able to issue them an ID)
- Giving an existing pet to a customer for adoption (customer must have a record)

Dogs can only get adopted to customers who own a garden. A customer can adopt 3 pets at most. Only Dogs can be trained.

The shelter should maintain a record of pets added to the shelter. The shelter needs to maintain another record that associates a customer with the pets they have adopted.

---

3. Tasks:

The total number of tasks you have to implement are **six** tasks across **three** parts.

3.1. Part 1: Adding a new pet to the shelter:

You need to provide classes for adding pets to the shelter. This is done by completing the **three** following tasks:

Task 1.1

You need to create classes to provide an appropriate hierarchy for pets. Your **ShelterManager** class (See Task 1.3) should create a pet object of the appropriate type when added to the shelter. You are provided with the interface **Pet** to start with (See Pet.java in Canvas).

All pets have the following **public** functionality:

- a method to get the pet ID (See Task 1.2 below)
- a method to get the pet type (either Cat or Dog)
- A method to check whether the pet is adopted or not
- A method to get care Instructions (dogs must feed three times a day and go on walks once a day, cats must feed two times a day)
- Dogs have a method to check whether the Dog is trained or not

You need to make sure that you create these classes with the appropriate methods/variables.

Task 1.2

You need to create a **PetID** class. A pet ID has two components - a single letter followed by a two-digit number. For example: `C01`

You must provide access to each component and an appropriate string representation of the ID.

Pet ID's are **unique**. You must guarantee that no two pets have the same ID.

Task 1.3

You are provided with **ShelterManager.java** template that has the signature of the methods that need to be implemented (Please **DO NOT** change the methods' signature). This class needs to provide a number of variables as well (Not included in the template).

In Taks 1.3 you only need to implement the methods needed for Part 1. The methods needed for Part 2 will be covered in **Task 2.2.**

For Part 1, this class needs to:

- Maintain a data structure of all pets added to the shelter.

And it needs to implement the following methods:

- `public Pet addPet(String petType)`
  This method adds a new pet of the specified type `petType` to the shelter and allocates it a pet ID. If the added pet is a dog, you can assume that it is untrained. On success, this method needs to return the Pet object.

- `public boolean updatePetRecord (PetID petID, boolean trained)`

  This method updates the training status of an existing pet depending on its PetID. You need to make sure that the pet does exist in the shelter before updating its record. You need to make sure that the pet can be trained depending on its type. The method returns true if the update is successful and false otherwise.

- `public int noOfAvailablePets(String petType)`
  This method returns the number of pets of the specified type (a Cat or a Dog) that are Not adopted.

3.2. Part 2: Giving an existing pet to a customer for adoption.

You need to provide classes for adopting pets by customers. This is done by completing the **two** following tasks:

Task 2.1

You need to create a **CustomerRecord** class and a **CustomerNumber** class.

A **CustomerRecord** has the customer name (comprising a first and last name – you need to create a **Name** class for storing customer first name and last name), the date of birth of the customer, a unique **CustomerNumber**, the date of issue and whether they own a garden or not (all fields cannot be altered).

The **CustomerNumber** has two components. The first component is the concatenation of the initial of the first name of the customer with an arbitrary serial number. The second component is the concatenation of the month of issue with the year of issue of the record. For example, the string representation of the **CustomerNumber** for a record issued to John Smith in January 2024 would have the form:

```
J10.12024
```

where the 10 is a serial number that, with the initials and year, guarantees the uniqueness of the customer number as a whole (note the two parts are separated by a dot ".").

You must guarantee the **uniqueness** of customer numbers.

Your CustomerRecord class must provide methods to access the customer name, the customer date of birth, the customer number, the date of issue of the record and whether the customer owns a garden.

You should use the java.util.Date class to represent dates in CustomerRecord. However, you **must not use deprecated methods of the Date class**. So, for example, in the test class, you can use java.util.Calendar to construct dates of birth and dates of issue of Customer Records. You can assume default time zone and locale.

Task 2.2

You need to implement the methods in **ShelterManager.java** that are needed for part 2.

For Part 2, this class needs to:

- Maintain a data structure of all existing customer records.
- Maintain a data structure of all existing customer numbers and the list/set of their adopted pets.

And it needs to implement the following methods:

- `public CustomerRecord addCustomerRecord(String firstName, String lastName, Date dob, boolean hasGarden)`

This method creates a customerRecord based on the given information if there is none:

  - the combination of firstName, lastName and dob are unique for each customer. If you are adding a customer with similar existing information, the method will throw an error.
  - You can assume that the date of issue is today.

The method adds the newly created record to the data structure of existing customers. On success, this method returns the CustomerRecord object.

- `public boolean adoptPet(CustomerRecord customerRecord, String petType)`

This method, given the customerRecord information and type of pet required (Cat or Dog):

  - determines whether the customer is eligible to adopt a pet of the specified type (see the rules below)
  - If the customer cannot adopt the pet, the method returns false and prints an appropriate indication of the failure (prints a message of the reason of why the customer cannot adopt the pet).
  - If the customer can adopt the pet and there is a pet available in the shelter, it gives them a pet of the specified type (at random). It then adds this pet to the list of adopted pets by the customer (so that the shelter has a record of adopted pets and the customers adopting them). It also returns true and print out a message saying what the customer is adopting.

The rules for determining whether or not a pet can be adopted are given below:

  - A customer cannot adopt more than three pets of all types (e.g. 2 Cats and a Dog)
  - if the available dog is a trained Dog, a customer must be at least 18 years old and have a garden to adopt it.
  - if the available dog is an untrained Dog, a customer must be at least 21 years old and have a garden to adopt it.
  - To adopt a Cat, a customer must be at least 18 years old.

If a pet is adopted, its record should not be removed from the shelter. Only the status of the pet will change from “not adopted” to “adopted”.

- `public Collection<Pet> adoptedPetsByCustomer (CustomerNumber customerNumber)`

  This method returns unmodifiable collection of all pets currently adopted by the customer with the specified customer number.

3.3. Part 3: Testing your code

You need to provide classes for testing your code. This is done by completing the following task:

Task 3.1:

You should provide test cases for your ShelterManager class and two other classes of your choice (created in Section 3). To do this you can use the simple test framework (**Assertions class**) provided for you in Canvas ->Assignments -> Shelter Management System. You can use Junit if you would like to. You should test the normal case, boundary conditions, and exceptional cases.

---

4. Notes:

- Make use of the Pet.java and ShelterManager.java provided for you.
- You **must not change** the methods' signature in the ShelterManager class and in the Pet interface. You can add additional methods if needed.
- You **can allow** your method in the ShelterManager class to throw an exception where needed.
- Your code must compile and run on university PCs.
- If we cannot test your implementation because you have changed the methods' signature, then that corresponding task will **score 0**.

---

5. Deliverable (What to submit)

You must submit your work to NESS as a **single** zip file named 'CSC8014_coursework_YourName.zip', where 'YourName' is replaced with your full name. The zip file must contain your solution including test classes (i.e. the implementation of the system and types outlined in Section 3 (Part 1, 2 and 3)).

Your solution should demonstrate::

- the sensible use of Java inheritance mechanisms, an understanding of how to use interfaces,

- the ability to handle collections of objects,
- the use of defensive programming, including use of immutability, appropriate error handling, validation of input and using appropriate accessors and modifiers on instance variables, methods, and classes.
- an understanding of when and how to override Object methods,
- the implementation of factories,
- the ability to implement simple algorithms,
- the ability to write Javadoc comments, and
- the ability to test your code.

6. Marking Scheme

Marks will be allocated for

- Overall structure and implementation of the solution (e.g. Classes and their relationships, correct implementation of rules specified in Sections 3 (part 1 and 2)) **(45 Marks)**
- Following good practice guidance mentioned in section 5 and appropriate comments. **(43 Marks)**
- Evidence of testing by implementation of appropriate test classes that test the normal case, boundary conditions, and exceptional cases **(12 Marks)**

7. Style guidelines

Adopt a consistent style, do not violate naming conventions (e.g. when to use upper/lower case letters in class, method and variable names) and make appropriate use of whitespace (Indentation and other spacing).

8. Further notes

Start early!! This is not a project to hack together during the last 1-2 days before the deadline.

Break the coursework down into separate tasks. Start with the simpler classes/methods first (e.g. **`Name`**, **`PetID`**, **`CustomerNumber`** and **`CustomerRecord`**) but leave the imposition of uniqueness and immutability until we cover these topics in lectures. You can implement the different types of Pets before implementing the methods in the **ShelterManager** class. Unit test classes as you progress through the coursework.

For each class you implement you should consider:

- whether to override Object methods (equals, toString, , etc.) and use valueOf,
- whether to use an interface-based hierarchy, and
- whether the class should be immutable.

You may have to defer parts of the coursework (or the implementation of certain aspects of a class) until we have covered material in lectures. In which case, you can make a start with a simpler solution that can be extended later. For example, we have not covered the Collections framework yet. To make a start, you can just use a single instance of a class.

For any questions, email me at Rouaa.yassin-kassab@ncl.ac.uk