Sprint 2 Guidance

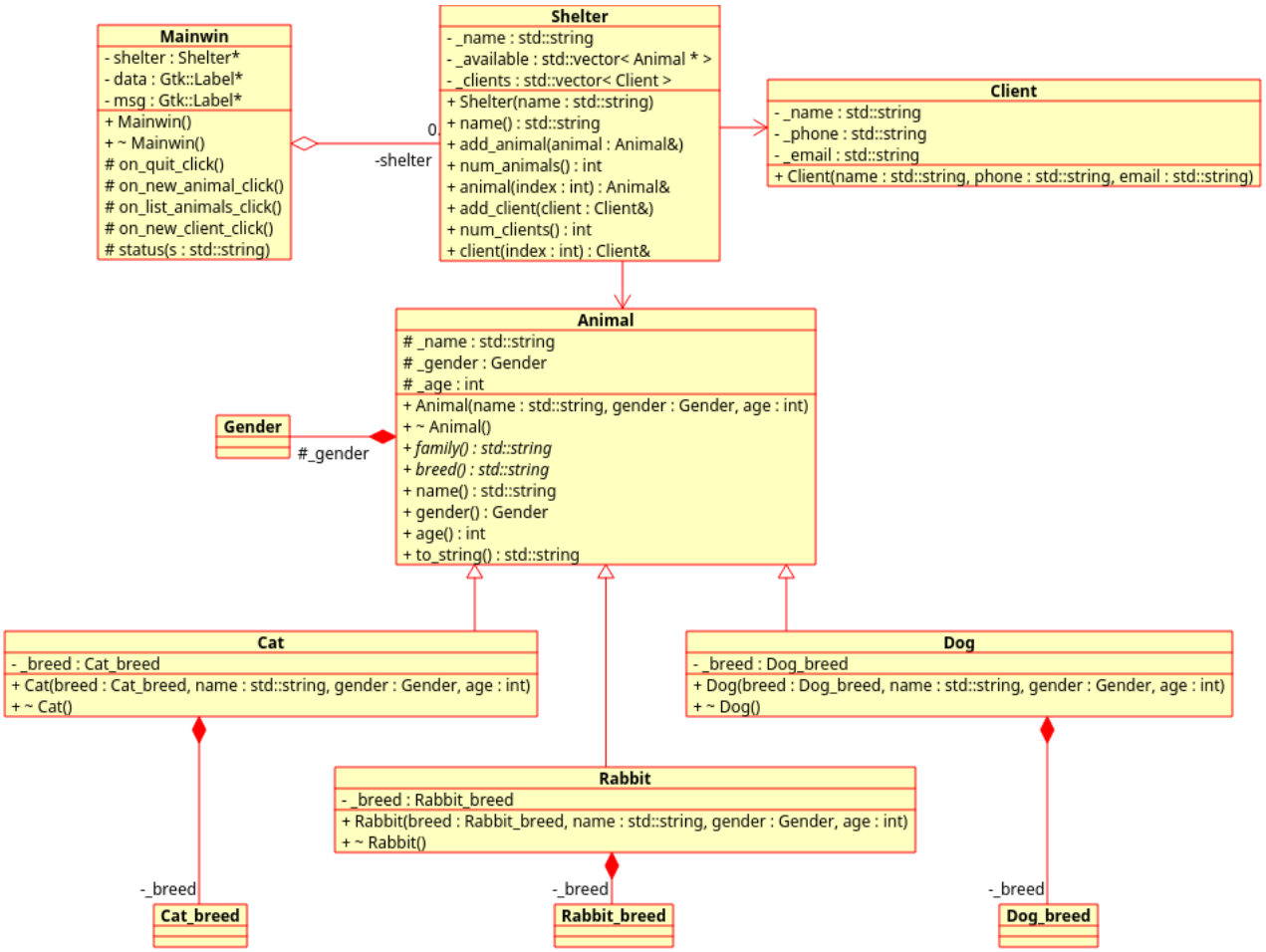
CSE 1325 – Fall 2019 – Homework #9
Due Tuesday, October 12 at 8:00 am

IMPORTANT: Do NOT use full_credit, bonus, or extreme_bonus subdirectories for the final project. Keep all files to be graded in the cse1325/P8 (or if you prefer, cse1325/P9) directory on GitHub for the duration of the final project.

The second sprint includes 29 points including two new animal families to create and list, or three if you count human clients. :-) Listing clients isn't allocated to this sprint, although if you can squeeze it in, it will simplify testing (yours and ours).

Your implementation is permitted to vary from any class diagram provided with the final project as needed without penalty, as long as you correctly implement both the letter of and the intent of the feature list. **If you have questions about the acceptability of changes that you are considering, contact a TA or the professor first!**

This class diagram is for sprint 2 only:



A brief description of each class follows.

Cat and Rabbit

Like Dog, these invariant classes inherit from Animal, and represents the cat and rabbit family, respectively. **You may use other or additional families if you prefer without penalty**; cat and rabbit are just suggestions. Don't forget that each class requires a constructor that delegates to Animal::Animal.

Each must override Animal::family() to return the type of the derived class, e.g., Cat::family will return "cat". **The override keyword is required for this homework.**

Each must also override Animal::breed to return whatever enumerated breed type that object represents, e.g., for a Cat instance it might return "Albissnian". You'll need an enum class for the cat and rabbit breeds, along with a way to convert to a string. **You must use a std::map to translate each Cat and Rabbit enumerated value to a string**, though you need not retrofit Dog. Pick at least 8 rabbit and 8 cat breeds from the Internet as you please.

Client

A client has a name, phone number, and email address. Client is an immutable class, so you'll need a constructor to store the data and (at least) an operator<< override to convert it all to a single string for display in the data area and in dialogs.

Shelter

Assuming you created an add_animal method accepting a reference or pointer and a vector of Animal pointers last sprint, nothing new should be needed to handle Cats and Rabbits.

You'll need to add similar methods for handling Clients, however, and a vector (objects, references, or pointers as you prefer) in which to store them. For example, you'll need an add_client method to store them, and a pair of methods to access the clients of the shelter.

Mainwin

For sprint 2, you'll need a way to create clients, e.g., Client > New, with a dialog for reading its attributes. Do some simple validation, but you needed (for example) verify that the phone number's area code is actually valid.

You'll need a way to select the family of animal to be created. This can be an additional submenu to Animal > New (e.g., Animal > New > Rabbit), a preceding dialog with a ComboTextBox or radio buttons listing the animal families prior to collecting the rest of the data, or something similar. Note that the breed enumeration, and thus drop-down breed listing, must be different for each family.

As a greater challenge, you could incorporate family into the same dialog as the other attributes, but this will require a callback so that when the family is selected, the ComboTextBox's list of breeds is updated. Neat trick, if you'd like to give it a try.

For Animal > List (or equivalent), **you must use polymorphism to list all families of animals in the shelter in a single list.** You may NOT use casting or other tricks. It's polymorphism or bust.

Be sure to do some basic data validation, as an unhandled exception or segfault will impact your grade.

Where We're Going

Sprint 3's solution may look something like this. Remember that the feature set is subject to change before each sprint begins, though I'll strive to avoid impacting those who are working ahead.