BIKE SHARING

07/11/2014



André Lago: [up201303313@fe.up.pt](mailto:up201303313@fe.up.pt)

Edgar Ramos: [up201305973@fe.up.pt](mailto:up201305973@fe.up.pt)

Leonardo Ferreira: [up201305980@fe.up.pt](mailto:up201305980@fe.up.pt)

**Tema de trabalho**

O nosso trabalho incide sobre os meios de transportes mais amigos do ambiente, as bicicletas, mais precisamente sobre a possibilidade de partilha das mesmas.

Por outras palavras, o trabalho tem em perspetiva a gestão de uma cooperativa própria para possibilitar a partilha de bicicletas, bem como um conjunto de situações que essa mesma partilha sugere (temos como exemplo a reparação de avarias, a manutenção de utilizadores, de empresas e dos postos de serviço, entre outros).

**Solução implementada**

A solução que implementamos visa à gestão de uma cooperativa de Bike-Sharing, possibilitando uma interface com diferentes menus, que permite acesso generalizado a utilizadores (registados os ocasionais) e aos serviços de manutenção da rede, cada um com privilégios e permissões diferentes.

Cada utilizador registado é identificado através de um nome, idade e palavra passe. Este pode alugar e devolver bicicletas, sob a regra que cada utilizador apenas pode alugar uma bicicleta de cada vez. O utilizador é representado pela classe ‘Utilizador’, cujos membros-dado representam aqueles descritos acima. Para além destes, a cada utilizador é associado uma lista de registos (no formato de um vetor de apontadores para objetos da classe ‘Registo’), que grava o historial de alugueres desse utilizador. Estes utilizadores pagam um valor fixo de utilização do serviço, estabelecido em 40€.

Os utilizadores ocasionais são identificados pela classe ‘Ut\_ocasional’, que é uma classe derivada de ‘Utilizador’. Contudo, estes apenas são identificados pelo nome, tipo e cartão de crédito, sendo-lhes atribuídos uma palavra-passe vazia e uma idade aleatória, pois estes membros não são necessários no sistema. Os utilizadores ocasionais apenas podem alugar e devolver bicicletas, segundo a mesma regra de que nenhum pode alugar mais do que uma bicicleta de cada vez. Quando devolvem a bicicleta, estes utilizadores recebem uma mensagem a informar o número de dias durante os quais alugaram a bicicleta, e o preço a pagar por essa duração do serviço.

Aos serviços de manutenção da rede são permitidas uma variedade grande de ações. Estes podem executar operações CRUD (create, read, update, delete) sobre utilizadores, empresas, bicicletas e postos de serviço.

**NOTA** : na implementação da solução, decidimos que sempre que um objeto de qualquer um destes referidos for apagado, todos os registos de aluguer que o envolvam devem também ser apagados.

Os registos de aluguer são constituídos por elementos identificativos das entidades envolvidas num aluguer de uma bicicleta. Desta forma, os registos identificam o ID da bicicleta, o nome do utilizador que a alugou, a data de entrega e devolução da bicicleta (a data é representada por um objeto da classe ‘Data’, referida à frente), o ID dos postos de levantamento e entrega da bicicleta e um booleano, que identifica se a bicicleta se avariou durante esse aluguer.

Cada posto de serviço é identificado por um objeto da classe ‘PostoServico’. Cada um destes objetos possui um ID que identifica o posto, uma lotação e uma ocupação, um vetor com o registo de utilização desse posto e dois vetores que armazenam as bicicletas existentes no posto. Um, possui bicicletas disponíveis para aluguer. O outro, possui as bicicletas que se encontram avariadas, pelo que se encontram indisponíveis, mas são consideradas no cálculo da ocupação do posto. No modo de gestão do sistema, é permitido fazer a reparação de uma ou mais destas bicicletas.

Cada bicicleta é representada por um objeto da classe ‘Bicicleta’. Este, possui um ID que identifica a bicicleta, o nome da empresa que forneceu essa bicicleta, o tipo, preço, tamanho e número de velocidades da bicicleta, um booleano que indica se a bicicleta se encontra avariada ou não e um vetor com o registo de utilização da bicicleta.

Cada empresa é representada por um objeto da classe ‘Empresa’, que possui um nome da empresa e um vetor de bicicletas da empresa. Todas as bicicletas do sistema devem estar associadas a uma empresa.

Por fim, existe a classe ‘Data’, criada para manter a ordem cronológica dos acontecimentos no programa. Esta, possui como membros ano, mês e dia, e possui métodos e construtores que permitem uma interface fácil com datas, com várias conversões já existentes.

Implementamos também os ficheiros ‘Tools.h’ e ‘Tools.cpp’ com algumas funções que foram úteis em várias partes do programa.

No que toca ao armazenamento da informação, optamos por um formato que mantém a informação em 4 ficheiros de texto, ‘rede.txt’, ‘rede\_user.txt’, ‘rede\_bikes.txt’ e ‘rede\_spots.txt’. O formato do armazenamento está especificado nos comentários do ficheiro ‘Rede.cpp’ e tem em atenção o facto de todos os objetos que possuem registo de utilização, o possuírem através de um vetor de apontadores para registos. Isto implica que o registo correspondente a vários objetos seja o mesmo objeto, apontado por apontadores diferentes. Desta forma, o processo de guardar e carregar a informação assegura-se de que os registos apontados são os mesmos, fazendo-o através do método especificado nos comentários desse ficheiro.

Esta solução apenas funcionará a 100% em Windows.

**Dificuldades encontradas**

Encontramos algumas dificuldades na conciliação das classes pois por vezes foi complicado perceber quais as dependências que cada uma delas tinha. Por outro lado, a programação paralela (com o uso de repositórios) por vezes torna-se complicada, pelo que é necessária muita atenção para que não haja sobreposição de código.

No que toca ao uso de repositórios (no nosso caso, GitHub), a divisão das classes entre nós não se mostrou fácil. Nesta solução, as classes que implementamos dependem muito umas das outras e fazer isso em separado e por pessoas diferentes causou alguns problemas nas interfaces entre classes. Contudo, de uma forma geral, consideramos a abordagem através de repositório positiva pois permitia manter todos os códigos sincronizados e atualizados.

Não obstante, outra das dificuldades encontradas na composição deste projeto centrou-se no facto de ter sido por vezes complicado, prevenir certas situações que são incorretas ou proibidas de se fazer como por exemplo: o mesmo utilizador alugar mais do que uma bicicleta ao mesmo tempo ou o mesmo querer alugar uma outra bicicleta sem ter antes devolvido a alugada primeiramente. Ter em atenção as datas de outros registos, a existência de outros alugueres, e outras situações como criação de objetos repetidos mostrou-se como um dos grandes obstáculos na criação do trabalho.

**Esforço dedicado pelos elementos do grupo**

O trabalho de grupo inicialmente foi distribuído uniformemente pelos elementos de grupo. Porém, é evidente que mediante as capacidades e conhecimentos de programação de cada um dos elementos do grupo, houve diferenças na quantidade de programação.

Inicialmente, as classes foram divididas entre os membros do grupo (Edgar com ‘Bicicleta’ e ‘Registo’, Leonardo com ‘PostoServico’ e ‘Empresa’, André ‘Utilizador’, ‘Data’ e ‘Rede’, mas em ‘Rede’ algumas partes foram feitas por todos em conjunto).

Por conseguinte, embora haja diferença na quantidade de programação, também devido à rapidez de programação de cada um dos elementos, podemos afirmar que o esforço dedicado pelos elementos do grupo foi aproximadamente igual, bem como as horas dedicadas ao mesmo.

**Diagrama UML**





