

# Gestão de Biblioteca da UPT

40103 – José Santos

40160 – Jorge Sousa

40445 – Pedro Teixeira

40593 – João Cerqueira



Informática

Engenharia de Software

Prof. Doutora Maria Paula Coutinho Dias Morais

10/05/2021



UNIVERSIDADE PORTUCALENSE

## 1 Introdução

No âmbito da unidade curricular de Engenharia de Software, foi-nos proposto construir especificações de desenho usando ferramentas e técnicas de UML, conhecer e saber aplicar padrões de desenho no desenvolvimento de um sistema e por fim elaborar e executar planos de teste.

Inicialmente, elaboramos um diagrama de pacotes que suporte o sistema de Gestão da UPT. Este diagrama descreve os pacotes ou pedaços do sistema divididos em agrupamentos lógicos mostrando as dependências entre eles, o que permite ilustrar a arquitetura de um sistema mostrando o agrupamento de suas classes.

De seguida desenvolvemos um diagrama híbrido relativo ao pacote de Gestão de Biblioteca. Este diagrama descreve a arquitetura do hardware e a sua relação com as diferentes componentes de software, definindo também os recursos onde estão os diferentes componentes, e explica como funciona o hardware e software de suporte à BGUPT.

Para o desenvolvimento das técnicas de UML (Diagrama de Pacotes e Diagrama Híbrido) utilizamos o *Lucidchart*. Quanto ao próximo passo, desenhar o diagrama de classes físico, procedemos à instalação de uma ferramenta no *Eclipse*, *Papyrus*, que permite gerar o código Java com base no diagrama. Neste diagrama utilizamos diferentes padrões de desenho de maneira a simplificar a conceptualização do código.

Selecionamos três dos requisitos fornecidos pela professora, e implementámo-los em Java. De seguida definimos e executamos um plano para testar as componentes do sistema desenvolvidas.

Por fim utilizamos um sistema de controlo de versões, *GitHub*, e ferramentas de desenvolvimento em equipa.

## Índice

<b>1</b>	<b><i>Introdução .....</i></b>	<b><i>2</i></b>
<b>2</b>	<b><i>Modelação em UML .....</i></b>	<b><i>5</i></b>
2.1	Diagrama de Pacotes.....	5
2.2	Diagrama Híbrido .....	6
2.3	Diagrama de Classes.....	7
<b>3</b>	<b><i>Diagrama Classes Final.....</i></b>	<b><i>11</i></b>
<b>4</b>	<b><i>Implementação em Java .....</i></b>	<b><i>12</i></b>
4.1	Requisitos implementados.....	12
4.2	Fase de Testes.....	12
4.3	Ferramentas Utilizadas.....	13
<b>5</b>	<b><i>Conclusão .....</i></b>	<b><i>14</i></b>



## Índice de Figuras

Figura 1 - Diagrama de Pacotes .....	5
Figura 2 - Diagrama Híbrido .....	6
Figura 3 - Factory Utilizadores .....	7
Figura 4 - Factory Publicação .....	8
Figura 5 - Strategy Multas .....	9
Figura 6 - Observer Newsletter .....	10
Figura 7 - GerirBiblioteca e ligações .....	10
Figura 8 - Diagrama Classes Final By Papyrus .....	11
Figura 9 - Classe TesteBiblioteca .....	12
Figura 10 - Projeto GitHub .....	13
Figura 11 - Colaboradores GitHub .....	13

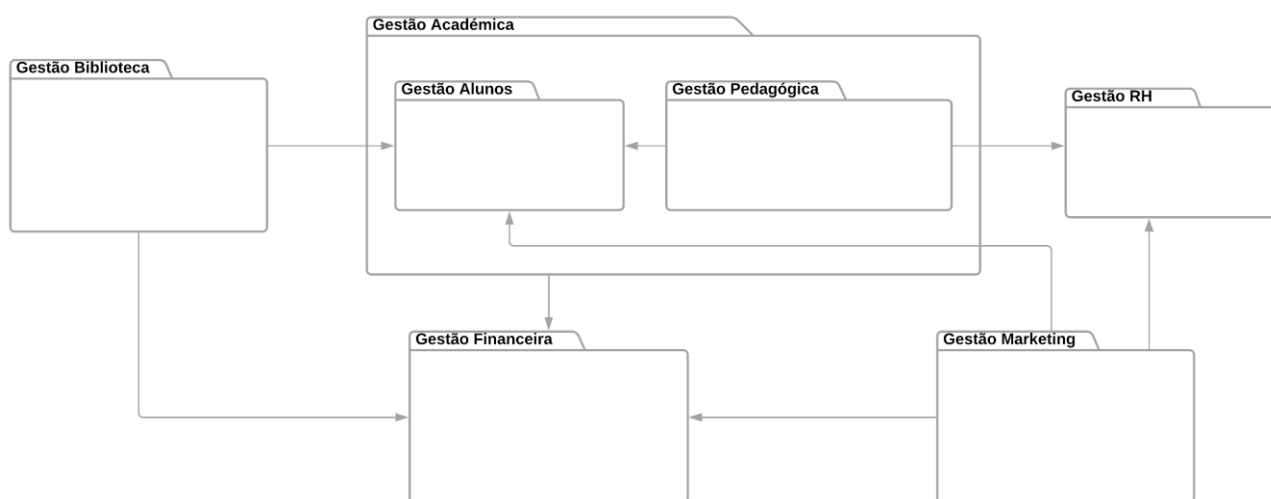
## 2 Modelação em UML

## 2.1 Diagrama de Pacotes

O diagrama de pacotes do sistema de Gestão da UPT mostra como se podem agrupar e quais as relações entre as diferentes componentes do sistema. Simultaneamente permite definir a arquitetura do sistema e dividir a complexidade do mesmo em partes menores para uma melhor gestão - meramente organizacional.

Considerando que, segundo os objetivos específicos, este sistema inclui seis pacotes (Gestão de alunos, Gestão acadêmica, Gestão de RH, Gestão de Biblioteca, Gestão Financeira e Marketing), desenvolvemos o diagrama de pacotes abaixo representado.

O pacote de Gestão Acadêmica depende do Gestão Financeira e ainda engloba a Gestão Pedagógica e Gestão de Alunos, sendo que o primeiro depende do segundo e do Gestão de RH. No que diz respeito ao pacote Gestão de Biblioteca este depende do Gestão de Alunos bem como do Gestão Financeira. Relativamente ao Gestão de Marketing, o mesmo depende da Gestão Financeira, Gestão de Alunos e Gestão de RH.



*Figura 1 - Diagrama de Pacotes*

## 2.2 Diagrama Híbrido

Um diagrama híbrido descreve a arquitetura do hardware e a sua relação com as diferentes componentes de software, definindo também os recursos onde estão os diferentes componentes.

O diagrama é composto por apenas um servidor de nome CatálogoBIB, sendo que este contém quatro componentes e uma base de dados. Todas as operações suportadas pelo Koha e DSpace são executadas usando o portal web da BGUPT logo efetuamos a ligação entre o portal web e as outras duas componentes. Ligamos também a Gestão Portal Web ao TomCat, que fornece um serviço para otimizar as pesquisas. De seguida efetuamos uma ligação entre o Koha e o DSpace à base de dados.

Existem três computadores ligados ao servidor através de uma ligação HTTP/HTTPS - TCP/IP - IPV4. Criámos também um pc aluno/staff que está ligado ao servidor por meio de uma rede Wifi. Existe também uma impressora da biblioteca que está ligada ao PC do atendimento geral por WIFI.

Por fim existem dois pacotes, Gestão RH e Gestão Alunos, que estão ligados ao componente Gestão Portal Web.

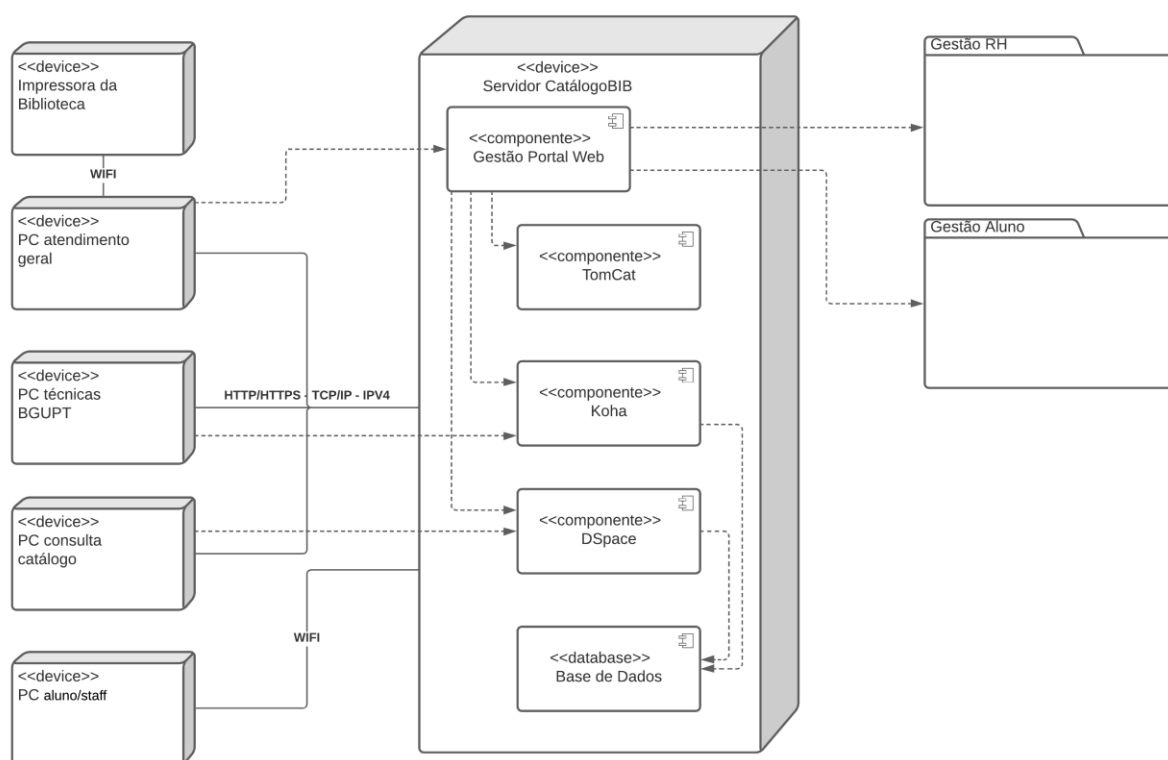


Figura 2 - Diagrama Híbrido

## 2.3 Diagrama de Classes

O diagrama de classes representa a estrutura e relação das classes que servem de modelo para o desenvolvimento do código. É uma modelagem muito útil para o desenvolvimento de sistemas, pois define todas as classes que o sistema necessita para o correto funcionamento.

Começamos por utilizar o **Padrão de Desenho Factory**, pois este permite-nos criar vários objetos do mesmo tipo com atributos diferentes. No nosso caso utilizamos por duas vezes este padrão já que, tanto os utilizadores como as publicações, têm diferentes objetos associados. A classe Utilizadores é uma interface e possui os métodos que estão em todos os objetos filhos (getNome(), getNumeroCartao() e getNewsletter()). Esta classe tem como subclasses Estudante, Staff (Docente e não Docente) e Externo. A interface foi ligada à classe UtilizadorFactory que possui os métodos de registo dos vários utilizadores e um getProxUtil(), método este que vai comparar o token recebido com os tipos de utilizador. Esta Classe está ligada à interface UtilizadoresFactoryInterface que apenas possui o método getProxUtil().

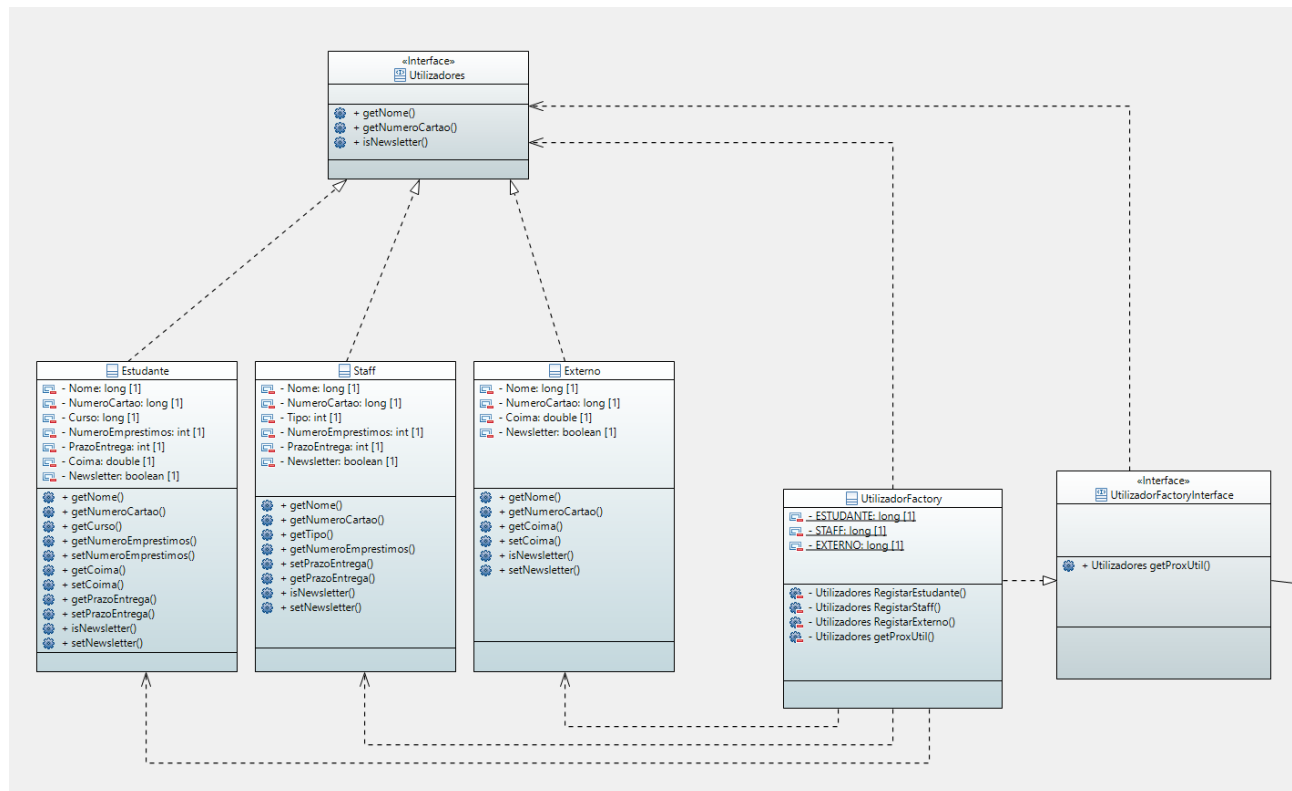


Figura 3 - Factory Utilizadores

No que diz respeito às publicações, a utilização do padrão Factory segue o mesmo modelo que os utilizadores, mas as subclasses da interface Publicacao são o Livro, PublicacaoPeriodica e ObraReferencia. Como em cima descrito, esta interface apenas possui os métodos que pertencem às três subclasses (getTitulo(), getAutor(), getEditor() e getTipo()). De seguida procedemos com a criação da classe PublicacaoFactory e da interface PublicacaoFactoryInterface. Estas foram criadas seguindo o mesmo modelo dos Utilizadores.

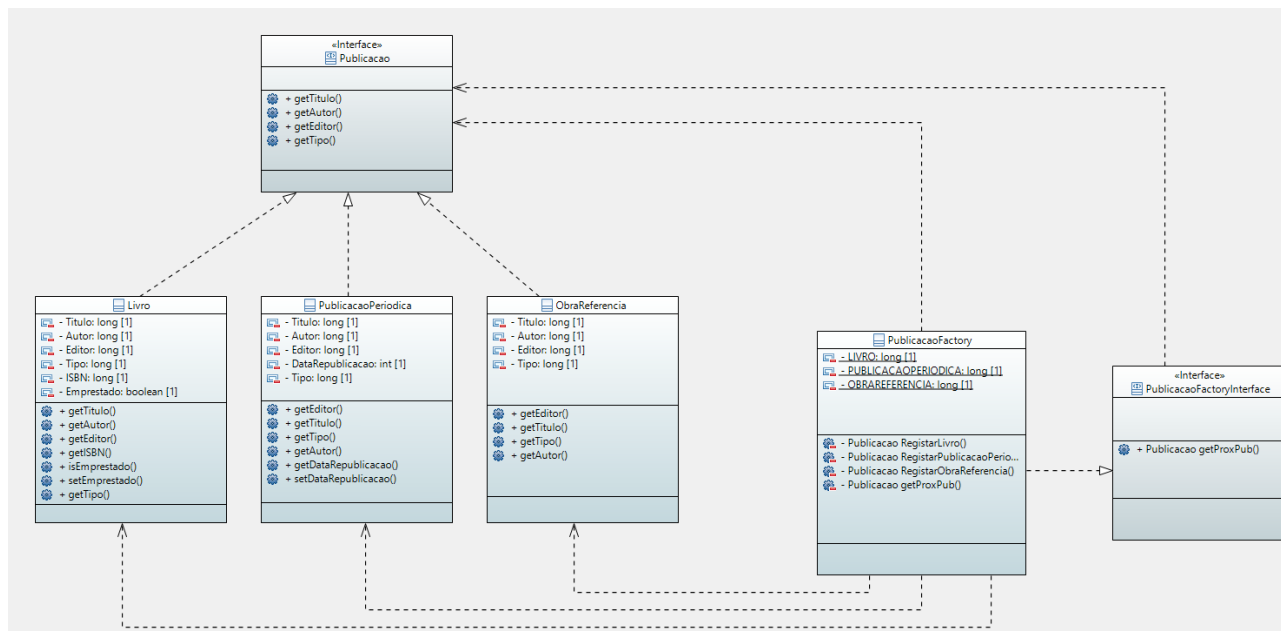


Figura 4 - Factory Publicação



O segundo padrão que utilizámos foi o **Padrão de Desenho Strategy** para calcular a multa que um determinado utilizador tem de dívida. Cada utilizador, dependendo do seu tipo, tem uma determinada taxa, se for um Estudante tem uma coima de 50 cêntimos por cada dia de atraso, se for um utilizador externo tem uma coima de 1 euro e por fim se for membro do Staff não paga multa. Como tal, decidimos utilizar o padrão de desenho Strategy para atender a estas especificações.

Criámos então quatro classes, MultaExterna, MultaEstudante, CalcularMulta e OrdemAtraso. As duas primeiras têm apenas um método taxa que recebe a coima do utilizador. Quanto ao CalcularMulta, este vai receber o número de dias em atraso e vai chamar as classes (MultaEstudante e MultaExterna) dependendo do tipo de utilizador. Por fim a classe OrdemAtraso vai servir como classe intermédia entre o CalcularMulta e o GerirBiblioteca.

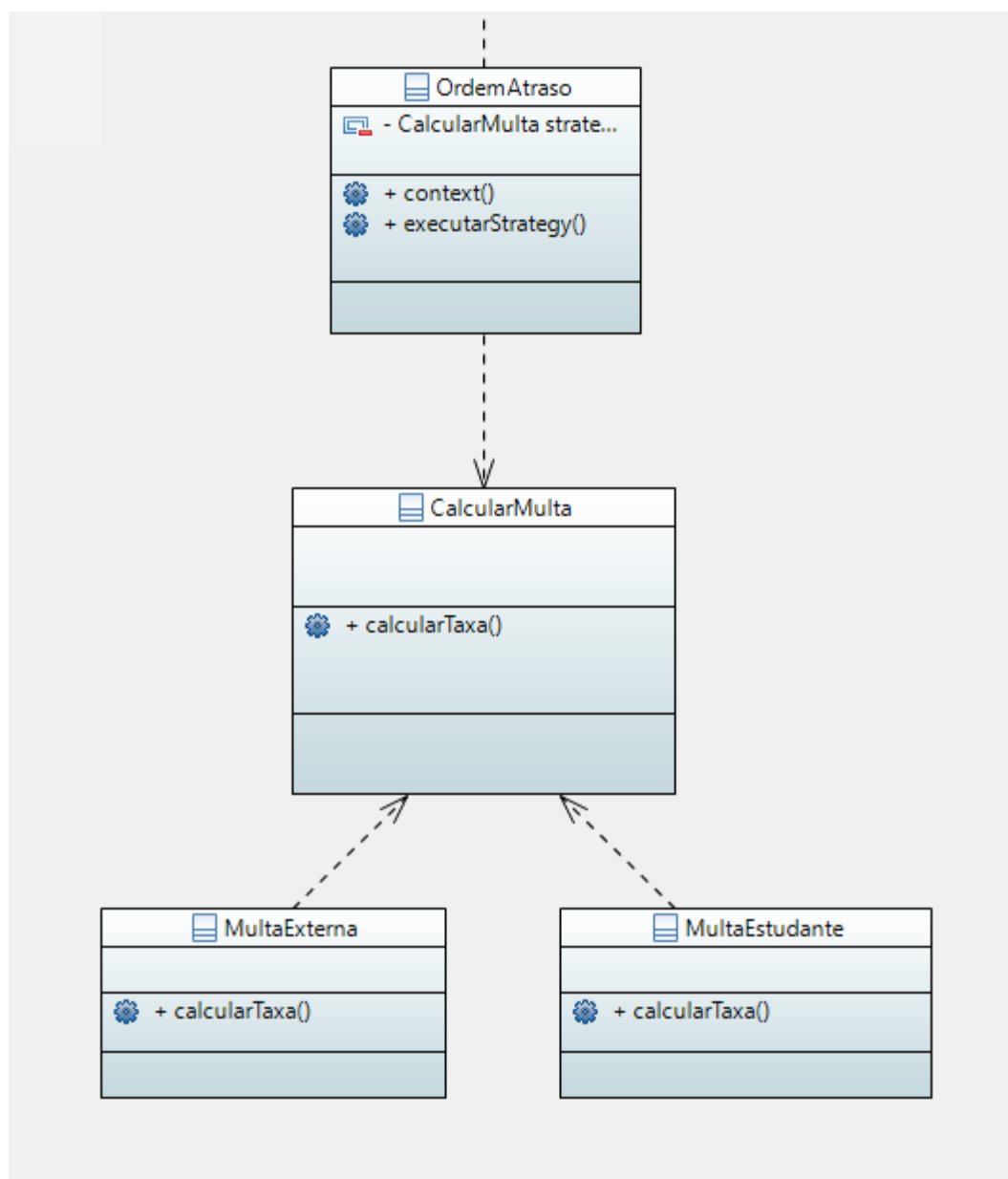


Figura 5 - Strategy Multas

O último padrão de desenho que utilizamos foi o **Padrão de Desenho Observer** para gerir a Newsletter. Na classe Subject temos um array de Utilizadores que vai permitir fazer o registo da Newsletter, remover a mesma caso já tenha sido registada, bem como notificar os Utilizadores.

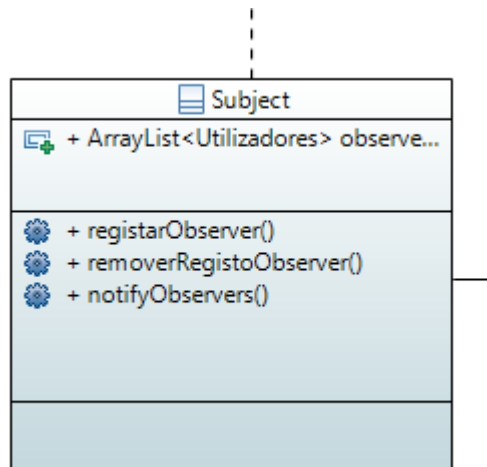


Figura 6 - Observer Newsletter

Por fim temos a classe **GerirBiblioteca** que faz a ligação entre todos os Padrões de Desenho e o Main. A classe **GerirBiblioteca** é responsável por criar os métodos necessários para o bom funcionamento da classe **Main** onde, por sua vez, criamos um menu para guiar o utilizador pelas diferentes opções fornecidas, sendo essas o registo do utilizador, o registo de uma publicação, imprimir tantos os registos dos utilizadores como da publicação e por fim o empréstimo de uma publicação a um determinado utilizador.

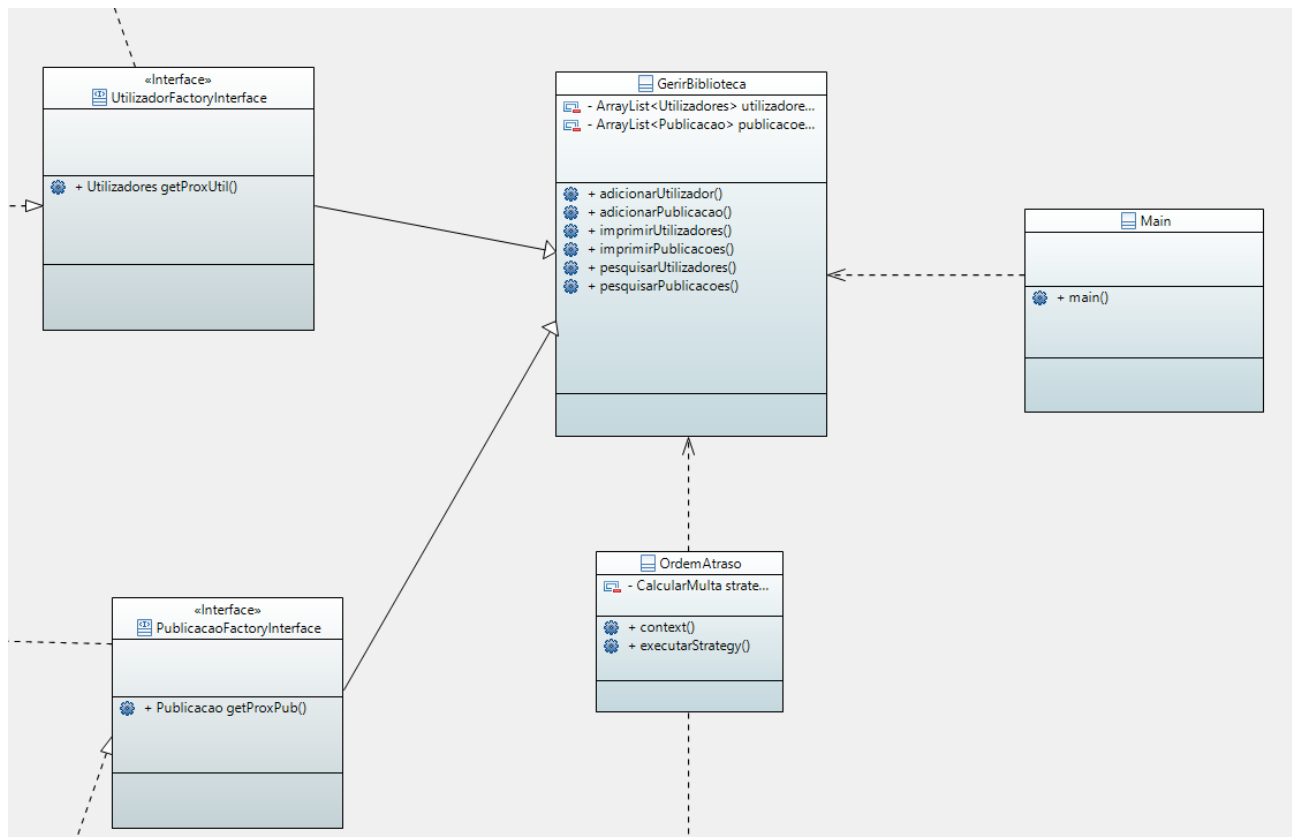


Figura 7 - GerirBiblioteca e ligações



## 4 Implementação em Java

### 4.1 Requisitos implementados

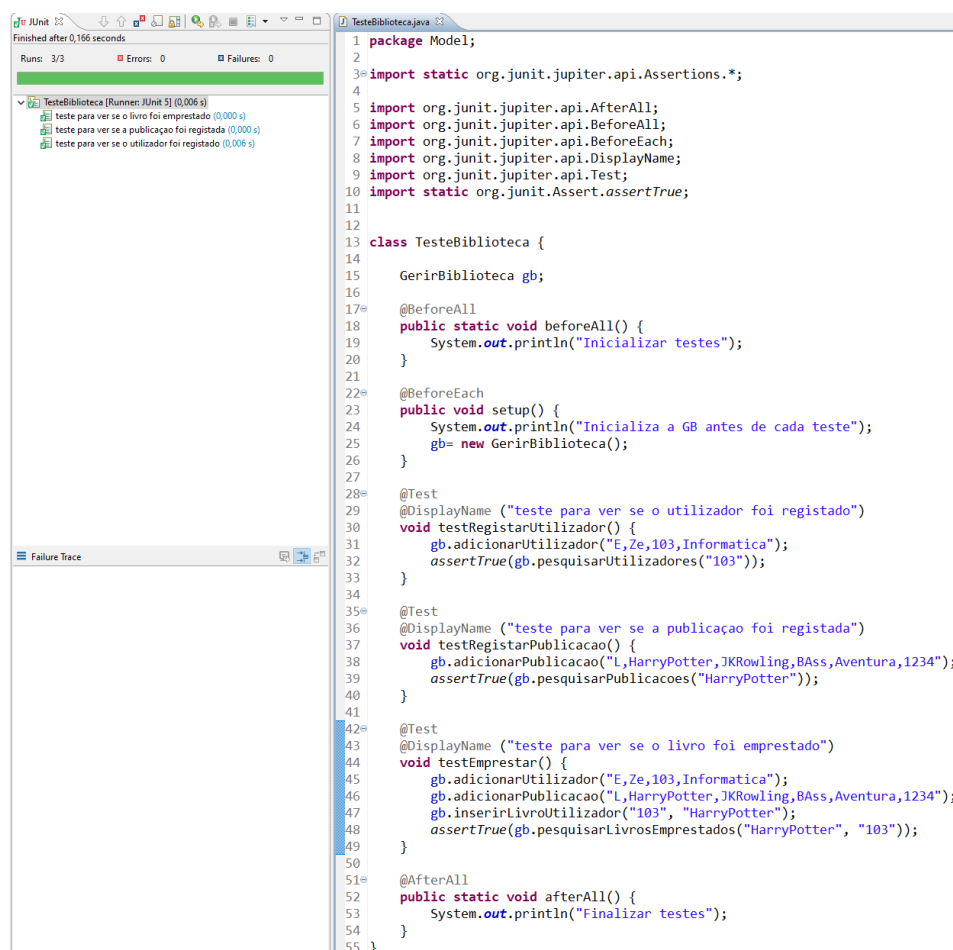
Para a criação do código em java tivemos que escolher três requisitos do catálogo fornecido pela professora. Optámos então pelos seguintes requisitos:

- O sistema deve permitir o registo de Utilizadores/Leitores;
- O sistema deve permitir o registo de publicações;
- O sistema deve permitir ao utilizador requisitar publicações;

Os dois primeiros requisitos foram escolhidos porque sem eles não seria possível implementar os restantes requisitos do catálogo. Quanto ao terceiro requisito foi escolhido porque pensamos que é aquele que mais lógica fazia implementar e também porque vai depender dos dois primeiros.

### 4.2 Fase de Testes

Na fase de testes procedemos à escrita do código que vai testar a implementação dos métodos que criámos para cada um dos requisitos escolhidos. Para tal criámos a classe a teste com a extensão do eclipse JUnit5. Desenvolve-mos três testes, `TesteRegistrarPublicacao`, `TesteRegistrarUtilizador` e `TesteEmprestarPublicacao`.



```

1 package Model;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 import org.junit.jupiter.api.AfterAll;
6 import org.junit.jupiter.api.BeforeAll;
7 import org.junit.jupiter.api.BeforeEach;
8 import org.junit.jupiter.api.DisplayName;
9 import org.junit.jupiter.api.Test;
10 import static org.junit.Assert.assertTrue;
11
12
13 class TesteBiblioteca {
14
15     GerirBiblioteca gb;
16
17     @BeforeAll
18     public static void beforeAll() {
19         System.out.println("Inicializar testes");
20     }
21
22     @BeforeEach
23     public void setup() {
24         System.out.println("Inicializa a GB antes de cada teste");
25         gb = new GerirBiblioteca();
26     }
27
28     @Test
29     @DisplayName("teste para ver se o utilizador foi registado")
30     void testRegistrarUtilizador() {
31         gb.adicionarUtilizador("E,Ze,103,Informatica");
32         assertTrue(gb.pesquisarUtilizadores("103"));
33     }
34
35     @Test
36     @DisplayName("teste para ver se a publicação foi registada")
37     void testRegistrarPublicacao() {
38         gb.adicionarPublicacao("L,HarryPotter,JKRowling,BAss,Aventura,1234");
39         assertTrue(gb.pesquisarPublicacoes("HarryPotter"));
40     }
41
42     @Test
43     @DisplayName("teste para ver se o livro foi emprestado")
44     void testEmprestar() {
45         gb.adicionarUtilizador("E,Ze,103,Informatica");
46         gb.adicionarPublicacao("L,HarryPotter,JKRowling,BAss,Aventura,1234");
47         gb.inserirLivroUtilizador("103", "HarryPotter");
48         assertTrue(gb.pesquisarLivrosEmprestados("HarryPotter", "103"));
49     }
50
51     @AfterAll
52     public static void afterAll() {
53         System.out.println("Finalizar testes");
54     }
55 }

```

Figura 9 - Classe TesteBiblioteca

## 4.3 Ferramentas Utilizadas

Para a realização deste trabalho utilizámos a ferramenta Papyrus bem como o GitHub.

De salientar que na utilização do Papyrus, apesar das diversas vantagens, sendo uma delas, gerar o código através da criação do diagrama de Classes Fisico na ferramenta, o Papyrus causou algumas dificuldades por exemplo, as classes criadas vinham com os métodos vazios visto que a ferramenta não deixava colocar os atributos dentro dos métodos no diagrama de classes.

No caso do GitHub, apesar de não termos dado tanto uso como esperado deu para perceber que é uma ferramenta bastante útil e uma mais valia na realização de trabalhos em grupo e para controlar versões.

Link: <https://github.com/ZAFSantos/Engenharia-de-Software.git>

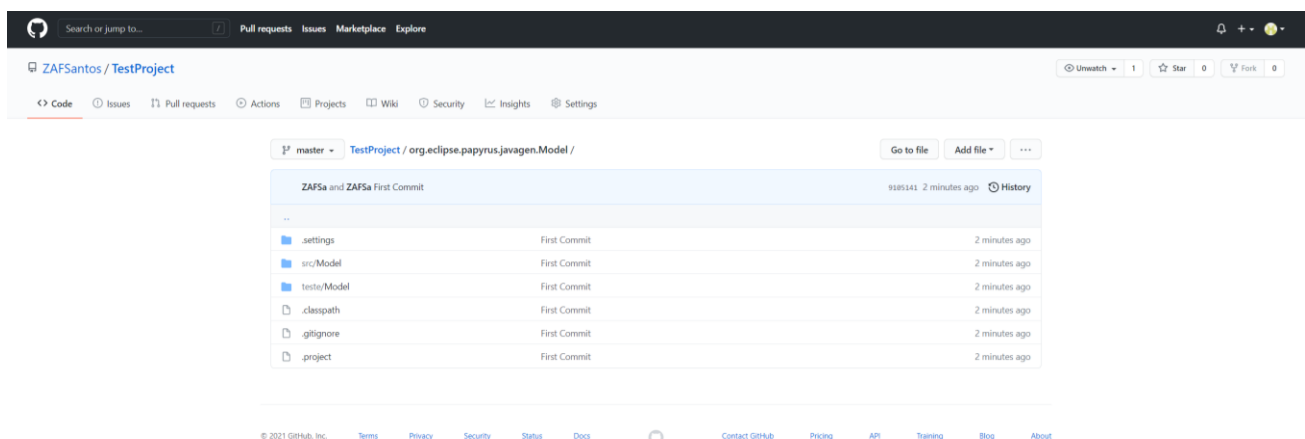


Figura 10 - Projeto GitHub

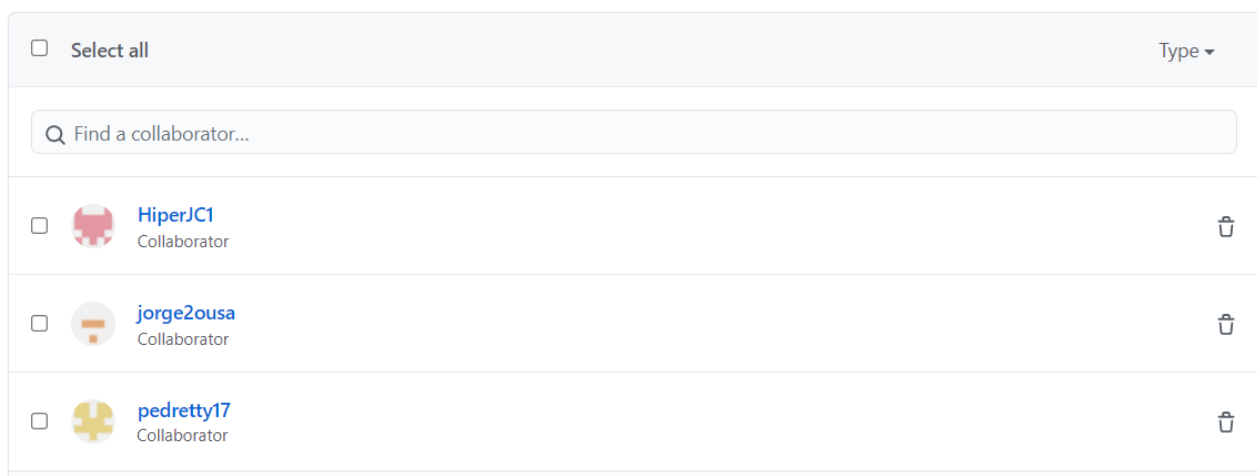


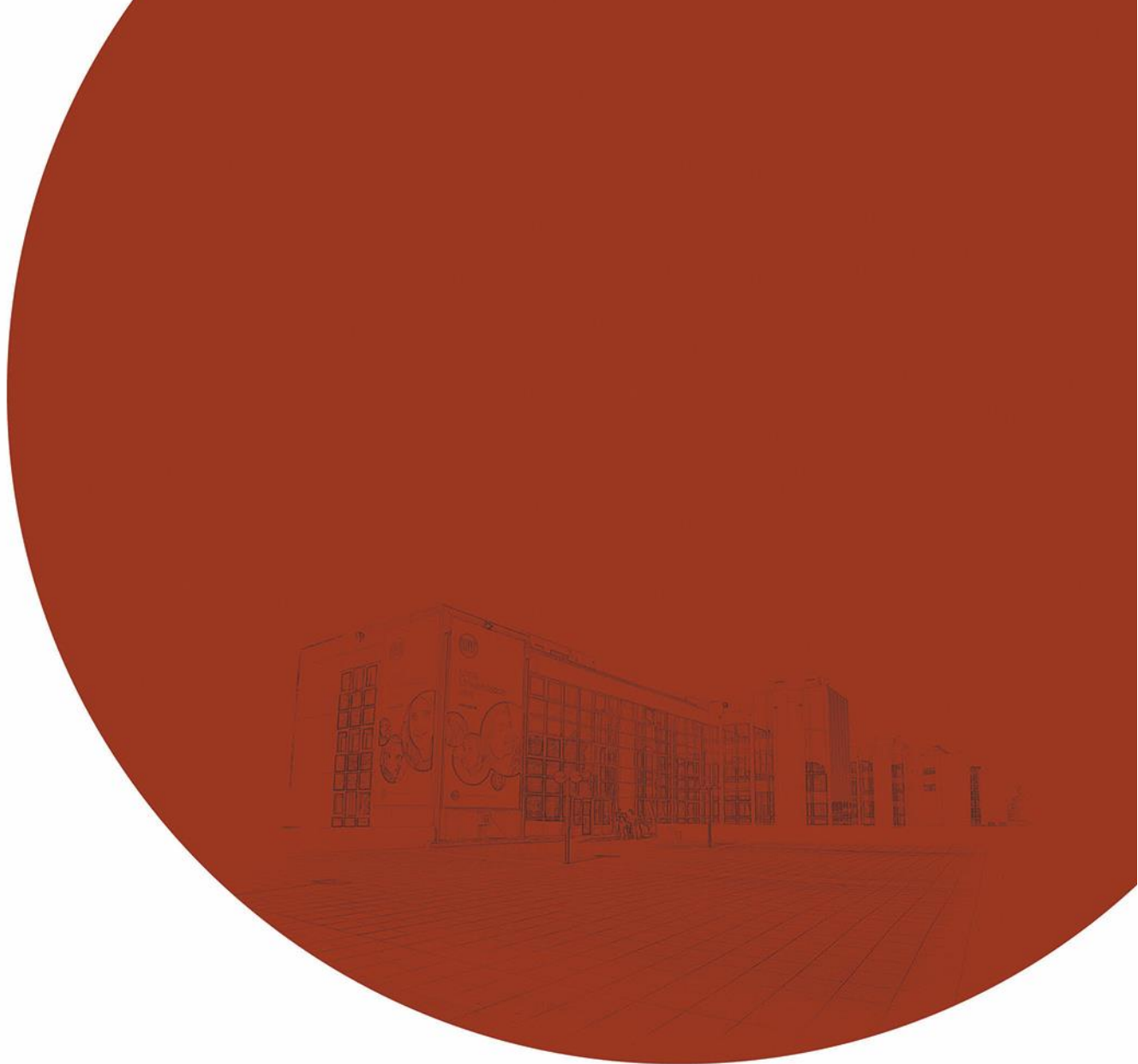
Figura 11 - Colaboradores GitHub

## 5 Conclusão

Com o cumprimento deste trabalho, ficámos a entender melhor como o sistema de informação da biblioteca da Universidade Portucalense funciona, bem como é suportado a nível de hardware. Para esse fim aplicámos os nossos conhecimentos adquiridos ao longo do semestre nas aulas de Engenharia de Software, desde a conceptualização do sistema, à elaboração dos diagramas (Pacotes, Híbridos e Classes), à implementação do código no eclipse com o auxílio da ferramenta Papyrus e por fim a realização de testes utilizando o JUnit5.

Consideramos que este trabalho foi bastante interessante bem como importante porque nos ajudou a compreender como funcionam os Padrões de Desenho, perceber os seus pontos fortes e fracos.

Com a finalização deste trabalho deparamo-nos com algumas dificuldades principalmente com a utilização da ferramenta Papyrus mas acabamos por cumprir todos os objetivos a que nos propusemos.



UNIVERSIDADE PORTUCALENSE