U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

ANO LETIVO 2017/2018

# Partilha de Bicicletas

Aplicação para utilizar e gerir o sistema de partilha de bicicletas

*Algoritmos e Estruturas de Dados*
*Mestrado Integrado em Engenharia Informática e Computação*

**Estudantes & Autores:**
*Pedro Lopes* up201603557@fe.up.pt
*Eduardo Silva* up201603135@fe.up.pt

FEUP - Faculdade De Engenharia da Universidade do Porto

*João Alves* up201603143@fe.up.pt

# Resumo

Este projeto foi realizado no âmbito da unidade curricular de Algoritmos e Estruturas de Dados, da Faculdade de Engenharia da Universidade do Porto, com o objetivo de desenvolver uma aplicação para a gestão da partilha de bicicletas numa cidade, utilizando a linguagem de programação C++.

No final do projeto, o usuário deverá conseguir usar a aplicação, não só para requisitar, usar e devolver uma bicicleta, como também para realizar o pagamento pela utilização do serviço e consultar informações quanto às estações de partilha que se encontram espalhadas pela cidade. Para além disso, para efeitos de teste e demonstração, também criamos um submenu, que nos permite aceder a diversas funções para simular uma situação prática da aplicação.

Na realização deste projeto, começámos por planear uma estrutura geral de classes que iriamos necessitar. De seguida, dividimos cada um dos “módulos” necessários para o funcionamento do programa por cada um, de forma a construir, passo a passo, as várias funções das várias classes.

# 1. Índice

Resumo.....2

1. Índice.....3

2. Solução Implementada .....4

3. Diagramas de UML .....7

4. Casos de Utilização .....8

5. Dificuldades Encontradas .....5

6. Distribuição de Trabalho ..... 10

7. Conclusão ..... 11

# 2. Solução Implementada

## Parte 1

Todo o projeto assenta no desenvolvimento de uma aplicação que pode ser usada tanto por usuários do dia a dia, como também pela própria administração de todo o sistema de partilha de bicicletas. Na prática, aconteceria que existiriam duas "aplicações" distintas, que separariam as duas funcionalidades acima referidas. Contudo, não sendo esse o objetivo do trabalho, criamos apenas uma aplicação, e admitimos que os utilizadores não sócios não tem acesso à parte administrativa do sistema.

Na base da solução que implementamos está a criação de classes, que se relacionam através da herança de atributos de umas classes para as outras. Usamos também ficheiros de texto como bases de dados, para guardar a informação relativa aos sócios, às estações de partilha, entre outros, de forma a mantermos informação de uma execução do programa para a outra. Por fim, decidimos adaptar um sistema de localização que funciona com base num referencial cartesiano, utilizando apenas o 1º quadrante, de forma a simplificar a implementação. A razão pela qual optamos por este método está explicada mais aprofundada no capítulo das Dificuldades Encontradas.

A classe principal, à qual se pode atribuir o conceito de superclasse, é a classe HQ ("Headquarters"), que serve como base para o funcionamento de todo o sistema. Nesta classe estão declarados três vetores: um vetor de apontadores para os usuários ativos no momento (que arrendaram uma bicicleta e ainda não a devolveram), outro vetor de apontadores para os sócios e, por fim, um vetor de apontadores para as estações de partilha. Estes vetores permitem-nos maior eficiência e acessibilidade na manipulação dos dados, que é necessária devido às funcionalidades que implementamos. Para além disso, na classe HQ também estão implementados todos os métodos relacionados com a manipulação e pesquisa de dados nestes vetores e nos ficheiros de texto.

Existe ainda a necessidade de existir uma classe para definir cada um dos utilizadores, que é a função da classe "User", cujos atributos são o nome, a sua localização, um booleano que indica se está ativo e a bicicleta que está a usar. No entanto, também é necessário distinguir entre um utilizador normal e um sócio. Portanto, criamos duas subclasses da classe "User": a classe "Member" e a classe Regular, que se referem aos sócios e aos utilizadores normais, respetivamente. A diferença entre estas duas subclasses reside no facto de, no caso dos sócios, ser guardado o tempo total do mês em que este arrendou a bicicleta e o balanço do preço que irá pagar, enquanto que no outro caso, apenas é necessário calcular e guardar o valor final do pagamento.

Para ser possível associar a cada objeto da classe "User" uma bicicleta, criamos uma classe "Bike". Ainda, para distinguir os vários tipos de bicicletas e o respetivo custo de cada uma, criamos quatro subclasses, "Urban_b", "Urban_simple_b", "Child_b" e "Race-b", em cada uma possui um só atributo, o custo da bicicleta por hora.

Para definir cada um dos pontos de partilha de bicicletas, implementamos a classe Station, cujos atributos são o nome, o número máximo de bicicletas que pode guardar, as bicicletas que se encontram disponíveis e a sua localização.

Por último, criamos uma classe “Date”, que facilita na manipulação do tempo (questão da manipulação do tempo também explicada no capítulo das Dificuldades Encontradas) para ser possível realizar os cálculos do custo da utilização do serviço.

Existem, ainda, os ficheiros “Funcs.ccp” e “Funcs.h,” que usamos para definir funções gerais, que não se enquadram necessariamente em nenhuma classe específica. Por isso, e para serem de mais fácil acesso, estão definidas em ficheiros à parte dos ficheiros das classes. Por fim, definimos ainda várias exceções para identificarmos e limitar determinada informação que o utilizador pode inserir ao executar o programa (por exemplo, tentar adicionar um utilizador com o mesmo nome).

## Parte 2

Para a implementação das funcionalidades na segunda parte do trabalho, não se verificou a necessidade de alterar a estrutura que já tinha sido programada na parte anterior. Houve apenas a necessidade de acrescentar classes e estruturas para que o nosso programa correspondesse ao pedido no enunciado.

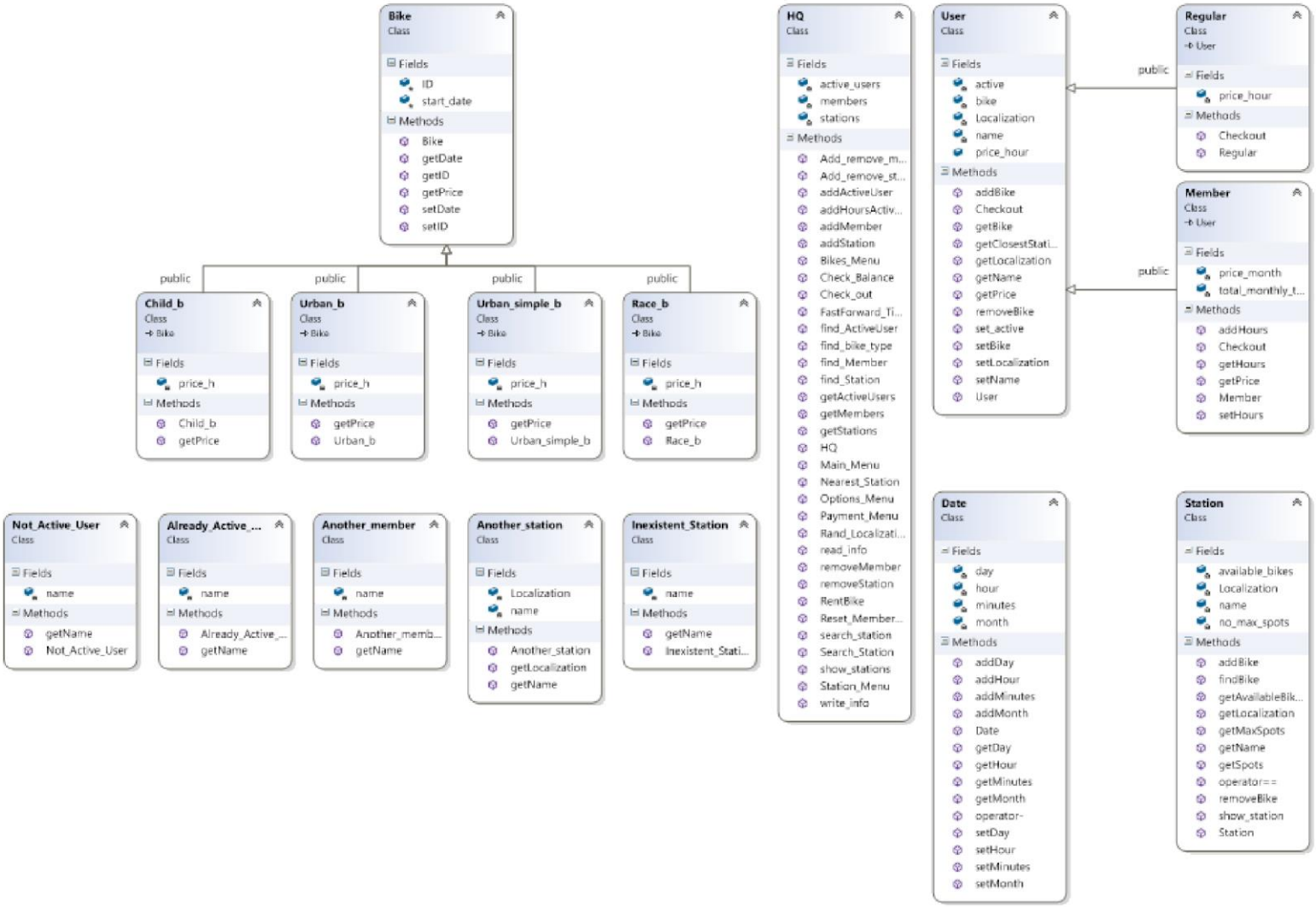
Em primeiro lugar, implementamos uma nova classe Part, cujos atributos são o nome, o fornecedor e o preço. Através do ficheiro BST.h disponibilizado no moodle, foram implementadas diversas funções para a compra das partes das bicicletas, dando a possibilidade ao utilizador de arranjar a peça mais barata de todas as oficinas, com base numa árvore binário de pesquisa.

Seguidamente temos a implementação da tabela de dispersão que permite registar a destruição das bicicletas, através das diversas funções que foram implementadas.

Por fim, temos a classe Shop, cujos atributos são o nome, a sua reputação e um vetor com o stock de bicicletas, que representa uma loja de bicicletas, onde é feita a compra das mesmas. Para além disso, e mais especificamente para esta classe, foi implementado um enum que permite definir os 4 tipos de bicicletas disponíveis, e com a ajuda de um typedef “bike_stock”, que na realidade define um par em que o primeiro elemento é um tipo de bicicleta (enum) e o segundo é a quantidade de bicicletas desse tipo, permite definir o stock como um vetor de apenas 4 elementos.

Todas as funcionalidades podem ser executadas através dos diversos menus, implementados na classe HQ, previamente definida. Para além disso, criou-se ainda uma fila de prioridade nesta classe, shops, cujo critério de organização é a reputação de cada loja, obtida através da avaliação dos consumidores.

# 3. Diagramas de UML



# 4. Casos de Utilização

## Parte 1

A nossa implementação tem como objetivo final funcionar como uma aplicação de telemóvel. Por isso, o utilizador tem acesso a vários submenus que lhe permitem tirar partido do serviço de partilha de bicicletas. Contudo, existe um submenu ao qual um utilizador normal não tem acesso: o menu “Options”. Este menu, apenas é visível para a parte administrativa do serviço, ou seja, nós que desenvolvemos o projeto e que o iremos testar.

Tendo isto em conta, um utilizador normal, quer seja sócio ou não, pode:

1. Submenu “Bike”:
    a. Arrendar uma bicicleta;
2. Submenu “Payment”:
    a. Consultar o seu balanço em tempo real - o que terá de pagar se entregar a bicicleta naquele preciso momento;
    b. Entregar a bicicleta para realizar o pagamento;
3. Submenu “Stations”:
    a. Aceder a diversas informações sobre as várias estações de partilha, podendo procurar uma estação específica ou a mais próxima de si e obter a descrição da mesma;
    b. Imprimir no ecrã informação de todas as estações.

Existe ainda o submenu “Options”, que nos permite “gerir” o serviço de partilha de bicicletas e testar várias situações práticas, que de outra forma seria quase impossível de testar:

a. Adicionar ou remover um sócio ao vetor de apontadores de sócios;
b. Adicionar ou remover uma estação ao vetor de apontadores de estações;
c. Avançar no tempo, de forma a alterar a posição dos utilizadores e o seu balanço.

## Parte 2

Nesta segunda parte, o objetivo final da implementação é diferente do objetivo que foi definido na primeira parte. Enquanto que inicialmente o objetivo era desenvolver a interface para um utilizador normal tirar partido do serviço de partilha de bicicletas, esta segunda parte relaciona-se muito mais com a gestão e manutenção deste serviço. Agora, no menu Options, é também possível:

a. Aceder à lista de lojas de bicicletas e comprar bicicletas;
b. Comprar partes de bicicletas;
c. Destruir bicicletas que não se encontram utilizáveis.

# 5. Dificuldades Encontradas

Na implementação de qualquer programa, são encontradas dificuldades no que toca à passagem das funcionalidades que o programa tem de ter para as funções que o programador tem de criar para satisfazer as mesmas. Este projeto não foi exceção.

Numa primeira abordagem ao enunciado e à esquematização do nosso projeto, encontramos duas grandes dificuldades.

A primeira relacionava-se com o sistema de localização que poderíamos usar para localizar tanto os utilizadores como as estações de partilha. Surgiram várias ideias, como criar um mapa baseado num mapa real (do género Google Maps) e guardá-lo numa estrutura, e definir um conjunto de estradas que os utilizadores poderiam percorrer para calcular qual seria a estação mais perto, entre outras. No entanto, chegamos à conclusão que usar um simples referencial cartesiano xOy iria ser a solução ideal, pois para além de ser fácil de trabalhar com esse referencial (usando as coordenadas x e y, com números inteiros e positivos apenas), também conseguimos pôr em prática o programa de uma forma mais ou menos realista. Ainda decidimos que a localização dos utilizadores, inicializada quando estes são criados, era definida pela função rand(), possibilitando a criação de várias situações de forma automática, sem ser necessária a atribuição manual das coordenadas.

A segunda dificuldade surgiu-nos quando tivemos de equacionar um sistema para realizar a contabilização do número de horas durante as quais cada utilizador arrenda uma bicicleta. Acrescido a esse problema, tinhamos também os sócios, que realizam o pagamento apenas ao final do mês, sendo necessário simular a passagem de um mês para haver o pagamento do serviço. Usar o próprio "timer" do computador seria uma opção totalmente inconcebível, pois seria impossível testar o balanço e pagamento de um sócio, visto que numa execução jamais esperariamos umas horas, quanto mais um mês, para fazer a acumulação do tempo de arrendamento de bicicletas. Optamos, assim, por criar uma classe "Date", que representa o tempo abstrato necessário para a implementação do programa. Esta classe "Date", cujos atributos são o mês, o dia, a hora e o minuto, permite-nos não só ter um

"tempo" definido de acordo com as nossas necessidades, como também manipulá-lo para realizar os testes e simular uma aplicação prática deste projeto.

Ao longo do resto do projeto, as dificuldades encontradas foram menos relevantes que as que enfrentamos no planeamento do projeto. Essas dificuldades estavam associadas ao desconhecimento de certas ferramentas utilizadas, como o programa para a criação da documentação e os diagramas UML, e de certas especificações da linguagem C++.

# 6.

## Parte 2

Não foram encontradas dificuldades relevantes o suficiente para as descrever aqui no relatório.

# Distribuição de Trabalho

## Parte 1

No que toca à distribuição de tarefas no desenvolvimento do projeto, durante o planeamento, todos opinaram sobre as diversas decisões tomadas no que toca à estrutura do programa, e foi atribuída uma de três grandes ramos do projeto a cada elemento: a classe “Bike”, a classe “Station” e a classe “User”. Apesar dessa distribuição, observou-se que existia bastante ligação entre as várias estruturas, e no fim, todos contribuiram para cada um dos grandes ramos, e como é óbvio, para as restantes classes e funções que foram criadas.

## Parte 2

Para esta segunda parte, decidimos adaptar a metodologia que usamos na primeira, dividindo cada um dos três principais ramos, a oficina, a sucata e a loja, a cada um dos elementos. Desta vez, como os conceitos e conhecimentos que tínhamos de aplicar a cada uma das estruturas era bastante distinto, não se verificou tanta distribuição de trabalho dispersa por todos os ramos. Portanto, foi algo que permitiu que cada um explorasse, da sua maneira, o conceito e o enunciado.

# 7. Conclusão

## Parte 1

Em suma, todo o trabalho revelou ser uma mais valia para todos os elementos do grupo, visto ter-nos dado a oportunidade de aprofundar conhecimentos em áreas como a linguagem C++, que apesar de já nos ser familiar, consegue ainda ser explorada, e também entender novos conceitos, como o UML e a documentação “Doxygen”, que podem revelar-se ser uma mais valia no futuro.

Analisando a evolução dos nossos conhecimentos da linguagem C++ e da gestão de desenvolvimento do projeto, sem dúvida que cada elemento se considera mais apto para o desenvolvimento de futuros projetos e melhoramento deste mesmo. Apesar de para nós estar bem executado, todos admitimos que existem diversos aspetos que podem ser melhorados em termos de eficiência e de implementação, que tornariam este projeto mais eficiente e talvez mais próximo de uma situação real e prática.

## Parte 2

A segunda parte do trabalho funcionou como um complemento da primeira parte, dando oportunidade aos alunos, não só de melhorar a estrutura de dados que já estava implementada, como também aplicar novos conceitos sobre Programação dentro do mesmo projeto. Tudo isto torna possível a evolução do conhecimento de um aluno, pois mais importante do que desenvolver um trabalho forte em quantidade, é também necessário olhar para o que já está feito, e analisar de que forma poderia ser melhorado.