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 4

Inheritance and Data Structures

Date Due: October 24, 2022, 11:59pm **Total Marks: 90**

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 `A4-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

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

Overview

For this assignment, you will extend the work we did in Assignment 3. We will be building a simple Pet Store Management System with a fixed number of kennels for animals. The objective is to allow a user of the system to keep track of the animals in the store, and to keep track of the staff. Methods are needed to assign animals to kennels, as well as create new animals in the system. Also, it should be easy to determine which kennels are empty, making it easier to assign animals to free kennels. As well, staff members can be assigned to animals (There will be multiple staff, some of which can be assigned to multiple animals). It should be easy for a user of the system to see which animals are assigned to each staff member, and which staff are assigned to each animal.

Class overview

In this assignment, you must implement the following three classes: a `StaffMember` class (Question 2), a `Manager` class (Question 3). These two classes will extend and use the `Person` and `BasicStaff` classes which are given on Canvas. More details are given in each question. You are also given a `PetStore` class (on Canvas), which you will modify in certain ways, as described in Question 4.

From these classes we will build a `PetStoreSystem` class (Question 5), which will be the main program for the pet store management system we are building. It will make use of all the classes in earlier questions. The system will communicate with the user through the console. Your job in Question 5 is to organize these tasks, and get the user to add animals, assign animals to staff members, assign/remove animals from kennels, etc. More details are found in Question 5.

General requirements

Each class should have a constructor, as well as the instance variables and methods specified in each question. All the instance variables should be declared private. In addition, each class (except `PetStoreSystem`) is to have a static main method that should be used to test the class methods following the principles of testing as discussed in class. Do not add any additional parameters to the methods, none are needed. For `PetStoreSystem`, it will also have a static main method, but in this case, the main method will be the main program for our pet store management system.

You are expected to continue to **document your classes** internally using Javadoc comments, and other comments to make your code readable for markers. Proper internal documentation includes:

- A comment just before the class header that gives an overview of the class. For example, if the class models an entity, the comment might state what entity the class models and specify the key features. Alternatively, if the class serves as a container, state what type of items the container stores, and how these items are accessed. If the class is an interface, state what it provides an interface for and whether there is anything special about the interface. Finally, if it has a control function, what is it doing and controlling? Recall that comments for a class appear before the class and begin with `/**`.

- A Jacvadoc comment just before each instance variable stating what is stored in the field, again beginning with `/**`

- A comment before each constructor and method stating what it does. Note that the comment only gives what the method does, not how it does it. If it isn't obvious from the code how it accomplishes its goal, comments on how it is done belong in the body of the method.

- Be sure to include `@param` and `@return` comments. Also, if a method has a precondition, specify the precondition in a `@precond` comment, and throw a runtime exception if it is not satisfied. Note that a *precondition* in this context is an **extra** condition on the input arguments, e.g., a label can't be negative, or an employee ID has to be length 6. Note that these additional comments and precondition checks have already been added to the Animal and BasicStaff classes of Assignment 3, but they should be added into the PetStore class, as well as the new classes.

- Single line comments as needed to improve the readability of your code. Over-use of single-line comments will result in a deduction of marks. Here's an example of over-commenting:

```
int line_counter = 0; // initialize line_counter to zero
while (line_counter < 10) {  // count up to 10
     line_counter += 1; // add 1 to line_counter
     // print a message to the console
     System.out.println(line_counter < 10);
}
// all done!
```

- Use descriptive variable names and method names.
- Include good use of white space, especially reasonable indentation to improve the readability of your code.

You will be required to provide **external documentation** as well, similar to Assignment 3. See Question 5 for details.

Exceptions

You are expected to practice using **exceptions**, by throwing exceptions when appropriate (e.g., a method pre-condition is violated), or by catching exceptions (such as when the user's actions cannot be completed). You will need to consider where to catch and handle exceptions, as well as which exceptions to try to handle.

Question 1 (19 points):

Purpose: Maintainance

Degree of Difficulty: Easy

Task

In this question you will make modifications to the `Animal` class that was created in Assignment 3. The features to be added are as follows:

- An integer label of the kennel currently assigned to the animal. A value of -1 is used if the animal has not been assigned to a kennel
- A list to store the animal's assigned staff members (`StaffMember` class as described in questions 2)
- `getAssignedKennel()` An accessor method for the animals assigned kennel
- `setAssignedKennel(int newKennelLabel)` A mutator method to set the animals assigned kennel
- `addStaff(StaffMember stf)` A method that adds a new StaffMember to the list of the animals assigned staff **HINT:** It may be a good idea to leave this method as a stub for now, and come back to it once you have completed question 2 and created the StaffMember class.
- `removeStaff(String employeeID)` A method that removes a StaffMember from the list of the animal's assigned staff **HINT:** It may be a good idea to leave this method as a stub for now, and come back to it once you have completed question 2 and created the StaffMember class.
- `hasStaff(String employeeID)` A method that checks to see if a StaffMember with `employeeId` is assigned to the animal. Returns true if the manager is found, false otherwise. **HINT:** It may be a good idea to leave this method as a stub for now, and come back to it once you have completed question 2 and created the StaffMember class.
- Additional testing added to the `main()` method to test the newly added features above

Hint: You can use the `LinkedList` or `ArrayList` class in the `java.util` library to store a list of staff members. It's a generic class; review the lecture to remind yourself how to use these.

What to Hand In

The completed `Animal.java` file.

Evaluation

Attributes: 4 marks. 2 marks for each attribute. Full marks if it is appropriately named, has an appropriate type, has appropriate Javadoc comment, and is declared `private`.

Methods: 10 marks. 2 marks for each added method. Full marks if it is appropriately named, has an appropriate interface (parameters, return value), has appropriate Javadoc comment, has an appropriate implementation (including exceptions), and is declared `public`.

Testing: 5 marks. 1 mark for each method adequately tested in `main()`. Full marks if each method's return value or effect is checked at least once.

Question 2 (17 points):

Purpose: Practising Inheritance

Degree of Difficulty: Easy

Task

In this question you will add a new class to the system, namely the `StaffMember` class. This class extends the `BasicStaff` class. Note that the `BasicStaff` class also extends the `Person` class. Both the `BasicStaff` and `Person` class are given on Canvas. For our purposes, a staff member will have the following features:

- A list of the staff member's assigned animals.
- `StaffMember(String fName, String lName, String sin, String employeeID)` A constructor that takes in the staff member's first name, last name, SIN, and employee ID. Initially the staff member should have no animals assigned to them.
- `assignAnimal(Animal a)` A method that assigns an animal to the staff member's list.
  Hint: You can make this a stub at first. You don't have to do everything in the order that the specifications are given. Once you have gotten attributes and constructors in place for Questions 1-3, you can come back to fill in some of the method stubs.
- `removeAnimal(String animalID)` A method that removes an animal from the staff member's list of assigned animals.
  Hint: You can make this a stub at first. You don't have to do everything in the order that the specifications are given. Once you have gotten attributes and constructors in place for Questions 1-3, you can come back to fill in some of the method stubs.
- `hasAnimal(String animalID)` A method that checks to see if an animal with `animalID` is currently assigned to the staff member. Returns true if the animal is found, false otherwise
  Hint: You can make this a stub at first. You don't have to do everything in the order that the specifications are given. Once you have gotten attributes and constructors in place for Questions 1-3, you can come back to fill in some of the method stubs.
- `toString()` A method that returns a string representation of all the information about the StaffMember in a form suitable for printing
- A main method that will test all of the above features

Hint: You can use the `LinkedList` or `ArrayList` class in the `java.util` library to store a list of assigned animals.

What to Hand In

The completed `StaffMember.java` program, and the unmodified `BasicStaff.java` file.

Evaluation

Attributes: 2 marks. 2 marks for each attribute. Full marks if it is appropriately named, has an appropriate type, has appropriate Javadoc comment, and is declared `private`.

Methods: 10 marks. 2 marks for each method, including the constructor. Full marks if it is appropriately named, has an appropriate interface (parameters, return value), has appropriate Javadoc comment, has an appropriate implementation (including exceptions), and is declared `public`.

Testing: 5 marks. 1 mark for each method, including the constructor. Full marks if each method's return value or effect is checked at least once.

Question 3 (8 points):

Purpose: Practising Inheritance

Degree of Difficulty: Easy

Task

The `Manager` class. This class extends the `StaffMember` class. For our purposes, a manager will have the following features:

- `Manager(String fName, String lName, String sin, String empID)` A constructor that takes in the Manager's first name, last name, SIN, and employeeID. Initially the manager should have no assigned animals.
- `toString()` A method that returns a string representation of all the information about the manager in a form suitable for printing. This string should start off the the classifier "Manager: " followed by the rest of the relevant information.
- A main method that will test all of the above features

What to Hand In

The completed `Manager.java` program.

Evaluation

Attributes: 2 marks. 2 marks if your Manager class defines no attributes specific to the Manager class.

Methods: 4 marks. 2 marks for each method, including the constructor. Full marks if it is appropriately named, has an appropriate interface (parameters, return value), has appropriate Javadoc comment, has an appropriate implementation (including exceptions), and is declared `public`.

Testing: 2 marks. 1 mark for each method, including the constructor. Full marks if each method's return value or effect is checked at least once.

Question 4 (6 points):

Purpose: Some more maintenance

Degree of Difficulty: Easy

Task

In this question, you will perform additional maintenance by making modifications to the `PetStore` class.

The **provided** `PetStore` class should be modified, adding two more methods:

- `availableKennels()` A method that returns a list of the currently empty kennels in the store. This can be an `ArrayList` or a `LinkedList`.
- `freeKennel(int kennelLabel)` A method that removes an animal from a specified kennel.

Make sure you update the tests in the `main` method and add new tests for the new methods.

What to Hand In

The revised `PetStore.java` program.

Evaluation

Methods: 4 marks. 2 marks for each new method, including the constructor. Full marks if it is appropriately named, has an appropriate interface (parameters, return value), has appropriate Javadoc comment, has an appropriate implementation (including exceptions), and is declared `public`.

Testing: 2 marks. 1 mark for each new method. Full marks if each method's return value or effect is checked at least once.

Question 5 (30 points):

Purpose: System Design

Degree of Difficulty: Moderate

Task

The `PetStoreSystem` class. The last class to write is one that will run the pet store system. This class has one purpose: to be the main program that allows the user to carry out the following tasks:

1. Quit
2. Create a new animal in the system (The animal will not be assigned to a kennel yet)
3. Create a new staff member in the system
4. Assign an animal to a specific staff member (This will also assign the staff member to the animal)
5. Show all available kennels in the store.
6. Assign an animal to an empty kennel
7. Remove the assigned animal from a specific kennel
8. Drop the staff-animal association (i.e., the animal and the staff member are dropped from each other's lists)
9. Display the current system state (all animals, staff members and kennels in the system)

These tasks are numbered, and references to these tasks appear in the description below. Your system should display the above items to the console, and allow the user to enter a number from 1-9 to choose a task. Other parts of the system may require further input from the user, also through the console.

Our system can be implemented using the following instance variables (attributes):

- A single PetStore object.
- A keyed dictionary of all the animals in the system, where the key is the animal's animalID. (Hint: Use a TreeMap.)
- A keyed dictionary of all the staff members in the system, where the key is the staff member's employee ID. (Hint: Use a TreeMap.)

The numbered tasks above will be implemented largely by implementing an instance method for each task (some, like task 1, don't need a method). Here are the methods we will need:

- `PetStoreSystem()` A constructor for the class. Initially, there should be no animals or staff. The PetStore object needs to be created, and for that, the store name and the integer labels for the first and last kennels should be obtained from the user using console input and output.
- `addAnimal()` A method that executes Task 2.
- `addStaff()` A method that executes Task 3.
- `assignStaffToAnimal()` A method that executes Task 4.
- `showEmptyKennels()` A method stub for Task 5. **Leave this method as a stub. Implementing this method will be a part of a future assignment.**
- `assignKennel()` A method that executes Task 6.
- `releaseAnimal()` A method stub for Task 7. **Leave this method as a stub. Implementing this method will be a part of a future assignment.**

- `dropAssociation()` A method that executes Task 8.
- `systemState()` A method that executes Task 9.
- `toString()` A method that returns a string representation of all the information about the `PetStoreSystem` in a form suitable for printing. For example it may include information about the store, like a list of animals and staff.
- A main method. The main method should construct a new instance of the `PetStoreSystem` class. Then it should go into a loop in which the user is presented a menu (see above), allowing the user to enter a choice to perform one of the tasks. The loop should terminate if the user asks to quit (task 1); before exiting, the system should display the current state of the system (as in task 9).

  Note for this assignment you do not have to implement task 5 and task 7, but there should be method stubs for these tasks that prevent the system from crashing if these tasks are selected.

Hint: You can use the `TreeMap<K,V>` class in the `java.util` library to store a keyed dictionary. Unlike the other classes, the `PetStoreSystem` class gets information from the user instead of having passing in arguments to each method. Every method above (except for `toString()` will require user input, even the constructor!)

Additional information

System Tasks: For each of the tasks, animals will be identified by their animalID, staff by their employeeID, and kennels by their (external) integer label. If quit is selected, the system should print out the current system state before exiting. **Note:** When adding a new staff member, the user should be asked if the new staff is a manager or not. If so, a Manager should be created, if not a StaffMember should be created. Recall that, due to inheritance, an object of a certain type can be assigned to a variable of an ancestor type. A Manager can therefore be assigned to a StaffMember variable, and a Manager can be placed into the dictionary of StaffMembers.

Input and Output: You will need to use console input for this system in a number of different ways. The Scanner is the tool you need to use, but its behaviour is confusing for novices. Suppose you want to get an integer from the console (on a line by itself), followed by a whole line of text. You might write this:

```
int value = in.nextInt();        // get an integer on a line by itself
String line = in.nextLine();     // get the next line

System.out.println("The integer: " + val);
System.out.println("The line: <" + line + ">");
```

thinking that the first line gets the integer, and the second line gets the ine of text after the integer.

The problem is that this sensible expectation is not accurate. The Scanner will skip white space that appears on the line before the integer you asked for, but it will leave white space on the line after the value. As a result, the nextLine() immediately following will notice these white space characters, skip over them, realize that they mark the end of a line, and then return an empty String! As a matter of fact, it will seem like the Scanner just didn't work.

```
The integer: 13
The line: <>
```

So whenever you are trying to get a non-string value (integer, double) on a line by itself, and then an entire line of text on the next line, you have to force the Scanner to skip over the newline characters after the non-string value, and before the line you actually want:

```
int value = in.nextInt();        // get an integer on a line by itself
String junk = in.nextLine();     // skip junk at the end of the line
String line = in.nextLine();     // get the next line

System.out.println("The integer: " + val);
System.out.println("The line: <" + rest + ">");
```

In the above code, the user types an integer, followed by the newline character. The integer is scanned in correctly, and the junk is consumed (and not used in any way). Finally, the second line can be scanned all at once.

```
The integer: 13
The rest: <>
The line: <Preeow world!>
```

This kind of behaviour seems un-intuitive, and perplexing, but it is a consequence of the sophistication of the tool, and of the token-by-token process enabled by the Scanner.

What to Hand In

The completed `PetStoreSystem.java` program.

Evaluation

Attributes: 6 marks. 2 marks for each attribute. Full marks if it is appropriately named, has an appropriate type, has appropriate Javadoc comment, and is declared `private`.

Methods: 20 marks. 2 marks for each method, including the constructor as well as the method stubs for `showEmptyKennels()` and `releaseAnimal()`. Full marks if it is appropriately named, has an appropriate interface (parameters, return value), has appropriate Javadoc comment, has an appropriate implementation (including exceptions), and is declared `public`.

Main: 4 marks. Your main method presents the user with options, and then uses the PetStoreSystem methods to complete the tasks.

Question 6 (10 points):

Purpose: Practising preparation of External Documentation.

Degree of Difficulty: Easy.

External Documentation: A well-designed application always includes external documentation, i.e., a written document describing the application. We'll keep it simple for now, but external documentation is part of the design and implementation of any application.

For this assignment, your external documentation should consist of two documents: `A4_status.pdf` and `A4_manual.pdf`. You should summarize your whole assignment here.

- `A4_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.

- `A4_manual.pdf`. Maintenance manual. For this assignment (as with A3), it is sufficient to include a UML class diagram showing all the features of each class, and the relationships (inheritance, uses and aggregation) 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.

What to Hand In

- `A4_status.pdf`, outlining the status of your assignment, as described above.
- `A4_manual.pdf`, giving the UML diagrams for classes in questions 1- .

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 1-5, and the relationships between them. Full marks if the UML diagram is complete and accurately describes them.