Design Document for Zookeeper's Challenge

Project Overview:
This project is a C++ application that simulates a zoo management system. The application reads animal data from an input file named arrivingAnimals.txt and enclosure assignments from another file called animalEnclosures.txt. Using object-oriented programming concepts along with various data structures, the program processes this information and generates a comprehensive report in zooReport.txt. The final report is organized by species, ensuring that all animal data is clearly categorized and easily accessible.

Requirements & Objectives:
The main objectives of this project are to implement robust file input/output operations, apply fundamental OOP design principles, and utilize key data structures such as vectors and maps. The program is designed to read from two input files and output a detailed report to zooReport.txt. In terms of OOP design, an Animal base class with properties like name, age, species, and an enclosure ID is implemented, along with four specialized subclasses—Hyena, Lion, Tiger, and Bear—that override virtual methods to exhibit unique behaviors. Control structures like loops and conditionals are employed to process file data, update animal records, and generate the final report.

Class Design:
The core of the application is built around the Animal class, which encapsulates common properties such as name, age, species, and an enclosureID (with a default value of -1). This class provides getter and setter methods along with a virtual function, getUniqueInfo(), to allow subclasses to provide species-specific details. The four subclasses—Hyena, Lion, Tiger, and Bear—each extend the Animal class by adding specific attributes (for example, Lion includes a maneLength attribute while Tiger includes a stripeCount attribute) and override the getUniqueInfo() method to return tailored information that distinguishes each animal type.

Data Structures & File I/O:
The program leverages several important data structures to manage animal data effectively. A std::vector<Animal*> is used to store pointers to dynamically allocated Animal objects, enabling flexible and efficient management of varying animal entries. To organize the data for reporting, a std::map<std::string, std::vector<Animal*>> groups animals by their species, and another std::map<std::string, int> is used to map animal names to their respective enclosure IDs as retrieved from animalEnclosures.txt. File streams (ifstream for input and ofstream for output) are used to read the input files and write the final report.

Program Flow & Implementation:
The application begins by processing input from arrivingAnimals.txt; it reads the file line by line, creates the appropriate Animal subclass objects based on the species provided, and stores these objects in the vector. Next, the program reads animalEnclosures.txt to build a map of enclosure

assignments, which is then used to update the enclosureID for each Animal object. With all data integrated, the program groups the animals by species using a map and proceeds to generate a detailed report in zooReport.txt. The report lists, for each species, every animal's name, age, enclosure ID, and unique information, and concludes with a total count for each group. This flow ensures that all animal data is systematically processed and clearly presented in the final output