Homework 06 – Look at all Those Chickens!

Problem Description

Hello! Please make sure to read all parts of this document carefully.

You find yourself in a feathery situation. When you signed up for birdwatching, you didn't exactly sign up for this. You are now bird-sitting for your next-door neighbors. They provide you with a list of their birds, but it appears to have a few errors! By the end of the day, you are expected to update their list of birds, as well as provide a log of the day's events. To do this, you will create **InvalidBirdException.java**, **Bird.java**, **Goose.java**, **Pigeon.java**, and **BirdSitting.java** to simulate your day spent with the birds. We will provide an example **BirdDriver.java** file, which will not be turned in. To complete this assignment, you will use your knowledge of abstract classes, inheritance, polymorphism, file I/O, and exceptions.

Remember to test for Checkstyle Errors and include Javadoc Comments!

Solution Description

`Provided Files`

`birds.csv` - A comma separated values (csv) file representing an example list of birds

- Note: For csv files, we recommend opening them in notepad rather than Excel, as there are times where Excel does not properly display the values and formatting

birdCare.csv - A comma separated values (csv) file representing an example list of bird care instructions

- Two example files for birdCare.csv have been provided for this homework
- Same note as above for csv files

Note: The files you will be interacting with will be in the following formats:

- Input: A list of the birds with their information (csv)
  - Each line represents one bird
  - Have the following csv line format
    - name
    - type of bird
    - bird info (hunger, happiness, honkPower/homeCity)
  - Examples of valid lines are below (see more in provided csv file):
    - `Loona,pigeon,90,80,San Francisco`
    - `Cheeto,goose,99,70,40`
- Input: Bird-sitting instructions (csv)
  - The bird's name
  - Bird-sitting instructions (pet or feed)
    - These instructions will correspond with which method to invoke on the bird
    - You can assume that this file will have no errors, and the bird-sitting instructions will exclusively be either pet or feed
    - The birds and their corresponding care instructions will not necessarily be in order between the two csv files. Therefore, a search would be required.

        - The instructions are guaranteed to have instructions for every valid bird.
    - Here are examples of valid lines for this file:
        - `Bingo,feed`
        - `Peaches,pet`
- Output: Updated list of birds at the end of the day (csv)
    - This updated list should exclude any birds that were not valid
    - The lines must follow the format of the original birds list
- Output: Log of events (txt)
    - This file will be formatted in readable sentences for easy understanding

`InvalidBirdException.java`

This is an **unchecked exception** with two constructors. This exception will be thrown when the specified bird is neither a Pigeon nor a Goose. Use **constructor chaining** for the two following constructors.

Constructors

- `InvalidBirdException()` - has the following message:
    - `Your bird is invalid!`
- `InvalidBirdException(String s)` - has the message as is the String passed into the parameters

`Bird.java`

This class represents a Bird. You must not be able to create an instance of this class (there is a keyword that prevents us from creating instances of a class).

Variables

Variables must be not allowed to be directly modified outside the class in which they are declared, unless otherwise stated in the description of the variable. **Hint**: there is a specific visibility modifier that can do this!

The Bird class must have these variables.

- `String name` - the name of the bird
- `int hunger` - how hungry the bird is
    - Value of the hunger is within the range of [0, 100]
    - You can assume that the constructor will always receive values within this range
    - You are not required to enforce this in the constructor, which means that any value outside of the range can be left as-is
- `int happiness` - how happy the bird is
    - Value of happiness is within the range of [0, 100]
    - Similar to the previous instance variable, you can assume that the constructor will always receive values within this range

Constructors

There will be one constructor for this class. The constructor ***must*** take the variables in the specified order.

- A 3-arg constructor that takes in `name`, `hunger`, and `happiness`, setting all variables appropriately.

Methods

Do not create any other methods than those specified. Any extra methods will result in point deductions. All methods must have the proper visibility to be used where it is specified they are used.

- `public String speak()`
  - The bird expresses how it feels through a series of bird sounds
  - The sound that the bird makes is unique to the bird species
  - Any concrete class that extends the Bird class must provide a method definition for this method
- `public String receivePets()`
  - The bird is cared for via prescribed pets
  - This method returns a String that represents the bird's reaction
  - Any concrete class that extends the Bird class must provide a method definition for this method
- `public String eatFood()`
  - The bird is cared for via a gourmet meal!
  - This method returns a String that represents the bird's reaction
  - Any concrete class that extends the Bird class must provide a method definition for this method
- `public boolean equals(Object o)`
  - Two Birds are equal if they have the same values for `name`, `hunger`, and `happiness`
  - Must override equals() method defined in Object class
- Getters and setters as necessary.

`Goose.java`

This class represents a Goose, which is a type of Bird and should have all the attributes of one.

Variables

All variables must be not allowed to be directly modified outside the class in which they are declared, unless otherwise stated in the description of the variable. **Hint**: there is a specific visibility modifier that can do this!

The Goose class must have the following variable:

- `int honkPower` – the goose's honk power

Constructors

You **must use constructor chaining** for your constructors. The constructors ***must*** take the variables in the specified order. (Hint: Refer to L10: Constructors as L12: Coding Basics and More Visibility Modifiers for constructor chaining and constructor chaining when a superclass is involved).

- A 4-arg constructor that takes in `name`, `hunger`, `happiness`, and `honkPower`, setting all variables accordingly.
- A 1-arg constructor that takes in just a `name` and assigns the following default values:
  - `hunger`: 70
  - `happiness`: 20
  - `honkPower`: 8

Methods

Do not create any other methods than those specified. Any extra methods will result in point deductions. All methods must have the proper visibility to be used where it is specified they are used.

- `public String speak()`
  - The bird expresses how it feels through a series of bird sounds
  - Returns a String of "honk" based on the goose's honkPower
    - All caps if the goose's happiness is less than 30
    - For example, for a goose with a honkPower of 6 and a happiness of 0.1 will have a speak method that returns:
      - `HONK HONK HONK HONK HONK HONK`
- `public String receivePets()`
  - The bird is cared for via prescribed pets!
  - Its happiness increases by 20
    - If happiness exceeds 100, set it to 100
  - If you pet the goose when its happiness is greater than or equal to 80 (before the happiness is increased), the goose hands you a gift when pet
    - Returns a String in the format:
      - `[name] waddles off and hands me a flower after being pet.`
  - Otherwise, return a String in the format:
    - `[name] honks at me for more pets.`
- `public String eatFood()`
  - The bird is cared for via a gourmet meal!
  - If the goose is full and has a hunger of 0, its happiness decreases by 10.
    - If happiness drops below 0, set it to 0
    - Returns a String in the format:
      - `[name] was not hungry but ate a little food anyways.`
  - Otherwise, set its hunger to 0 and return the String in the format:
    - `[name] thoroughly enjoyed the meal.`
- `public boolean equals(Object o)`
  - Two Birds are equal if they have the same values for `name`, `hunger`, `happiness`, and `honkPower`
  - Must override equals() method defined in the superclass, as well as call on the superclass's equal method.
- Getters and setters as necessary.

`Pigeon.java`

This class represents a Pigeon, which is a type of Bird and should have all the attributes of one.

Variables

All variables must be not allowed to be directly modified outside the class in which they are declared, unless otherwise stated in the description of the variable. **Hint**: there is a specific visibility modifier that can do this!

The Pigeon class must have the following variable:

- `String homeCity` – the pigeon's home city

Constructors

You **must use constructor chaining** for your constructors. The constructors ***must*** take the variables in the specified order. (Hint: Refer to L10: Constructors as L12: Coding Basics and More Visibility Modifiers for constructor chaining and constructor chaining when a superclass is involved).

- A 4-arg constructor that takes in `name`, `hunger`, `happiness`, and `homeCity`, setting all variables accordingly.
- A 1-arg constructor that takes in just a `name` and assigns the following default values:
  - `hunger`: 90
  - `happiness`: 90
  - `homeCity`: New York

Methods

Do not create any other methods than those specified. Any extra methods will result in point deductions. All methods must have the proper visibility to be used where it is specified they are used.

- `public String speak()`
  - The bird expresses how it feels through a series of bird sounds
  - This method returns a String as following:
    - `coo coo coo`
  - In addition, if the pigeon's hunger is greater than 40, return the String in all caps.
- `public String receivePets()`
  - You attempt to pet the pigeon.
  - However, being a pigeon of the city, it does not welcome being pet.
  - This method returns a String in the format:
    - `[name] was displeased and flew away when I tried to pet it.`
  - Any concrete class that extends the Bird class must provide a method definition for this method
- `public String eatFood()`
  - Let's feed the pigeon some high-quality food.
  - Pigeons are always ready to consume more bread. The pigeon's hunger is set to 0.
  - Additionally, the pigeon's happiness increases by 50
    - If the happiness exceeds 100, set it to 100
  - This method returns a String in the format:
    - `[name] happily pecked up the bread.`
- `public boolean equals(Object o)`
  - Two Birds are equal if they have the same values for `name`, `hunger`, `happiness`, and `homeCity`
  - Must override equals() method defined in the superclass, as well as call on the superclass's equal method.
- Getters and setters as necessary.

`BirdSitting.java`

This class represents your time bird-sitting and contains methods to open and parse csv files. As each bird is interacted with, their reactions will be added to the day's log and saved to a txt file.

Variables

All variables must be not allowed to be directly modified outside the class in which they are declared, unless otherwise stated in the description of the variable. **Hint**: there is a specific visibility modifier that can do this!

The BirdSitting class must have these variables:

- `File birdsList` - the file that contains the information for all of the birds

Constructors

When applicable, you **must use constructor chaining** for your constructors. The constructors ***must*** take the variables in the specified order. (Hint: Refer to L10: Constructors as L12: Coding Basics and More Visibility Modifiers for constructor chaining and constructor chaining when a superclass is involved).

- A 1-arg constructor that takes in a File object of the birds list file as a parameter and sets fields accordingly
    - Each bird has its own line (see Provided Files section for more information)
- A 1-arg constructor that takes in a String representing the name of a file (including the file extension)
    - This constructor must chain to the other constructor
    - Hint: this should be done in 1-2 lines

Methods

For this homework, the use of **additional private helper methods** is highly encouraged. All methods must have the proper visibility to be used where it is specified they are used.

- `public void startBirdSitting(File birdCare)`
    - This method throws a FileNotFoundException
    - This method goes through birdsList for each bird
    - For each bird, you would need to instantiate the bird and perform the specified method of care as specified in the birdCare file by calling the appropriate method
        - Hint: Use the split() method for the Strings
        - If the specified bird in birdsList is neither a goose nor pigeon, throw an InvalidBirdException
        - As this method would be run through a driver class, a try-catch block would be used in the main method of the driver class to handle the exception
    - Afterwards update the birds list, both updating the instance variable and overwriting the old file
        - Hint: Keep track of all the updated values for the bird information and instance variables in a String
        - Note: Do not try to read the file and write to it at the same time – this method is intended to rewrite the file
    - Afterwards also call summarizeDay() to create a new txt file containing the bird responses
        - The new file should be named as the original birdCare file name with "_summary.txt" appended to the end
            - Example: `birdCare1_summary.txt`
        - Hint: Keep track of the bird responses in a String variable as they are returned from the receivePets() and eatFood() methods
- `public void startBirdSitting(String fileName)`
    - This method chains to the overloaded version of this method that accepts a File
- `private void summarizeDay(String birdResponses, String summaryDestination)`
    - This method saves the content contained in the String passed in to the file destination as specified

    - Be sure to include the new line characters
- `public void birdInventory()`
    - This method instantiates all birds in birdsList
    - For each bird, call and print the returned String of their speak() method to check up on how they are doing
    - Hint: An array can be used here
- Getters and setters as necessary.

Example Output

Run the provided **BirdDriver.java**. For this homework, we are providing one example birds.csv file and two example birdCare.csv files. Keep in mind that getting the same output does not guarantee full points on this assignment, as this is only one example. You are encouraged to test out different scenarios.

The content of the output of startBirdsitting(“birdCare1.csv”) should match the following:

```
Content of updated birds.csv
```

```
Bingo,pigeon,0,50,Reykjavik
Plato,pigeon,80,70,Athens
Walnut,goose,0,90,3
Hawk,pigeon,0,100,Atlanta
Pizza,goose,90,30,6
Tango,goose,40,50,2
```

```
Content of birdCare1_summary.txt
```

```
Bingo happily pecked up the bread.
Plato was displeased and flew away when I tried to pet it.
Walnut thoroughly enjoyed the meal.
Hawk happily pecked up the bread.
Pizza honks at me for more pets.
Tango honks at me for more pets.
```

```
Output in command line
```

```
coo coo coo
COO COO COO
honk honk honk
coo coo coo
honk honk honk honk honk honk
honk honk
```

The content of the output of startBirdsitting(“birdCare2.csv”) should match the following:

```
Content of updated birds.csv
```

```
Bingo,pigeon,65,0,Reykjavik
Plato,pigeon,0,100,Athens
Walnut,goose,20,100,3
Hawk,pigeon,10,90,Atlanta
Pizza,goose,0,10,6
Tango,goose,0,30,2
```

```
Content of birdCare2_summary.txt
```

```
Bingo was displeased and flew away when I tried to pet it.
Plato happily pecked up the bread.
Walnut waddles off and hands me a flower after being pet.
Hawk was displeased and flew away when I tried to pet it.
Pizza thoroughly enjoyed the meal.
Tango thoroughly enjoyed the meal.
```

```
Output in command line
```

```
COO COO COO
coo coo coo
honk honk honk
coo coo coo
HONK HONK HONK HONK HONK HONK
honk honk
```

Reuse your code when possible. Certain methods can be reused using certain keywords. If you use the `@Override` tag for a method, you will not have to write the Javadocs for that method. These tests and the ones visible on Gradescope are by **no means comprehensive so be sure to create your own**!

Rubric

[5] `InvalidBirdException.java`

- [2.5] Both constructors are correctly defined
- [2.5] Exception is an unchecked exception

[15] `Bird.java`

- [2.5] Correct instance variables
- [2.5] Correctly creates constructor
- [6] Correctly sets up abstract methods
  - [2] speak()
  - [2] receivePets()
  - [2] eatFood()
- [4] Correctly implements specified methods
  - [3] equals()
  - [1] Necessary setters and getters

[20] `Goose.java`

- [1] Correct instance variables
- [4] Correctly creates constructors
- [15] Correctly implements specified methods
  - [4] speak()
  - [4] receivePets()
  - [4] eatFood()
  - [2] equals()
  - [1] Necessary setters and getters

[20] `Pigeon.java`

- [1] Correct instance variables

- [4] Correctly creates constructors
- [15] Correctly implements specified methods
    - [4] speak()
    - [4] receivePets()
    - [4] eatFood()
    - [2] equals()
    - [1] Necessary setters and getters

[40] `BirdSitting.java`

- [1] Correct instance variables
- [1] Getters and setters for instance variables
- [3] Correct constructors
- [30] Correctly implements specified methods
    - [14] startBirdSitting()
    - [8] summarizeDay()
    - [8] birdInventory()
- [5] Correctly throws exception

We reserve the right to adjust the rubric, but this is typically only done for correcting mistakes. **Regrade requests will be closed for this homework unless stated otherwise.**

Allowed Imports

To prevent trivialization of the assignment, you are only allowed to import the following classes. You are *not* allowed to import any other classes or packages.

- `java.util.Scanner`
- `java.io.File`
- `java.io.FileNotFoundException`
- `java.io.IOException`
- `java.io.PrintWriter`

Feature Restriction

There are a few features and methods in Java that overly simplify the concepts we are trying to teach or break our auto grader. For that reason, do not use any of the following in your final submission:

- `var` (the reserved keyword)
- `System.exit`

Javadoc

For this assignment, you will be commenting your code with Javadoc. Javadoc comments are a clean and useful way to document your code's functionality. For more information on what Javadoc comments are and why they are awesome, the online overview for them is extremely detailed and helpful.

You can generate the Javadoc overview for your code using the command below, which will put all the files into a folder named "javadoc". Note you should execute this after adding Javadoc comments to your code:

```
$ javadoc *.java -d javadoc
```

The relevant tags that you need to include are `@author`, `@version`, `@param`, and `@return`. Here is an example of a properly Javadoc'd class:

```
import java.util.Scanner;

/**
 *This class represents a Dog object.
 *@author George P.Burdell
 *@version 1.0
 */
 public class Dog {

    /**
     *Creates an awesome dog(NOT a dawg!)
     */
     public Dog() {
     ...
     }

     /**
      *This method takes in two ints and returns their sum
      *@param a first number
      *@param b second number
      *@return sum of a and b
      */
      public int add(int a,int b) {
      ...
      }
}
```

A more thorough tutorial for Javadoc Comments can be found here.

Take note of a few things:

1. Javadoc comments begin with `/**` and end with `*/`.
2. Every class you write must be Javadoc'd and the @author and @verion tag included. The comments for a class should start with a brief description of the role of the class in your program.
3. Every non-private method you write must be Javadoc'd and the `@param` tag included for every method parameter. The format for an @param tag is `@param <name of parameter as written in method header> <description of parameter>`. If the method has a non-void return type, include the `@return` tag which should have a simple description of what the method returns, semantically.

Checkstyle can check for Javadoc comments using the -a flag, as described in the next section.

Checkstyle

For this assignment, we will be enforcing style guidelines with Checkstyle, a program we use to check Java style. Checkstyle can be downloaded here.

To run Checkstyle, put the jar file in the same folder as your homework files and run

```
java -jar checkstyle-6.2.2.jar -a *.java
```

The Checkstyle cap for this assignment is **20 points**. This means that up to 20 points can be lost from Checkstyle errors.

Collaboration

Collaboration Statement

To ensure that you acknowledge a collaboration and give credit where credit is due, **we require that you place a collaboration statement as a comment at the top of at least one .java file that you submit**. That collaboration statement should say either:

I worked on the homework assignment alone, using only course materials.

or

In order to help learn course concepts, I worked on the homework with [give the names of the people you worked with], discussed homework topics and issues with [provide names of people], and/or consulted related material that can be found at [cite any other materials not provided as course materials for CS 1331 that assisted your learning].

Allowed Collaboration

When completing homeworks for CS1331 you may talk with other students about:

- What general strategies or algorithms you used to solve problems in the homeworks
- Parts of the homework you are unsure of and need more explanation
- Online resources that helped you find a solution
- Key course concepts and Java language features used in your solution

You may **not** discuss, show, or share by other means the specifics of your code, including screenshots, file sharing, or showing someone else the code on your computer, or use code shared by others.

Examples of approved/disapproved collaboration:

- **approved**: "Hey, I'm really confused on how we are supposed to implement this part of the homework. What strategies/resources did you use to solve it?"
- **disapproved**: "Hey, it's 10:40 on Thursday... Can I see your code? I won't copy it directly I promise"

In addition to the above rules, note that it is not allowed to upload your code to any sort of public repository. This could be considered an Honor Code violation, even if it is after the homework is due.

Turn-In Procedure

Submission

To submit, upload the files listed below to the corresponding assignment on Gradecope:

- InvalidBirdException.java
- Bird.java

- Goose.java
- Pigeon.java
- BirdSitting.java

Make sure you see the message stating "HW06 submitted successfully". From this point, Gradescope will run a basic autograder on your submission as discussed in the next section.

You can submit as many times as you want before the deadline, so feel free to resubmit as you make substantial progress on the homework. We will only grade your last submission: be sure to **submit every file each time you resubmit**.

Gradescope Autograder

For each submission, you will be able to see the results of a few basic test cases on your code. Each test typically corresponds to a rubric item, and the score returned represents the performance of your code on those rubric items only. If you fail a test, you can look at the output to determine what went wrong and resubmit once you have fixed the issue.

The Gradescope tests serve two main purposes:

- Prevent upload mistakes (e.g. non-compiling code)
- Provide basic formatting and usage validation

In other words, the test cases on Gradescope are by no means comprehensive. Be sure to thoroughly test your code by considering edge cases and writing your own test files. You also should avoid using Gradescope to compile, run, or checkstyle your code; you can do that locally on your machine.

Other portions of your assignment can also be graded by a TA once the submission deadline has passed, so the output on Gradescope may not necessarily reflect your grade for the assignment.

Important Notes (Don't Skip)

- Non-compiling files will receive a 0 for all associated rubric items
- Do not submit `.class` files
- Test your code in addition to the basic checks on Gradescope
- Submit every file each time you resubmit
- Read the "Allowed Imports" and "Restricted Features" to avoid losing points
- Check on Piazza for a note containing all official clarifications