Instituto Superior de Engenharia
Politécnico de Coimbra

Instituto Politécnico de Coimbra

Instituto Superior de Engenharia de Coimbra

Disciplina Programação Avançada

Trabalho Prático – Meta 1

Docente: Álvaro Santos

Pedro Jorge Fernandes Morais – 2018020733 – LEI

Maria Beatriz Girão Abreu – 2020130937 – LEI-CE

domingo, 1 de maio de 2022

Índice

Índice de Figuras

Índice de Tabelas

1 Opções tomadas na implementação

1.1 Utilização de *HashMap*

De forma a guardar todas os dados referentes aos alunos, docentes, propostas, candidaturas e propostas atribuídas decidimos optar por guardar utilizando *HashMap*. Os *HashMap* permitem que para cada valor exista uma chave associada, facilitando quando se pretende obter os dados de um aluno, docente, proposta, candidatura ou proposta atribuída especifica. Tem ainda a particularidade de não permitir a duplicação de chaves, facilitando assim a verificação de entradas duplicadas. No caso dos alunos foi escolhido como chave o seu número de aluno, devido a este ser único a cada aluno, para os docentes o seu email, para as propostas o identificador da proposta, para as candidaturas o número do aluno candidato, pois este apenas pode apresentar uma candidatura, e nas propostas atribuídas o identificador da proposta que está a ser atribuída a um aluno.

1.2 Tratamento de Exceções

Para o tratamento de possíveis exceções que possam ocorrer ao longo da execução do programa foram desenvolvidas duas classes, *ExceptionsTypes* e *ExceptionOccurred*.

A classe *ExceptionTypes* é do tipo enumerável, tendo como objetivo representar os vários tipos de exceções que podem ocorrer.

Já a classe *ExceptionOccurred* segue o padrão de programação *Singleton*, e tem como objetivo guardar o tipo de exceção que ocorreu para depois ser mostrada uma mensagem personalizada ao utilizador na interface.

1.3 Padrões *Command* e *Facade*

Para que o utilizador tenha a possibilidade de realizar operações de *undo* e *redo* decidimos utilizar os padrões de programação *Command* e *Facade*. O padrão *Command* permite especificar o comportamento de um comando quando este é executado e revertido. O padrão *Facade* permite criar uma interface mais simples para que outras classes possam utilizar os diversos comandos criados. Estes padrões são utilizados na gestão de atribuição manual de propostas a alunos e na gestão atribuição manual de docentes orientadores a alunos com proposta atribuída.

2 Diagrama Máquina de Estados

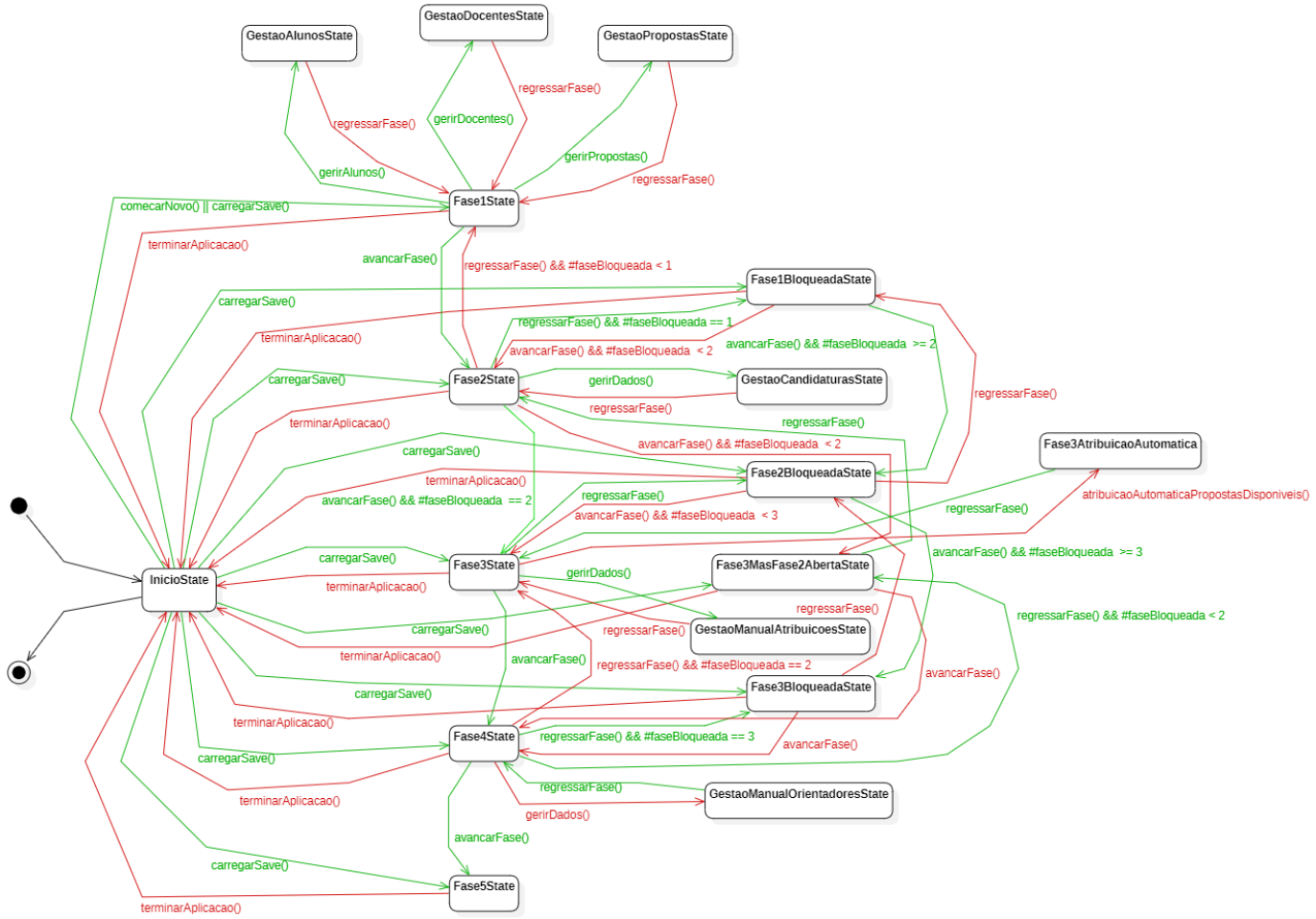

Figura 1 - Diagrama da máquina de estados

2.1 *InicioState*

O estado *InicioState* é o primeiro estado da aplicação. Este permite a criação de uma nova execução da aplicação, ou então importar os dados de uma execução anterior através de ficheiros binários, restaurando os dados e passando para o estado em que a execução anterior foi terminada.

2.2 *Fase1State*

Este estado diz respeito à primeira fase do programa. Aqui é tratada toda a gestão de alunos, docentes e propostas/autopropostas de estágio e de projeto, aos quais correspondem os estados *GestaoAlunosState*, *GestaoDocentesState* e *GestaoPropostasState*. Os mesmos são acedidos, respetivamente, pelas funções *gerirAlunos()*, *gerirDocentes()* e *gerirPropostas()*.

É possível também passar para a fase 2, escolhendo bloquear ou não a fase 1, e sair do programa, com a opção de guardar o estado atual.

2.3 *Fase1BloqueadaState*

Este estado corresponde ao bloqueio da fase 1, ou seja, apenas é permitido consultar os dados tratados na mesma. Para ser bloqueada, em cada ramo, o número total de propostas tem de ser igual ou superior ao número de alunos.

Através da função *avancarFase()*, é possível seguir para as próximas que são representadas pelos seguintes estados:

- *Fase2State*, se a fase 2 ainda não estiver bloqueada;
- *Fase2BloqueadaState*, se a fase 2 estiver bloqueada.

2.4 *GestaoAlunosState*

Este estado dá ao utilizador a possibilidade de adicionar, editar, remover e consultar alunos, bem como a importação e exportação destes dados para ficheiros *CSV*.

Há, ainda, a opção de regressar ao *Fase1State*.

2.5 *GestaoDocentesState*

Neste estado existem as opções de adicionar, editar, remover e consultar docentes, além de importar e exportar os respetivos dados para ficheiros *CSV*.

É possível, também, regressar ao *Fase1State*.

2.6 *GestaoPropostasState*

Aqui é possível adicionar, editar, remover e consultar propostas/autopropostas, além de importar e exportar os respetivos dados para ficheiros *CSV*.

Além disso, pode-se voltar ao *Fase1State*.

2.7 *Fase2State*

Este estado permite controlar as candidaturas, através do estado *GestaoCandidaturasState*, listar alunos (com e sem candidatura registada) e as respetivas autopropostas, e, listar propostas de projeto/estágio.

Existem as funcionalidades para regressar à fase anterior, avançar para a próxima, bloqueando ou não a atual.

Através da função *regressarFase()*, é possível voltar às anteriores que são representadas pelos seguintes estados:

- *Fase1State*, se a fase 1 ainda não estiver bloqueada;
- *Fase1BloqueadaState*, se a fase 1 estiver bloqueada.

Por outro lado, é permitido avançar para outros estados:

- *Fase3MasFase2AbertaState*, se a fase 2 ainda não estiver bloqueada;
- *Fase3State*, quando a fase 2 se encontra bloqueada.

Por fim, pode-se sair do programa.

2.8 *Fase2BloqueadaState*

Com este estado, a fase 2 é bloqueada, estando disponíveis apenas os dados para consulta. É permitido regressar ao *Fase1BloqueadaState*. Pode avançar para os estados:

- *Fase3State*, se a fase 3 ainda não estiver bloqueada;
- *Fase3BloqueadaState*, se a fase 3 estiver bloqueada.

2.9 *GestaoCandidaturasState*

Neste estado, é possível inserir, consultar, editar e eliminar candidaturas, assim como importar e exportar esses dados para ficheiros *CSV*.

Através da função *regressarFase()*, o utilizador pode voltar à fase 2 (estado *Fase2State*).

2.10 *Fase3State*

No *Fase3State*, dá para atribuir propostas automaticamente, listar alunos e propostas, e exportar esses dados para ficheiros *CSV*. Este estado permite ainda fazer uma atribuição manual de propostas, passando para o estado *GestaoManualAtribuicoesState*.

Também é possível regressar à fase anterior, avançar para a próxima (bloqueando esta ou não) e sair do programa.

2.11 *Fase3MasFase2AbertaState*

A fase 3 só pode ser bloqueada se a fase 2 estiver na mesma condição, uma vez que é necessário que todos os alunos com candidaturas tenham projeto atribuído. Assim, se a fase 2 ainda não estiver fechada, a partir do *Fase3MasFaseAberta2State* é possível atribuir propostas automaticamente a alunos com autopropostas ou que estejam designados a propostas de docentes, e ainda regressar ao *Fase2State*, avançar para o *Fase4State* e sair do programa.

2.12 *Fase3BloqueadaState*

Neste estado, só é possível consultar os dados tratados na fase 3, não havendo maneira de os alterar. A partir deste, pode-se voltar ao *Fase2BloqueadaState*, passar para o *Fase4State* e sair do programa.

2.13 *Fase3AtribuicaoAutomatica*

É através deste estado que a atribuição automática de propostas aos alunos é feita, tendo por base as suas classificações, restrição de acessos a estágios e opções indicadas no estado *GestaoCandidaturasState*.

Pode-se regressar ao *Fase3State*, através da função *regressarFase()*.

2.14 *GestaoManualAtribuicoesState*

Com este estado, o utilizador consegue atribuir manualmente as propostas e removê-las através de operações *undo* e *redo*. Pode, ainda, remover todas as propostas (que não sejam autopropostas ou de docentes com aluno associado) e regressar ao *Fase3State*.

2.15 *Fase4State*

Neste estado decorre a fase de atribuição de orientadores (*GestaoManualOrientadoresState*), onde se pode associar automaticamente docentes proponentes de projetos, gerir manualmente orientações, listar os alunos com proposta atribuída, consultar dados de docentes, exportar os dados para ficheiros *CSV*. Para alem de sair do programa, poderá ainda:

- regressar ao *Fase3State*, se a fase 3 não estiver bloqueada;
- regressar ao *Fase3BloqueadaState*, se a fase 3 estiver bloqueada;
- regressar ao *Fase3MasFase2Aberta*, se as fase 2 e 3 não estiverem bloqueadas;
- avançar para o *Fase5State*.

2.16 *GestaoManualOrientadoresState*

Este estado permite adicionar, alterar e remover orientadores, podendo ainda realizar operações de *undo* e *redo*. É também possível consultar as propostas atribuídas e regressar à fase 4.

2.17 *Fase5State*

Este estado disponibiliza apenas a consulta do que foi implementado nas fases anteriores, impossibilitando regressar às mesmas. É possível consultar lista de estudantes com propostas atribuídas, lista de estudantes sem propostas atribuídas e com opções de candidatura, conjunto de propostas disponíveis, conjunto de propostas atribuídas e número de orientações, bem como exportar os dados dos alunos, docentes, propostas, candidaturas e propostas atribuídas para ficheiros *CSV*.

3 Outros padrões de Programação

3.1 *Factory*

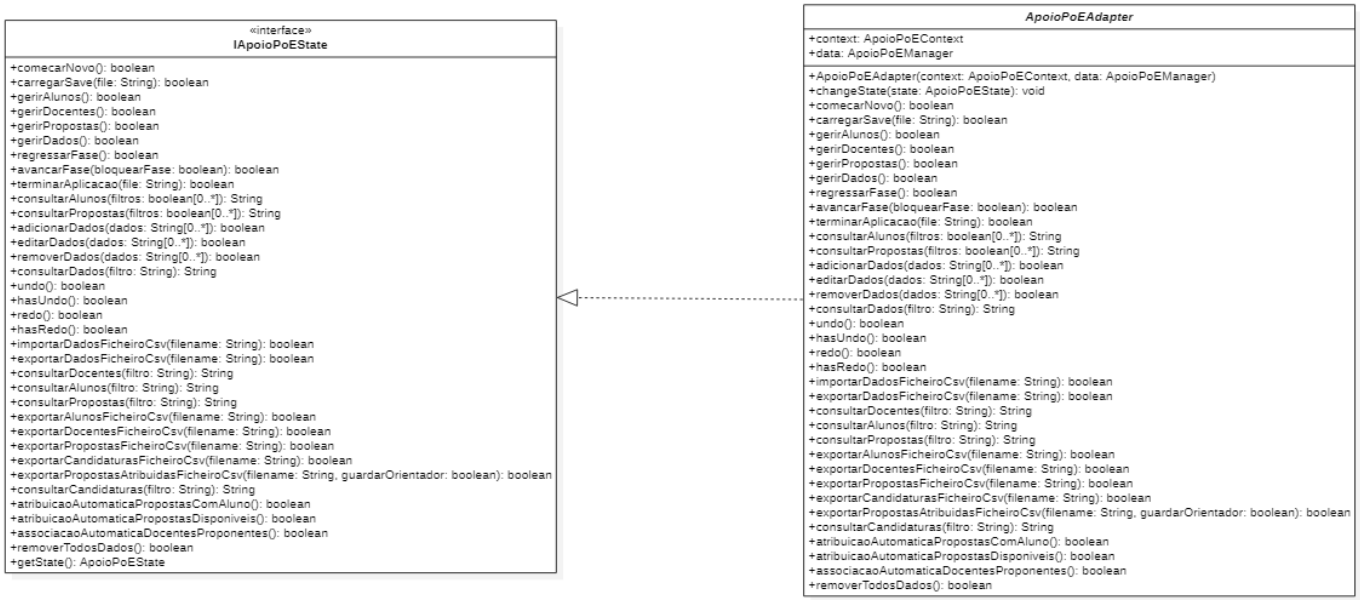

Figura 2 - *Factory*, Interface e Classe Abstrata

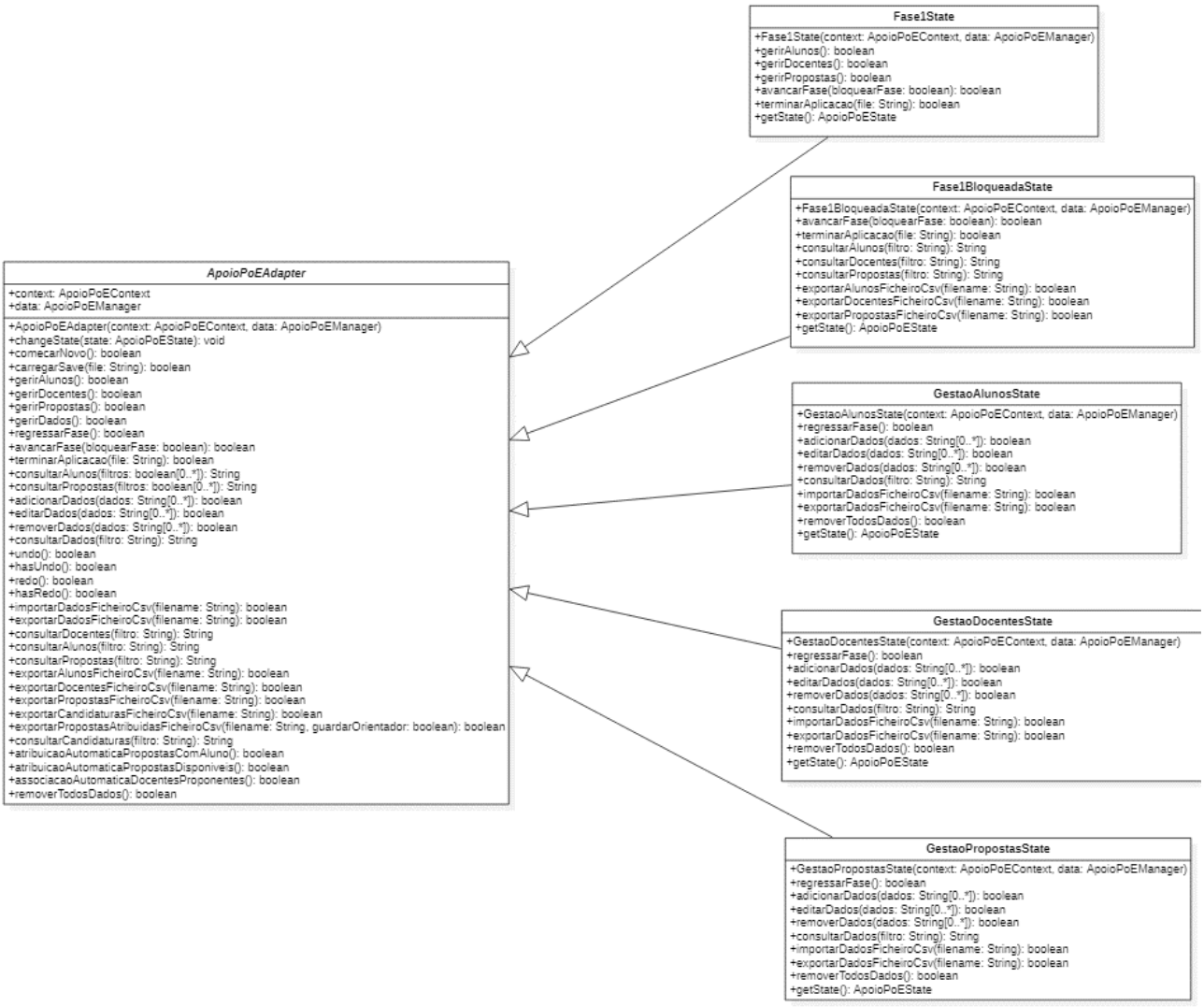

Figura 3 - *Factory*, Classes derivadas da classe Abstrata

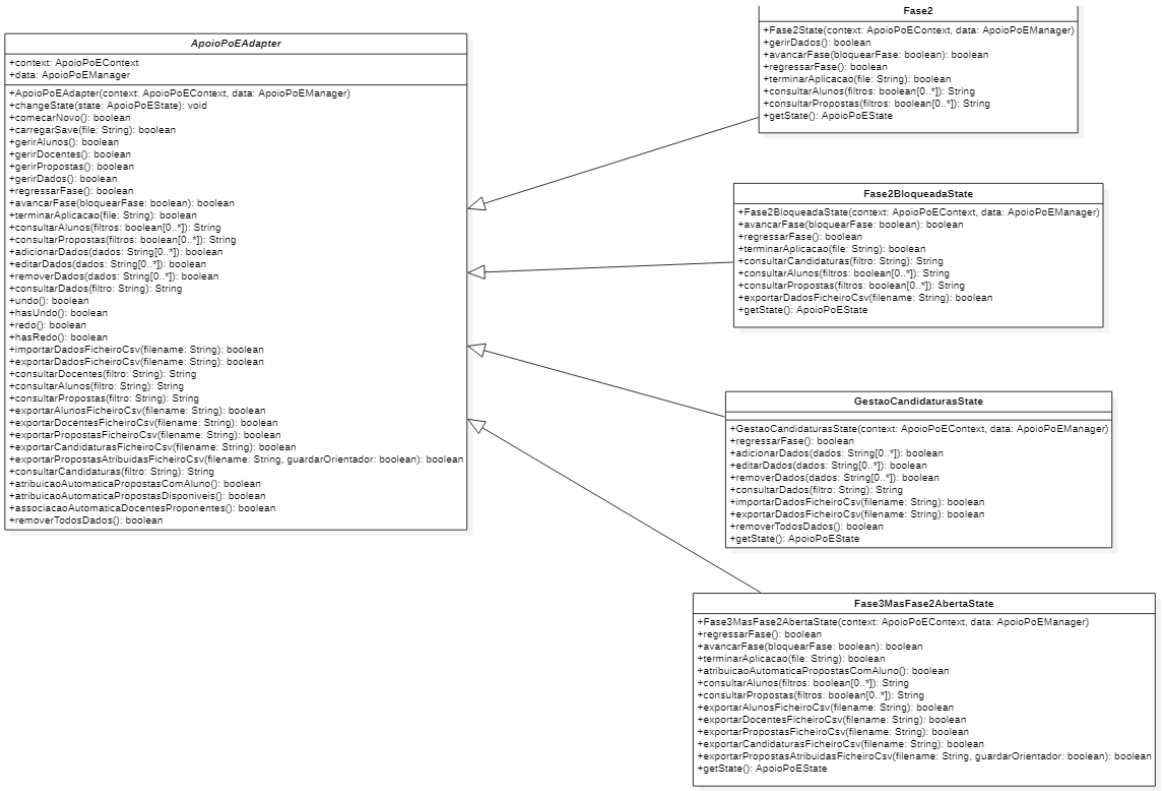

Figura 4 - *Factory*, Classes derivadas da classe abstrata

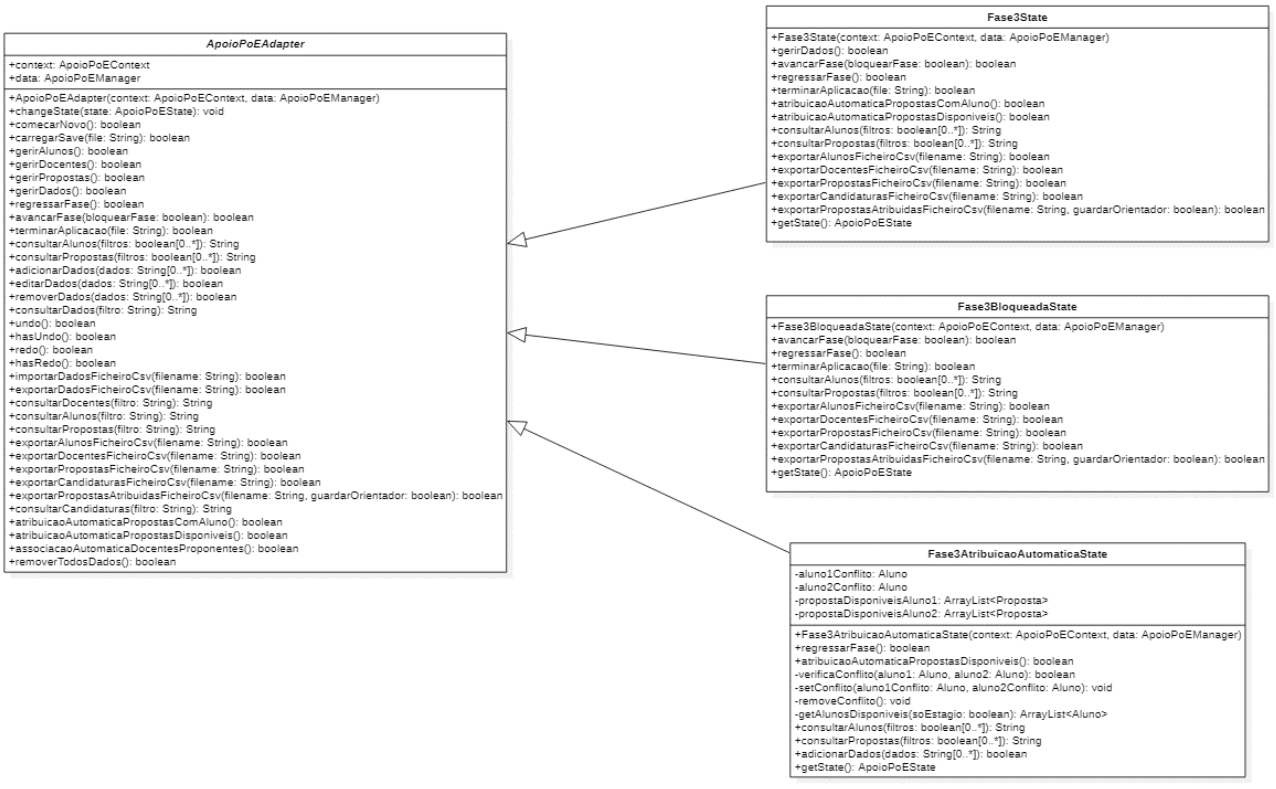

Figura 5 - *Factory*, Classes derivadas da classe abstrata

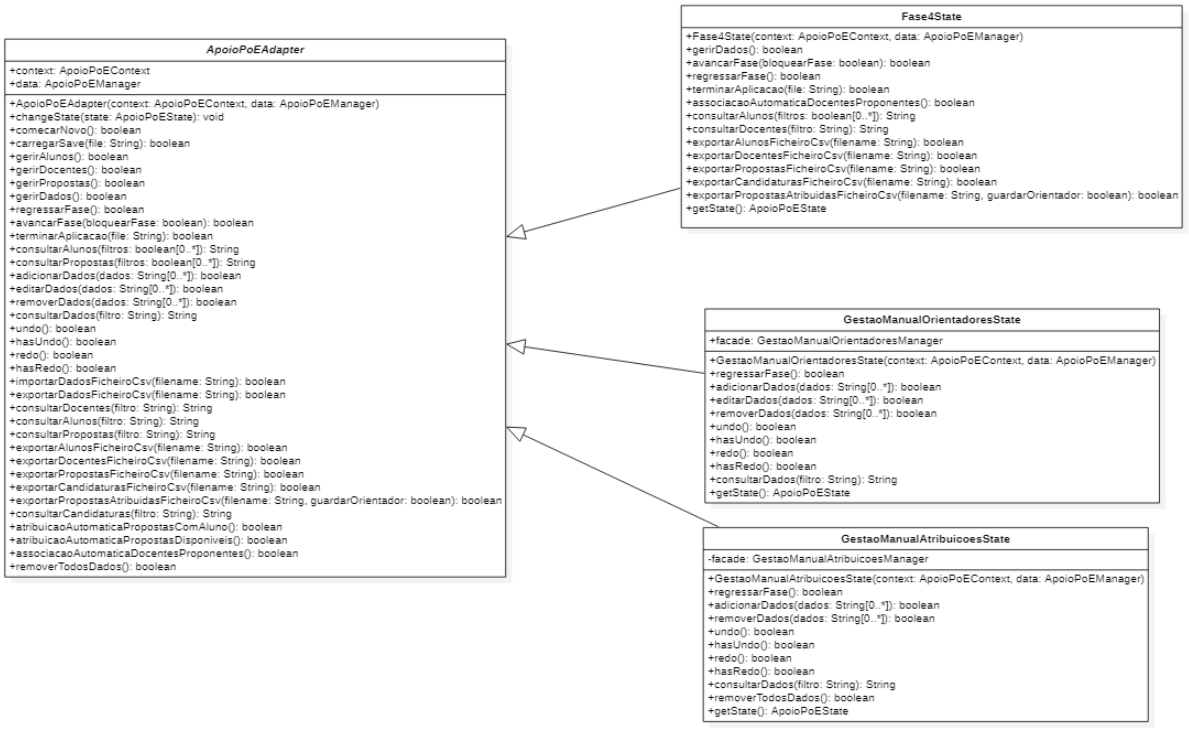

Figura 6 - *Factory*, Classes derivadas da classe abstrata

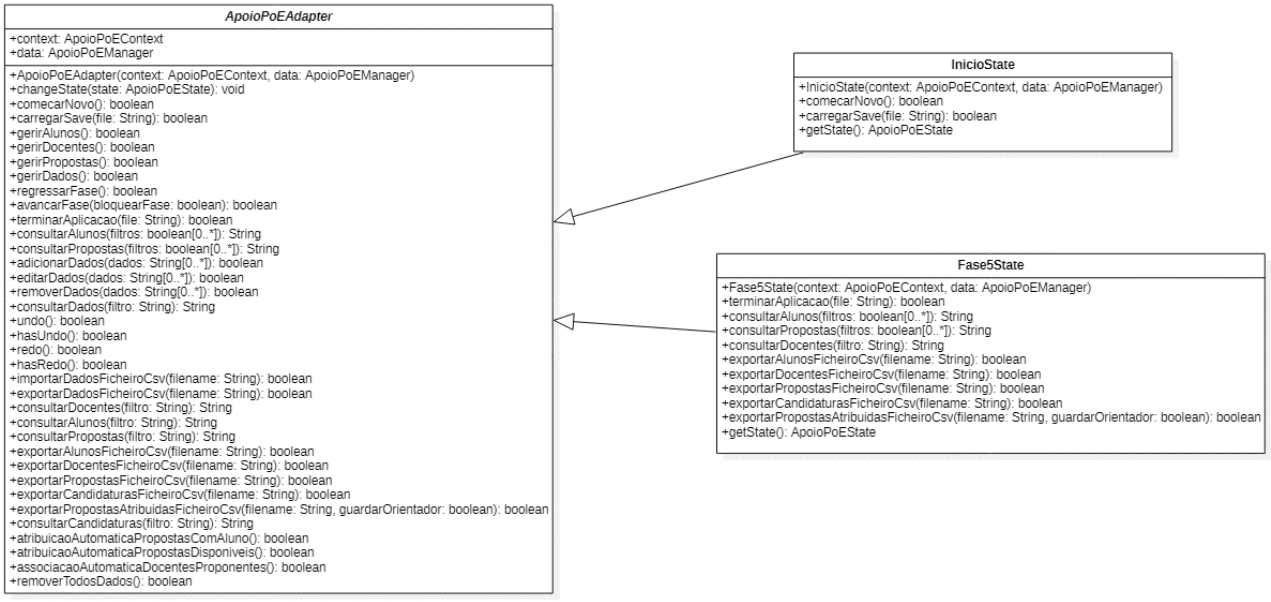

Figura 7 - *Factory*, Classes derivadas da classe abstrata

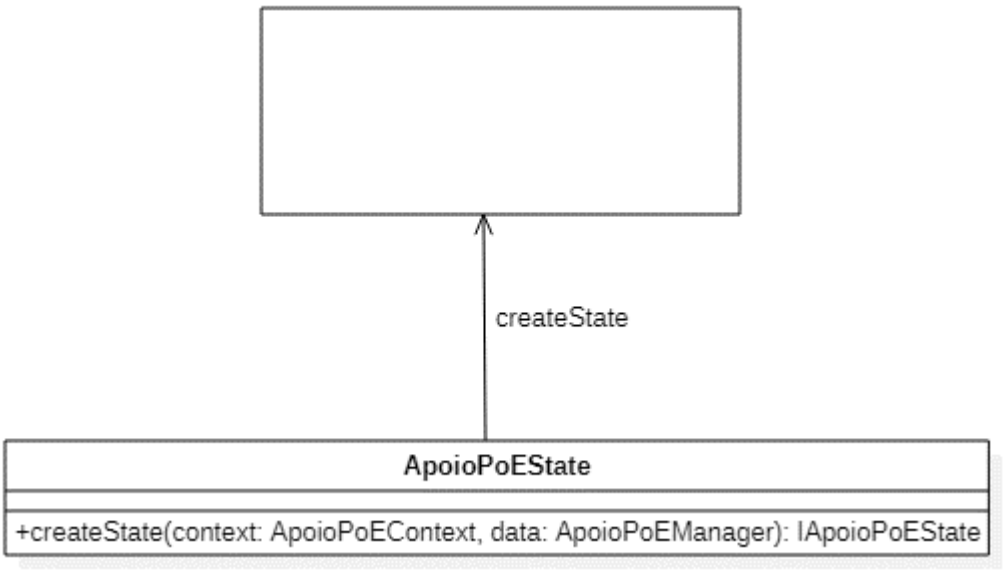

Figura 8 - *Factory*, Classe *Factory* que cria o objeto necessário

Dentro do retângulo desenhado fica toda a estrutura demonstrada nas figuras 2 a 7.

3.2 *Singleton*

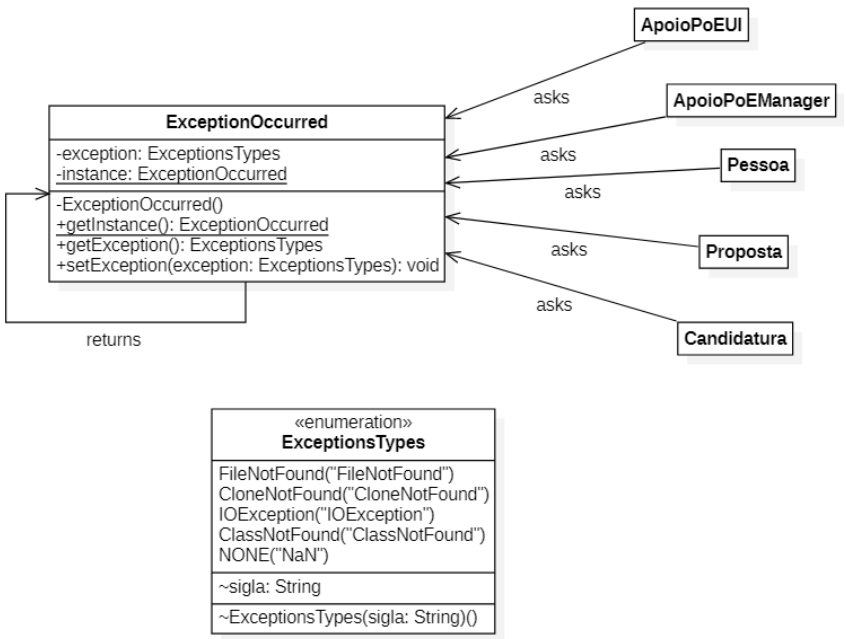

Figura 9 - Padrão de Programação *Singleton*

3.3 *Command*

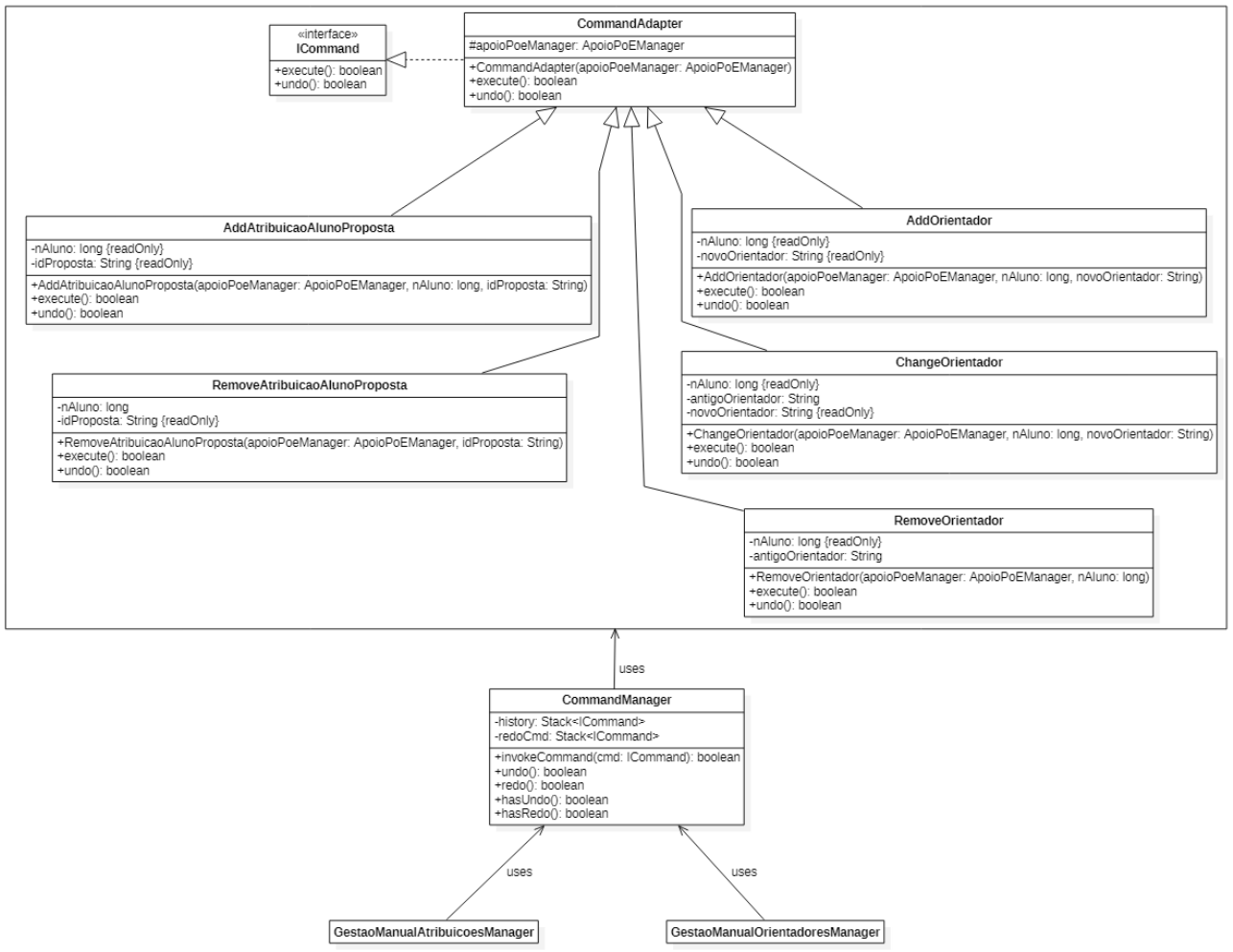

Figura 10 - Padrão de Programação *Command*

3.4 *Facade*

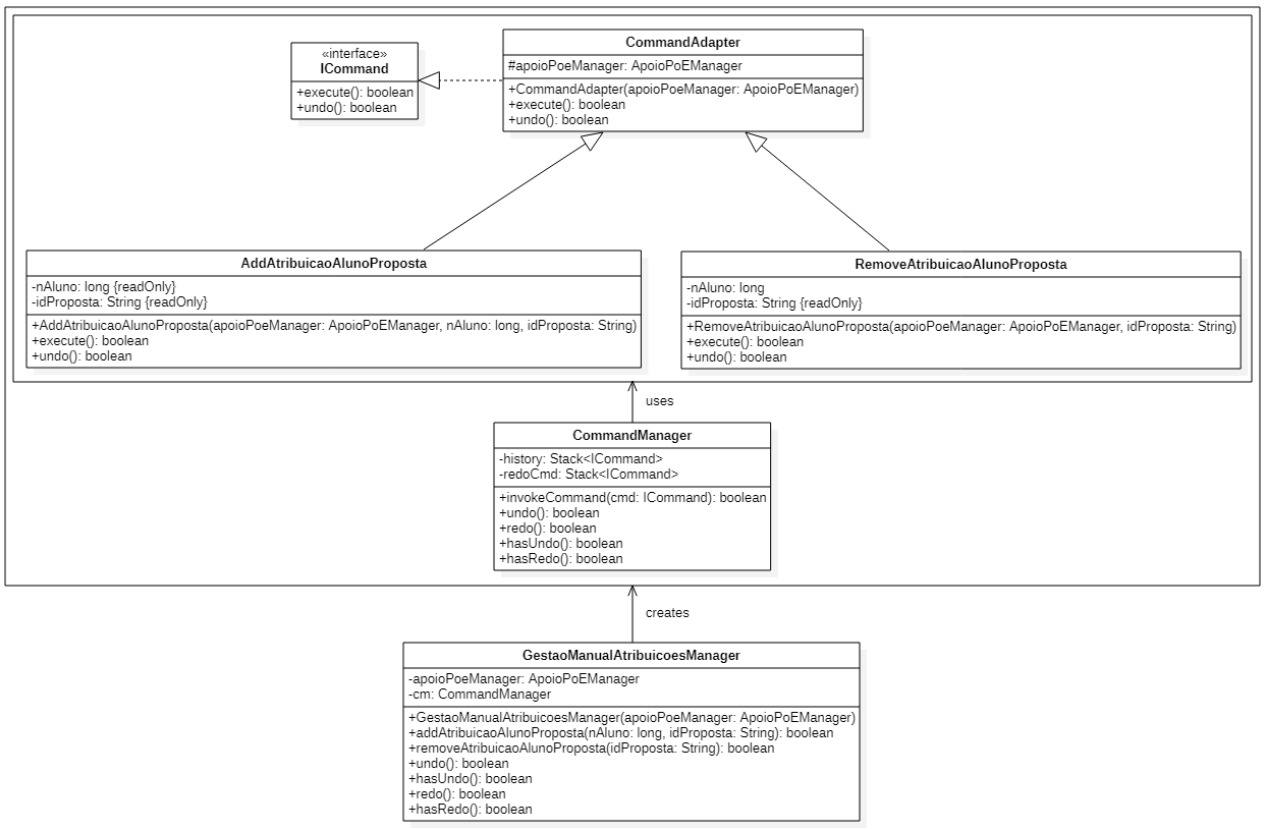

Figura 11 - Padrão de Programação *Facade*, Gestão Manual de Atribuições

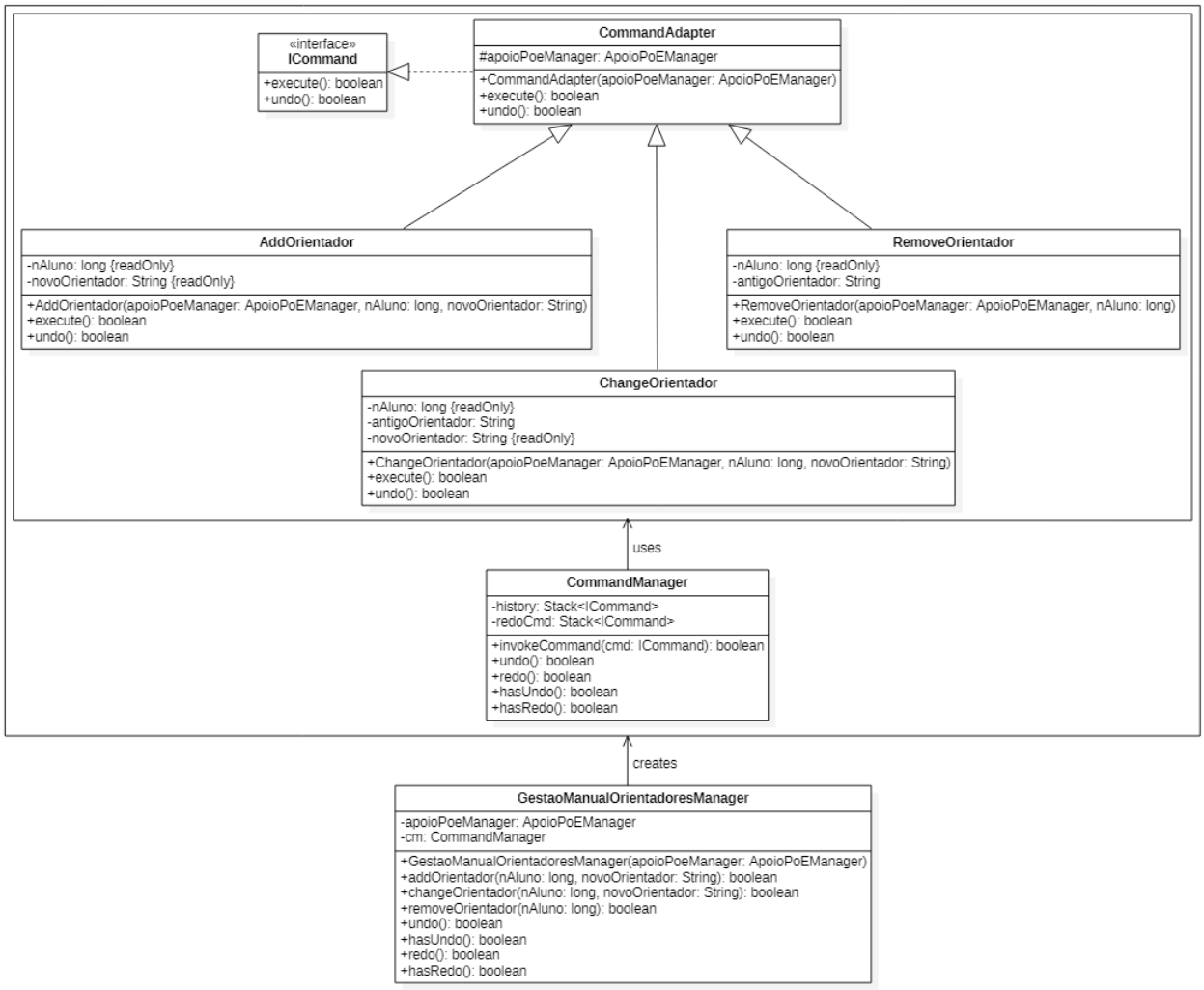

Figura 12 - Padrão de Programação *Facade*, Gestão Manual de Orientadores

3.5 *Proxy*

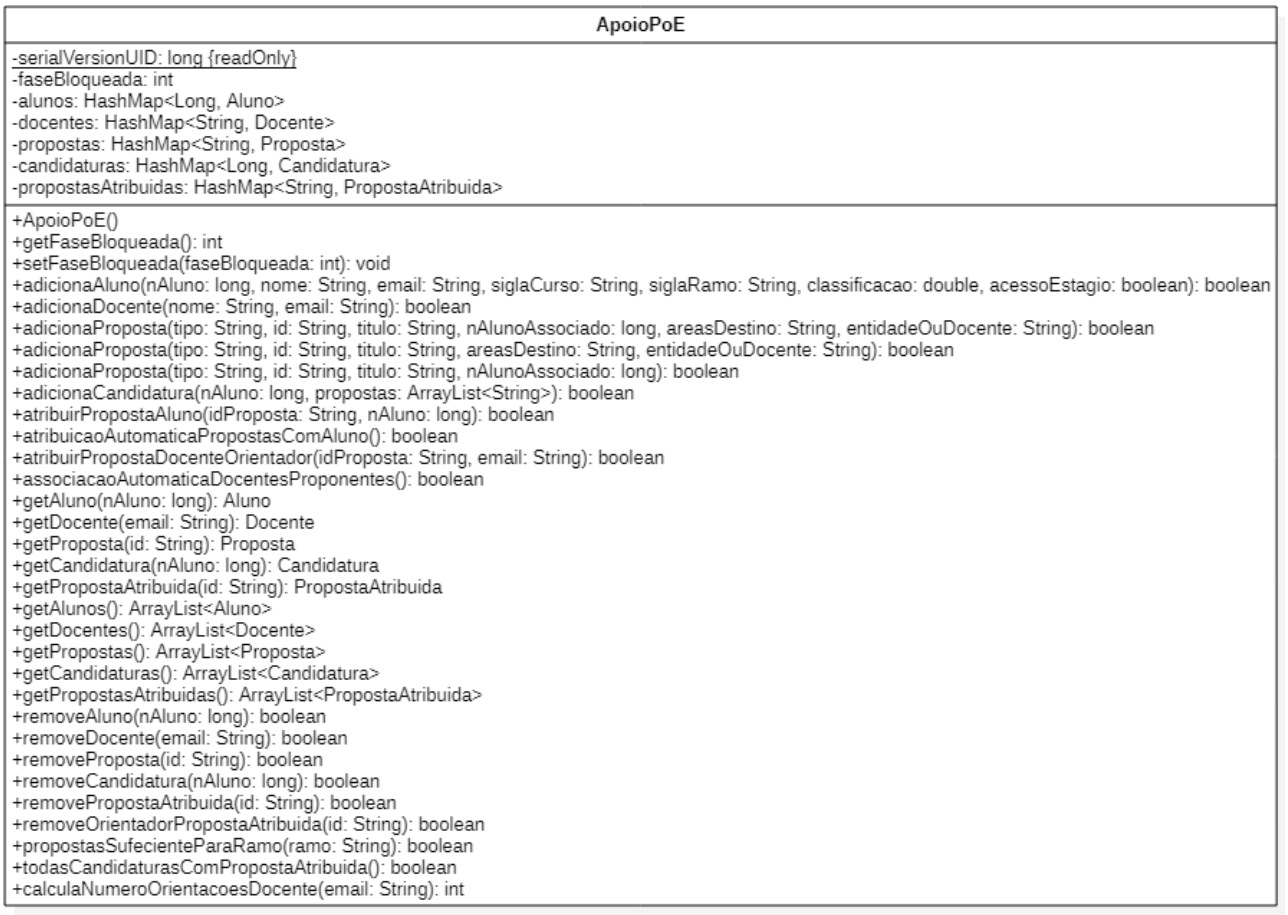

Figura 13 - Padrão de Programação *Proxy*, classe com os dados e lógica

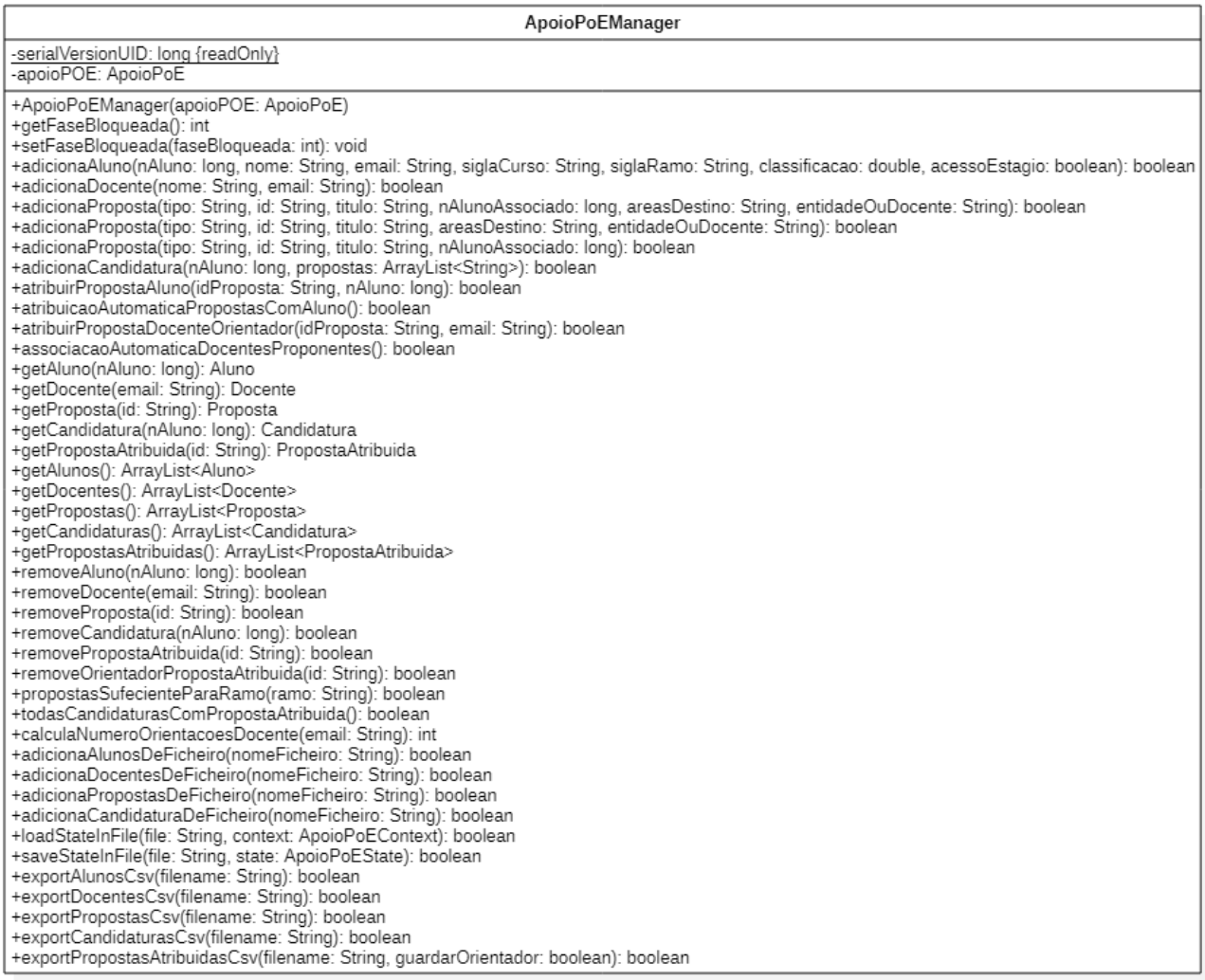

Figura 14 - Padrão de Programação *Proxy*, classe *Proxy*, adiciona métodos para importar, exportar dados e fazer *save* e *load* dos dados e estado para ficheiros binários

3.6 *Decorator*

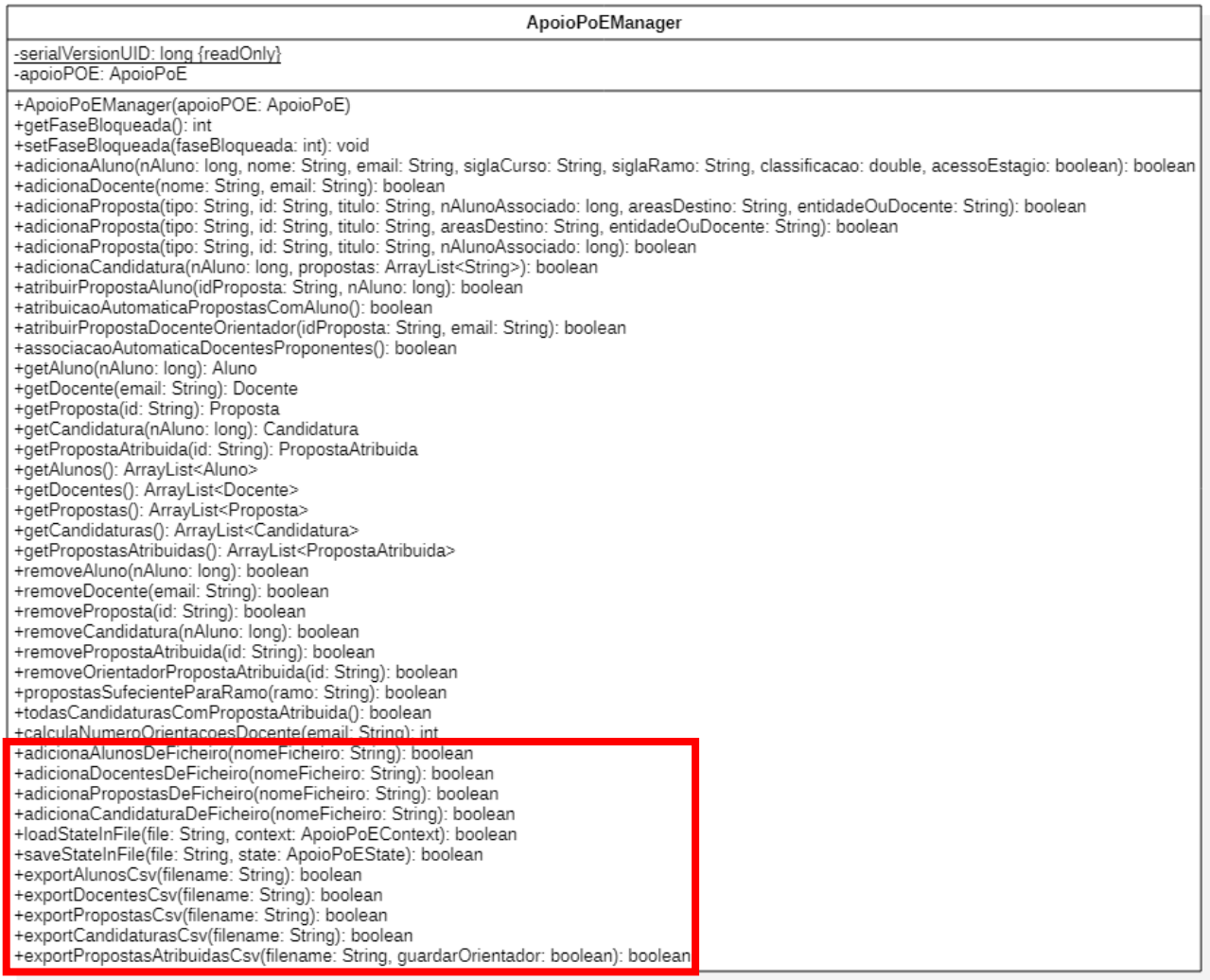

Figura 15 – Padrão de Programação *Decorator*, Adiciona as classe destacadas no retângulo vermelho à classe da Figura 13

4 Classes Utilizadas

4.1 *pt.isec.pa.apoio_poe.model.cmd*

4.1.1. *ICommand*

A classe *ICommand* é a interface base para todos os comandos. Todos os comandos implementados terão uma ação de *execute()* e outra de *undo()*. A operação *execute()* executa o comando e retorna *true* caso este tenha sido executado com sucesso. A operação *undo()* reverte as alterações que tinham sido realizadas com a operação *execute()*.

4.1.2. *CommandManager*

A classe *CommandManager* é o gestor de comandos, responsável por executar os comandos, guardar um histórico dos comandos realizados e guardar também os comandos que foram revertidos, podendo voltar a executá-los.

4.1.3. *CommandAdapter*

A classe *CommandAdapter* tem como objetivo disponibilizar implementações padrão para as operações da interface *ICommand*. Desta forma as classes que estenderem da classe *CommandAdapter* não necessitarão de implementar todas as operações possíveis, apenas terão de redefinir as que realmente necessitam.

4.1.4. *AddAtribuicaoAlunoProposta*

A classe *AddAtribuicaoAlunoProposta* representa a implementação de um comando concreto. Esta classe tem como objetivo atribuir um aluno a uma proposta. Nesta classe estão redefinidas as operações de *execute()* e *undo()*, com o objetivo de criar uma nova atribuição ou revertendo o comando, retirando essa atribuição.

4.1.5. *RemoveAtribuicaoAlunoProposta*

A classe *RemoveAtribuicaoAlunoProposta* representa a implementação de um comando concreto. O objetivo desta classe é remover uma atribuição de um aluno a uma proposta. Para tal são redefinidas as operações de *execute()* e *undo()*, indo estas remover uma atribuição ou reverter este comando, voltando a adicionar a atribuição.

4.1.6. *AddOrientador*

A classe *AddOrientador* representa a implementação de um comando concreto. Esta classe tem como objetivo adicionar a uma proposta atribuída um docente orientador. Para isso redefine as operações de *execute()* e *undo()*, onde estas vão adicionar um docente a uma atribuição que ainda não tenha docente, ou remover o docente orientador na operação de *undo()*.

4.1.7. *ChangeOrientador*

A classe *ChangeOrientador* representa a implementação de um comando concreto. O objetivo desta classe é alterar o docente orientador a uma proposta atribuída que já tenha um orientador atribuído. Para tal redefine as operações de *execute()* e *undo()*, onde estas vão alterar o antigo docente orientador pelo novo especificado pelo utilizador, ou reverter esta operação colocando o antigo docente orientador.

4.1.8. *RemoveOrientador*

A classe *RemoveOrientador* representa a implementação de um comando concreto. Esta classe tem como objetivo remover o docente orientador de uma proposta atribuída, ficando esta sem docente orientador. Para isso redefine as operações de *execute()* e *undo()*, onde estas vão remover o atual docente orientador, ou revertendo a operação colocando o antigo docente orientador.

4.2 *pt.isec.pa.apoio_poe.model.data*

4.2.1. *Pessoa*

A classe *Pessoa* representa uma classe abstrata com as informações básicas de uma pessoa no contexto da aplicação. Tem como objetivo servir de classe mãe para as classes *Docente* e *Aluno*.

4.2.2. *Docente*

A classe *Docente* representa toda a informação necessária sobre os docentes no contexto desta aplicação. Para tal deriva da classe *Pessoa* de forma a utilizar as funcionalidades por esta disponibilizadas.

4.2.3. *Aluno*

A classe *Aluno* representa um aluno no contexto desta aplicação. Tem como objetivo guardar toda a informação relativa a um aluno. Para isso deriva da classe *Pessoa* com o objetivo de utilizar as funcionalidades por esta disponibilizadas.

4.2.4. *AlunoClassificacaoComparator*

A classe *AlunoClassificacaoComparator* representa um comparador de dois alunos pela classificação dos mesmos. Esta classe tem como objetivo ser utilizada para ordenar os alunos pela sua classificação, útil no momento de atribuição de propostas aos alunos de forma automática. Para tal implementa a interface *Comparator* e o seu método *compare()*.

4.2.5. *Proposta*

A classe *Proposta* representa uma classe abstrata com as informações básicas de todas as propostas. Esta classe tem como objetivo servir de classe mãe para as classes *Projeto*, *Estagio*, *Autoproposto* e *PropostaAtribuida*.

4.2.6. *Projeto*

A classe *Projeto* representa uma proposta do tipo projeto. Esta classe tem como objetivo guardar os dados referentes a uma proposta de projeto. Para tal deriva da classe *Proposta*, de forma a utilizar as funcionalidades por esta disponibilizadas.

4.2.7. *Estagio*

A classe *Estagio* representa uma proposta do tipo estágio. Esta classe tem como objetivo guardar toda a informação referente a uma proposta de estágio. Para isso deriva da classe *Proposta*, com o objetivo de utilizar as funcionalidades que esta disponibiliza.

4.2.8. *Autoproposto*

A classe *Autoproposto* representa uma proposta de estágio/projeto apresentada por um aluno. Para tal guarda todas as informações referentes a uma autoproposta de um aluno. Esta classe também deriva da classe *Proposta*, utilizando também as funcionalidades que esta disponibiliza.

4.2.9. *PropostaAtribuida*

A classe *PropostaAtribuida* representa a atribuição de uma proposta a um aluno e qual o docente orientador atribuído a esta proposta. Esta classe tem como objetivo guardar toda a informação necessária para representar a atribuição de uma proposta a um aluno e qual o docente orientador para esta proposta.

4.2.10. *Candidatura*

A classe *Candidatura* tem como objetivo representar a candidatura de um aluno a uma ou várias propostas. Para isso guarda toda a informação necessária, nomeadamente o aluno candidato e as propostas a que este se está a candidatar.

4.2.11. *ApoioPoE*

A classe *ApoioPoE* representa todos os dados do programa. Tem como objetivo guardar os vários alunos, docentes, propostas e candidaturas que são criados no decorrer da execução da aplicação. Disponibiliza também métodos que permitem manipular esta informação.

4.2.12. *ApoioPoEManager*

A classe *ApoioPoEManager* representa um *proxy* para a classe *ApoioPoE*. Tem como objetivo fazer a gestão de um objeto *ApoioPoE*, disponibilizando métodos semelhantes aos da classe *ApoioPoE* redirecionando todo o processamento para a classe *ApoioPoE*. Nesta classe é também implementado o padrão *Decorator*, onde são disponibilizadas novas funcionalidades à classe *ApoioPoE* sem que esta sofra alterações à sua estrutura, nomeadamente a introdução de dados vinda de ficheiros e a exportação dos mesmos, bem como a realização do *load* e *save* do estado e dados da aplicação num determinado ponto da sua execução.

4.3 *pt.isec.pa.apoio_poe.model.exceptionsHandling*

4.3.1. *ExceptionsTypes*

A classe *ExceptionTypes* representa uma classe enumerável. Esta classe tem como objetivo representar as diversas possíveis exceções que podem decorrer durante a execução da aplicação.

4.3.2. *ExceptionOccurred*

A classe *ExceptionOccurred* representa uma classe segundo o padrão *Singleton*. Esta classe tem como objetivo representar uma exceção quando esta ocorre, de forma que a interface com o utilizador possa apresentar uma mensagem personalizada conforme o tipo de exceção que tenha ocorrido.

4.4 *pt.isec.pa.apoio_poe.model.facade*

4.4.1. *GestaoManualAtribuicoesManager*

A classe *GestaoManulaAtribuicoesManager* representa uma classe segundo o padrão *Facade*. Esta classe tem como objetivo fornecer uma interface mais simples para as classes que vão utilizar as funcionalidades que esta esconde, neste caso a criação e remoção de atribuições de propostas a alunos, realizada através do padrão *Command*.

4.4.2. *GestaoManualOrientadoresManager*

A classe *GestaoManualOrientadoresManager* representa uma classe segundo o padrão *Facade*. Esta classe tem como objetivo esconder funcionalidades mais complexas, de forma a criar uma interface mais simples para as classes que utilizem estas funcionalidades. Esta classe refere-se a associação, alteração e desassociação de um docente orientador a uma proposta atribuída, implementados segundo o padrão *Command*.

4.5 *pt.isec.pa.apoio_poe.model.fsm*

4.5.1. *ApoioPoEState*

A classe *ApoioPoEState* representa uma classe enumerável com todos os estados possíveis. Esta classe tem ainda o objetivo de criar instâncias das classes que representam os diversos estados, usando para isso o padrão *Factory*.

4.5.2. *IApoioPoEState*

A classe *IApoioPoEState* representa uma interface base para todos os estados da aplicação. Esta interface tem como objetivo criar todos os métodos a serem implementar pelos diversos estados, sendo estes métodos representativos da transição entre estados e também das operações que podem ser realizadas em cada estado.

4.5.3. *ApoioPoEContext*

A classe *ApoioPoEContext* representa a classe que será utilizada pelas diversas interfaces com o utilizador para estas receberem e alterarem os dados. Esta classe tem como objetivo garantir que as alterações aos dados são apenas realizadas no contexto de um estado, não possibilitando a alteração direta dos dados.

4.5.4. *ApoioPoEAdapter*

A classe *ApoioPoEAdapter* representa as implementações por omissão dos métodos definidos na interface *IApoioPoeState*. Esta classe tem como objetivo criar implementações por omissão, desta forma os diversos estados apenas terão de redefinir os métodos que utilizem. Fornece ainda um método que permite alterar o estado atual no contexto.

4.5.5. *InicioState*

A classe *InicioState* representa a implementação de um estado. Esta classe tem como objetivo dar a possibilidade ao utilizador de criar uma nova execução com os dados vazios, ou então carregar um estado guardado anteriormente.

4.5.6. *Fase1State*

A classe *Fase1State* representa a implementação de um estado. Esta classe tem como objetivo permitir ao utilizar alternar entre os modos de gestão de alunos, docentes ou propostas.

4.5.7. *Fase1BloqueadaState*

A classe *Fase1BloqueadaState* representa a implementação de um estado. Esta classe tem como objetivo fornecer ao utilizador a possibilidade de consultar e exportar os dados dos alunos, docentes e propostas após a primeira fase ser fechada a novas alterações.

4.5.8. *Fase2State*

A classe *Fase2State* representa a implementação de um estado. Esta classe tem como objetivo possibilitar ao utilizador a gestão de candidaturas, bem como consultar dados sobre os alunos e propostas. Podendo ainda regressar à primeira fase e avançar para a terceira fase.

4.5.9. *Fase2BloqueadaState*

A classe *Fase2BloqueadaState* representa a implementação de um estado. Esta classe tem como objetivo permitir ao utilizador consultar os dados das candidaturas e poder exportar estes dados, após a segunda fase ter sido fechada a novas alterações.

4.5.10. *Fase3State*

A classe *Fase3State* representa a implementação de um estado. Esta classe tem como objetivo proporcionar ao utilizador três diferentes mecanismos para atribuir propostas ao utilizador, dois de forma automática, tendo um deste uma resolução de conflitos entre alunos, e ainda uma outra forma de gerir as atribuições de propostas a alunos de forma manual. Permite ainda obter dados sobre os alunos e sobre as várias propostas.

4.5.11. *Fase3MasFase2AbertaState*

A classe *Fase3MasFase2AbertaState* representa a implementação de um estado. Esta classe tem como objetivo apenas fornecer uma forma de atribuir propostas aos alunos, sendo esta totalmente automática, limitando assim as possibilidades do utilizador, devido à segunda fase ainda se encontrar aberta a alterações, podendo ser alteradas as candidaturas.

4.5.12. *Fase3BloqueadaState*

A classe *Fase3BloqueadaState* representa a implementação de um estado. Esta classe tem como objetivo apresentar dados sobre os alunos e propostas ao utilizador após a terceira fase ter sido fechada a alterações. Possibilita ainda a exportação de dados sobre os alunos, docente, propostas, candidaturas e propostas atribuídas.

4.5.13. *Fase3AtribuicaoAutomaticaState*

A classe *Fase3AtribuicaoAutomaticaState* representa a implementação de um estado. Esta classe tem como objetivo a atribuição automática de alunos a propostas segundo a sua classificação e as suas opções de candidatura. Tem ainda o objetivo de permitir ao utilizador a resolução de conflitos entre dois alunos.

4.5.14. *Fase4State*

A classe *Fase4State* representa a implementação de um estado. Esta classe tem como objetivo possibilitar ao utilizador duas formas de atribuir docentes orientadores a propostas atribuídas, uma forma automática e uma forma manual. Permite ainda obter dados sobre os alunos e os docentes e exportar dados sobre os alunos, docentes, propostas, candidaturas e propostas atribuídas.

4.5.15. *Fase5State*

A classe *Fase5State* representa a implementação de um estado. Esta classe tem como objetivo apresentar informação sobre os alunos, propostas e docentes. Permitindo também exportar dados sobre os alunos, docentes, propostas, candidaturas e propostas atribuídas.

4.5.16. *GestaoAlunosState*

A classe *GestaoAlunosState* representa a implementação de um estado. Esta classe tem como objetivo possibilitar ao utilizador a gestão manual de alunos, bem como importar e exportar alunos através de ficheiros *csv*.

4.5.17. *GestaoDocentesState*

A classe *GestaoDocentesState* representa a implementação de um estado. Esta classe tem como objetivo permitir ao utilizador gerir manualmente os docentes, bem como a possibilidade de importar ou exportar docentes para ficheiros *csv*.

4.5.18. *GestaoPropostasState*

A classe *GestaoPropostasState* representa a implementação de um estado. Esta classe tem como objetivo fornecer ao utilizador uma forma de gerir as propostas de forma manual, e ainda possibilitar que este importe ou exporte os dados das propostas para ficheiros *csv*.

4.5.19. *GestaoCandidaturasState*

A classe *GestaoCandidaturasState* representa a implementação de um estado. Esta classe tem como objetivo fornecer ao utilizador uma forma de gerir as candidaturas manualmente, bem como permitir que o utilizador importe ou exporte candidaturas para ficheiros *csv*.

4.5.20. *GestaoManualAtribuicoesState*

A classe *GestaoManualAtribuicoesState* representa a implementação de um estado. Esta classe tem como objetivo permitir ao utilizador que este gira as atribuições de alunos a propostas de forma manual. O utilizador tem ainda a possibilitar de reverter ou refazer as alterações aos dados que realizou através de comandos *undo* e *redo*.

4.5.21. *GestaoManualOrientadoresState*

A classe *GestaoManualOrientadoresState* representa a implementação de um estado. Esta classe tem como objetivo possibilitar que o utilizador faça a gestão manual da atribuição de docentes orientador a propostas que estejam atribuídas a alunos. O utilizador pode ainda reverter ou refazer as alterações que realizou através de comandos *undo* e *redo*.

4.6 *pt.isec.pa.apoio_poe.ui.text*

4.6.1. *ApoioPoEUI*

A classe *ApoioPoEUI* representa a interface com o utilizador em modo texto. Esta classe tem como objetivo permitir ao utilizador interagir com a aplicação permitindo a alteração de dados através dos vários estados e métodos criados.

4.7 *pt.isec.pa.apoio_poe.utils*

4.7.1. *PAInput*

A classe *PAInput* representa uma forma mais simples de pedir dados ao utilizador em formato de texto. Esta classe tem como objetivo fornecer métodos e verificações aos dados inseridos pelo utilizador. Esta classe foi fornecida nas aulas práticas, sendo que foi adicionado um método que permite ao utilizador não inserir nada, útil para a consulta de dados, quando o utilizador pretende ver todos os dados e não apenas de um aluno/docente/proposta/candidatura específica.

4.8 *pt.isec.pa.apoio_poe*

4.8.1. *Main*

A classe *Main* representa o início da aplicação. Esta classe tem como objetivo criar a máquina de estados e fornecer esta às várias interfaces que se pretendam executar.

5 Relacionamento entre Classes

5.1 Máquina de Estados

A interface *IApoioPoEState* é implementada pela classe *ApoioPoEAdapter*, que por sua vez, é estendida pelas classes que representam a máquina de estados: *Fase1BloqueadaState*, *Fase1State*, *Fase2BloqueadaState*, *Fase2State*, *Fase3AtribuicaoAutomatica*, *Fase3BloqueadaState*, *Fase3MasFase2AbertaState*, *Fase3State*, *Fase4State*, *Fase5State*, *GestaoAlunosState*, *GestaoCandidaturasState*, *GestaoDocentesState*, *GestaoManualAtribuicoesState*, *GestaoManualOrientadoresState*, *GestaoPropostasState*, *InicioState*.

A classe *ApoioPoEState* está responsável por gerir os estados, na medida em que tem uma *factory* que cria uma instância da classe conforme o estado atual.

A classe *ApoioPoEContext* está responsável por guardar os dados da execução da aplicação, bem como o estado atual. Redirecionando ainda a invocação dos métodos por parte da interface para a redefinição realizada pelo estado atual, ou caso este não implemente um método para a sua definição por omissão encontrada na classe *ApoioPoEAdapter*.

5.2 Comandos

A interface *ICommand* é implementada pela classe *CommandAdapter*, que é estendida pelas classes que representam os comandos concretos, *AddAtribuicaoAlunoProposta*, *RemoveAtribuicaoAlunoProposta*, *AddOrientador*, *ChangeOrientador* e *RemoveOrientador*.

Estes comandos são geridos e invocados pela classe *CommandManager*, que conta com funcionalidades *undo* e *redo*, que os guarda num histórico, permitindo reverter ou refazer comandos.

5.3 Dados

A classe *Pessoa* implementa interfaces *Cloneable* e *Serializable*. A interface *Cloneable* permite que sejam criados clones de objetos *Pessoa*, e a *Serializable* possibilita guardar objetos *Pessoa* em ficheiros binários.

Além disso, esta classe é estendida pelas classes *Aluno* e *Docente*, que por sua vez, implementam a interface *Comparable*. Assim, os objetos de *Aluno* podem ser ordenados pelo número de aluno, e os objetos de *Docente* por email.

Existe, também, a classe *AlunoClassificacaoComparator* que implementa a interface *Comparator*. Deste modo existe também a possibilidade de ordenar os alunos pela sua classificação.

A classe *Proposta* implementa a interface *Comparable*, que vai permitir ordenar as propostas pelos respetivos códigos (*idProposta*). Implementa, ainda, as interfaces *Cloneable*, permitindo que sejam criados clones de objetos *Proposta*, e, *Serializable*, podendo guardar objetos *Proposta* em ficheiros binários. Desta classe estendem outras, que representam os tipos de proposta existentes: *Autoproposto*, *Estagio*, *Projeto* e *PropostaAtribuida*.

A classe *ApoioPoE* implementa a interface *Serializable* e está responsável por gerir todo o funcionamento da aplicação utilizando as classes anteriormente mencionadas. Esta classe é gerida pelo *ApoioPoEManager*, que funciona como *proxy*, redirecionando para as funções do *ApoioPoE* e trata da importação e exportação dos dados para ficheiros.

A classe Candidaturas implementa as interfaces *Comparable* (objetos candidatura serão ordenados pelos números de aluno que lhes correspondem), *Cloneable* e *Serializable*. Aqui existem métodos que permitem adicionar ou remover propostas à candidatura.

5.4 *Facade*

A classe *GestaoManualAtribuicoesManager* é responsável por adicionar e remover atribuições de propostas a alunos, de acordo com os comandos que lhe estão associados (comandos geridos pela *CommandManager*). Compete, depois, à *GestaoManualAtribuicoesState*, gerir essa informação e fornecer uma interface mais simples para estes comandos serem utilizados por outras classes.

A classe *GestaoManualOrientadoresManager* é responsável por adicionar, alterar e remover atribuições de propostas a alunos, de acordo com os comandos que lhe estão associados (comandos geridos pela CommandManager). Compete, depois, à *GestaoManualAtribuicoesState*, gerir essa informação e fornecer uma interface mais simples para as outras classes utilizarem os comandos.

5.5 *ExceptionsHandling*

A classe *ExceptionTypes* é responsável por indicar qual o tipo de exceção que ocorreu, ou poderá ocorrer. A classe *ExceptionOccurred* contém o objeto da classe ExceptionTypes de forma a guardar qual a exceção que ocorreu com o objetivo de mostrar uma mensagem ao utilizador.

5.6 *UI*

A classe *ApoioPoeUI* trata de todos os dados introduzidos pelo utilizador no formato de texto. À medida que a informação é introduzida, envia-a para a máquina de estados, que por sua vez comunica com a *ApoioPoEManager*, e esta com a *ApoioPoE*.

5.7 *Utils*

A classe *PAInput* é utilizada na interface com o utilizador em formato de texto de forma a facilitar a verificação dos dados inseridos pelo utilizador aos longo da execução da aplicação.

6 Funcionalidades Implementadas

Tabela 1 - Funcionalidades Fase 1

Funcionalidade	Implementado	Razão
Alternar entre modos de gestão de alunos, docentes e propostas de estágio	Totalmente implementado	
Inserção, Consulta, Edição e Eliminação de dados referentes a alunos	Implementado parcialmente	Apenas foi implementada a consulta de dados, sendo esta a única requerida para a primeira meta do trabalho prático.
Inserção, Consulta, Edição e Eliminação de dados referentes a docentes	Implementado parcialmente	Apenas foi implementada a consulta de dados, sendo esta a única requerida para a primeira meta do trabalho prático.
Inserção, Consulta, Edição e Eliminação de dados referentes a propostas	Implementado parcialmente	Apenas foi implementada a consulta de dados, sendo esta a única requerida para a primeira meta do trabalho prático.
Eliminar todos os dados de alunos, docentes ou propostas	Totalmente implementado	
Importação de dados de alunos, docentes ou propostas a partir de ficheiros *csv*	Totalmente implementado	
Exportação de dados de alunos, docentes ou propostas para ficheiros *csv*	Totalmente implementado	
Fechar a fase	Totalmente implementado	
Avançar para a fase seguinte	Totalmente implementado	

Tabela 2 - Funcionalidades Fase 2

Funcionalidade	Implementado	Razão
Inserção, Consulta, Edição e Eliminação de dados referentes a candidaturas	Implementado parcialmente	Apenas foi implementada a consulta de dados, sendo esta a única requerida para a primeira meta do trabalho prático.
Eliminar todos os dados de candidaturas	Totalmente implementado	
Importação de dados de candidaturas a partir de ficheiros *csv*	Totalmente implementado	
Exportação de dados de candidaturas para ficheiros *csv*	Totalmente implementado	
Obter lista de alunos	Totalmente implementado	
Obter lista de propostas	Totalmente implementado	
Fechar a fase	Totalmente implementado	
Regressar à fase anterior	Totalmente implementado	
Avançar para a fase seguinte	Totalmente implementado	

Tabela 3 - Funcionalidades Fase 3

Funcionalidade	Implementado	Razão
Atribuição automática das autopropostas ou propostas de docentes com aluno associado	Totalmente implementado	
Atribuição automática de uma proposta disponível aos alunos sem atribuições definidas, com base nas suas classificações, restrição de acesso a estágios e opções indicadas no processo de candidatura	Totalmente implementado	
Atribuição manual de propostas disponíveis aos alunos sem atribuição ainda definida	Totalmente implementado	
Remoção manual de uma atribuição previamente realizada ou de todas as atribuições	Totalmente implementado	
Consultar propostas atribuídas	Totalmente implementado	
Operações de *undo* e *redo* na gestão manual	Totalmente implementado	
Importação de dados de candidaturas a partir de ficheiros *csv*	Totalmente implementado	
Exportação de dados de alunos, docentes, propostas, candidaturas ou propostas atribuídas para ficheiros *csv*	Totalmente implementado	

Obter lista de alunos	Totalmente implementado	
Obter lista de propostas	Totalmente implementado	
Fechar a fase	Totalmente implementado	
Regressar à fase anterior	Totalmente implementado	
Avançar para a fase seguinte	Totalmente implementado	

Tabela 4 - Funcionalidades Fase 4

Funcionalidade	Implementado	Razão
Associação automática dos docentes proponentes de projetos como orientador dos mesmos	Totalmente implementado	
Atribuição, consulta, alteração e eliminação de um orientador aos alunos com propostas atribuídas	Totalmente implementado	
Operações de *undo* e *redo* na gestão manual	Totalmente implementado	
Exportação de dados de alunos, docentes, propostas, candidaturas ou propostas atribuídas para ficheiros *csv*	Totalmente implementado	
Obtenção de listas de alunos com propostas atribuídas	Totalmente implementado	
Obtenção de dados diversos sobre atribuição de orientadores	Totalmente implementado	
Fechar a fase	Totalmente implementado	
Regressar à fase anterior	Totalmente implementado	
Avançar para a fase seguinte	Totalmente implementado	

Tabela 5 - Funcionalidades Fase 5

Funcionalidade	Implementado	Razão
Lista de estudantes com propostas atribuídas	Totalmente implementado	
Lista de estudantes sem propostas atribuídas e com opções de candidatura	Totalmente implementado	
Conjunto de propostas disponíveis	Totalmente implementado	
Conjunto de propostas atribuídas	Totalmente implementado	
Número de orientações por docente, em média, mínimo, máximo, e por docente especificado	Totalmente implementado	
Outros dados	Não implementado	Não foram encontrados outros dados que se considerassem pertinentes mostrar ao utilizador nesta fase.
Exportação de dados de alunos, docentes, propostas, candidaturas ou propostas atribuídas para ficheiros *csv*	Totalmente implementado	

Tabela 6 - Outras Funcionalidades

Funcionalidade	Implementado	Razão
Gravação/carregamento do estado da aplicação usando um formato binário	Totalmente implementado	
Interface em modo Texto	Implementado parcialmente	As interfaces relativas à inserção, consulta e remoção de alunos, docentes, propostas e candidaturas não foram implementadas, devido a estas funcionalidades não terem sido implementadas na primeira meta
Interface em modo Gráfico	Não implementado	Apenas requerido para a segunda meta do trabalho prático.