**Report**

**b)**The existing system provides pet owners a way of logging and saving pet information they enter over any period of time. The application is a GUI that opens to multiple windows, it has button interaction capabilities, picture upload and display options, text output boxes, and text input boxes for the user to enter information. There is a login for returning users and a sign-up page for new users. The user can add multiple pets and each pet can have their own pet profile. They can log information regarding their pet’s medication, exercise, and diet. The user can also upload images to set as the pet profile picture. The option to generate a report of the information that has been accumulated over time is formatted into a PDF file that can easily be shared with pet sitters and veterinarians.

One of the main features that can be improved upon is the report generation. The generated report needs to be formatted better so that the user can obtain a well-organized and easy to understand report detailing their pet information. Another feature that can be added to improve the application is the ability to find pet services near the user’s area. There could be a ‘near me’ functionality added to the application to provide a convenience of finding sitters or veterinarian services. One other improvement we could implement is to upgrade the application design and have it consistent across all the windows. For example, the window buttons and fonts are not consistent. It should also validate the contact number of the pet owner using OTP mechanism so that pet sitter can easily contact pet owner in case of emergency. We can also improve this application to have a notification service that can remind pet owner for the follow up appointment with veterinarians. User experience can also be improved by aligning the text fields and buttons properly.

**c)**

Prototype

The prototype design pattern could be used to efficiently handle the pet information log: medication, diet, and exercise. These UI information windows are displayed using the same layout, design, and asks for the same information. The only differences are the titles and where the information is stored. The prototype pattern allows us to create the information page objects from an existing default object. The objects that are created are the same type and we would need to store the information over time.

Flyweight

Since one pet owner can have multiple pets of the same type, the flyweight design pattern allows the sharing of information between similar objects while minimizing memory usage. This design pattern would benefit large scale users like animal rescues and veterinary clinics. An example of this would be a large-scale rescue that has 25+ dog type pet profiles. All the dog profiles can share copies of the same attributes, but when we need to get the information from a specific dog, we can use an external data structure to pass the specific dog object information to the client. This helps to mitigate large scale memory usage.

ObjectPool:

ObjectPool can be used for PetMe+ since we need to create only a limited number of different kinds of objects (i.e. a pet). By using an object pool, we could avoid creating new objects each time, and instead reuse the ones that it has finished using rather than letting them be garbage-collected. The demand for objects in the program does not vary over time and therefore, ObjectPool would work well. It also ensures that they don't consume an excessive amount of memory with objects awaiting reuse.

Abstract Factory

Since all the pet profiles follow the same general format, we can use the abstract factory design pattern to create a family of related pet objects that have similar attributes. Since there is only one concrete factory, it will be easy to change the concrete factory uses. This gives our application the flexibility to add different attributes that are specifically tailored for certain pets. Using an abstract factory will allow us to use a pet type factory and a care type factory, so that each pet will have the correct care instructions. An example of this would be the difference in grooming; dogs would go to a groomer, while reptiles would need to have their tanks cleaned. Therefore, tailoring custom services for each type of pet while maintaining the integrity of the family of pet objects.

Singleton

Since the database and location managers are getting used across different parts of the project and it is also expensive to initialize the database manager and location manager. We can use singleton for such objects so that we are not creating unnecessary number of objects and not consuming runtime memory. We can do lazy loading for these kinds of objects so that the runtime memory is not consumed up front. It will be initialized when we want to use these objects.

Screenshots:

  












