Design Document: Zoo Application

Project Overview

The Zoo Application is designed to process, organize, and manage data about animals in a zoo. It includes functionalities such as reading animal data from files, updating animal names, and generating a report detailing animal information and species counts. The primary goal is to implement and extend the application with robust data structures and methods that ensure efficient data handling.

Components

Animal Class:

-Represents the base class for all animal types.

- Contains common properties like `name`, `age`, and `species`.

- Provides getter and setter methods for encapsulation.

Hyena, Lion, Tiger, and Bear Subclasses:

- Extend the `Animal` class to add specific attributes such as `favoriteFood` for Hyena, `isAlpha` for Lion, `stripeColor` for Tiger, and `weight` for Bear.

- Override the `toString()` method for detailed string representations of each animal.

ZooApp Class:

- Handles the core application logic.

- Implements methods for reading animal data, updating names, and writing reports.

Files Used:

- `arrivingAnimals.txt`: Stores incoming animal data.

- `animalNames.txt`: Contains updated names for animals.

- `newAnimals.txt`: Generated report of processed animal data.

Data Structures

List(`ArrayList<Animal>`):

- Stores all animals processed by the application.

- Enables dynamic resizing and efficient iteration.

Map (`HashMap<String, Integer>`):

- Tracks species counts.

- Provides efficient lookups for species statistics.

Interactions

`readAnimalData()`:

- Reads data from `arrivingAnimals.txt`, parses animal attributes, and populates the `List` and `Map`.

- Demonstrates interaction between file reading and dynamic object creation.

`readAndUpdateNames()`:

- Updates animal names based on data from `animalNames.txt`.

- Illustrates iteration over `List` and modification of object properties.

`writeAnimalReport()`:

- Generates `newAnimals.txt` with detailed animal information and species counts.

- Combines formatted output of `List` and `Map`.

Code Review

Review of `readAnimalData()`

Observations:

Strengths:

- Clear structure for parsing data and handling different species.

- Exception handling for invalid entries.

Improvements:

- Use `try-with-resources` for closing the `BufferedReader` automatically.

- Add validation for `parts[3]` to ensure safe parsing of optional attributes.

Revised Code:

```java

private static void readAnimalData(String filePath, List<Animal> animals, Map<String, Integer> speciesCount) throws IOException {

try (BufferedReader reader = new BufferedReader(new FileReader(filePath))) {

String line;

while ((line = reader.readLine()) != null) {

String[] parts = line.split(",");

if (parts.length < 3) {

System.err.println("Skipping invalid entry: " + line);

continue;

}

String name = parts[0].trim();

int age = Integer.parseInt(parts[1].trim());

String species = parts[2].trim();

try {

Animal animal = switch (species.toLowerCase()) {

case "hyena" -> new Hyena(name, age, parts.length > 3 ? parts[3].trim() : "unknown");

case "lion" -> new Lion(name, age, parts.length > 3 && Boolean.parseBoolean(parts[3].trim()));

case "tiger" -> new Tiger(name, age, parts.length > 3 ? parts[3].trim() : "unknown");

case "bear" -> new Bear(name, age, parts.length > 3 ? Double.parseDouble(parts[3].trim()) : 0.0);

default -> null;

};

if (animal != null) {

animals.add(animal);

speciesCount.put(species, speciesCount.getOrDefault(species, 0) + 1);

} else {

System.err.println("Unknown species: " + species);

}

} catch (NumberFormatException e) {

System.err.println("Skipping invalid numeric value: " + line);

}

}

}

}

```

Experiment

Alternative Data Structures

Set(`HashSet<Animal>`):

- Use for storing unique animals to avoid duplicates.

- Trade-offs:

- Faster lookups.

- Does not preserve order.

Queue(`LinkedList<Animal>`):

- Process animals in a first-in-first-out (FIFO) manner.

- Suitable for tasks involving sequential processing or scheduling.

TreeMap(`TreeMap<String, Integer>`):

- Replace `HashMap` to maintain species count in sorted order.

- Useful for producing sorted reports directly.