Project 11: Pet Boarding – Part 3

Due: Skeleton Code **(**ungraded – checks class names, method names, parameters, and return types)
Completed Code – **Thursday, December 1, 2016 by 11:59 p.m.** (Web-CAT correctness tests)

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

From Project 10
- PetNameComparator.java, PetNameComparatorTest.java
- BoardingCostComparator.java, BoardingCostComparatorTest.java
- PetBoardingList.java, PetBoardingListTest.java [modified as described below]

New in Project 11
- InvalidCategoryException.java
- PetBoardingPart3.java, PetBoardingPart3Test.java

Recommendations

You should create a new folder for Project 11 and copy your relevant Project 10 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 3 of three that involves 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 implemented four additional classes: (1) PetNameComparator, which implements the Comparator interface; (2) BoardingCostComparator, which implements the Comparator interface; and (3) PetBoardingList, which represents a list of pets and includes several specialized methods. In Part 3 (Project 11), you are to add exception handling. You will need to do the following: (1) create a new class named InvalidCategoryException which extends the Exception class, (2) add try-catch statements to catch a IOException in the main method of the PetBoardingPart3 class, and (3) modify the readPetFile method in the PetBoardingList class to catch/handle an InvalidCategoryException and a NumberFormatException in the event that either type of exception is thrown while reading the input file.

Note that the main method in PetBoardingPart3 should create a PetBoardingList object and then invoke the readPetFile method on that PetBoardingList object to read data from a file and add pets to the list. You can use PetBoardingPart3 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 the indicated source files and write one or more test methods to ensure the classes and methods meet the specifications. You should create a jGRASP project upfront and then add the source files as they are created. All of your files should be in a single folder.

- **Cat, Dog, ServiceDog, Horse, PetNameComparator, BoardingCostComparator**

  Requirements and Design: No changes from the specifications in Projects 9 and 10.

- **InvalidCategoryException.java**

  Requirements and Design: InvalidCategoryException is a user defined exception created by extending the Exception class. This exception is to be thrown and caught in the readPetFile method in the PetBoardingList class when a line of input data contains an invalid pet category. The constructor for InvalidCategoryException takes a single String parameter representing *category* and invokes the super constructor with the following String:

  ```
      "For category: " + "\"" + category + "\""
  ```

  This string will be the toString() value of an InvalidCategoryException when it occurs. For a similar constructor, see InvalidLengthException.java in 11_Exceptions\Examples\Polygons from this week's class notes.

- **PetBoardingList.java**

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

  Design: In addition to the specifications in Project 10, the existing readPetFile method must be modified to catch the following: InvalidCategoryException, NoSuchElementException, and NumberFormatException.

  - `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. For each *incorrect* line scanned (i.e., a line of data contains an invalid category, invalid numeric data, or missing data), your method will need to handle the invalid items properly.

- If the line of data begins with an invalid category, your program should throw an InvalidCategoryException (see description above). The code that adds a record with an invalid pet category to the excluded records array should be placed in the catch block for InvalidCategoryException.
- If a line of data has a valid category, but includes invalid numeric data (e.g., the value for *weight* contains an alphabetic character), a NumberFormatException (see notes on last page) will be thrown automatically by the Java Runtime Environment (JRE).
- If a line of data has a valid category but has missing data, a NoSuchElementException will be thrown automatically by the JRE.

The code in the while loop that reads in the pet records and checks for pet category should be in a try statement followed by three catch blocks: one for each of InvalidCategoryException, NumberFormatException, and NoSuchElementException. In each catch block, a String object should be created consisting of

```
        e + " in: " + line
```

where *e* is the exception and *line* is the record with the invalid data. The String object should be added to the excludedRecords array.

The file *pet_boarding_data2.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). Note that four of these pet records will each cause an exception to be thrown. These are listed in the "Excluded Records Report" on page 6.

```
Critter Sitter
C,Barb Jones,Callie,Siamese,9.0,7,9
H,Pam Racer,Trigger,Palomino,1200,14,none
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
D,Jackie Williams,Lassie,Collie
S,Pat Atkins,Duke,Labrador Retriever,45.0,7,drug dog
D,Sam Boone,Lucy,Beagle,24.5,eight days
```

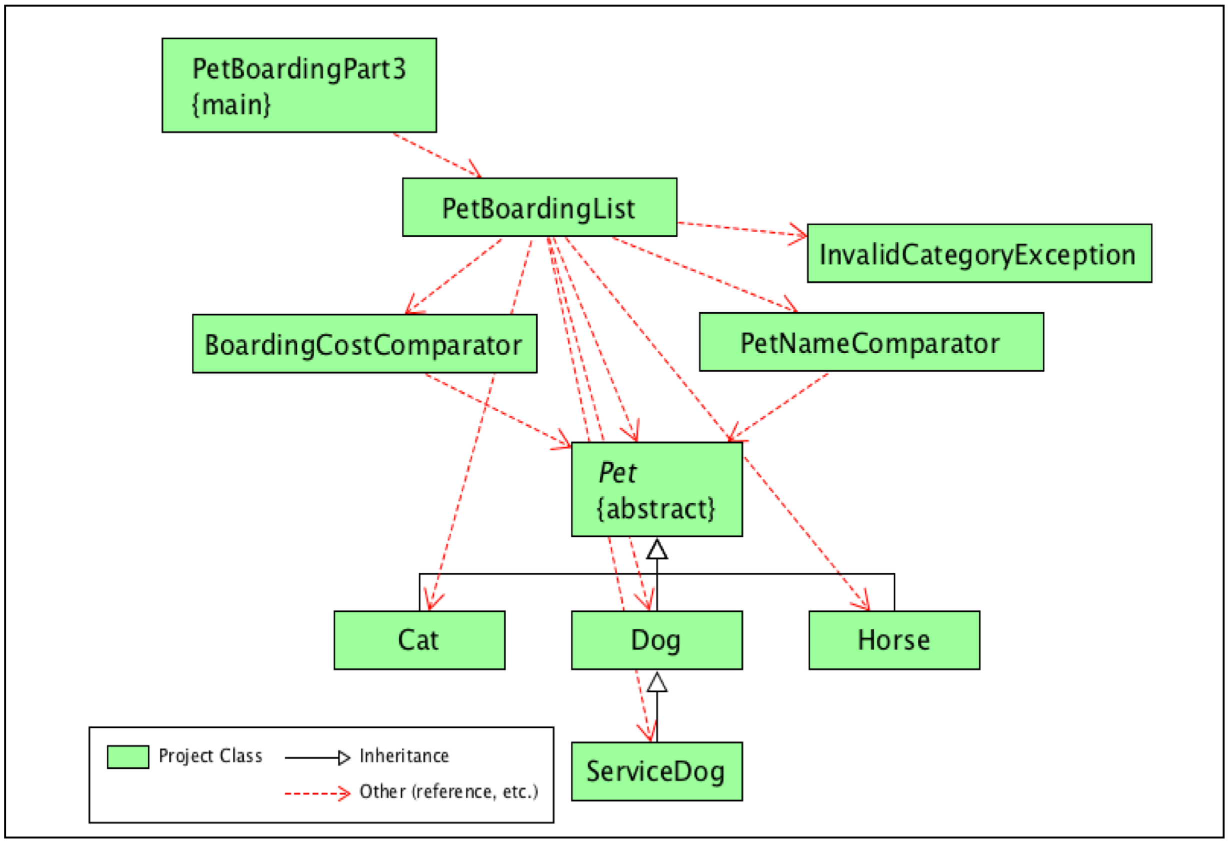
- **PetBoardingPart3.java**

  Requirements and Design: The PetBoardingPart3 class has only a main method as described below. In addition to the specifications in Project 10, the main method should be modified to catch and handle an IOException if one is thrown in the readPetFile method.

  - As before, the `main` method 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 the next page. The main method should not include the *throws IOException* in the declaration. Instead, the main method should include a try-catch statement to catch IOException when/if it is thrown in the readPetFile method in the PetBoardingList class. This exception is most likely to occur when an incorrect file name is passed to the readPetFile method. After this exception is caught and the appropriate message is printed in the main method, your program should end. See the second example output on the following page.

    Example data files can be downloaded from the Lab web page, and the program output for *pet_boarding_data2.csv* begins on the next page.

UML Class Diagram



Example Output when file name is missing as command line argument

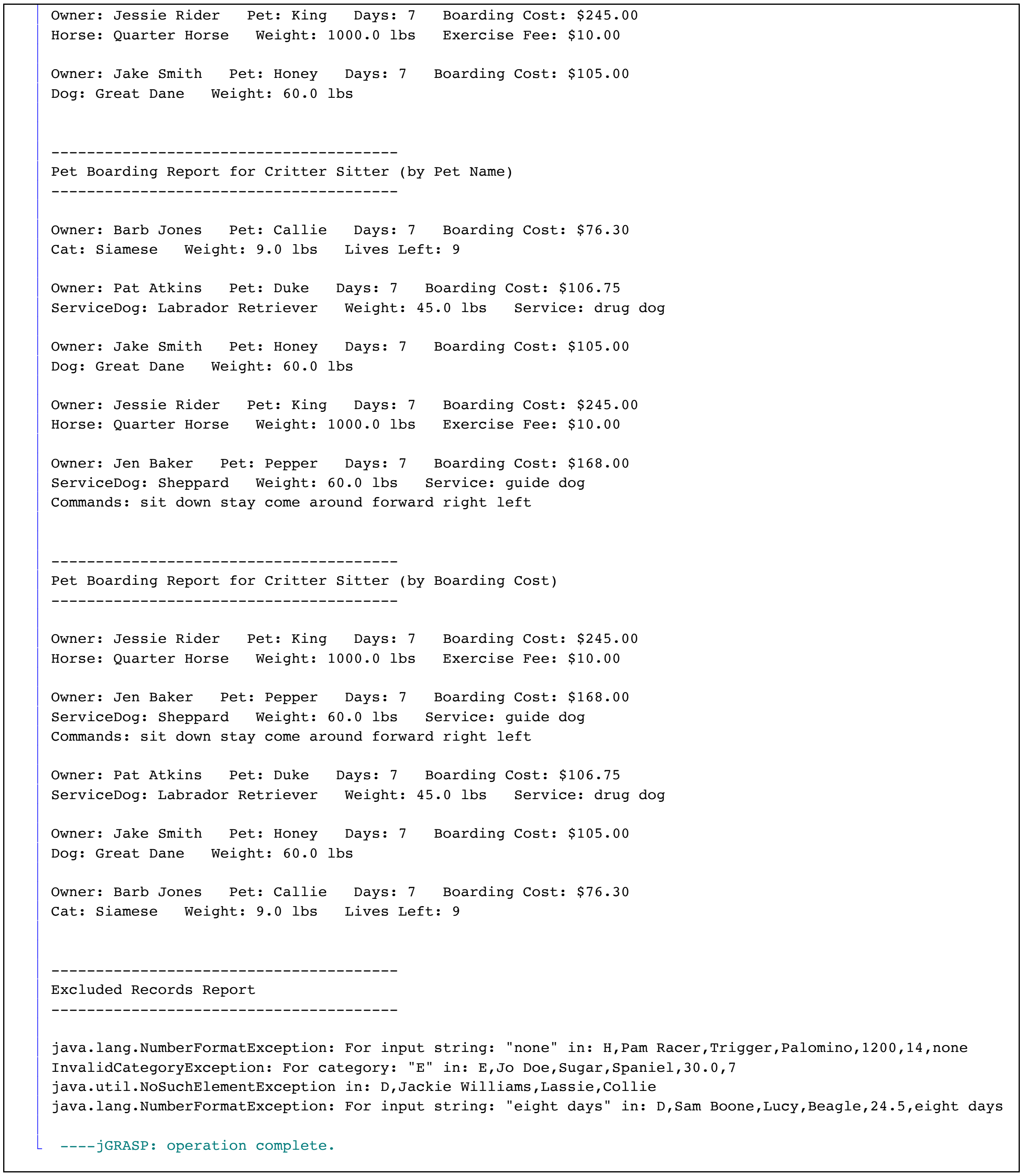
```
 ----jGRASP exec: java PetBoardingPart3
File name expected as command line argument.
Program ending.

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

Example Output when attempting to read a file that is not found (fake_data.csv)

```
 ----jGRASP exec: java PetBoardingPart3 fake_data.csv
Attempted to read file: fake_data.csv (No such file or directory)
Program ending.

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

Example Output for *pet_boarding_data2.csv*

```
 ----jGRASP exec: java PetBoardingPart3 pet_boarding_data2.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
---------------------------------------

java.lang.NumberFormatException: For input string: "none" in: H,Pam Racer,Trigger,Palomino,1200,14,none
InvalidCategoryException: For category: "E" in: E,Jo Doe,Sugar,Spaniel,30.0,7
java.util.NoSuchElementException in: D,Jackie Williams,Lassie,Collie
java.lang.NumberFormatException: For input string: "eight days" in: D,Sam Boone,Lucy,Beagle,24.5,eight days

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

Notes:

This project assumes that you are reading each double value as a String using next() and then parsing it into a double with Double.parseDouble(...) as shown in the following example.

. . . Double.parseDouble(myInput.next());

This form of input will throw a java.lang.NumberFormatException if the value is not a double.

If you are reading in each double value as a double using nextDouble(), for example

```
. . . myInput.nextDouble();
```

then a java.util.InputMismatchException will be thrown if the value read in is not a double.

You can either change your input to use Double.parseDouble(...) or you can catch the java.util.InputMismatchException and handle it the same way that you handled the NumberFormatException.

If you have mixed the two forms of input in your program and you want to keep both, then you will need to catch and handle both of the exceptions.