Project: Bunnies – Part 3

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 for this project). 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 - - test files are required (see note 3, p. 8):

From Bunnies – Part 1 (see note 4, p. 8) regarding static *count* in Bunny)

- Bunny.java [modify constructor to throw NegativeValueException]
- PetBunny.java, PetBunnyTest.java
- HouseBunny.java, HouseBunnyTest.java [modify constructor to throw NegativeValueException]
- JumpingBunny.java, JumpingBunnyTest.java [modify constructor to throw NegativeValueException]
- ShowBunny.java, ShowBunnyTest.java [modify constructor to throw NegativeValueException]

From Bunnies – Part 2

- BunnyList.java, BunnyListTest.java
- CostComparator.java, CostComparatorTest.java

New in Project 11

- NegativeValueException.java, NegativeValueExceptionTest.java
- BunniesPart3.java, BunniesPart3Test.java

Recommendations

You should create new folder for Part 3 and copy your relevant Part 1 and Part 2 source files to it (i.e., do not include BunniesPart1.java and BunniesPart2.java). You should create a jGRASP project and add these source files as well as those created in Part 3. You may find it helpful to use the "viewer canvas" feature in jGRASP as you develop and debug your program.

Specifications

Overview: This project is Part 3 of three that that will involve calculating the estimated monthly cost of owning a bunny where the amount is based on the type of bunny, its weight, and various additional costs. In Part 1, you developed Java classes that represent categories of bunny: pet bunny, house bunny (a subclass of pet bunny), jumping bunny, and show bunny. In Part 2, you implemented three additional classes: (1) CostComparator that implements the Comparator interface, (2) BunnyList that represents a list of bunnies and includes several specialized methods, and (3) BunniesPart2 which

contains the main method for the program. Note that the main method in BunniesPart2 should create a BunnyList object, read the data file using the readBunnyFile method. BunniesPart2 then prints the summary, the bunnies listed by name and the bunnies listed by estimated monthly cost, and the list of excluded records. In Part 3 (this project), you are to add exception handing and invalid input reporting. You will need to do the following: (1) create a new class named NegativeValueException which extends the Exception class, (2) add try-catch statements to catch FileNotFoundException in the main method of the BunniesPart3 class, and (3) modify the readBunnyFile in the BunnyList class to catch/handle NegativeValueException, NumberFormatException, and NoSuchElementException in the event that these type exceptions are thrown while reading the input file.

Note that the main method in BunniesPart3 should create a BunnyList object and then invoke the readBunnyFile method on the BunnyList object to read data from a file and add bunnies to the bunnyList array in the BunnyList object. You can use BunniesPart3 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. 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.

- **Bunny, PetBunny, HouseBunny, JumpingBunny, and ShowBunny**

  Requirements and Design: The constructors for these classes must be modified to check numeric parameters specific to their respective classes for negative values and to throw a NegativeValueException as appropriate. For example, if -8.5 is passed into the Bunny constructor (via *super*) as the bunny's *weight*, the constructor should throw a NegativeValueException. A second example, if -0.5 is passed into the HouseBunny constructor as *wearAndTear*, the constructor should throw a NegativeValueException. Since these constructors for Bunny, PetBunny, HouseBunny, JumpingBunny, and ShowBunny are not catching this exception, they must include NegativeValueException in the throws clauses of their respective headers. Since the Bunny constructor gets called when any constructors in Bunny subclasses are called, subclass constructors must also have throws negativeValueException in the constructor headers.

  Testing: Since the constructors in Bunny, PetBunny, HouseBunny, JumpingBunny, and ShowBunny may throw a NegativeValueException, any method that calls one of these constructors must either catch the exception or it must throw the exception (i.e., include NegativeValueException in a throws clause for your test method headers).

- **NegativeValueException.java**

  Requirements and Design: NegativeValueException is a user defined exception created by extending the Exception class with an empty body. The constructor for NegativeValueException should be public and parameterless, and it should invoke the super constructor with the String message `"Numeric values must be nonnegative"`. The inherited toString() value of a NegativeValueException will be the name of the exception and the message. This exception is to

be caught in the readBunnyFile method in the BunnyList class when a line of input data contains a negative value for one of the numeric input values: weight, wearAndTear, trainingCost, groomingCost. The NegativeValueException is to be thrown in the "bunny" constructor that is responsible for setting the field in question. The following shows how the constructor would be called: `new NegativeValueException()` For a similar constructor, see InvalidLengthException.java in Examples/Polygons from this week's lecture notes on Exceptions.

Testing: Here is an example of a test method that checks to make sure a negative value for *value* in the constructor for Bunny throws a NegativeValueException. Note that creating a HouseBunny invokes the constructor in Bunny. You should consider adding test methods to check for negative values for the other numeric fields.

```
@Test public void negativeValueExceptionTest() {
   boolean thrown = false;
   try {
      HouseBunny hb = new HouseBunny("Spot", "Mixed", -0.08);
   }
   catch (NegativeValueException e) {
      thrown = true;
   }
   Assert.assertTrue("Expected NegativeValueException to be thrown.",
                     thrown);
   /* or alternatively: */
   Assert.assertEquals("Expected NegativeValueException to be thrown.",
         true, thrown);
}
```

- **BunnyList.java**

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

  Design: In addition to the specifications in Bunnies – Part 2, the existing readBunnyFile method must be modified to catch following: NegativeValueException, NumberFormatException, and NoSuchElementException. When these exceptions occur, an appropriate message along with the offending line/record should be added the excludedRecords array.

  - `readBunnyFile` has no return value, accepts the data file name as a String, and has a throws clause for FileNotFoundException. This method creates a Scanner object to read in the file and then reads it in line by line. The first line of the file contains the name of the list, and each of the remaining lines contains the data for a bunny. After reading in the list name, the "bunny" lines should be processed as follows. A bunny line (or record) is read in, a second Scanner is created on the line, and the individual values for the bunny are read in. Be sure to "trim" each value read in. All values should be read as strings. Non-String values should be "parsed" into their respective values using the appropriate wrapper class (e.g., Double.parseDouble(..)). After the values on the line have been read in, an "appropriate" bunny object is created and added to the bunny array using the addBunny method. If the

bunny type is not recognized, the message `"Invalid Bunny Category in:\n"` and the record/line should be added to the excluded records array using the addExcludedRecord method. The data file is a "comma separated values" file; i.e., if a line contains multiple values, the values are delimited by commas. So after you set up the Scanner for the bunny lines, you need to change the delimiter to a "," by invoking `useDelimiter(",")` on the Scanner object. Each bunny line in the file begins with a category for the bunny. Your switch statement should determine which type of Bunny to create based on the first character of the category (i.e., P, H, J, and S for PetBunny, HouseBunny, JumpingBunny, and ShowBunny respectively, ignoring case). The second field in the record is the name, followed by breed, and weight, as well the values appropriate for the category of bunny represented by the line of data. That is, the items that follow weight correspond to the data needed for the particular category (or subclass) of Bunny. For each *incorrect* line scanned (i.e., a line of data that contain missing data or invalid numeric data), your method will need to handle the invalid items properly. If a line includes invalid numeric data (e.g., the value for *weight*, a double, contains an alphabetic character), a NumberFormatException (see note 1, p. 8) will be thrown automatically by the Java Runtime Environment (JRE). Your readBunnyFile method should catch and handle NumberFormatException, NoSuchElementException (for missing values), and NegativeValueException (thrown in bunny constructors when a negative value is passed in) as follows. In each catch clause, a String consisting of the exception, the comment `" :\n"`, and the *line* with the invalid data should be added to the excludedRecords array. For example, in the catch clause for NumberFormatException e, the String resulting from the following expression should be added to the excludedRecords array.

```
e + " in:\n" + line
```

The file *bunnies2.txt* is available for download from the course web site. Below are example data records (the first line/record containing the bunny list name is followed by bunny lines/records):

```
Bunny Collection
Pet bunny, Floppy, Holland Lop, 3.5
Pet bunny, Hopper, Holland Lop Mixed, -4.5
Pet bunny, Hopster, Holland Lop Mixed, 3..8
house Bunny, Spot, Really Mixed, 5.8, 0.15
mouse Bunny, Spots, Mixed, 0.8, 0.15
House Bunny, Spotster, Sorta Mixed, 5.8, -0.25
House Bunny, Sam, Mixed, 5.8
Jumper Bunny, Speedy, English, 6.3, 25.0
Jumper Bunny, Speeder, English, 6.1, -15.0
fighting bunny, Slugger, Big Mixed, 16.5, 21.0
Show bunny, Bigun, Flemish Giant, 14.6, 22.0
show bunny, Big John, Flemish Giant, 15.6, -20.1
```

- **BunniesPart3.java**

  Requirements and Design: The BunniesPart3 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 FileNotFoundException if one is thrown in the readBunnyFile method of the BunnyList class.

  In Part 3, `main` reads in the file name from the command line as was done in Bunnies – Part 2, creates an instance of BunnyList, and then calls the readBunnyFile method in the BunnyList class to read in the data file. After successfully reading in the file, the main method then prints the summary, bunny list by name, bunny list by cost, and the list of excluded records. The main method should not include the *throws FileNotFoundException* in the declaration. Instead, the main method should include a try-catch statement to catch FileNotFoundException when/if it is thrown in the readBunnyFile method in the BunnyList class. This exception will occur when an incorrect file name is passed to the readBunnyFile method. After this FileNotFoundException is caught in main, print the messages below and end.

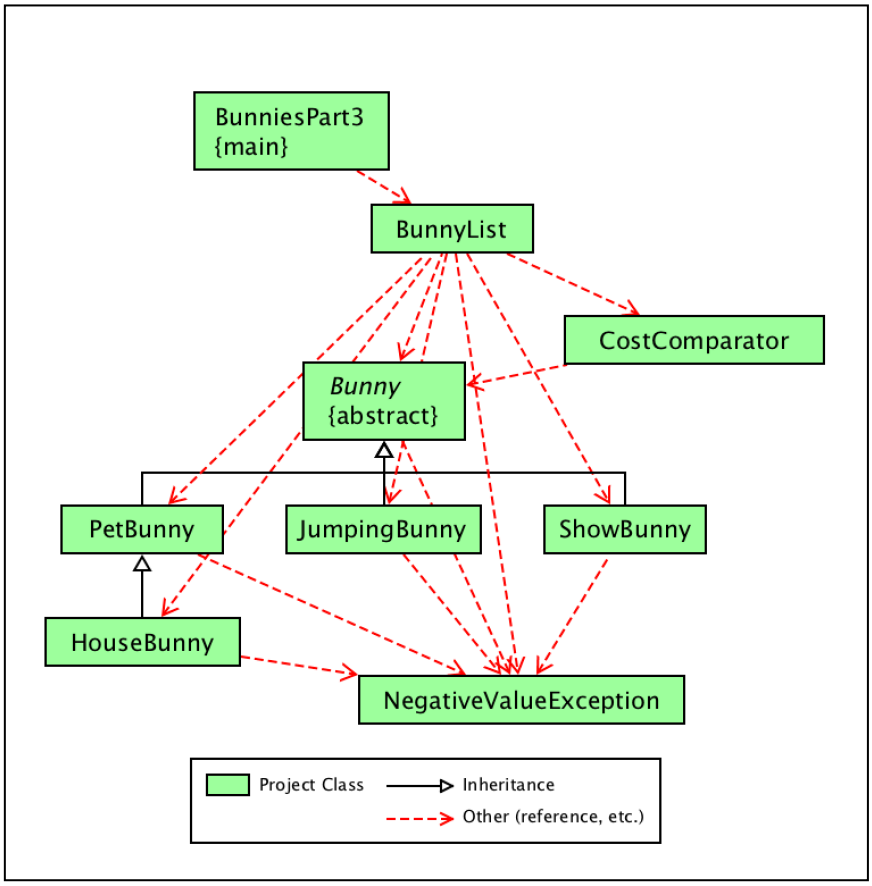
```
*** File not found.
Program ending.
```

  Also, if the user runs the program without a command line argument (e.g., `args.length == 0`), main should print the following message and end.

```
*** File name not provided by command line argument.
Program ending.
```

  An example data file can be downloaded from the Lab web page. The program output for *bunnies2.txt* begins on the next page after the UML class diagram. See note 2 on p. 8 for testing your main method.

UML Class Diagram



Example Output for bunnies2.txt

```
 ----jGRASP exec: java BunniesPart3 bunnies2.txt
------------------------------
Summary for Bunny Collection
------------------------------
Number of Bunnies: 4
Total Estimated Monthly Cost: $124.38


------------------------------
Bunnies by Name
------------------------------

Bigun (ShowBunny)   Breed: Flemish Giant   Weight: 14.6
Estimated Monthly Cost: $62.15 (includes $22.00 for grooming)

Floppy (PetBunny)   Breed: Holland Lop   Weight: 3.5
Estimated Monthly Cost: $6.48

Speedy (JumpingBunny)   Breed: English   Weight: 6.3
Estimated Monthly Cost: $40.75 (includes $25.00 for training)

Spot (HouseBunny)   Breed: Really Mixed   Weight: 5.8
Estimated Monthly Cost: $15.01 (includes 15.0% for wear and tear)
```

```
------------------------------
Bunnies by Cost
------------------------------

Floppy (PetBunny)   Breed: Holland Lop   Weight: 3.5
Estimated Monthly Cost: $6.48

Spot (HouseBunny)   Breed: Really Mixed   Weight: 5.8
Estimated Monthly Cost: $15.01 (includes 15.0% for wear and tear)

Speedy (JumpingBunny)   Breed: English   Weight: 6.3
Estimated Monthly Cost: $40.75 (includes $25.00 for training)

Bigun (ShowBunny)   Breed: Flemish Giant   Weight: 14.6
Estimated Monthly Cost: $62.15 (includes $22.00 for grooming)


------------------------------
Excluded Records
------------------------------

NegativeValueException: Numeric values must be nonnegative in:
Pet bunny, Hopper, Holland Lop Mixed, -4.5

java.lang.NumberFormatException: multiple points in:
Pet bunny, Hopster, Holland Lop Mixed, 3..8

Invalid Bunny Category in:
mouse Bunny, Spots, Mixed, 0.8, 0.15

NegativeValueException: Numeric values must be nonnegative in:
House Bunny, Spotster, Sorta Mixed, 5.8, -0.25

java.util.NoSuchElementException in:
House Bunny, Sam, Mixed, 5.8

NegativeValueException: Numeric values must be nonnegative in:
Jumper Bunny, Speeder, English, 6.1, -15.0

Invalid Bunny Category in:
fighting bunny, Slugger, Big Mixed, 16.5, 21.0

NegativeValueException: Numeric values must be nonnegative in:
show bunny, Big John, Flemish Giant, 15.6, -20.1


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

Example Output for bunnies_not_a_real_file.txt

```
 ----jGRASP exec: java BunniesPart3 bunnies_not_a_real_file.txt
*** File not found.
Program ending.
 ----jGRASP: operation complete.
```

Example Output for missing command line argument

```
----jGRASP exec: java BunniesPart3
*** File name not provided by command line argument.
Program ending.

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

Notes

1. **Exceptions for numeric items –** This project assumes that you are reading each double value as 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 an 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 is not a double. Since an InputMismatchException is a subclass of NoSuchElementException, this exception will be caught in your catch clause for NoSuchElementException but will be reported as an InputMismatchException.

   Web-CAT is looking for NumberFormatException rather than InputMismatchException. Therefore, you must use Double.parseDouble(...) as described above to in order to pass the tests in Web-CAT.

2. **Testing your main method –** You will need three test methods for BunniesPart3Test.java: (1) test with a good file name, (2) test with a bad file name, and (3) test with no file name (i.e., the user did not provide a command line argument). In the latter two cases, the bunnyCount should be zero after calling the main method in BunniesPart3.

3. **General note on test files** – The data files for Part 2 (bunnies1.txt) and Part 3 (bunnies2.txt) have been uploaded in Web-CAT. If you have test methods that read bunnies1.txt, you can retain these and then add new test methods that read bunnies2.txt as needed.

4. **Static count in Bunny** – If you are incrementing bunnyCount in the Bunny constructor you may find it necessary to decrement bunnyCount when a Bunny subclass detects a negative value. The bunnyCount should be decremented within any subclass constructor that has an if statement checking for a negative value. That is, if true, bunnyCount should be decremented then the program should have a statement: `throw new NegativeValueException();`
   This will abandon the constructor and the object will not be created.

5. NegativeValueException exception is thrown in one of the subclass constructors. The exception causes the constructor to end, and thus the instance is not created. However, since the call to the super constructor incremented bunnyCount before the exception in a subclass, bunnyCount should be decremented in the subclass where the NegativeValueException exception is thrown.

6. **Skeleton Code (ungraded)** – You can submit to this Web-CAT assignment to check the coverage of your test methods. Just submit your project test files along with your project source files.

7. **Assert.assertArrayEquals** – When the Assert.assertArrayEquals is used in a JUnit test method, it does an element by element compare of the arrays using the equals method with type Object as the parameter. Thus, it is important that the equals method in the Bunny class overrides the equals method inherited from the Object class and that it is working correctly. Otherwise, using Assert.assertArrayEquals will likely not work properly. For example, if you have not overridden the equals method from the Object class, since the elements are objects rather than primitives, the addresses are compared instead of the objects' fields themselves. This means that you are likely get a *false* since the addresses are only equal if the references are aliases. Therefore, to make the Assert.assertArrayEquals work for arrays of Bunny, the *equals* method inherited from Object should be overridden in the Bunny class.