UNIVERSITY OF SASKATCHEWAN

Department of Computer Science
176 Thorvaldson Building
110 Science Place, Saskatoon, SK, S7N 5C9, Canada
Telephine: (306) 966-4886, Facimile: (306) 966-4884

Assignment 5

Simple GUIs, and Three-Layer Architecture

Date Due: November 17, 2022, 11:59pm **Total Marks: 73**

General Instructions

- **Read these instructions carefully, especially the first assignment. You are responsible for all the instructions here.** They will not change in arbitrary and punitive ways over the semester; any changes will be highlighted.
- **This assignment is individual work.** You may discuss questions and problems with anyone, but the work you hand in for this assignment must be your own work.
- **Assignments are being checked for plagiarism.** We are using state-of-the-art software to compare every pair of student submissions.
- **Make sure your name and student number appear at the top of every document you hand in.** These conventions assist the markers in their work. Failure to follow these conventions will result in needless effort by the markers, and a deduction of grades for you.
- **Assignments must be submitted to Canvas.** It is your responsibility to ensure that your upload worked correctly. See *Instructions for Submission* for more details.
- **The assignment is due at the time stated above.** The due date should appear on your Canvas calendar on the day it is due.
- **Canvas has a built-in late penalty mechanism, which will automatically apply to any files submitted after due date.** After the grace period, 0.6% of your grade will be deducted per hour with 15% deductions per day, increasing by 15% every day thereafter.
- **You do not need to obtain permission to submit an assignment after the due date.** Canvas will be configured to accept submissions for a week after the due date. Extensions for medical and compassionate reasons can still be granted; contact your instructor as usual.
- **Programming questions must be written in Java.** Do not submit IntelliJ projects (unless the assignment says so directly). Do not submit the compiled `.class` files (unless the assignment says so directly). If you submit an IntelliJ project when it was not required, you will get zero for that question!
- **In general, non-programming questions must be submitted as either `.txt` or `.pdf` files.** If other file types are used, you may receive a grade of zero for that question. If the marker cannot open a file because you did not follow these instructions, you will definitely get a zero for anything in that file.

Instructions for submissions

- **Assignment Submissions**: You will submit a single ZIP file containing all your questions solution files. This will help mitigate the problem of Canvas renaming your files if you resubmit something. Please see *What to Hand In* for each question.
  - Name the file `A5-abc123-CMPT270.zip`, where you should use your own NSID instead of `abc123`. This naming convention will assist the markers with their work, and we encourage all students to adopt it.
  - Each question will describe one or more files to submit. These are to be included in the same ZIP file described above. All the files mentioned by all questions in a single assignment go into one ZIP file that you submit.
  - Each question will mention the name of the file(s) to submit, and file format requirements. Failure to follow file format requirements may result in severe mark deductions. If you change or modify any file you've previously submitted, you have to recreate the whole ZIP file, and resubmit it.
  - You must submit a ZIP file, and no other archival format (e.g., no RAR, 7ZIP, etc.). If you use these other tools, and rename it with ZIP, the markers will not be able to open it, and you will get zero, because it could not be opened.

Version History

- **31/10/2022**: released to students

Overview

This assignment builds on A4. Each question is a software development task that will extend the behaviour of your application, or bring new object oriented techniques into play. Some of these you have seen, but some will be presented after the assignment has been released. Don't leave your review of the lecture material to the last minute.

Codebase

You could start from your own codebase, submitted for A4. If it was working well, you could continue to work on it. That would be ideal, as you will benefit from the full experience of dealing with your past code. On the other hand, you may also start with the Solutions for A4 (supplied with A5 as starter code). It's not your codebase, so you may need to study it to become familiar with it. It should not be too surprising, if you gave A4 a convincing effort. This exercise of studying someone else's code is also valuable, for a different reason: developers are frequently required to work with code designed or implemented by another developer.

Workload

This work is pretty straight-forward. The work does not involve solving problems where the solution is unknown. It involves working with code, refactoring, adding functionality, as well as testing, and debugging. For you, having a completed final project is the way to get the best mark, but the real value of this assignment is the experience you get by completing it. The whole assignment should take you 10-12 hours. It's a lot of work, but none of it is intellectually difficult. You should try to work on it over the course of 2 weeks. Do not expect to finish if you start on the day it is due.

Outline

1. Question 1 completes some of the incomplete tasks from A4. Its main purpose is to make you familiar with the new code base, or re-familiarize yourself with your old code.

2. Question 2 cleans up the user interaction a bit with the idea of "programming to an interface." We'll create a class to do all the interaction on the console, and this class will have a fixed interface.

3. Question 3 allows you to create a new user interaction module so that we are no longer bound to the console. It will be easy to do because we are replacing the class from Question 2 with a new class that has the same interface, but uses the Java GUI components, instead of the console.

4. Question 4 will get you to apply the Singleton pattern twice, so that Animal objects and StaffMember objects are contained in their own class.

5. Questions 5 & 6 will get you to apply the Command pattern, which will replace methods with objects.

6. Question 7 asks you for external documentation, just as A3 and A4 did. You should create a UML diagram and update it for every question. It will add a little time to each question, but it will mean that you have something to hand in no matter what stage you are at. Also, rushing to complete the UML close to the deadline is stressful. Having a partially completed diagram ready to be modified will reduce your stress near the deadline.

7. Question 8 asks you to produce a JAR file. Attend the tutorials and watch the TA's demo how to do this. Then try it on your own a few times, so that there are no surprises when you get close to the deadline. Don't leave it until the deadline, because there will be no one online to help you.

How to get started

1. Read the whole assignment. It might take you an hour.

2. Read the UML diagram provided, and browse through the starter files. Nothing in these files should be unfamiliar to you, since this is basically what you were working on in A4.

3. Make a plan. This plan should take about 10-15 minutes to make. Look at your schedule, and try to schedule a few hours to work on this assignment every day or two. Each question provides a time estimate, from the point of view of the instructors. It's a guideline, and you may need more or less time. Even if you don't finish every question, you can stop after any question, and still have something incomplete but working to hand in.

4. Consult with your instructors during office hours and with the TAs during help desk hours, and after tutorials. If you don't start soon, you will waste your opportunities to get help.

What to Hand In

You will submit all your Java source code in one ZIP file. It will contain all the files needed for all the questions. Each individual question will remind you which files are important for each question, but you will only submit one copy, and all your Java files will be part of the ZIP file.

You will create an executable JAR file for A5Q6. Instructions for creating JAR files are posted, and there is a video demo of the process on the Lecture Video site.

Question 1 (6 points):

Purpose: To familiarize yourself with the code-base, by adding a couple of methods.

Degree of Difficulty: Easy

Task

Recall the `PetStoreSystem` class from Assignment 4. We left a couple of methods incomplete (left as method stubs in A4). In this question, your job is to implement these methods and provide a demonstration of them working.

- Task 5: `showEmptyKennels()` should display a list of available kennels in the `PetStore` that do not have any animal assigned to them.
  - In the starter files, there is a `PetStore` method called `availableKennels()` that should make Task 5 very easy.
- Task 7: `releaseAnimal()` is the opposite of task 6. This method will ask the user for the animalID of an animal, and then free the kennel that the animal has been assigned to, allowing it to be assigned to another animal. This method should not remove the animal from the `PetStoreSystem`, nor should it change any of the associations between animal and staff. The animal should remain in the system, but is just not assigned to a kennel anymore.
  - Remember that the information about the kennel assignment is stored in `PetStore`, and in `Animal` classes.

Implement the `showEmptyKennels()` and `releaseAnimal()` methods in the provided `PetStoreSystem` class. After these are implemented, run the `main()` method of the `PetStoreSystem` class, and demonstrate that your methods are working by doing something like this:

- Create a `PetStore` with a handful of kennels.
- Add at least one animal to the system.
- Assign some animal(s) to kennels.
- Show that the available kennels have changed.
- Release at least one animal.
- Show the available kennels has changed.

Copy-and-paste the console input/output into a file titled `a5q1_demo.txt`. Don't make this a complicated file to read. Keep it short. It's a demonstration, not a proof, and not a thorough test.

You should of course test your methods thoroughly, but that's for your satisfaction, not for grades.

Once everything is working, rename the class `PetStoreSystema5q1`. This renaming is strictly to help with marking.

Hint: You will be required to submit a UML diagram for the whole system in one of the later Questions. To prevent a long stressful diagramming session near the deadline, create a UML diagram for Question 1 when you finish it, and each time you finish a question, update the diagram.

What to Hand In

- The completed `PetStoreSystema5q1.java` program with task 5 and task 7 implemented. **Make sure this file is part of the ZIP file you submit for A5. You do not need to submit it separately.**
- A document named `a5q1_demo.txt` which shows the console input/output for task 5 and task 7.

Be sure to include your name, NSID, student number and course number at the top of all documents.

Evaluation

2 marks `showEmptyKennels()` correctly displays available kennel labels to console.

2 marks `releaseAnimal()` correctly frees the kennel where an animal was assigned.

2 marks `a5q1_demo.txt` shows that your two methods work.

Question 2 (14 points):

Purpose: Cleaning up the user interface

Degree of Difficulty: Moderate. (2-3 hours) Getting the console input/output working is tricky because of the Scanner. Make sure you do a lot of testing to avoid errors you cannot find later!

Task

Perhaps you noticed in A4 that coding the work on the console was a bit repetitive and problematic. In this question, we'll use interfaces to help clean that up. By cleaning it up first, we can later set the stage for practising our understanding of GUIs.

Study the given Java file `InputOutputInterface.java`. It declares four general-purpose methods:

- `readString(String prompt)`: A method with this name should display the given prompt, ask for input, and return the inputted String.
  - You could use a method like this to ask the user for any kind of string value.
- `readInt(String prompt)`: A method with this name should display the given prompt, ask for input, and return the inputted integer.
  - You could use a method like this to ask the user for any integer value.
- `readChoice(String[] options)`: A method with this name should display each of the given options, ask for input, and return the inputted integer corresponding to the index of one of the options.
  - You could use a method like this as the main "menu" for the `PetStoreSystem`, displaying all the options, and returning an integer.
- `outputString(String outString)`: A method to show the given string to the user.
  - You could use a method like this to display the state of the PetStore, or report a user input error. There is no input needed.

This interface is useful because it generalizes interaction between an application and a user, which can be implemented in different ways. Each implementation could be different, but the application could still use it because it conforms to the standard.

In this question, we will implement these methods for use with the console. That will require writing a class called `ConsoleIO` that implements this interface.

- There should be one attribute, the scanner used by the methods.
- The methods in `ConsoleIO` will use `System.in` (with the Scanner, of course) and `System.out` to interact with the user.
- Your methods can check for validity of user input, but if the user's input was invalid, your method should deal with the problem before returning. This may require catching exceptions from the Scanner object, and dealing with it within the method.

You should test your implementation of `ConsoleIO`. In `ConsoleIO.main()`, make sure that you use all 4 of the methods, in several orders. Running `main()` will require you to enter data from the console, so the testing will not be as automated as some other testing could be.

- `main()` will call the input methods, and it require you to type some input values.
- To check if the input methods are working, you'll display the input values on the console using `outputString()`, and use a visual check.

Don't start the next part of the question until you are relatively sure `ConsoleIO` is working. Make sure you test thoroughly. It's easy to make mistakes you don't know you are making with a tool as complicated as the Scanner.

Once we have this `ConsoleIO` class defined, we will add a new attribute to the `PetStoreSystem` class, namely a `ConsoleIO` object, which will be the object that all the `PetStoreSystem` methods use to communicate with the user.

- **Note:** If you are working with the solution to A4 provided by the instructors, you will notice that all the console IO happens in the `PetStoreSystem` class. However, if you are working with your own code, you may have designed A4 so that console IO happens in other places. If that's the case, you want to revise your system so that only the `PetStoreSystem` does any interaction with the user, and only using the `ConsoleIO` object!

Then we have to go through the `PetStoreSystem`, and re-write those places where the code uses a Scanner directly, or uses `System.out` directly, replacing them with appropriate calls to the `ConsoleIO` methods.

All console input/output in the `PetStoreSystem` must be done through `ConsoleIO`.

Benefits:

- The `PetStoreSystem` code will be tidier.
- All the user-input input can be checked in a standard uniform way, and all in the same class!
- Once converted to use `ConsoleIO`, we can replace it with another class that also implements `InputOutputInterface` but uses a GUI, and we will not have to change `PetStoreSystem` at all, because it was programmed to work with an interface, and not with a specific class.

But we will see these benefits in the next question. In this question, we want to implement `ConsoleIO`, and modify `PetStoreSystem` so that only uses the methods described by `InputOutputInterface` which are provided by the class `ConsoleIO`.

Once everything is working, rename the class `PetStoreSystema5q2`. This renaming is strictly to help with marking.

What to Hand In

- The revised `PetStoreSystema5q2.java`, with a (new) attribute to remember the `ConsoleIO` object, and all the tasks using this object to communicate with the user.
- Your implementation of `ConsoleIO.java`.
- **Make sure these files are part of the ZIP file you submit for A5. You do not need to submit them separately.**

Be sure to include your name, NSID, student number and course number at the top of all documents.

Evaluation

5 marks `PetStoreSystema5q2`

- Has a single attribute for an object implementing the `InputOutputInterface`.
- Uses the `InputOutputInterface` for all user interactions.
- No direct calls to `System.in`, `Scanner`, or `System.out` remain in this class.

9 marks `ConsoleIO`

- Has a single attribute for a Scanner object, for input.
- The methods that "read" input, use the Scanner object.
- The output method can use `System.out` directly.
- The "read" methods should manage simple user input error, e.g., not entering an integer in `readInt()`. The read methods must not throw exceptions if the user typed something invalid.

Question 3 (8 points):

Purpose: Moving to a Graphical User Interface.

Degree of Difficulty: Easy. (60-90 min)

Task

Now that we have revised the `PetStoreSystem` class to use the `InputOutputInterface`, we can substitute the `ConsoleIO` implementation with another implementation that uses some of Java's simpler GUI tools.

Study the given Java file `AbstractDialogIO.java`. It's an abstract class that implements the `InputOutputInterface`, which we studied in the previous question. The `AbstractDialogIO` class implements one method:

- `readChoice(String[] options)`: This method creates a dialog box, shows the options, and returns the index of the user's choice from the options.

In this question, we will implement a new class called `DialogIO` that extends `AbstractDialogIO` (and therefore implements `InputOutputInterface`). It should use `JOptionPane` to communicate with the user:

- `readString(String prompt)`: Uses `JOptionPane` to acquire input from the user, and returns a String.
- `readInt(String prompt)`: Uses `JOptionPane` to acquire input from the user, and returns an integer.
- `outputString(String outString)`: Uses `JOptionPane` to display the given string to the user.

Notice that you do not need to implement the `readChoice()` because it will be inherited from `AbstractDialogIO`.

You should test your implementation of `DialogIO`. In `DialogIO.main()`, make sure that you use all 4 of the methods. Running `main()` will require you to enter data in dialog boxes that pop up on your monitor, so the testing will not be as automated as some other classes could be.

- `main()` will call the input methods, and it require you to type some input values.
- To check if the input methods are working, you'll display the input values using `outputString()`, and use a visual check.

Don't start the next part of the question until you are relatively sure `DialogIO` is working.

Once we have this `DialogIO` class defined, we will modify the `PetStoreSystem`, replacing the `ConsoleIO` object with a `DialogIO` object. If you have done the previous question correctly, everything should just fall into place with a single change.

Once everything is working, rename the class `PetStoreSystema5q3`. This renaming is strictly to help with marking.

What to Hand In

- The revised `PetStoreSystema5q3.java` replacing the `ConsoleIO` object with the `DialogIO` object, and no other changes.
- Your implementation of `DialogIO.java`.
- **Make sure these files are part of the ZIP file you submit for A5.**

Be sure to include your name, NSID, student number and course number at the top of all documents.

Evaluation

1 mark `PetStoreSystema5q3`

- Has a single attribute for an object implementing the `InputOutputInterface`.
- Uses the `DialogIO` object.

7 marks `DialogIO`

- Extends `AbstractDialogIO`.
- Uses `JOptionPane` for interacting with the user.
- The "read" methods should manage simple user input error, e.g., not entering an integer in `readInt()`. The read methods must not throw exceptions if the user typed something invalid.

Question 4 (15 points):

Purpose: Practising the Singleton pattern.

Degree of Difficulty: Easy. (1-2 hours) These are not difficult, if you have studied the Singleton pattern.

Task

Many of the tasks performed by our `PetStoreSystem` need to access the collection of `Animal`'s, the collection of `StaffMember`'s, and the `PetStore`. In order to provide this access but prevent more than one instance of any of these entities, you are to use the **Singleton Pattern** to implement them.

Note: This exercise is a little artificial, since one might presume that a `PetStoreSystem` could be programmed to have multiple PetStores, and each `PetStore` would need a collection of Animal objects. But, we will use the Singleton pattern here anyway, just for practice.

Implement the following classes:

- `AnimalMapAccess`. This class creates a single `TreeMap` instance. It starts out empty.
  - Singleton Attribute: `dictionary`: `TreeMap<String, Animal>`
  - Static method: `getInstance()`: `TreeMap<String, Animal>` returns a reference to the dictionary containing all Animal objects known to the system.
  - If the PetStoreSystem wants to do add, release, or lookup a Animal, it has to ask for this singleton object first using `getInstance()`.
- `StaffMapAccess`. This class creates a single `TreeMap` instance. It starts out empty.
  - Singleton Attribute: `dictionary`: `TreeMap<String, StaffMember>`
  - Static method: `getInstance()`: `TreeMap<String, StaffMember>` returns a reference to the dictionary containing all StaffMember objects known to the system.
  - If the PetStoreSystem wants to do add, release, or lookup a StaffMember, it has to ask for this singleton object first using `getInstance()`.
- `PetStoreAccess`. This one is a bit trickier.
  The problem is that the `PetStore` constructor needs to know some initial values for the attributes (the `PetStore` name, the first and last kennel labels). Since the `PetStoreSystem` will interact with the `PetStore` class only through the `PetStoreAccess` class, we need another static method to get the initial values to create the PetStore.
  - Singleton Attribute: `storeInstance`: `PetStore`
  - Static method: `void initialize(String storeName, int fLabel, int lLabel)` This function should create the PetStore, but only if it has not been created already. This static method should throw an exception if the `PetStore` object has already been created and initialized. This exception is not one that you should catch; it's the sign that the program has a bug, so fix the bug!
  - Static method: `getInstance(): PetStore` returns the PetStore known to the PetStoreSystem. This method should throw an exception if the PetStore has not already been created and initialized. This exception is not one that you should catch; it's the sign that the program has a bug, so fix the bug!
  - If the PetStoreSystem wants to do perform any action on the PetStore, it has to ask for this singleton object first using `getInstance()`.
- `IOAccess`. This class creates a single `InputOutputInterface` object.
  We want a single `InputOutputInterface` object that the whole `PetStoreSystem` will use to communicate with the user. It will either be a single `ConsoleIO` object, or a single `DialogIO` object. To get this to work, you have to ask the user. Depending on the user's answer, the `IOAccess` class will create a `ConsoleIO` object as the singleton, or a `DialogIO` object as the singleton. Of course, you have

to use one of these two objects to communicate with the user about the choice. For example, you could create a `ConsoleIO` object, and use it to ask which style of IO they want, and then if they choose `DialogIO`, create the `DialogIO` object, and use it as the singleton.

- Singleton Attribute: `io`: `InputOutputInterface`
- Static method: `getInstance()`: `InputOutputInterface` returns an object that implements the `InputOutputInterface`.

Remember that the constructors for Singleton classes should be private.

Once you have created these classes, revise your `PetStoreSystem`, so that each method that needs access to the Animal objects, StaffMember objects, `PetStore`, or `InputOutputInterface` uses the classes above. For example, we might use the following code in `addStaff()`:

```
StaffMapAccess.getInstance().put(employeeID, StaffMemberObject);
```

This singleton object, stored in `StaffMapAccess` `PetStore` , replaces the `TreeMap` attribute currently used in the `PetStoreSystem`. In other words, `PetStoreSystem` will not store a reference to the `TreeMap` for StaffMember objects; instead it will use the dictionary provided by StaffMapAccess. Likewise, replace the `TreeMap` attribute for the the Animal objects with direct access to AnimalMapAccess, and the `PetStore` attribute with direct access to the `PetStoreSystem` object. Finally, you should change the code in `PetStoreSystem` to use the IOAccess object, as follows:

```
IOAccess.getInstance().outputString("Invalid option: "+ task +", try again.");
```

Once everything is working, rename the class `PetStoreSystema5q4`. This renaming is strictly to help with marking.

What to Hand In

- The revised `PetStoreSystema5q4.java`.
- Your implementation of
  - `AnimalMapAccess.java`
  - `StaffMapAccess.java`
  - `IOAccess.java`
  - `PetStoreAccess.java`
- **Make sure these files are part of the ZIP file you submit for A5. You do not need to submit them separately.**

Be sure to include your name, NSID, student number and course number at the top of all documents.

Evaluation

- `AnimalMapAccess`
  - 1 mark. The class allows a single instance to be created only.
  - 1 mark. The method `getInstance()` is static, and returns a reference to the (only) instance.
- `StaffMapAccess`.
  - 1 mark. The class allows a single instance to be created only.
  - 1 mark. The method `getInstance()` is static, and returns a reference to the (only) instance.
- `PetStoreAccess`
  - 1 mark. The class allows a single instance to be created only.
  - 1 mark. The method `getInstance()` is static, and returns a reference to the (only) instance.
  - 1 mark. The method `initialize()` is static, and creates the instance.
  - 1 mark. Exceptions are used to signal illegal use of the static methods, i.e., asking for the object before it is initialized, and trying to initialize a second time.
- `IOAccess`.
  - 1 mark. The class allows a single instance to be created only.
  - 1 mark. The class lets the user choose between `ConsoleIO` and `DialogIO`.
  - 1 mark. The method `getInstance()` is static, and returns a reference to the (only) instance.
- `PetStoreSystema5q4`
  - 1 mark. The `PetStoreSystem` no longer creates an instance for `TreeMap<String, Animal>`, but uses `AnimalMapAccess` directly.
  - 1 mark. The `PetStoreSystem` no longer creates an instance for `TreeMap<String, StaffMember>`, but uses `StaffMapAccess` directly.
  - 1 mark. The `PetStoreSystem` no longer creates an instance for `PetStore`, but uses `PetStoreAccess` directly.
  - 1 mark. The `PetStoreSystem` no longer creates an instance for `InputOutputInterface`, but uses `IOAccess` directly.

Question 5 (9 points):

Purpose: Practising with the Command Pattern

Degree of Difficulty: Moderate. (1-2 hours). After solving the first one, the remainder should be very easy.

Task

The current version of the `PetStoreSystem` class has a method to handle each of the tasks. In general, each method does the following things:

- Obtains user input for the data needed for the task
- Checks the data for validity
- Invokes the methods from other classes to do the task
- Handles the result of the task

For this question, your job is to refactor the tasks in the `PetStoreSystem` class. This question will go through the steps in detail for one of the tasks, and when you have done the steps once, you will repeat the exercise for the others.

Study the given Java file `Command.java`. It defines in interface the requires a single method,

```
public interface Command {
    public abstract void execute();
}
```

In this question, we will move the logic of the method `addStaff()` out of `PetStoreSystem` to a new class called `AddStaff`, with requirements to be as follows:

- `AddStaff implements Command`.
- Uses the default constructor `AddStaff()` provided by Java. You don't need to write a constructor.
- Has a single method `public void execute()` whose body is basically copy/paste from `addStaff()`. If you have done everything well so far, then this is really just copy/paste from `PetStoreSystem` to a new class `AddStaff`. If you haven't quite refactored properly, you might have more trouble.
  - Because you are using `StaffMapAccess`, you have direct access to the StaffMember objects in the system.
  - Because you are using `IOAccess`, you can communicate to the user.

When you are done, you should be able to use the following to demonstrate that the method works:

```
    public static void main(String[] args) {
        Command cmd = new AddStaff();
        cmd.execute();
    }
```

In this short program, a new `Command` object is created, and then immediately executed. If you have refactored properly using the singleton classes from the previous question, running this program will let you try out the task of adding a new staff member to the system. It's not a test, of course. You can imagine how you might test `AddStaff` in ways that we could not test the methods in `PetStoreSystem`.

When you are satisfied that everything works, you should be able to use `AddStaff` in `PetStoreSystem`, by replacing the body of `addStaff()`, as follows:

```
public void addStaff() {
    Command cmd = new AddStaff();
    cmd.execute();
}
```

This is not the final version of `PetStoreSystem`, so don't be concerned that we are creating an object and using it every time we call `addStaff()`.

Finally, now that you have implemented the `Command` pattern for `addStaff()`, repeat the steps for every task! You'll need a class for every task, and that class should implement the `Command` interface. Each of these classes should take the name of the task they are implementing, e.g., `AddAnimal`, `AssignStaffToAnimal`, etc.

Hints:

- The only slightly tricky one here is `SystemState`, because it uses `PetStoreSystem.toString()`. Just copy the code from `toString()` into `SystemState.execute()`. No problem!
- The PetStoreSystem menu has a "quit" option, but you don't need to create a Quit command. Instead, you can re-use the SystemState object. Remember that the loop in `PetStoreSystem` will stop the program when the user selects the option to quit, but before the system actually stops, the state of the system is displayed.

What to Hand In

- The revised `PetStoreSystema5q5.java`.
- Your implementation of the Command classes that implement the 8 tasks.
- **Make sure these files are part of the ZIP file you submit for A5.**

Be sure to include your name, NSID, student number and course number at the top of all documents.

Evaluation

- For each of 8 tasks:
  - 1 mark. A class is defined to implement the Command interface. It has no constructor, and no attributes. The class has a method `public void execute()` that performs the task.
- `PetStoreSystem`
  - 1 mark only. You replaced the body of each of the task methods with a simpler version that creates a Command object, and then executes it.

Question 6 (6 points):

Purpose: Getting the most out of the Command pattern.

Degree of Difficulty: Moderate. (60-90 min). You are mostly deleting code.

Task

The current version of the `PetStoreSystem` class has a method to handle each of the tasks. By this point in our refactoring, each of them should look like this:

```
public void addStaff() {
    Command cmd = new AddStaff();
    cmd.execute();
}
```

Of course, we can do better!

What you have to do in this question is to create an array of `Command` objects, something like this:

```
Command[] commands = new Command[9];
```

You can decide where you want this array: as a local variable in `main()`, or as an attribute of the `PetStoreSystem`, or as a static variable. It really doesn't matter.

Then you are going to initialize this array with objects, one for each task. If you put the proper `Command` in the right place, you can use the index returned by `InputOutputInterface.readChoices()` to access the command that the user asked for:

```
task = IOAccess.getInstance().readChoice(options);
commands[task].execute();
```

Notice that this line of code doesn't know what the command is doing, and isn't calling a `PetStoreSystem` method by name. It's simply using the `Command` interface to do one of those tasks. For that reason, you can delete all the methods in `PetStoreSystem` that we had previously used, e.g., `addStaff()`. They are no longer needed once we have loaded up the commands, e.g. AddStaff, in the array.

What to Hand In

- The revised `PetStoreSystema5q6.java`.
- **Create a ZIP file for all the Java source (.java) files you created for A5, including all the different PetStoreSystem versions.**

Be sure to include your name, NSID, student number and course number at the top of all documents.

Evaluation

- `PetStoreSystem`
  - 2 marks: the named methods `addStaff()`, `addAnimal()`, etc. that performed the tasks in previous versions of the `PetStoreSystem` have been deleted.
  - 2 marks: An array of `Command` has been created, and initialized with `Command` objects that perform the various tasks.
  - 1 mark: The main program uses the `Command` objects in the `Command` array to perform the tasks.

Question 7 (10 points):

Purpose: Practising preparation of External Documentation.

Degree of Difficulty: Moderate. (60-90 min). This will take time, but it's not hard. Don't leave it to the last minute.

For this assignment, your external documentation should consist of two documents: `A5_status.pdf` and `A5_manual.pdf`. You should summarize your whole assignment here.

- `A5_status.pdf`. Describe the status of your assignment. What is working and what is not working? What is tested and what is not tested? If it is only partially working, the previous point should have described how to run that part or parts that work. For the part or parts not working, describe how close they are to working. For example, some of the alternatives for how close to working are (i) nothing done; (ii) designed but no code; (iii) designed and part of the code; (iv) designed and all the code but anticipate many faults; or (v) designed and all the code but with a few faults; (vi) working perfectly and thoroughly tested.

  Also include the number of hours you spent working on this assignment, and when you started. Indicate whether you gave yourself enough time, or whether unexpected events impacted your work. This is a self-reflection exercise, to help you refine your expectations for future assignments and projects.

- `A5_manual.pdf`. Maintenance manual. For this assignment (as with A4), it is sufficient to include a UML class diagram showing all the features of each class, and the relationships (inheritance, uses and associations) amongst the classes (if any). Future assignments may require more information!

  UML diagrams should be generated with a program like StarUML (`http://staruml.io/download`) or an online tool, such as `https://www.lucidchart.com`. Make sure that you use the correct arrows and arrowheads. Incorrect arrowheads will lose several marks. You can also draw out a UML diagram by hand and scan it. Just make sure it is easily readable.

  Hint: Create a UML diagram for Question 1 when you finish it, and each time you finish a question, update the diagram. This will help you avoid spending a very long time at the end documenting just the last question.

What to Hand In

- `A5_status.pdf`, outlining the status of your assignment, as described above.
- `A5_manual.pdf`, giving the UML diagrams for classes in questions 1-6.

Be sure to include your name, NSID, student number and course number at the top of all documents.

Evaluation

Status: 5 marks Full marks will be given if the status document is complete, and represents an objective assessment of your assignment status. Your mark does not depend on your actual status (the rest of the assignment marks will reflect what we think of your work). It depends on the quality of your own assessment of your status. An honest assessment and reflection is worth something to you as a professional-in-training.

Manual: 5 marks The UML diagram shows the classes from Questions 1-6, and the relationships between them. Full marks if the UML diagram is complete and accurately describes them.

Question 8 (5 points):

Purpose: Creating a JAR file.

Degree of Difficulty: Moderate. (15-30 min). You should probably not leave this for the last minute.

Task

Create a JAR file for A5, making the `PetStoreSystema5q6` class the main class for the JAR. No Java coding here, but you have to learn how to create a JAR file.

Even if you don't finish every question, you can still submit a JAR file for whatever you got working. In this case, use the `PetStoreSystema5qN` class as the main class for the JAR, where N is the question number for your most advanced working system.

What to Hand In

- A JAR file for A5, making the `PetStoreSystema5q6` class the main class for the JAR.
- This question adds no Java code, so there is no additional Java code to submit. All of the Java requirements are in earlier questions.

Evaluation

- 2 marks: A JAR file was submitted.
- 3 marks: The marker can execute your Java application because the JAR file was created correctly. The application does not have to work perfectly to get this mark. If you did not quite get all the work done for all 6 previous questions, you can still get this mark.