Project 10: Pet Boarding – Part 2

Due: Skeleton Code (ungraded – checks class names, method names, parameters, and return types)
Completed Code – **Thursday, November 17, 2016 by 11:59 p.m.**

Deliverables

Your project files should be submitted to Web-CAT by the due date and time specified above (see the Lab Guidelines for information on submitting project files). You may submit your skeleton code files until the project due date but should try to do this by Friday (there is no late penalty since this is ungraded). You must submit your completed code files to Web-CAT before 11:59 PM on the due date for the completed code to avoid a late penalty for the project. You may submit your completed code up to 24 hours after the due date, but there is a late penalty of 15 points. No projects will be accepted after the one-day late period. If you are unable to submit via Web-CAT, you should e-mail your project Java files in a zip file to your lab instructor before the deadline. The Completed Code will be tested against your test methods in your JUnit test files and against the usual correctness tests. The grade will be determined, in part, by the tests that you pass or fail and the level of coverage attained in your Java source files by your test methods.

Files to submit to Web-CAT:

From Project 9
- Pet.java
- Cat.java, CatTest.java
- Dog.java, DogTest.java
- ServiceDog.java, ServiceDogTest.java
- Horse.java, HorseTest.java

New in Project 10
- PetNameComparator.java, PetNameComparatorTest.java
- BoardingCostComparator.java, BoardingCostComparatorTest.java
- PetBoardingList.java, PetBoardingListTest.java
- PetBoardingPart2.java, PetBoardingPart2Test.java

Recommendations

You should create a new folder for Project 10 and copy your relevant Project 9 source and test files to it. You should create a jGRASP project and add the class and test files as they are created.

Specifications – Use arrays in this project; ArrayLists are not allowed!

Overview: This project is Part 2 of three that will involve the boarding of pets. In Part 1, you developed Java classes that represent categories of pets including cats, dogs, service dogs, and horses. In Part 2, you will implement four additional classes: (1) PetNameComparator, which implements the Comparator interface; (2) BoardingCostComparator, which implements the Comparator interface; (3) PetBoardingList, which represents a list of pets and includes several specialized methods; and (4) PetBoardingPart2, which contains the main method for the program. Note that the main method in PetBoardingPart2 should declare a PetBoardingList object and then call the readPetFile method which will add pets to the PetBoardingList as the data is read in from a file. You can use

PetBoardingPart2 in conjunction with interactions by running the program in the canvas (or debugger with a breakpoint) and single stepping until the variables of interest are created. You can then enter interactions in the usual way. In addition to the source files, you will create a JUnit test file for each class and write one or more test methods to ensure the classes and methods meet the specifications. <u>You should create a jGRASP project upfront and then add the source and test files as they are created. All of your files should be in a single folder</u>.

- **Cat, Dog, ServiceDog, and Horse**

  Requirements and Design: No changes from the specifications in Project 9.

- **Pet.java**

  Requirements and Design: <u>In addition to the specifications in Project 9</u>, the Pet class should implement the Comparable interface for Pet, which means the following method must be implemented in Pet.
  - `compareTo`: Takes a Pet as a parameter and returns an int indicating the ordering from <u>lowest to highest</u> based on the owner's *last name* followed by *first name*. To do this, you will need to extract the owner's *last name* and *first name* from the owner field (assume that the first name is separated from the last name with a space and that there may be other spaces in the last name; e.g., Gigi de la Hoya). To avoid uppercase and lowercase issues you should make the extracted String values either all uppercase or all lowercase prior to comparing them.

- **PetBoardingList.java**

  Requirements: The PetBoardingList class provides methods for reading in the data file and generating reports.

  Design: The PetBoardingList class has fields, a constructor, and methods as outlined below.

  (1) **Fields:** All fields below should be private.
  (a) *listName* is the name of the PetBoardingList.
  (b) *petList* is an array that can hold up to 100 Pet objects.
  (c) *petCount* tracks the number of pet objects in the petList array.
  (d) *excludedRecords* is an array that can hold up to 30 String elements representing records that are read from the data file but not processed.
  (e) *excludedCount* tracks the number of records that have been added to the excludedRecords array.
  (f) listCount is a <u>static</u> field initialized to zero, which tracks the number of PetBoardingList objects created.

  (2) **Constructor:** The constructor has no parameters, initializes the instance fields in PetBoardingList, and increments the static field listCount. The readPetFile method defined below will populate the PetBoardingList object.

(3) **Methods:** Usually a class provides methods to access and modify each of its instance variables (i.e., getters and setters) along with any other required methods. The methods for PetBoardingList are described below.

- `getListName` returns the String representing the listName.
- `setListName` has no return value, accepts a String, and then assigns it to listName.
- `getPetList` returns the Pet array representing the petList.
- `setPetList` has no return value, accepts a Pet array, and then assigns it to petList (used in conjunction with `setPetCount` to provide a new Pet array and Pet count).
- `getPetCount` returns the current value of petCount.
- `setPetCount` has no return value, accepts an int, and assigns it to petCount.
- `getExcludedRecords` returns the String array representing the excludedRecords.
- `setExcludedRecords` has no return value, accepts a String array, and then assigns it to excludedRecords (used in conjunction with `setExcludedCount` to provide a new excluded records array and new excluded records count).
- `getExcludedCount` returns the current value of excludedCount.
- `setExcludedCount` has no return value, accepts an int, and sets excludedCount to it.
- `getListCount` is a static method that returns the current value of listCount.
- `resetListCount` is a static method that has no return value and sets listCount to 0.
- `readPetFile` has no return value and accepts the data file name as a String. Remember to include the `throws IOException` clause in the method declaration. This method creates a Scanner object to read in the file and then reads it in line by line. The first line contains the pet list name and each of the remaining lines contains the data for a pet. After reading in the list name, the "pet" lines should be processed as follows. A pet line is read in, a second scanner is created on the line, and the individual values for the pet are read in. After the values on the line have been read in, an "appropriate" Pet object is created. The data file is a "comma separated values" file; i.e., if a line contains multiple values, the values are delimited by commas. So when you set up the scanner for the pet lines, you need to set the delimiter to use a "," by calling the `useDelimiter(",")` method on the Scanner object. Each pet line in the file begins with a category for the pet (C, D, S, and H are valid categories for pets indicating **C**at, **D**og, **S**erviceDog, and **H**orse respectively). The second field in the record is the pet's owner, followed by the data for the name, breed, weight, and days to be boarded. The last items correspond to the data needed for the particular category (or subclass) of Pet. If the record has a valid category and has been successfully read in, the appropriate Pet object should be created and added to petList. If a record has an invalid category, no Pet is created and the record should be added to the excluded records array with an appropriate prefix message (`*** invalid category ***`) and its count should be incremented. The file *pet_boarding_data.csv* is available for download from the course web site. Below are example data records (the first line/record containing the pet list name is followed by pet lines/records):

```
Critter Sitter
C,Barb Jones,Callie,Siamese,9.0,7,9
D,Jake Smith,Honey,Great Dane,60.0,7
E,Jo Doe,Sugar,Spaniel,30.0,7
S,Jen Baker,Pepper,Sheppard,60.0,7,guide dog,sit,down,stay,come,around,forward,right,left
H,Jessie Rider,King,Quarter Horse,1000,7,10.0
S,Pat Atkins,Duke,Labrador Retriever,45.0,7,drug dog
```

- `generateReport` processes the pet list array using the original order from the file to produce the Pet Boarding Report which is returned as a String. See the example output on pages 7 - 9.
- `generateReportByOwner` makes a copy of the pet list array and sorts the copy using the natural ordering, and processes the sorted array to produce the Pet Boarding Report (by Owner) which is returned as a String. See the example output on pages 7 - 9.
- `generateReportByPetName` makes a copy of the pet list array and sorts the copy by pet name, and processes the sorted array to produce the Pet Boarding Report (by Pet Name) which is returned as a String. See the example output on pages 7 - 9.
- `generateReportByBoardingCost` makes a copy of the pet list array and sorts the copy by boarding cost, and processes the sorted array to produce the Pet Boarding Report (by Boarding Cost) which is returned as a String. See the example output on pages 7 - 9.
- `generateExcludedRecordsReport` processes the excludedRecords array to produce the Excluded Records Report which is returned as a String. See the example output on pages 7 - 9.

Code and Test: See examples of file reading and sorting (using Arrays.sort) in the class notes. The natural sorting order for Pet objects is determined by the compareTo method from the Comparable interface. If *petList* is the variable for the array of Pet objects, it can be sorted with the following statement.

```
Pet[] pList = Arrays.copyOf(petList, petCount);
Arrays.sort(pList);
```

The sorting order based on pet name is determined by the PetNameComparator class which implements the Comparator interface (described below). It can be sorted with the following statement.

```
Pet[] pList = Arrays.copyOf(petList, petCount);
Arrays.sort(pList, new PetNameComparator());
```

The sorting order based on boarding cost is determined by the BoardingCostComparator class which implements the Comparator interface (described below). It can be sorted with statements similar to those above, except that a new BoardingCostComparator is passed to the Arrays.sort method.

- **PetNameComparator.java**

  Requirements and Design: The PetNameComparator class implements the Comparator interface for Pet objects. Hence, it implements the following method.

  - `compare(Pet p1, Pet p2)` that defines the ordering from lowest to highest based on the name of the pet.

  Note that the *compare* method is the only method in the PetNameComparator class. An instance of this class will be used as one of the parameters when the Arrays.sort method is used to sort by "pet name". For an example of a class implementing Comparator, see class notes 10B Comparing Objects.

- **BoardingCostComparator.java**

  Requirements and Design: The BoardingCostComparator class implements the Comparator interface for Pet objects. Hence, it implements the following method.

  - `compare(Pet p1, Pet p2)` that defines the ordering from highest to lowest based on the boarding cost of each pet.

  Note that the *compare* method is the only method in the BoardingCostComparator class. An instance of this class will be used as one of the parameters when the Arrays.sort method is used to sort by "boarding cost". For an example of a class implementing Comparator, see class notes 10B Comparing Objects.

- **PetBoardingPart2.java**

  Requirements and Design: The PetBoardingPart2 class has only a main method as described below.

  - `main` reads in the file name as the first argument, args[0], of the command line arguments, creates an instance of PetBoardingList, and then calls the readPetFile method in the PetBoardingList class to read in the data file and generate the five reports as shown in the output examples beginning on page 7. If no command line argument is provided, the program should indicate this and end as shown in the first example output on page 7. An example data file, *pet_boarding_data.csv* can be downloaded from the Lab assignment web page, and this file will be available in Web-CAT. Any .csv data files that you create for your test methods will need to be uploaded to Web-CAT along with your source and test files.

  Code and Test: In your JUnit test file for the PetBoardingPart2 class, you should have at least two test methods for the main method. One test method should invoke PetBoardingPart2.main(args) where args is an empty String array, and the other test method should invoke PetBoardingPart2.main(args) where args[0] is the String representing the data file name. Depending on how you implemented the main method, these two test methods should cover the code in main. In the first test method, you should reset *listCount*, call your main method, then assert that `getListCount()` is zero for the test with no file name. In the second test method, you should reset *listCount*, call your main method assert that `getListCount()` is one with *pet_boarding_data.csv* as the file name.

  In the first test method, you can invoke main with no command line argument as follows:

```
// You should be checking for args.length == 0
// in PetBoardingPart2, and the following should exercise
// the code for true.
PetBoardingList.resetListCount();
```

```
String[] args1 = {};  // an empty String[]
PetBoardingPart2.main(args1);
Assert.assertEquals("Pet Boarding List count should be 0. ",
                    0, PetBoardingList.getListCount());
```

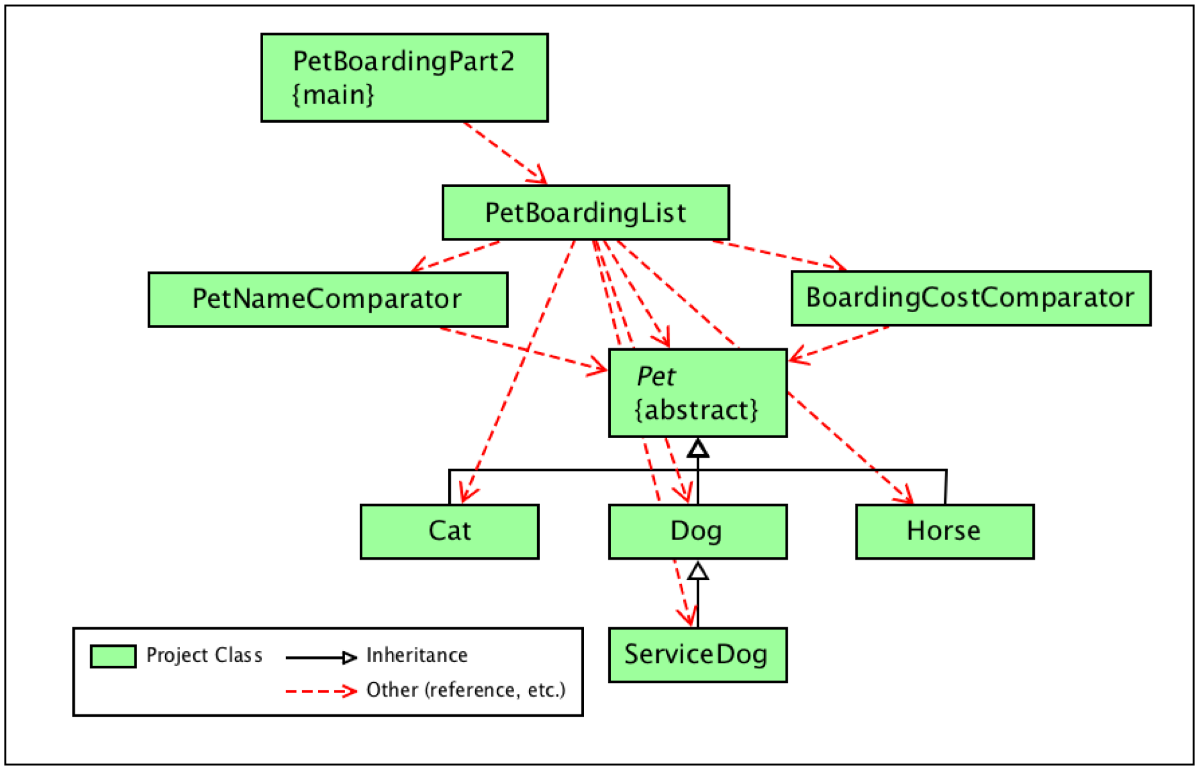
In the second test method, you can invoke main as follows with the file name as the first (and only) command line argument:

```
PetBoardingList.resetListCount();
String[] args2 = {"pet_boarding_data.csv"};
// element args2[0] is the file name
PetBoardingPart2.main(args2);
Assert.assertEquals("Pet Boarding List count should be 1. ",
                    1, PetBoardingList.getListCount());
```

If Web-CAT complains that the default constructor for `PetBoardingPart2` has not been covered, you should include the following line of code in one of your test methods.

```
// to exercise the default constructor
PetBoardingPart2 app = new PetBoardingPart2();
```

UML Class Diagram



Example Output when no file name is provided as a command line argument

```
 ----jGRASP exec: java PetBoardingPart2
File name expected as command line argument.
Program ending.

 ----jGRASP: operation complete.
```

Example Output when *pet_boarding_data.csv* is the command line argument

```
 ----jGRASP exec: java PetBoardingPart2 pet_boarding_data.csv

---------------------------------------
Pet Boarding Report for Critter Sitter
---------------------------------------

Owner: Barb Jones   Pet: Callie   Days: 7   Boarding Cost: $76.30
Cat: Siamese   Weight: 9.0 lbs   Lives Left: 9

Owner: Jake Smith   Pet: Honey   Days: 7   Boarding Cost: $105.00
Dog: Great Dane   Weight: 60.0 lbs

Owner: Jen Baker   Pet: Pepper   Days: 7   Boarding Cost: $168.00
ServiceDog: Sheppard   Weight: 60.0 lbs   Service: guide dog
Commands: sit down stay come around forward right left

Owner: Jessie Rider   Pet: King   Days: 7   Boarding Cost: $245.00
Horse: Quarter Horse   Weight: 1000.0 lbs   Exercise Fee: $10.00

Owner: Pat Atkins   Pet: Duke   Days: 7   Boarding Cost: $106.75
ServiceDog: Labrador Retriever   Weight: 45.0 lbs   Service: drug dog


---------------------------------------
Pet Boarding Report for Critter Sitter (by Owner)
---------------------------------------

Owner: Pat Atkins   Pet: Duke   Days: 7   Boarding Cost: $106.75
ServiceDog: Labrador Retriever   Weight: 45.0 lbs   Service: drug dog

Owner: Jen Baker   Pet: Pepper   Days: 7   Boarding Cost: $168.00
ServiceDog: Sheppard   Weight: 60.0 lbs   Service: guide dog
Commands: sit down stay come around forward right left
```

```
Owner: Barb Jones   Pet: Callie   Days: 7   Boarding Cost: $76.30
Cat: Siamese   Weight: 9.0 lbs   Lives Left: 9

Owner: Jessie Rider   Pet: King   Days: 7   Boarding Cost: $245.00
Horse: Quarter Horse   Weight: 1000.0 lbs   Exercise Fee: $10.00

Owner: Jake Smith   Pet: Honey   Days: 7   Boarding Cost: $105.00
Dog: Great Dane   Weight: 60.0 lbs



---------------------------------------
Pet Boarding Report for Critter Sitter (by Pet Name)
---------------------------------------

Owner: Barb Jones   Pet: Callie   Days: 7   Boarding Cost: $76.30
Cat: Siamese   Weight: 9.0 lbs   Lives Left: 9

Owner: Pat Atkins   Pet: Duke   Days: 7   Boarding Cost: $106.75
ServiceDog: Labrador Retriever   Weight: 45.0 lbs   Service: drug dog

Owner: Jake Smith   Pet: Honey   Days: 7   Boarding Cost: $105.00
Dog: Great Dane   Weight: 60.0 lbs

Owner: Jessie Rider   Pet: King   Days: 7   Boarding Cost: $245.00
Horse: Quarter Horse   Weight: 1000.0 lbs   Exercise Fee: $10.00

Owner: Jen Baker   Pet: Pepper   Days: 7   Boarding Cost: $168.00
ServiceDog: Sheppard   Weight: 60.0 lbs   Service: guide dog
Commands: sit down stay come around forward right left



---------------------------------------
Pet Boarding Report for Critter Sitter (by Boarding Cost)
---------------------------------------

Owner: Jessie Rider   Pet: King   Days: 7   Boarding Cost: $245.00
Horse: Quarter Horse   Weight: 1000.0 lbs   Exercise Fee: $10.00

Owner: Jen Baker   Pet: Pepper   Days: 7   Boarding Cost: $168.00
ServiceDog: Sheppard   Weight: 60.0 lbs   Service: guide dog
Commands: sit down stay come around forward right left

Owner: Pat Atkins   Pet: Duke   Days: 7   Boarding Cost: $106.75
ServiceDog: Labrador Retriever   Weight: 45.0 lbs   Service: drug dog

Owner: Jake Smith   Pet: Honey   Days: 7   Boarding Cost: $105.00
Dog: Great Dane   Weight: 60.0 lbs
```

```
Owner: Barb Jones   Pet: Callie   Days: 7   Boarding Cost: $76.30
Cat: Siamese   Weight: 9.0 lbs   Lives Left: 9



---------------------------------------
Excluded Records Report
---------------------------------------

*** invalid category *** E,Jo Doe,Sugar,Spaniel,30.0,7

 ----jGRASP: operation complete.
```