RELATÓRIO DE TRABALHO PRÁTICO

Trabalho Prático LP2 – PetLovers

AURELIEN BOUÇA & ELDEN CARONES

ALUNO Nº 6163 & 4459

Trabalho realizado sob a orientação de:
Luís Ferreira

Linguagens de Programação II

Licenciatura em Engenharia de Sistemas Informáticos

Barcelos, junho de 2020

Índice

1 Introdução

Como forma de avaliação da unidade curricular de linguagens de programação, foi-nos proposto o desenvolvimento de uma solução em C# que aporte uma resolução a um problema real e de complexidade moderada.

Sendo 2020 um ano atípico, quer pela disrupção da sociedade, devido ao confinamento pandémico e mais recentemente pela segregação racial, inúmeros problemas poderiam ser explorados. Optamos por desenvolver uma solução que consiga ajudar aqueles que não conseguem falar e que não se conseguem defender, acabam por ser marginalizados com indiferença e abandono – animais de rua.

É assim, o grande objetivo deste trabalho prático iniciar o desenvolvimento de uma plataforma que seja o principal ponto de recolha e disponibilização de informação de uma associação de animais de rua, disponibilizando o mesmo nível de informação atempada, atual e com valor a todos os níveis da associação. Assim contribuiremos para que os voluntários passem a estar mais preocupados com o tratamento e carinho aos animais do que com questões repetitivas ou tarefas burocráticas e manuais.

2 Objetivos

Como parte da avaliação do trabalho prático, foram-nos propostos diferentes objetivos, sendo eles separados em 3 grupos distintos: gerais, concretos e outros. Abaixo o detalhe a conclusão de cada.

2.1 Objetivos gerais

1. Consolidar conceitos basilares do Paradigma Orientado a Objetos – cumprido;

   Com a evolução das 3 fases de desenvolvimento da solução, começando o desenvolvimento sem metodologias nem respeitando as boas práticas de programação orientada a objetos, foi notória a vantagem de desenvolver com um paradigma orientado a objetos.

2. Analisar problemas reais – cumprido;

   A análise do problema, traduzindo figuras reais e problemas reais para uma solução em C# foi uma aprendizagem de pensar sistemas e de como implementar algo complicado, mas que tem de permanecer simples para o utilizador final.

3. Desenvolver capacidades de programação em C# - cumprido;

   Toda a solução foi desenvolvida em C#. Todas as tecnologias utilizadas, foram utilizadas com C#.

4. Potenciar a experiência no desenvolvimento de software – cumprido.

   As diferentes fases de desenvolvimento da solução levaram a que a experiencia de desenvolvimento fosse levada a outro nível. Foi possível identificar que cada decisão tomada, boa ou má, teria consequências mais tarde na expansão e modificação da solução.

5. Assimilar o conteúdo da Unidade Curricular – cumprido

A conclusão do trabalho prático cela a assimilação do conteúdo da UC.

2.2 Objetivos Concretos

1. Saber identificar e implementar classes e objetos num problema real – cumprido.

   Foram identificadas e implementadas as diferentes classes que se traduziram em objetos quem suportam a resolução do nosso problema.

2. Saber produzir código com qualidade, de acordo com a norma CLS e bem documentado – cumprido.

   Todo o código foi desenvolvido e documentado seguindo as normas indicadas pelo docente, respeitando as normas CLS e os documentos XML.

3. Saber gerar a API com a documentação do código que produziu;
4. Saber aplicar os pilares da POO: Herança, Encapsulamento, Abstração e Polimorfismo – cumprido.
5. Saber estruturar devidamente uma solução em bibliotecas de classes – cumprido através da utilização do NTier.
6. Saber estrutura uma solução por camadas, seguindo padrões como NTier ou MVC – cumprido (NTier)
7. Saber analisar o código que produziu e explorar das vantagens que o C# oferece.

3 Análise e Desenvolvimento

3.1 Análise do problema

Hoje, uma associação especializada em recolha e abrigo de animais de rua, não pode ter uma gestão de *sebenta e lápis*.

Se o objetivo continua a ser proteger, alimentar e recolher animais de rua, com a crescente população de animais nos canis, a sua gestão deixou de ser trabalho de uma pessoa passando a ser necessário apoio em todas as áreas e de todas as pessoas quanto possíveis.

Com a escolha da atual solução e antes de proceder ao desenvolvimento, decidimos que iríamos dedicar mais um bocado de tempo ao desenho das classes, deixando a solução capaz de ser expansível e modular.

Decidimos assim explorar as diferentes classes e quais a forma mais correta das estruturas de dados a utilizar.

Decidiu-se que a solução poderias expandir-se em 4 partes:

- Gestão de pessoas: Pessoa, Funcionário, Socio & Veterinário;
- Gestão de animais: Animal, Cão, Gato, Boletim Sanitário & Tratamentos;
- Organização de animais dentro do canil: Parque, Setor (sénior, gatil, maternidade, etc.);
- Gestão Sócio: socio & Quotas;

Para avaliação do trabalho prático decidimos explorar a Gestão de animais e parte da Gestão de pessoas.

Sabe-se que uma associação que recolhe animais de rua, não pode misturar cães e gatos, optando muitas vezes por apenas prestar apoio a um tipo de animal (canil vs. gatil). Para albergar os dois seriam necessárias condições para que os gatos se sentissem em segurança e que os cães não se sentissem provocados.

Todos os animais estão protegidos em parques e cada um tem um boletim sanitário próprio onde irão ser registados os diferentes tratamentos efetuados assim como a data de nascimento ou outras informações relevantes.

3.2 Análise estruturada em objetos

De forma a traduzir o problema em análise foram identificadas diferentes classes que serviriam de referencia para estruturar o problema: Pessoa, Animal e Boletim Sanitário. Pessoas e Animais por razões obvias e boletim sanitário pois queríamos uma *estrutura* que nos permitisse registar a *vida* do animal durante a estadia no abrigo.

Por estarmos a programar por *objetos* e para melhor tirar partido dos conceitos de herança, polimorfismos e encapsulamento, convencionamos que a classe Pessoa não seria instanciada servindo de base de derivação das classes secundárias Sócio, Funcionário, veterinário, reaproveitando assim a maior parte do código desenvolvido. Decidimos que um sócio voluntário não iria derivar da classe socio (só sócios podem ser voluntários). Decidimos que um Sócio teria antes um perfil de voluntário.

As classes Cão e Gato, derivadas da classe base Animal respeitam os mesmos princípios evocados anteriormente de herança. No entanto, sabendo que um animal pode estar alojado em diferentes parques ao longo da sua estadia, dependendo das necessidades do canil, normalizou-se a relação Animal Parque para podermos ter uma relação 1-N|N-1 de forma a podermos escalar a relação e reorganizar tanto quanto necessário. O Parque pertenceria a um setor. Por ser uma relação mais estável, optamos por não normalizar a relação.

Sabendo que cada Animal tem um boletim sanitário, decidimos que a Classe veterinário estaria ligada ao animal através dos tratamentos administrados e assim normalizando a classe permitindo que relações muitos para muitos possam ser expandidas.

Em relação às estruturas de dados utilizadas decidiu-se que a estrutura de dados utilizadas seriam *<Lista>system.generics.* A única exceção foi o boletim sanitário, por termos de armazenar diferentes tipos de dados, o ArrayList system.collections permite uma maior liberdade e versatilidade.

O Diagrama de classes completo pode ser consultado no ficheiro do Visual Paradigma *Lp2 201920.vpp* e na solução desenvolvida.

3.3 Paradigma de desenvolvimento

Por defendermos que uma classe só, não deve ter responsabilidades unilaterais, desde a impressão de dados no ecrã até ao armazenamento dos mesmos em ficheiros ou base de dados, decidimos seguir o paradigma de N-Tier.

Foram criadas assim 3 camadas distintas de programação:

BO -> Objetos/ instâncias a utilizar

BR -> Regras aplicadas aos objetos de negocio e ponte de comunicação entre a camada de apresentação e de manipulação de dados.

DAL -> Camada responsável pela manipulação de dados.

Existem também as camadas de apresentação e de exceções que foram criadas com o objetivo de modular o programa de forma a garantir independência e reaproveitamento.

4 Conclusão e sugestões futuras

Ao longo de todo o desenvolvimento da solução, passamos por diferentes estágios de aprendizagem assim como diferentes níveis de abstração na programação. Quanto menor a abstração, mais limitada a evolução da solução. Quanto maior da estruturação e o nível de modularidade maior a liberdade de evolução ou alteração da solução.

Achamos que a não inclusão de interfaces nos facilitou o desenvolvimento de forma direta, mas que nos limita no reuso de código e na lógica de programação. Se a classe Animal derivasse de um Interface Animal, poderíamos dizer que a Lista Animal Parque poderia receber um objeto que respeita a interface e assim se *amanha* o canil abrigar outro tipo de animais não seria necessário desenvolvimento adicional.

Também a relação Boletim Sanitário/ Tratamentos poderia ser explorada mais extensivamente sendo mais abstrata a abrangente.

No geral, foi um trabalho que nos enriqueceu e nos permitiu explorar um pouco da programação por objetos e todas as possibilidades que este paradigma nos oferece. O facto de não termos perdido tempo com *menus e janelinhas* também nos ajudou a concentrar mais exaustivamente no core da solução.

Bibliografia

https://stackoverflow.com/questions/3007711/is-file-empty-check
https://stackoverflow.com/questions/35539928/auto-increment-id-c-sharp
https://stackoverflow.com/questions/759133/how-to-display-list-items-on-console-window-in-c-sharp