Veterinary clinic
by Ilja Kirjanov

Documentation

Intro

The main idea of this project is creating a program that allows User to operate on data of a veterinary clinic using GUI. Due to wide functionality the idea of the program can be easily changed to any other, so it can be any kind of a shop or any other service where databases are in usage.

In current state program allows User to add/delete/update objects of classes Dog, Cat, Owner, Doctor.

The main tools used in project are Python 3.8 and MySQL database. Also Python libraries as Tkinter (8.6) and MySQL-connector-python (8.0.19).
Additional libraries: PIL, fontTools.

Classes and structure

App.py

The central file of the program is called App. It consists of class App that allows program to start using Tkinter. Also there are classes and outside functions. Every class is a separated Tkinter page, and between pages we can comfortably switch. The graph of structure is presented below.

App
In class App are defined: min size and icon of the window, dictionary involving pages of the program, and method show_frame to show chosen frame.

Start
Just after running the program we will see the StartPage. Where we can see some greeting, button Start and button Reset database, which allows us to delete each row in each table from database.

Menu
After pressing button Star on StartPage we proceed to the next page Menu. On this page we see button Database and navigational buttons Exit and Back. If some functionalities were added we can efficiently include buttons linking to it.

Database
After pressing button Database on Menu page we proceed to the next page Database. On this page we see some buttons with icons illustrating classes of objects we can work with like Dog, Cat, Owner, Doctor. Navigational buttons Exit and Back are included. At this moment we have to decide what class we are interesting in.

Dog and Cat
If User choices dog or cat several possibilities are presented.
First of all, the table of the objects of the chosen class is presented at the bottom of the window.

On the left side of the window there is option to add new object of the corresponding to his choice class. There are obligatory characteristics and all fields have to be filled to create an object and add it to the database. User should choose one of the doctors and one of the owners presented in the database to create a pet.

In the center there is an option to delete an object with chosen ID from the shown table.

On the right side is the option to update object's attributes. No necessity to fill every field.

Doctor and Owner

If User choices doctor or owner similar to another option possibilities are presented.

The main difference is that User should not choose any other objects during adding the doctor or owner to database.
That means that User firstly has to create a doctor and owner and then create and connect pet to them.

Outside functions

At the file App.py on the bottom you can see definitions of functions for each class of objects and common functions for the pages.

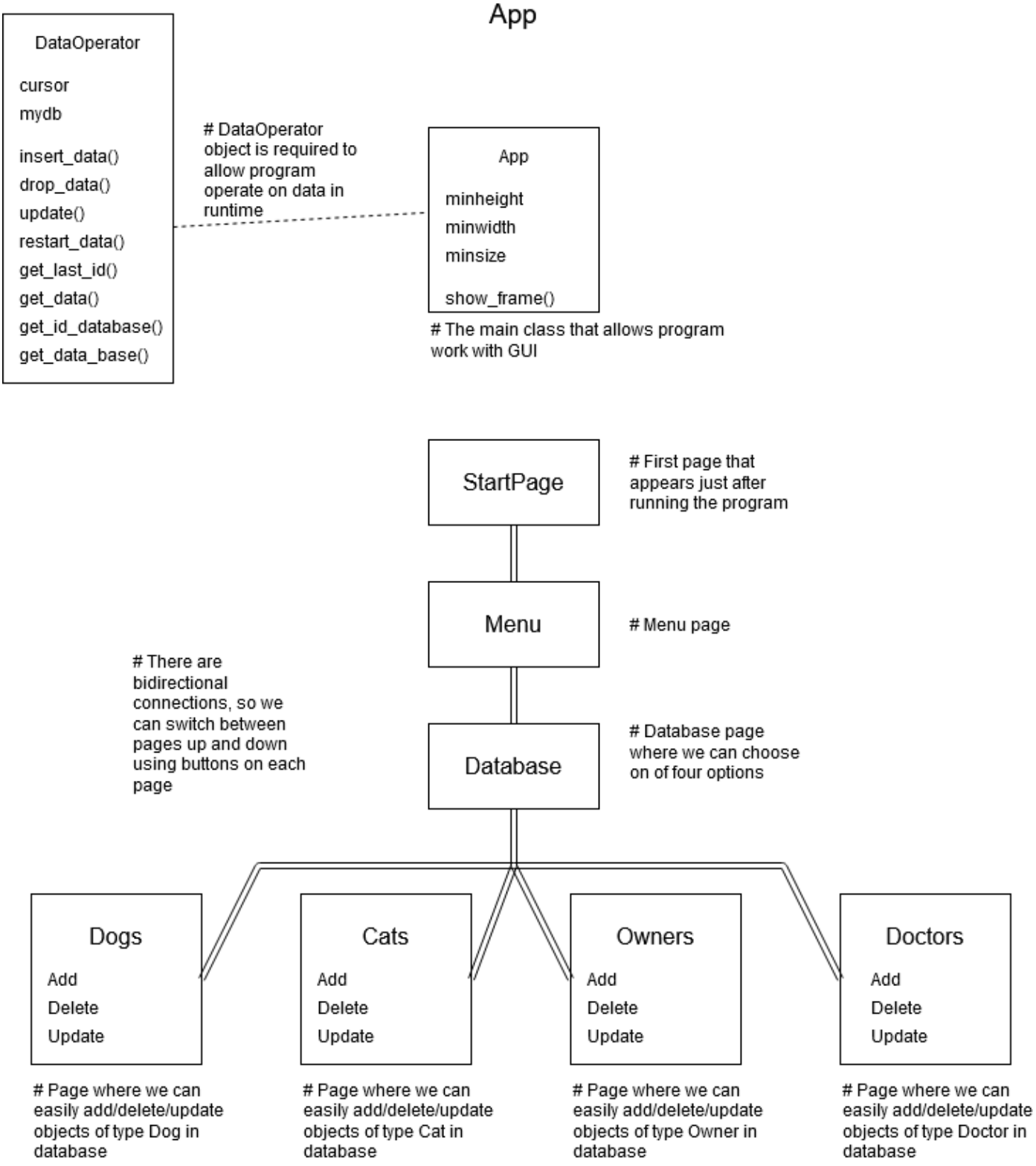
Let me show and explain Dog, Cat, Doctor and Owner methods on the example of Owner:

reset_values_owner(name, surname, contact) – function that resets field values in add option
create_owner(name, surname, contact) – creates owner object and calls method insert_data() of class DataOperator to add owner object to the database.
call_create_owner_and_reset_values(name, surname, contact, list_of_owners) - calls functions reset_values_owner() and create_owner().
show_owner_database() – allows to create a table illustrating all objects of class Owner from database.
delete_owner_and_reset_id_value(owner_id, list_of_owners) – function that deleted Owner object with chosen ID from the database.
call_update_owner_and_reset_values() – update owner object characteristics in database.

For Doctor, Dog and Cat were created same functions.

Below is presented structure of App with short comments.

App


DataOperator
cursor
mydb
insert_data()
drop_data()
update()
restart_data()
get_last_id()
get_data()
get_id_database()
get_data_base()
# DataOperator object is required to allow program operate on data in runtime
App
minheight
minwidth
minsize
show_frame()
# The main class that allows program work with GUI
StartPage
First page that appears just after running the program
Menu
Menu page
There are bidirectional connections, so we can switch between pages up and down using buttons on each page
Database
Database page where we can choose on of four options
Dogs
Add
Delete
Update
Page where we can easily add/delete/update objects of type Dog in database
Cats
Add
Delete
Update
Page where we can easily add/delete/update objects of type Cat in database
Owners
Add
Delete
Update
Page where we can easily add/delete/update objects of type Owner in database
Doctors
Add
Delete
Update
Page where we can easily add/delete/update objects of type Doctor in database

logger.py

This file contains only one function that allows to log in to the MySQL database and get cursor to it.

get_cursor() – returns cursor to the database.

This method is used during the creation of objects of type DataOperator and DataBaseOperator.

data_operator.py

DataOperator

In this file class DataOperator is initialized. The main idea of this class is operating on data in tables.
During initialization of DataOperator object linker to the database mydb is set using imported function get_cursor() from logger.py file.

Method of class DataOperator:
insert_data(obj) – takes object of type Person/Pet/Owner/Doctor/Dog/Cat and puts it in a corresponding table of the database.
drop_data(name_table, id_) – deletes object with given ID from the given table.
update(name_table, id_, dict) – updates object with given ID from the given table got from update field.
restart_data() – drops all rows from tables Dogs, Cats, Doctors, Owners. As well as all methods can be efficiently extended.
get_last_id() – sets last ID for parent classes Person and Pet in order to avoid dublication of ID's while adding objects to tables. Idea is such that ID's for pair doctors and owners as well as dogs and cats should be different.
get_data(table_name, id) – returns string from table corresponding to the object with given ID.
get_data_base(table_name) – returns whole data from the table.
get_id_data_base(table_name) – returns list of ID's from the whole table. Method is used to show the list of ID's of doctors and owners during addition of pet.

database_operator.py

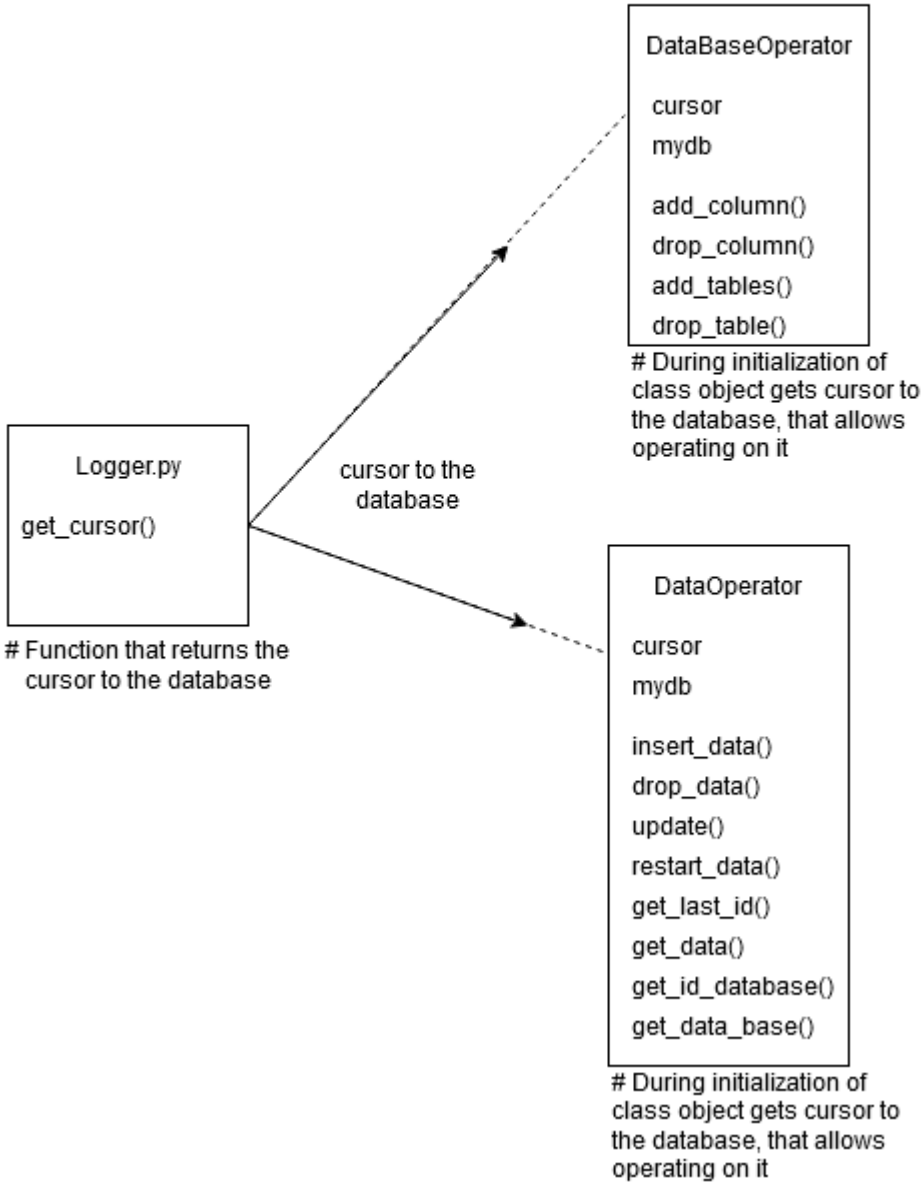
DataBaseOperator

In this file class DataBaseOperator is initialized. The main idea of this class is operating on the structure of database and separated tables.
During initialization of DataBaseOperator object linker to the database mydb is set using imported function get_cursor() from logger.py file.

Method of class DataBaseOperator:
add_column(name_table, column_name, data_type) – using MySQL query adds new column with given name and type at the end of the given table.
drop_column(name_table, column_name) - using MySQL query drops the column from the given table with given name.
add_tables() – main method for creating tables. In present state it adds 6 tables with connections between them. Can be comfortably modified using SQL queries.
drop_table(table_name) – drops table from the database.

Below is presented flow chart illustrating dependencies and structure of classes these two classes.

Operating on data



pet.py

Pet

In this file class Pet is initialized. Pet is a parent class for all pets, in present state for Dog and Cat but as usual we can easily inherit from it any other animal pet.
This class has some basic attributes like **name, age, breed, whether pet castrated or not, link to the doctor and to the owner**. Also it has class_counter and id that protects database from overriding ID's in table.
Class Pet has setters and getters for each attribute and also get_tuple() method that is used in setting data to the table and method get_info() prints the info about what the class it is.

dog.py

Dog

In this file class Dog is initialized. Dog is a child class of Pet. It inherits all attributes and methods, has additional attributes like **sound and whether a tail is cut or not**. For these additional attributes are initialized getters and setters. Method get_tuple and get_info() are overridden.

cat.py

Cat

In this file class Cat is initialized. Cat is a child class of Pet. It inherits all attributes and methods. There are no additional attributes, method get_info is overridden.

person.py

Person

In this file class Person is initialized. Person is a parent class for all persons, in present state for Owners and Doctors but as usual we can easily inherit any other type of person from it.
This class has some basic attributes like **name, surname, contact**. Also it has class_counter and id that protects database from overriding ID's in table.
Class Person has setters and getters for each attribute and also get_tuple() method that is used in setting data to the table.

owner.py

Owner

In this file class Owner is initialized. Owner is a child class of Person. It inherits all attributes and methods. There are no additional attributes.

doctor.py

Doctor

In this file class Doctor is initialized. Doctor is a child class of Person. It inherits all attributes and methods, has additional attributes like **position and salary**. For these additional attributes are initialized getters and setters. Method get_tuple is overridden.

Below is presented the flow chart illustrating dependencies and structure of classes these six classes.

Veterinary clinic