CPSC 8700 Assignment 2 (50 points) Fall 2024

Building a Zoo where Animals live

This assignment is done in teams of 2, or alone. You are not allowed to share the details of your program with others. In this assignment you will demonstrate several skills:

1. Combining design patterns: Abstract Factory, Singleton and Clone.
2. Input redirection
3. Inheritance and abstract classes
4. Building a simple makefile with *make* and *make clean*
5. Creating a tarball

Grading Rubric

5 - Create only one instance of the Zoo – use of Singleton pattern.

3 - Static var/func to keep total count of zoo animals

4 - Correctly reading input data via input redirection

6 - Each of 5 animal types inherit from interface Animal – use of Factory pattern
5 - Each of 5 animal types have type specific members and can be counted (static var and func)

6 - Overloaded stream insertion << operator

6 - Able to clone animals – use of Clone pattern

6 - Driver that clearly demonstrates all program components are working

5 – Output as shown or similar (animal food and sounds may be different)

4 - Tarball without subdirectories

Program Execution:

This program will create a Zoo “Wild Things” with several animals. To know what/how many animals to create you will get data via input redirection. User will specify how many animals of each kind to create. Animals can also be listed in a different order. For example, your input file may have the following. Please make sure not to hard code the numbers, as they may vary.

```
tiger 2
wolf 2
kangaroo 1
lemur 3
serpent 2
```

To execute your program on the command line, you will type:

```
$$ zoo < inputfile
```

where *inputfile* will contain data about how many/which animals to create (shown above). $$ is a command prompt.

Instructions

1. You should be able to create only one instance of the Zoo. You should be able to get the total count of animals living in the Zoo.

2. You will have 5 types of animals: tiger, wolf, lemur, kangaroo, and serpent. All of them will inherit from interface called Animal. Each animal will have defining characteristics appropriate to that animal and have a name.

Each object will print a message: ”I am a tiger , my name is Ronald, I roar, and I eat meat”, “I am a wolf, my name is Rex, I growl, and I eat chicken”, “I am a lemur, my name is Fluffy, I squeak, and I eat fruit”, “I am a kangaroo, my name is Jack, I click, and I eat carrots”, “ I am a serpent, my name is George, I hiss and I eat mice”. (Hint: overloaded operator << and might have to have class members called food, sound, name...) You should be able to get the count of each animal type as well.

3. Each class should have a cloning capability because you will want to clone some or all of the animals.

4. Be creative in your driver. You should output all created objects (all tigers, etc.) and their characteristics (as described above). If animals were cloned, you will output them as well, as shown below.

Your output will look similar to this:

```
Zoo “WildThings” is home to the following animals:
______________________________

There are total 10 animals in the Zoo:
   2 tigers, 2 wolves, 1 kangaroo, 3 lemurs and 2 serpents.


Our animals would like to introduce themselves:
   Hi, I am a tiger, my name is Ronald, I roar, and I eat meat
   Hi, I am a tiger, my name is Stripy, I roar, and I eat meat
   Hi, I am a wolf, my name is Rex, I growl and I eat chicken
  …
Zoo has some cloned animals:
  tiger Ronald was cloned, and his clone is named Sam.
  lemur Fluffy was cloned, and his clone is named Fuzzy.
```

Details

Specific implementation details are intentionally omitted to give you a chance to come up with a unique solution. Skills listed on top of the file are required.

What to submit:

tarball *without* subdirectories or object files.

Recommendations:

1. Work in modular fashion. Type a few lines of code at a time and compile to make sure it has no syntax errors. Use print statements and comments while debugging.

2. Make backups in another directory. Especially when tarring your files.

3. Create interfaces first, then derived classes. Add functionality a little bit at a time.

4. Open more than one terminal window, so you can edit in one and compile in another.

5. Do test your code on SoC Linux machines.