Sprint 1 Guidance

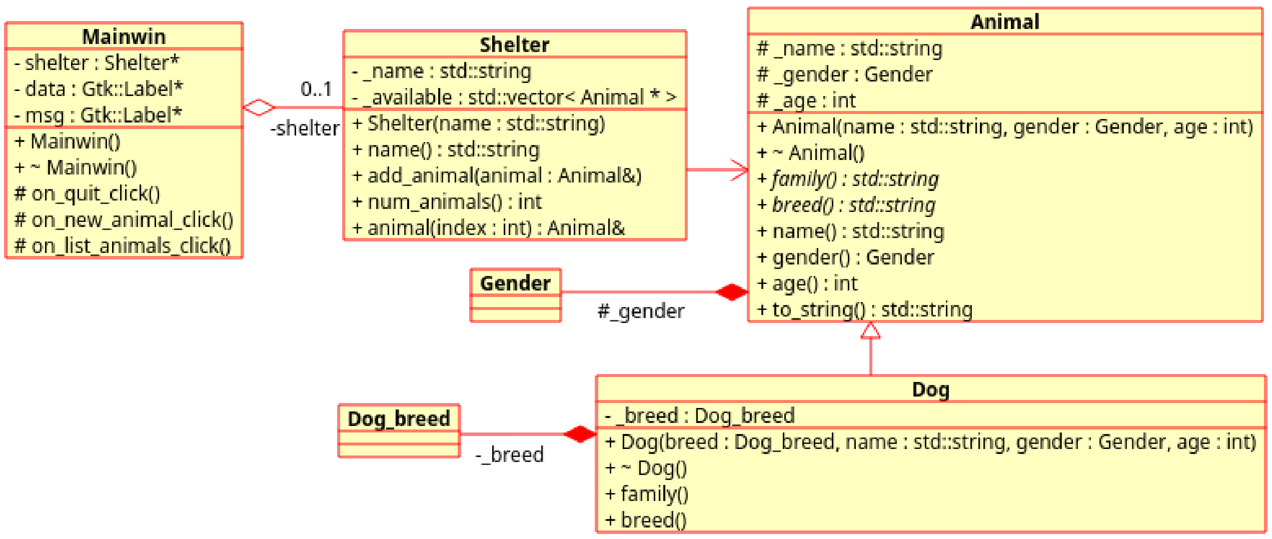
CSE 1325 – Fall 2019 – Homework #8
Due Tuesday, October 5 at 8:00 am

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 directory on GitHub for the duration of the final project.

The first sprint includes 31 "points" (a measure of the expected amount of work required to implement those features). This includes a main function; a Mainwin class as your graphical user interface (see the user interface document); Animal and Dog classes with associated enum classes; and class Shelter.

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 1 only:



A brief description of each class follows.

Animal

Animal is a pure virtual invariant class from which all animal families (such as Dog, Cat, and Rabbit) are derived, and represents the common attributes of all animals seeking adoption at the shelter. It includes the name of this animal, its gender, and its age. (You may instead store its birthdate and calculate the age. This is *much* better, but unfortunately C++ lacks a native Date class.)

The 3 attributes are set by a constructor. Three getter methods provide read-only access to the associated attributes. Pure virtual method family will return the family's name, a constant in the derived classes. Pure virtual method breed will return the animal's breed based on the unique enum class provided by each derived class.

You may also need an operator<< function for this class, for example to be used with an ostringstream when formatting the animal's attributes for the main window's the data area. (Hint: Since functions cannot use polymorphism, you'll need a virtual Animal::to_string method for operator<< to call to make this work. Another quirk of C++, I'm afraid!)

Dog

This invariant class inherits from Animal, and represents the dog family. Don't forget that Dog requires a constructor that delegates to Animal::Animal.

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

Dog must also override Animal::breed to return whatever enumerated breed type that object represents, e.g., for a Dog instance it might return "retriever". You'll need an enum class for the dog breeds, along with a way to convert to a string. Pick at least 8 dog breeds from the Internet as you please.

Shelter

Each shelter has a name, and (for sprint 1) a collection of Animal objects (remember that polymorphism requires either references or (most commonly) pointers, and you'll eventually need to call methods on classes derived from Animal polymorphically). You'll need a getter for the name, a method to add an animal, and (for now) a pair of methods to access the animals in the shelter.

Mainwin

Mainwin derives from Gtk::Window, and implements the main window for the application. You'll save a LOT of time if you baseline from one of our existing examples, e.g., Mav's Ultimate Sweet Shop from P6).

For sprint 1, this need only include a menu bar with 3 menu items:

- File → Quit, which exits the program.
- Animal → New, which open a dialog allowing you to enter the name, gender, and age of a dog. When you click OK, an instance of Dog will be created and added to the shelter. Creating cats, rabbits, and whatever other animals you like will be part of sprint 2.
- Animal → List Available, which will just list a table of the available animals in the data area of the main window.

Note that because the larger design includes roles as a bonus feature, for which certain menu items will need to become insensitive (also called disabled or "greyed out"), you should define the menu items on your drop-down menus as class attributes if you plan to implement this bonus feature. This will make it fairly easy to set their sensitivities when another role logs in.

Mainwin instances its own Shelter by default (pick a default name). Later, we'll have Open and Save features to persist our store data via files.

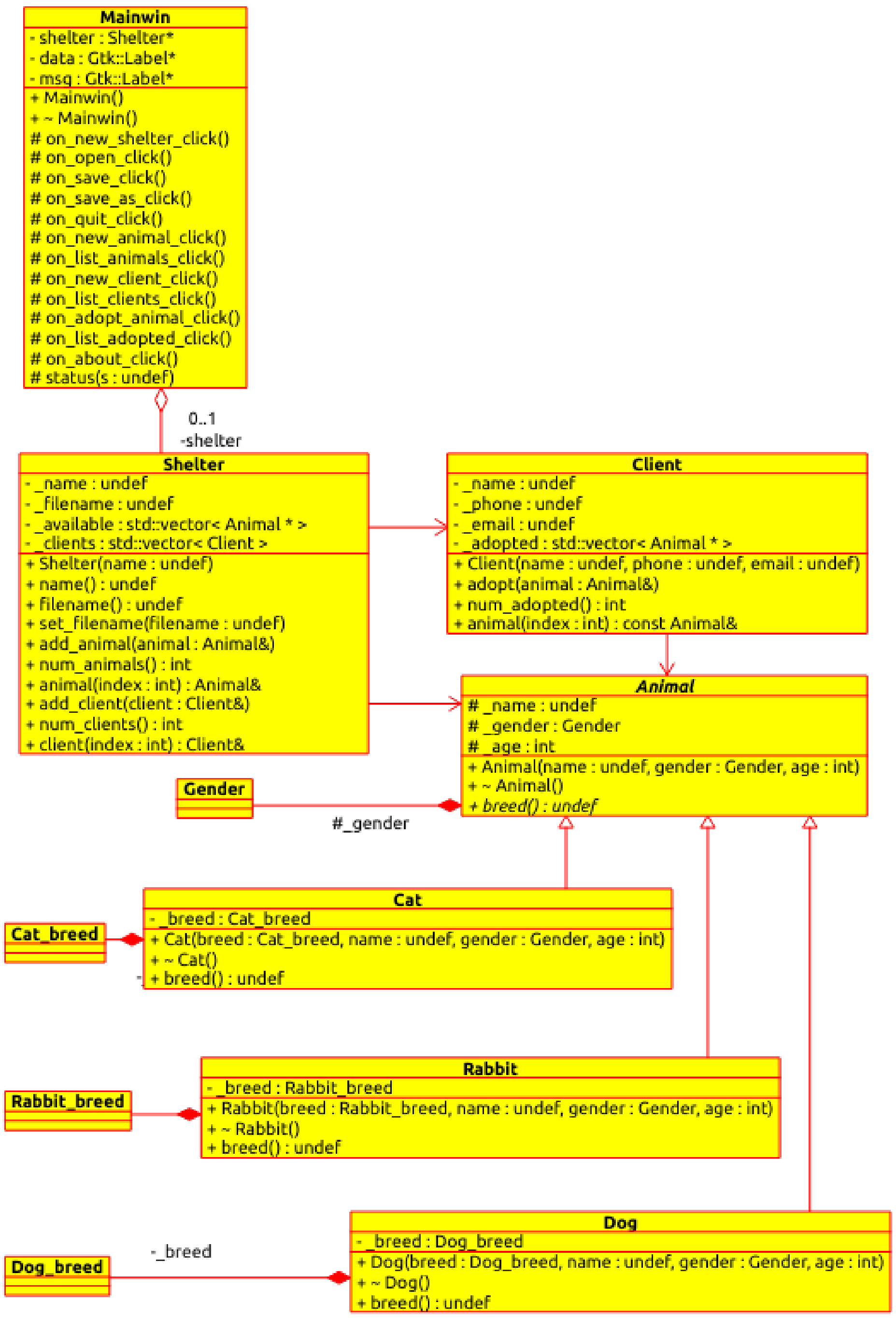
Main

Our main function is the same as it always is – use our Gtk::Application::create factory to create an app, instance Mainwin, and then use app's run method to execute our Mainwin instance.

Where We're Going

The current draft of the full class diagram implementing many of the requested features looks something like this. A good portion of this diagram isn't required in the first few sprints, however.

We'll flesh out this sprint's class diagram a bit more in sprint 2. Remember that both the feature list and the priority of the features is subject to change each sprint.



Mainwin
- shelter : Shelter*
- data : Gtk::Label*
- msg : Gtk::Label*
+ Mainwin()
+ ~ Mainwin()
# on_new_shelter_click()
# on_open_click()
# on_save_click()
# on_save_as_click()
# on_quit_click()
# on_new_animal_click()
# on_list_animals_click()
# on_new_client_click()
# on_list_clients_click()
# on_adopt_animal_click()
# on_list_adopted_click()
# on_about_click()
# status(s : undef)
0..1
-shelter
Shelter
- _name : undef
- _filename : undef
- _available : std::vector< Animal * >
- _clients : std::vector< Client >
+ Shelter(name : undef)
+ name() : undef
+ filename() : undef
+ set_filename(filename : undef)
+ add_animal(animal : Animal&)
+ num_animals() : int
+ animal(index : int) : Animal&
+ add_client(client : Client&)
+ num_clients() : int
+ client(index : int) : Client&
Client
- _name : undef
- _phone : undef
- _email : undef
- _adopted : std::vector< Animal * >
+ Client(name : undef, phone : undef, email : undef)
+ adopt(animal : Animal&)
+ num_adopted() : int
+ animal(index : int) : const Animal&
Animal
# _name : undef
# _gender : Gender
# _age : int
+ Animal(name : undef, gender : Gender, age : int)
+ ~ Animal()
+ breed() : undef
Gender
#_gender
Cat
- _breed : Cat_breed
+ Cat(breed : Cat_breed, name : undef, gender : Gender, age : int)
+ ~ Cat()
+ breed() : undef
Cat_breed
Rabbit
- _breed : Rabbit_breed
+ Rabbit(breed : Rabbit_breed, name : undef, gender : Gender, age : int)
+ ~ Rabbit()
+ breed() : undef
Rabbit_breed
Dog
- _breed : Dog_breed
+ Dog(breed : Dog_breed, name : undef, gender : Gender, age : int)
+ ~ Dog()
+ breed() : undef
Dog_breed
-_breed