# PROG 1400 - Assignment 4

# Polymorphism & Interfaces

Assignment Value: *12*% of overall course mark.

Due Date: **See due dates designated on the Assignment 4 dropbox on Brightspace.**

Late submissions will receive the standard late submission penalty as stated in the course outline.

#### Assignment Instructions:

Use IntelliJ to create a Java Swing UI application, to demonstrate your understanding of polymorphism, interfaces, simple file I/O and basic Regex validation.

#### Submissions:

UML class diagram (done in Visio) – Submit to the Assignment 4 dropbox on Brightspace.

#### You will submit your program code for this assignment via GitHub. While you will have frequent commits/pushes of your assignment code to GitHub as your work on it, the instructor needs to know which version to mark and when it was committed. So, when you have completed all assignment work, put a “Ready for Marking” comment on the last code committed into GitHub. Then submit a simple text document to the Brightspace Dropbox that contains the git Commit ID string (e.g. “b180b37”) that identifies that commit. It is this Dropbox submission that will be used to determine late penalties, so make sure to do so prior to the assignment deadline.

#### Once you have committed the code, make sure to visit the repository page on GitHub’s website to verify that the final version has been pushed to GitHub as that is where the instructor will go to get your code.

#### Evaluation:

To insure the greatest chance of success on this assignment, be sure to check the marking rubric contained at the end of this document or in Brightspace. The rubric contains the criteria your instructor will be assessing when marking your assignment.

## Program – Poly Penguins – Animal Tracking App

### Design and Coding

Prior to coding the application, you should work out a design for the objects in your application. This will be diagrammed using Visio. For example, each Animal in the system shares a common attribute of Species. Make sure your classes do not duplicate any code by using Java inheritance and/or aggregation to share common attributes/methods, or define additional classes to capture differences. Your solution should include at least one interface, and show which classes implement it. Upload the Visio diagram as a JPEG or PNG along to Brightspace.

Some guidelines for the UML diagram:

* All classes in your program (besides Main) should be included in the class diagram.
* Including GUI classes in your diagram is not required.
* Constructor methods should included for all classes that have them.
* Unless necessary, your diagram does not need to include all getter/setter methods.
* Your Character class hierarchy must have at least three levels of inheritance (including Character).
* Your solution must include at least one example of inheritance.
* Your solution must include at least one example of aggregation or composition (has-a, instead of is-a).
* Your solution must include at least one example of an interface.
* Your solution must include at least one example of an abstract class.
* All classes, attributes and methods should follow proper UML class diagram standards and conventions.
* Proper line indicators denoting inheritance, association, aggregation/composition and interfaces should be included wherever required.

### Program Requirements

Use Java and Java Swing to create a simple graphical Antarctic Animal Tracking program, to demonstrate your understanding of polymorphism, interfaces, simple file I/O and basic Regex validation.

Researchers at an Antarctic wildlife research facility need an application to help them keep track of penguins, sea lions and walruses. All animals in the study are outfitted with a tracking device that holds GPS information on its movements over time.

The application will use a simple UI that allows the researchers to record the species, sex, weight and GPS information of each animal captured. GPS data will be added to a GPS Log text file. Presumably a future enhancement to the program will be to save animal data to a database, but for now, all animals will be saved to a list in memory. Both the saved animal entries from the list and the GPS data from file can be displayed as a report in the program.

Each animal observed has some distinct characteristics that are entered for only that species:

* For penguins only, the program needs to track blood pressure (double).
* For sea lions only it needs to track the number of spots each has (integer).
* For walrus only, their dental health “good”, “average” or ”poor” (String).

### Data Validation Rules

* Weight must be a whole number greater than 0.
* GPS coordinates for latitude must range from -90 to 90 and have 7 places after the decimal.
* GPS coordinates for longitude must range from -180 to 180 and have 7 places after the decimal.
* There must be a space between latitude and longitude values.
* Penguins: Blood Pressure must be double greater than 0.
* Seal Lions: Number of Spots must be a whole number greater than 0.

#### Screen Descriptions

##### **Data Entry Screen**

The data entry screen will allow the user to enter all details for an observed animal of any of the three species. Species and gender will be displayed in as combobox choices (Penguin, Sea Lion and Walrus for species, Male and Female for Gender). Weight in Kg should only accept whole numbers in a textfield. For penguins and sea lions, their blood pressure (double) or number of spots (integer) values use textfields, while for walrus dental health will be displayed in a combobox. Appropriate labels for the species-specific should be displayed no matter which species is being entered.

GPS coordinates can be added into a textfield. When the Add GPS button is clicked, the entered GPS value will be validated via Regex. If it is valid, it will be added to the GPS textarea list.

When the Add Entry button is clicked, your program should use regular expressions to validate all user-entered data to ensure proper values. If any information is missing or invalid, display an error message indicating what is wrong. If all information is valid, create the selected animal object as its specific type, and add to a super-typed list that should hold all animal types in a single list. Once an animal entry has been saved, the screen should clear, to allow entry of the next animal.

At any time, the View Reports button will move the user to the Reports screen.

**Reports Screen**

The Reports screen should have a title label, which will display the title of the currently viewed report. The text area for displaying report data should be disabled.

The screen should include three buttons:

* **Show New Entries** – Will print to screen all attributes for all animals that have been entered in this session. (ie. Stored in memory)
* **Show GPS Logs** – Will read all GPS locations from the GPS Log text file, and print all to screen.
* **Back** – Will clear the report screen and return the user to the Data Entry screen.

#### Program Structure

Your program must use inherited classes for the program Frame and each of the two screens (as Panels), as well as for other objects used in the system.

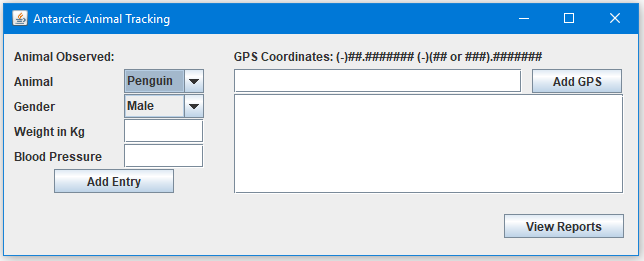
Your program must include an interface called **ILoggable**, which will be used to write GPS data, to either the GPS Log file, or to the console (for debugging purposes). The interface must be implemented by each of the three animal types (write to file), and on your GPS class (write to console).

Each of your classes should make proper use of constructors, getter and setter methods, as well as any standard methods you deem necessary.

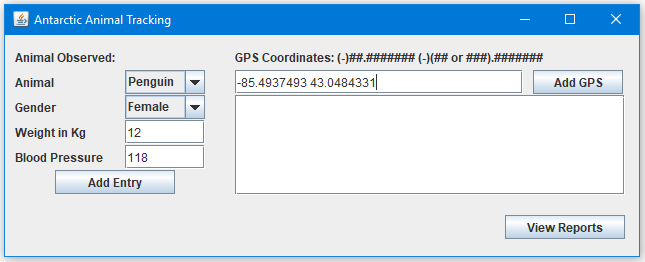
Your solution should also include at least one example of an abstract class and at least one abstract/override method. Examples of static properties or methods should also be included.

### Sample Screenshots

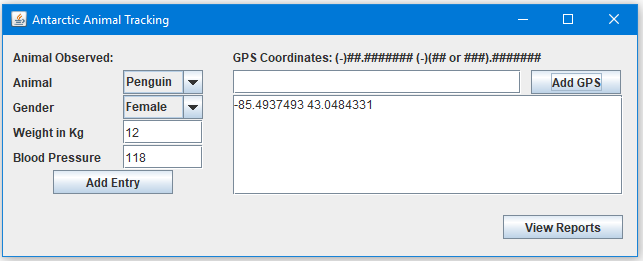
###### Data Entry screen at startup



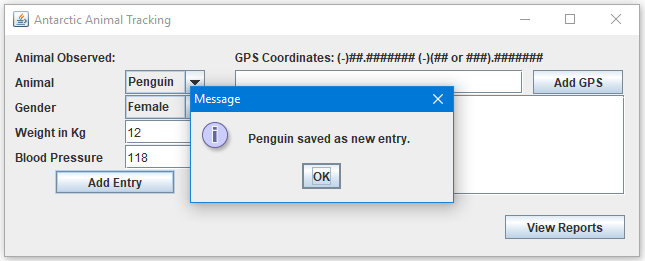
###### Data Entry screen, after adding data, before pressing Add GPS button



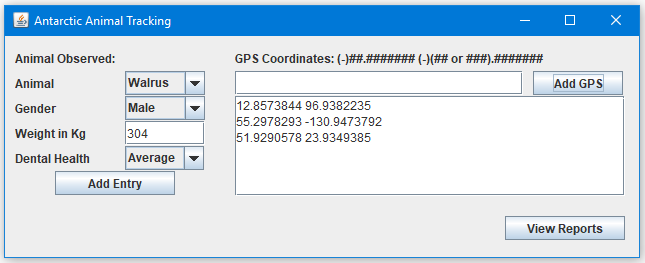
###### Data Entry screen, after adding data and AFTER pressing Add GPS button

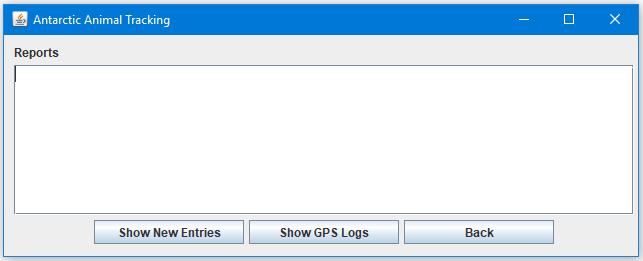


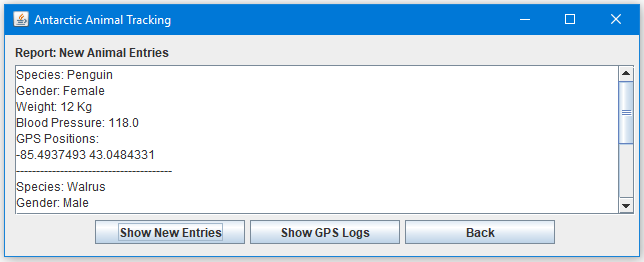
###### Data Entry screen, after adding all valid data and pressing Add Entry button

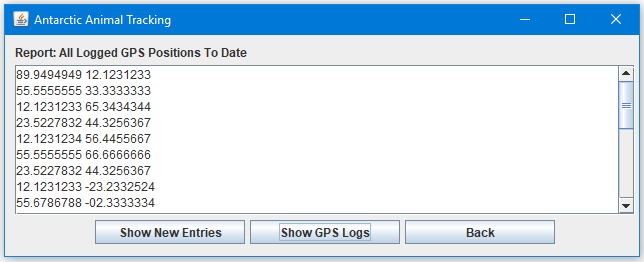


###### Data Entry screen, after adding a new record with multiple GPS entries

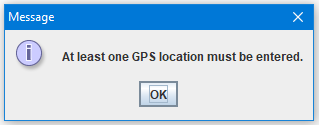
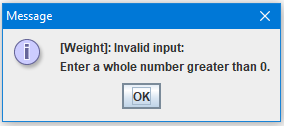
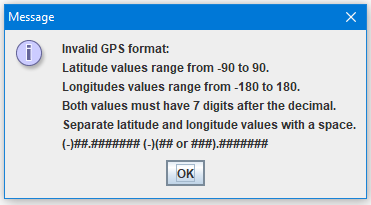


*Reports screen, when first opened*

*Reports screen, displaying New Entries Report (data from all animals added during this session).*

*Reports screen, displaying GPS Logs Report (data retrieved from GPS Log text file).*

*Examples of some failed validation messages*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **A4 – Poly Penguins – Animal Tracking** | | |  | Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  |  |
| **Criteria** | **Insufficient (0 pts)** | **Developing (1-2 pts)** | **Sufficient (3-4 pts)** | **Excellent (5 pts)** | **Mark** | **X** |
| **Class Diagram** | Submission missing entirely, or it contains too many issues and/or omissions. | An effort was made, but submission has multiple issues, major issues, or omissions. | A good effort, but submission has a few issues and/or areas for improvement. | Submission demonstrates a strong understanding of class diagram concepts, meets all requirements and fully covers the expected scope of the program, including inheritance, aggregation and interfaces. Proper UML notation is used throughout. Displays a high level of analysis, planning and attention to detail. Submission is well-formatted, proofread and presentable. |  |  |
| **Program Requirements** | Missing entirely, or it contains too many issues and/or omissions. | An effort was made, but has multiple issues, major problems, or omissions. | A good effort, but has a few issues and/or areas requiring improvement | Program works as expected, with all required functionality, and runs without errors. |  |  |
| **UI** | Missing entirely, or it contains too many issues and/or omissions. | An effort was made, but has multiple issues, major problems, or omissions. | A good effort, but has a few issues and/or areas requiring improvement | The user interface is properly structured, functional, appropriate and easy to use. Layout is clean and well-formatted. |  |  |
| **Inheritance, Aggregation & Abstraction** | Missing entirely, or it contains too many issues and/or omissions. | An effort was made, but has multiple issues, major problems, or omissions. | A good effort, but has a few issues and/or areas requiring improvement | Program includes at least one example each of inheritance, aggregation and abstraction, and implementation demonstrates strong understanding of concepts. |  | 2 |
| **Polymorphism** | Missing entirely, or it contains too many issues and/or omissions. | An effort was made, but has multiple issues, major problems, or omissions. | A good effort, but has a few issues and/or areas requiring improvement | Program includes example(s) of polymorphism, and implementation demonstrates a strong conceptual understanding. |  | 2 |
| **Interface** | Missing entirely, or it contains too many issues and/or omissions. | An effort was made, but has multiple issues, major problems, or omissions. | A good effort, but has a few issues and/or areas requiring improvement | Program includes and implements an interface called ILoggable, which is properly used to log GPS data from all specified classes by writing to text file. Implementation demonstrates a strong conceptual understanding. |  |  |
| **File I/O** | Missing entirely, or it contains too many issues and/or omissions. | An effort was made, but has multiple issues, major problems, or omissions. | A good effort, but has a few issues and/or areas requiring improvement | All expected file reading/writing works as expected, uses proper exception handling, and shows a strong understanding of the concepts. |  |  |
| **Regex & Data Validation** | Missing entirely, or it contains too many issues and/or omissions. | An effort was made, but has multiple issues, major problems, or omissions. | A good effort, but has a few issues and/or areas requiring improvement | Regular expressions are properly used to apply all required data validation to all user-entered values. Invalid data is captured and displays appropriate and informative error messages to user. |  |  |
| **Code Efficiency** | Missing entirely, or it contains too many issues and/or omissions. | An effort was made, but has multiple issues, major problems, or omissions. | A good effort, but has a few issues and/or areas requiring improvement | Program demonstrates reasonable code efficiency. (e.g. large segments of code that could be shorter, unnecessary code duplication, code not encapsulated in functions, etc.) |  |  |
| **Comments & Best Coding Practices**  (At least 60% of the reqs must be complete) | Little to no effort was made or contains too many errors / omissions. | A reasonable effort was made, but there are multiple areas for improvement. | A good effort was made, but at least one error or omission exists. | Organizational or explanatory comments are used extensively, most are meaningful and easily understood. A consistent naming convention was used for most of the program and deviated very little. Source code was clean, consistently well-formatted and easy to read. |  |  |
|  |  |  | **Total:** |  | **/60** | |