Padrões e Desenho de Software - Trabalho teórico prático

Autor: Vinícius Benite Ribeiro [82773], 2020-05-11

Professor: José Luís Oliveira

DETI - Universidade de Aveiro

1 Introdução

1.1 Objetivo

O objetivo deste trabalho Teórico-Prático é consolidar e aplicar os conceitos programáticos de PDS, nomeadamente princípios, boas práticas e padrões de software. Os dois padrões apresentados neste trabalho são o Adapter e o Strategy.

2 Padrão Adapter

2.1 Descrição

Em engenharia de software, o adapter é um padrão estrutural que, basicamente, possibilita o uso de uma interface de uma classe existente, com uma outra nova interface de outra classe, sem termos que modificar o source code da interface antiga [1]. É um padrão relativamente fácil de compreender, pois o mundo real está cheio de adaptadores, por exemplo um adaptador VGA:HDMI, para ligar cabos VGA a novos computadores que só possuem entradas HDMI.

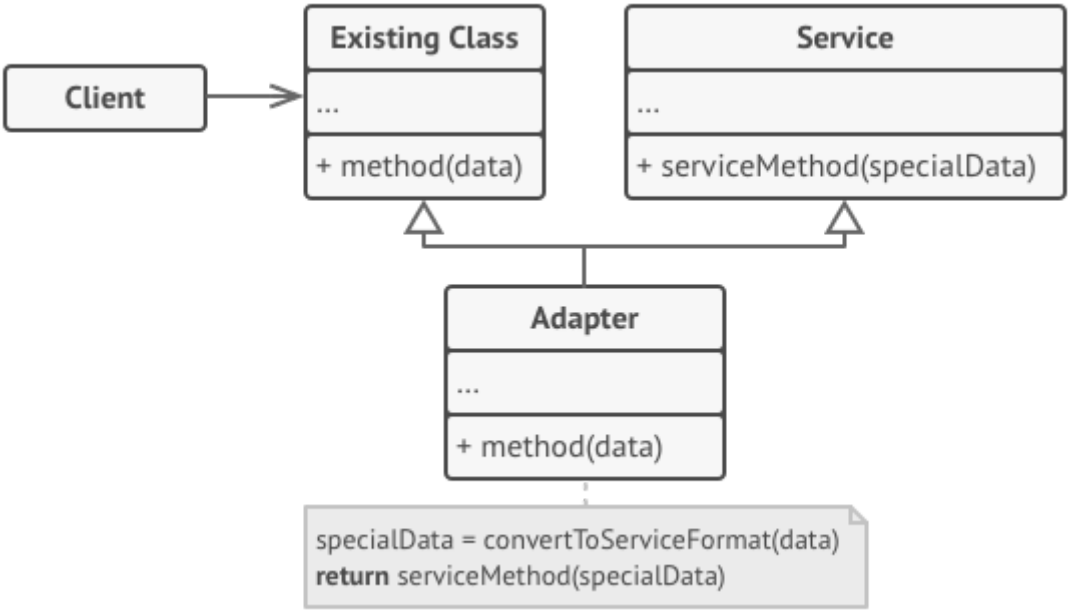
O cliente segue os seguintes passos ao usar o padrão:

1. O cliente faz um pedido ao adapter ao chamar um método de uma interface desejada;
2. O adapter traduz o pedido baseado na interface desejada;
3. É retornado para o cliente os resultados desta chamada.

Vale notar que, o cliente não sabe que o adapter existe. O cliente faz um pedido a uma interface qualquer, que, por sua vez, implementa a interface dessa interface e delega o pedido a interface da classe antiga [2].

Há duas maneiras de se implementar este padrão:

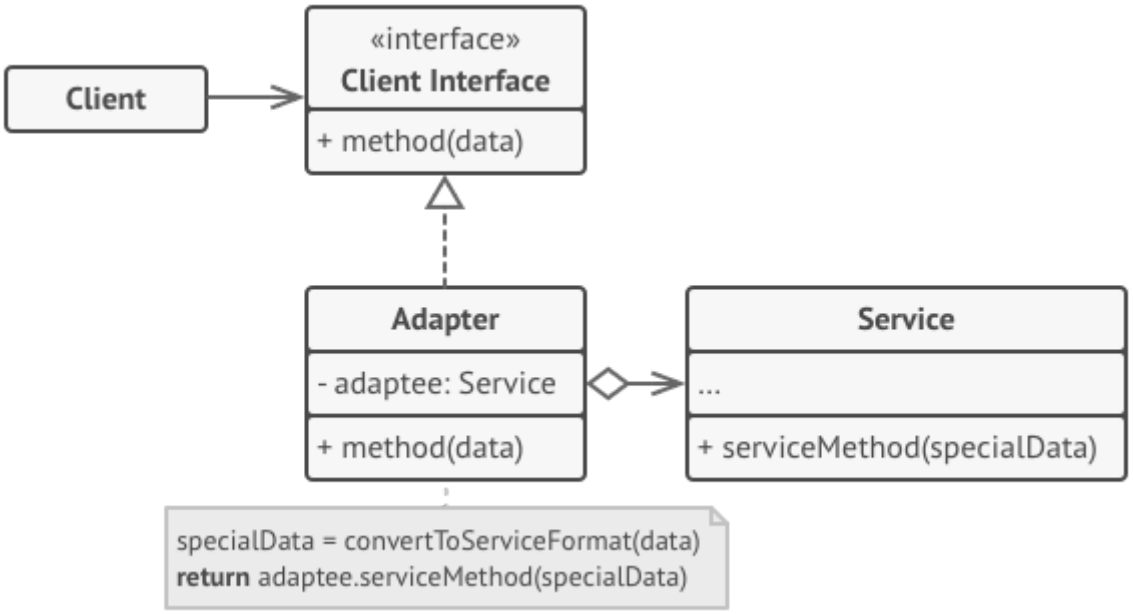
1. Um adapter para classes, que é implementado pelo uso de herança;



Fonte: https://refactoring.guru/design-patterns/adapter

A classe adapter não precisa de nenhum wraper para os objetos, pois herda o comportamento tanto do Client, como do Service. A “adaptação” ocorre quando fazemos override dos métodos.

2. Um adapter para objetos, que faz o uso de composição.



Fonte: https://refactoring.guru/design-patterns/adapter

O Adapter é uma classe capaz de trabalhar com o Client e o Service: implementa a interface do cliente enquanto quebra o objeto de serviço. O adapter recebe chamadas do Client através da interface do adapter e as converte em chamadas para o objeto de serviço agrupado em um formato que ele possa entender.

Aqui temos uma analogia ao mundo real para ajudar a compreender o padrão:

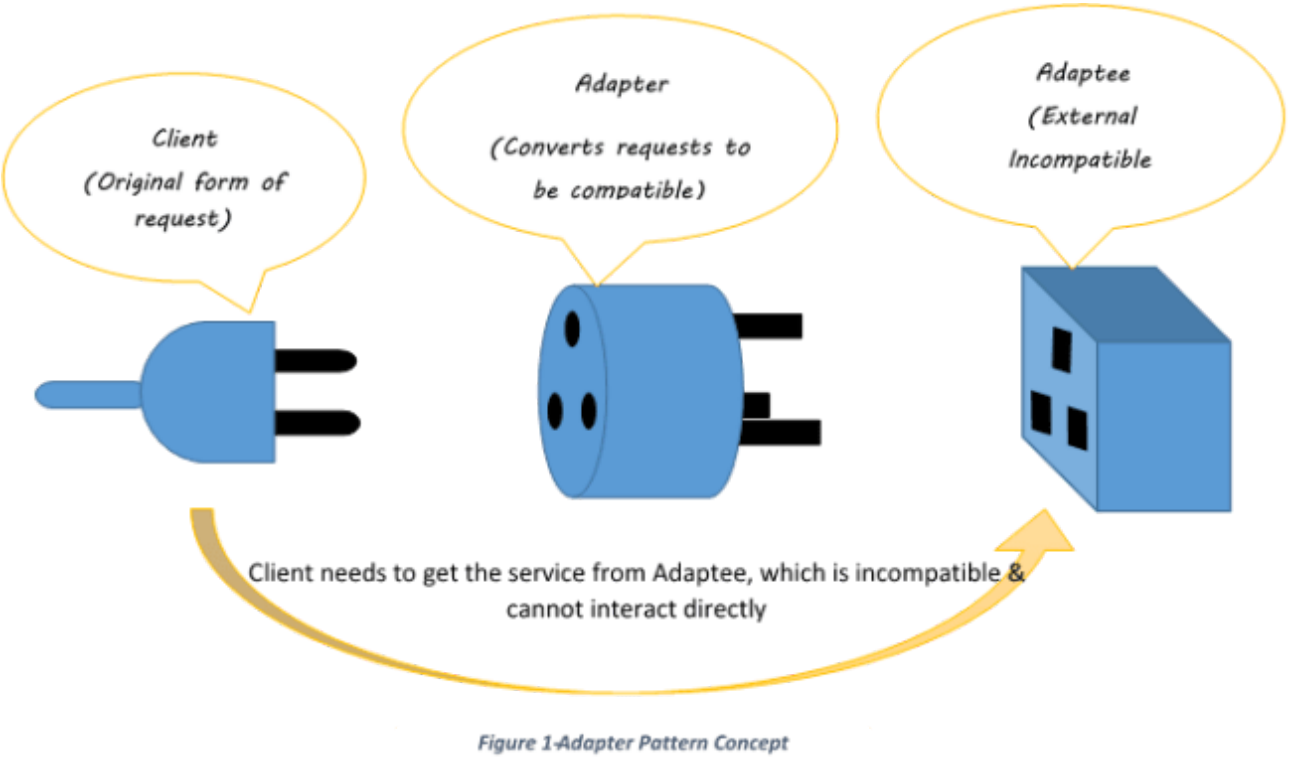


Figure 1-Adapter Pattern Concept

Fonte: https://medium.com/@pramodayajayalath/adapter-design-pattern-3307ada690db/

O adapter permite o reuso de código antigo sem a necessidade de grandes modificações no código. Em uma comparação rápida com o padrão decorator, o adaptor somente converte objetos para torná-los compatíveis, o decorator, por sua vez, adiciona novas funcionalidades a esse objeto.

É recomendável o uso deste padrão quando temos diferentes interfaces com comportamento semelhante e métodos diferentes.

Java.util.Arrays#asList(), java.io.InputStreamReader(InputStream) (retorna um Reader) e java.io.OutputStreamWriter(OutputStream) (retorna um Writer) são exemplos de uso do adapter no JDK.

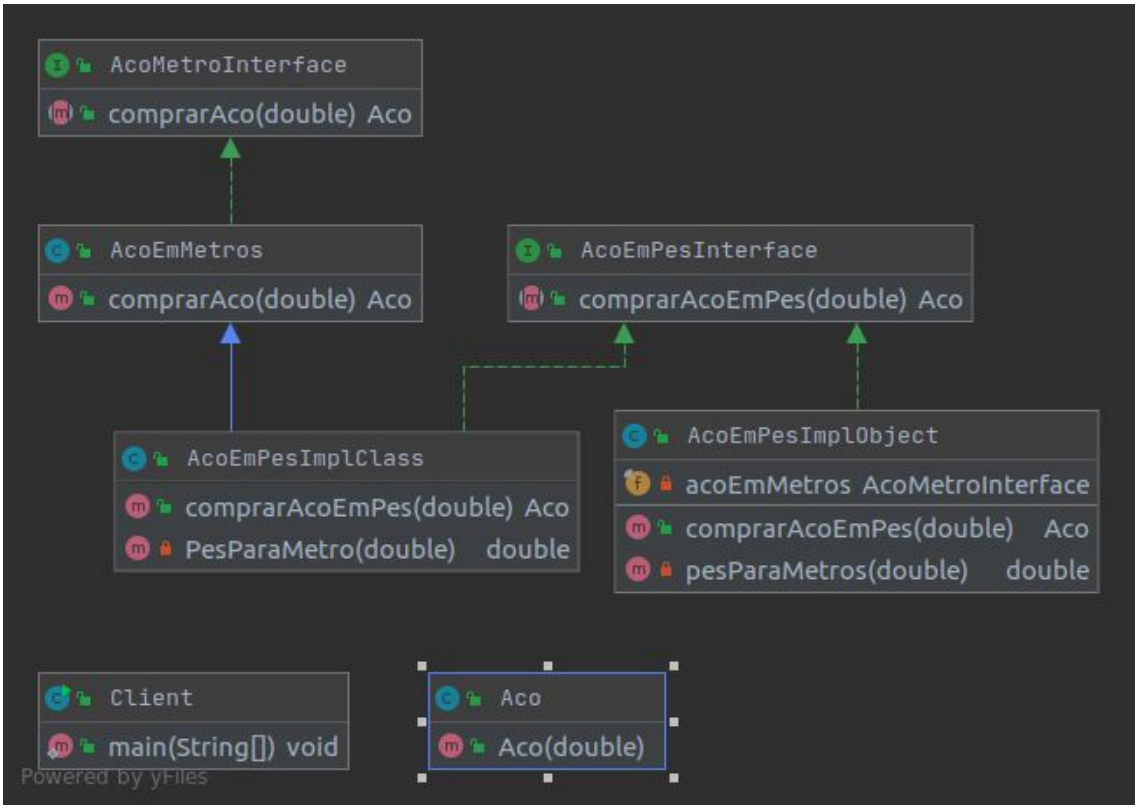
2.2 Problema

A) O nosso cliente tem uma empresa que trabalha com a manufatura do aço. A empresa é de Portugal, portanto, trabalha com o sistema internacional de unidades, já tendo um sistema informático implementado. Porém, o cliente começará a fazer negócios com empresas do Estados Unidos, que trabalham com o sistema imperial. O cliente quer que o sistema informático trabalhe com ambas as unidades.

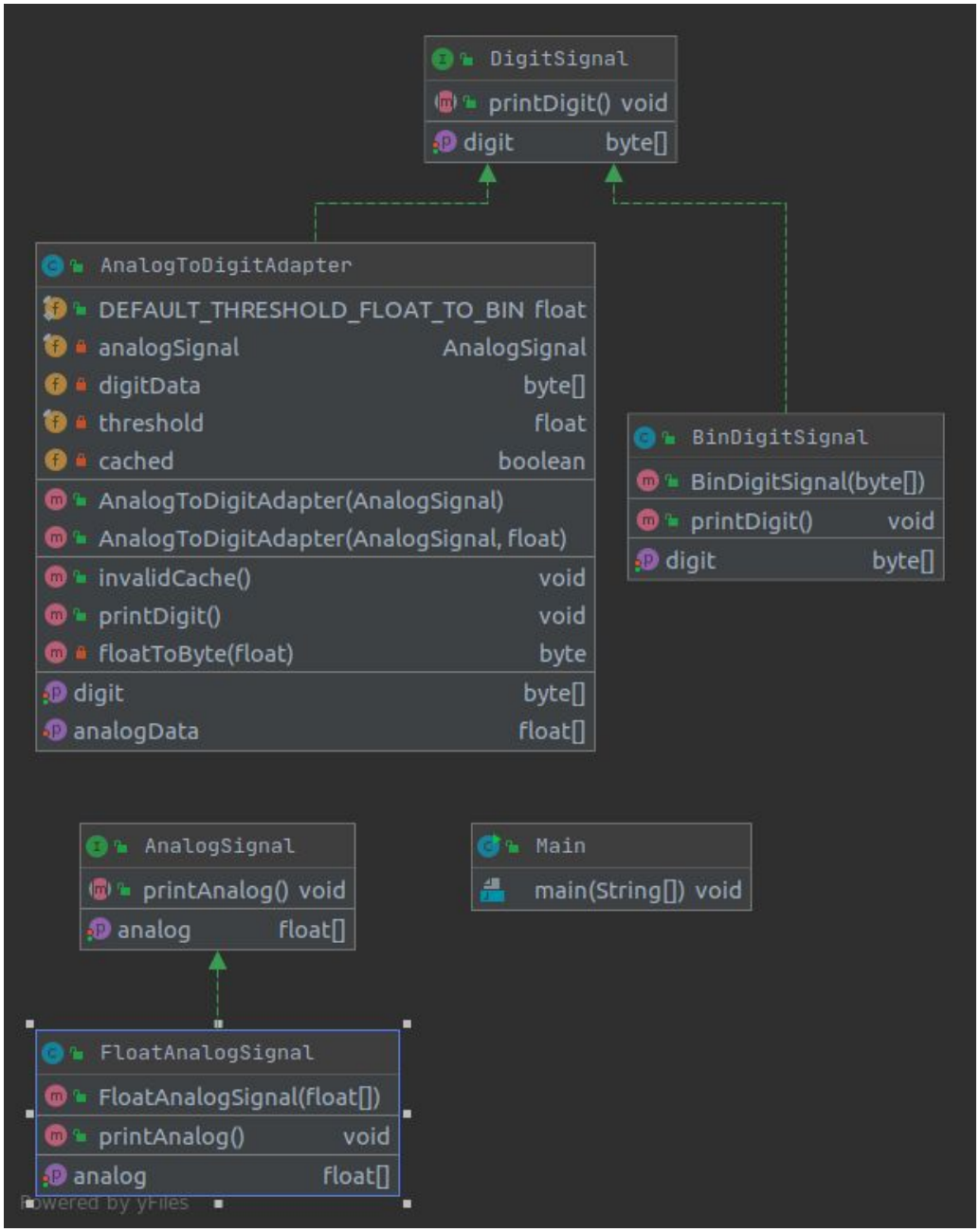
B) Simular um exemplo para converter dados analógicos para o formato binário.

2.3 Solução

A)



Código disponível no Github.

B) 

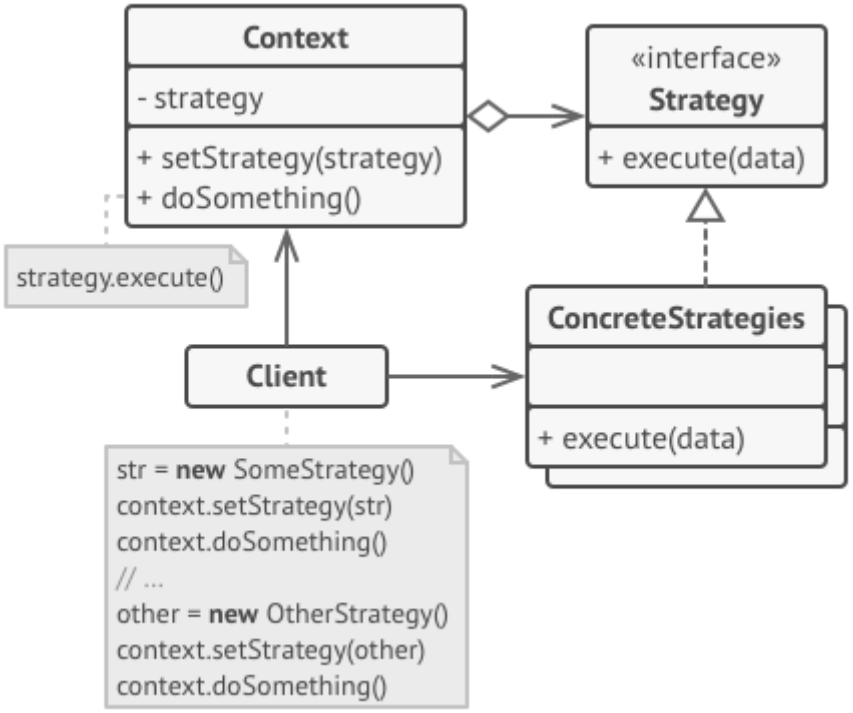
3 Padrão Strategy

3.1 Descrição

Strategy é um padrão comportamental em design de software que, permite definir vários algoritmos diferentes, em classes diferentes e por fim, fazer os seus respectivos objetos permutáveis [1].

Este padrão permite ao programador pegar em um certo algoritmo, cujo o objectivo final pode ser atingido por diferentes maneiras e separar esses algoritmos em classes separadas, chamadas de

strategies. Temos nesse padrão, uma classe chamada Context, que armazena um apontador para as estratégias disponíveis. O Context delega o trabalho a ser realizado. Importante notar que, o Context não seleciona a melhor estratégia e nem tem conhecimento sobre as mesmas. A única função do Context é apontar para a estratégia selecionada. Assim, o Context se torna independente do código das estratégias.



Fonte: https://refactoring.guru/design-patterns/strategy/

A interface Strategy é a mesma para todos os algoritmos. Ela declara um método para correr uma determinada estratégia. ConcreteStrategies são os algoritmos propriamente ditos. A classe Context chama o método para executar a estratégia desejada. O Client seleciona uma estratégia e passa para o Context. Através de setters do Context, é possível mudar a estratégia associada em runtime.

Uma possível analogia ao mundo real seria: qual a melhor maneira de ir a estação de comboios? A pé, bicicleta ou autocarro?

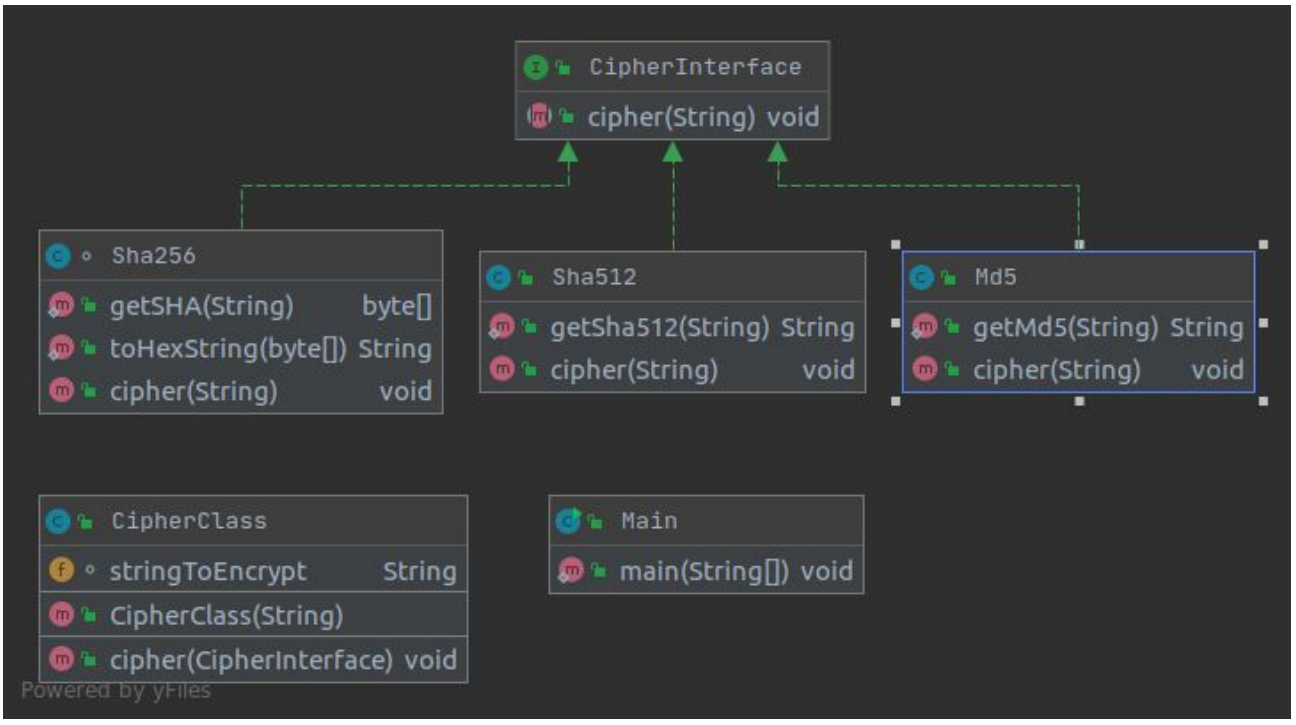
Recomenda-se o uso do padrão strategy quando pretendemos usar diferentes algoritmos/classes com a mesma finalidade e poder alterar a estratégia em runtime. Também é recomendável usar este padrão para isolar a camada business de uma classe de algoritmos que não são importantes para o contexto da classe. Se você tiver switchs com muitas variáveis, talvez é melhor usa o padrão Strategy.

Javax.servlet.http.HttpServlet, HttpServletRequest(), HttpServletResponse(), java.util.Comparator#compare() e javax.servlet.Filter#doFilter() são exemplos do uso do padrão Strategy no JDK.

3.2 Problema

Precisamos encriptar certas palavras definidas pelo usuário. Temos 3 diferentes algoritmos de encriptação disponíveis: MD5, SHA-512 e SHA-256 [3]. O cliente pode escolher qual deseja usar em runtime.

3.3 Solução



Código disponível no Github.

4 Referências e recursos

Project resources

- Git: https://github.com/viniciusbenite/pds-tp

Reference materials

[1] Freeman, Eric; Freeman, Elisabeth; Sierra, Kathy; Bates, Bert (2004). *Head First Design Patterns* (paperback). O'Reilly Media. p. 244. ISBN 978-0-596-00712-6. OCLC 809772256. Retrieved 2013-04-30.

[2] https://www.geeksforgeeks.org/adapter-pattern/

[3] https://www.geeksforgeeks.org/md5-hash-in-java/?ref=rp