## Zoo Design README

### Overview

This README explains the approach taken to design the components of a zoo. Since a zoo contains many entities, it covers only some critical components to the zoo’s identity and operations. The design focuses on some animals in a zoo, the employees that work at the zoo, and cages within the zoo. The design is done in c++ using object oriented techniques, inheritance, and polymorphism.

### Compilation and execution

There are 4 files that contain the declaration and definitions of several classes and their attributes and behaviors.

zoo.h : Contains the Zoo class which has functions and attributes for the Zoo as a whole

cage.h : Contains the Cage class which has attributes describing a cage in a zoo

animal.h : Contains the Animal, FlyingAnimal, LandAnimal, Crow, Parrot, Cheetah, and Goat classes which their respective attributes and actions

employee.h : Contains the Employee and Manager classes which have the actions that these workers can take and information pertinent to describing them

To keep it simple to read and build, the declaration and definitions for each of the classes and associated member functions are done in the .h files. The zoo\_main.cpp file has different examples of what can happen in the zoo.

To compile and run this, you should use a compiler that supports c++ 11 and execute the following (under Windows):

To compile: g++ -std=c++11 zoo\_main.cpp -o zoo\_main

To run: ./zoo\_main

This does use certain semantics such as shared pointers that are not available natively in previous versions of c++.

### The classes and design

zoo\_main.cpp runs through almost all the different actions that are defined for the zoo such as creating animals, cages, employees, and managers. Also, it runs through examples of animals taking actions such as eating, moving, and making sounds. Furthermore, it does some changes such as modifying an animals cage, promoting an employee, and even changing an employee’s manager.

The best way to go through the Zoo setup is to look at the code but the different classes are briefly explained here.

The Zoo class

The Zoo class maintains maps between each animal, cage, and employee and their respective numeric identifications. These maps store a shared pointer to these objects so that the memory is managed by reference count automatically. Without shared pointers, we would have to define a destructor that cleans up the data allocated in these maps. But this still wouldn’t handle any objects allocated on the heap but not inserted on the map, which would lead to memory leaks. This is give a way to obtain the full information of each of these using just their unique id. The functions called by zoo\_main here to add the different types of objects and make modification are contained here as well as operations common to a zoo.

The Cage class

The Cage class is simple as each cage simply contains dimensions and function to compute its volume.

The Employee class

The Employee class has the attributes and actions common to a Zoo employee. It has a mix of non-virtual and virtual functions depending on what actions a manager may do differently to demonstrate polymorphism.

The Manager class

The Manager class inherits from Employee and polymorphically re-implements some of the member functions such as doWork and showInfo. It also contains a map containing the Employee information for any employees that report to the manager. The manager can also remove and add employees under him/her.

The Animal class

The Animal class is set up as an abstract base class since it does not make sense to define a general animal in the zoo since they can be specified by the type of animal it is or species. To achieve this, Animal has some pure virtual functions such as makeSound and eat that must be implemented by derived classes that are not abstract. The accessors and modifiers are shared amongst animals so these are defined at the Animal level. The common attributes defined here amongst animals are the name and cage (if in a cage) for each animal.

The FlyingAnimal class

The FlyingAnimal class inherits from Animal. It is abstract as it does not implement the makeSound or eat functions which were pure virtual in the Animal class. It does however implement the move behavior, since flying is common to all Animals that fall under this class. Flying animals have an additional wing span attribute.

The LandAnimal class

The LandAnimal class inherits from Animal. It is also abstract for the same reason as the FlyingAnimal class. This makes sense since LandAnimal is still a relatively generic concept. It does implement the move behavior so there is a default behavior to how LandAnimals move. Land animals have an addition attribute for number of legs.

The Crow class

The Crow class inherits from FlyingAnimal. It directly defines the virtual makeSound and eat functions so that it is not abstract and can be instantiated.

The Parrot class

The Parrot class inherits from FlyingAnimal. It defines the makeSound and eat functions making it instantiable. In addition it also has another attribute not defined in any of the base classes, which is its feather color.

The Cheetah class

The Cheetah class inherits from LandAnimal. It defines the makeSound and eat function so that it is instantiable. In addition, it overrides the move function defined in the LandAnimal class since a cheetah moves much faster than a typical land animal.

The Goat class

The Goat class inherits from LandAnimal. It also defines makeSound and eat so that we can instantiate goats. It has an additional attribute not defined in its hierarchy which is its number of legs.