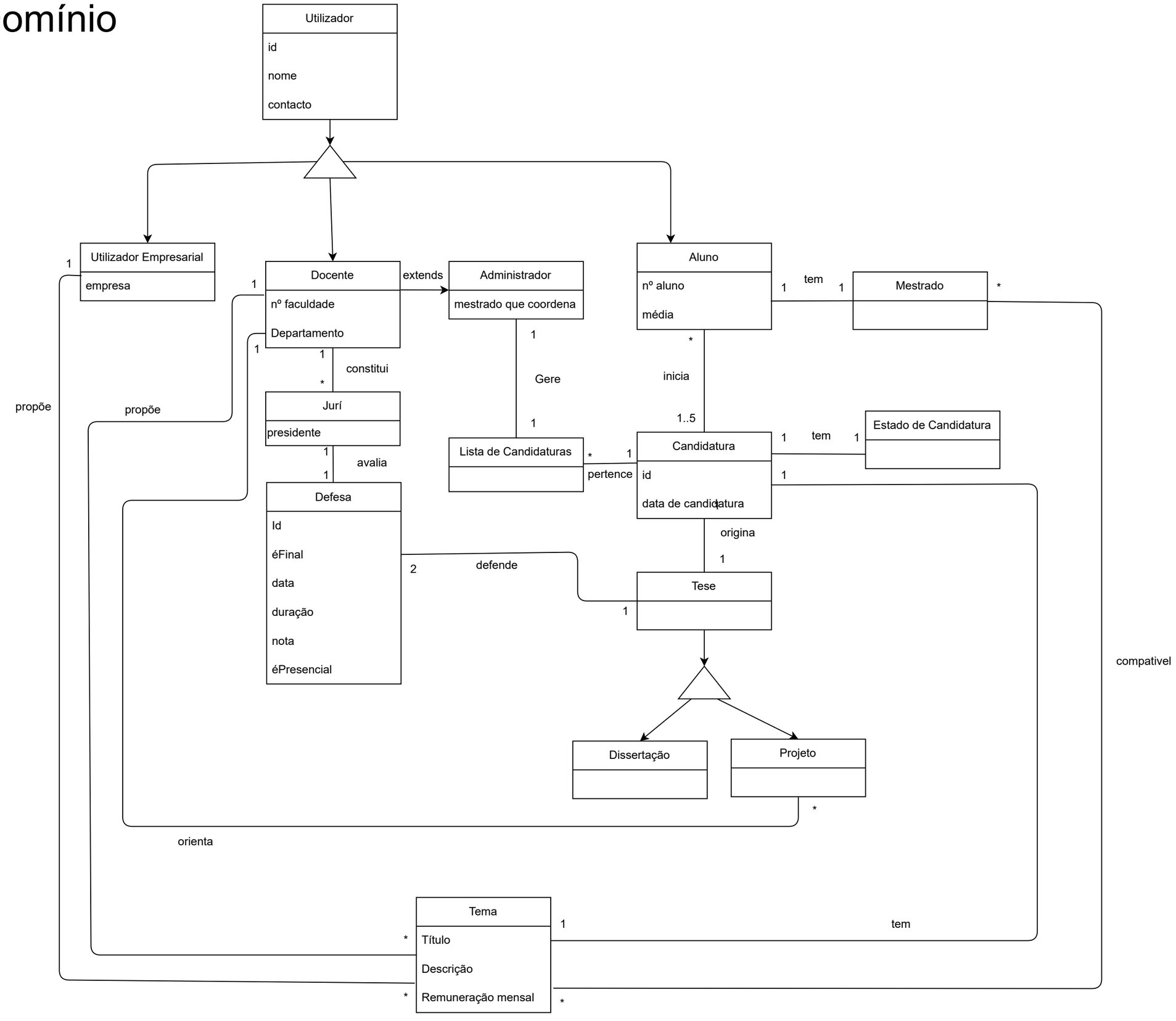
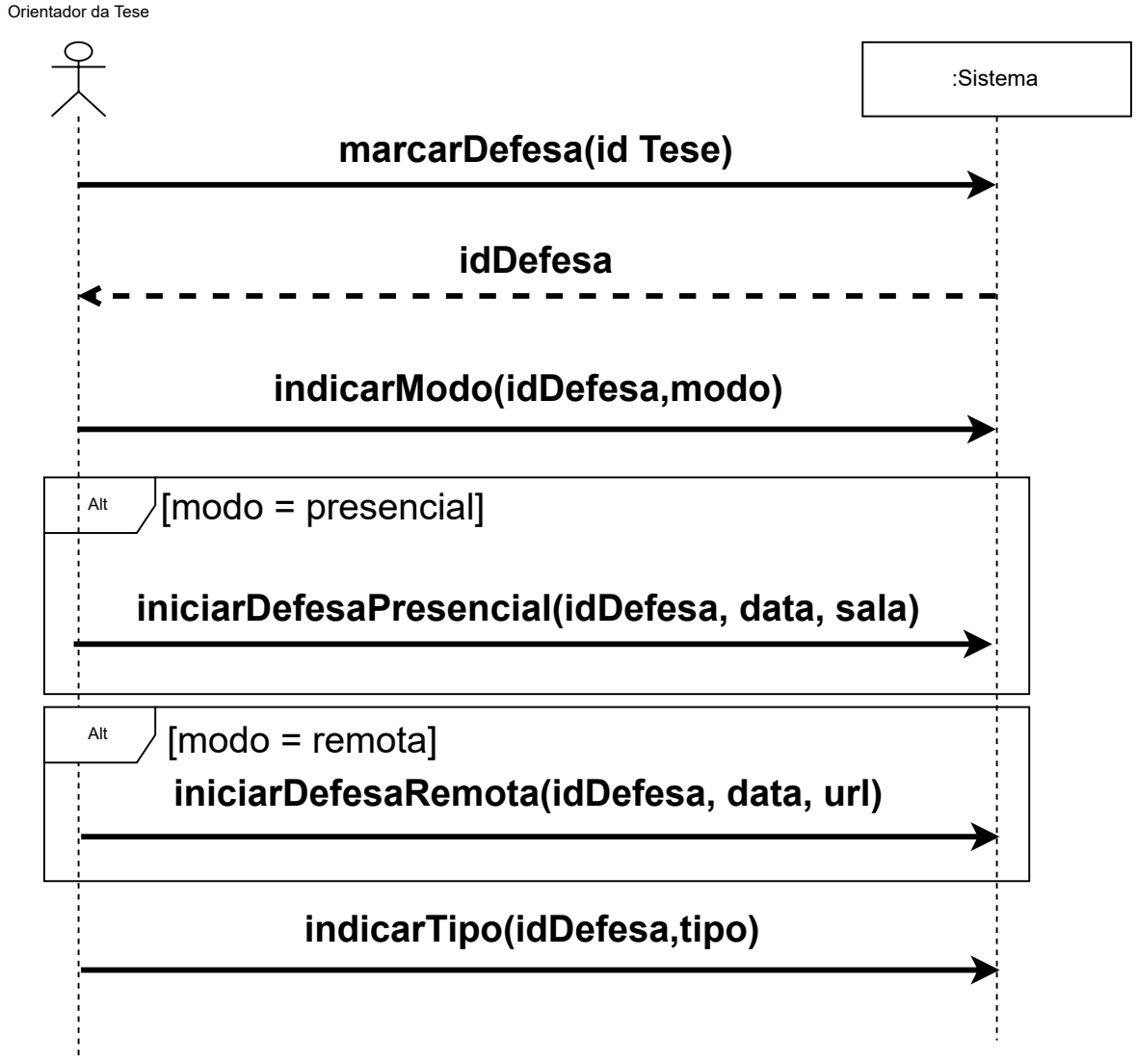
Modelo de Domínio



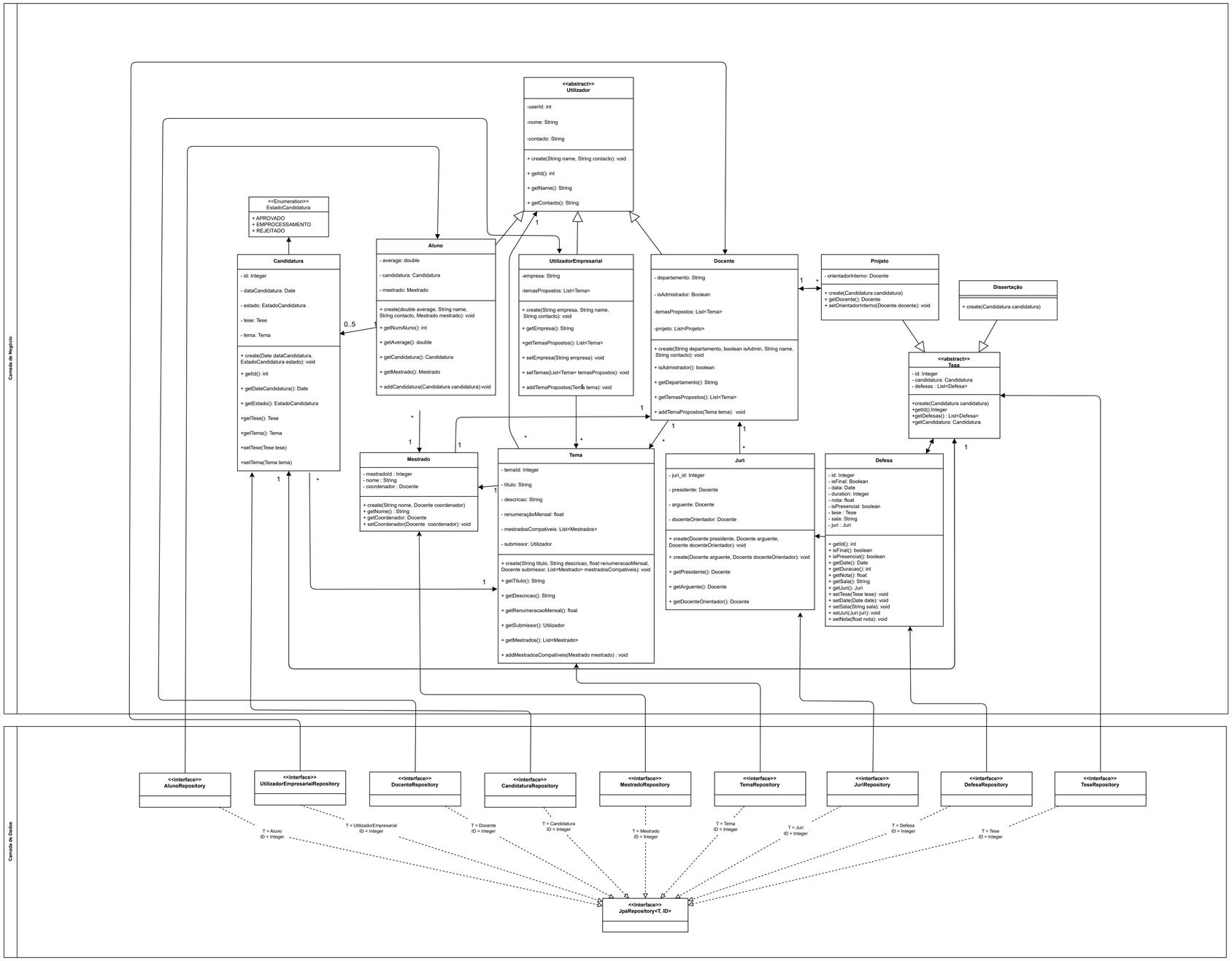
Desenhar o SSD para o caso de uso K.

K. Marcação da Defesa da proposta de tese, por parte do orientador da tese



Nota:
modo refere-se a se a defesa é presencial ou remota
tipo refere-se é uma defesa final ou de proposta (que neste SSD é 'proposta';
Omite-se a escolha de opções de tipo, pois o SSD indica-nos que é uma defesa de proposta

Diagrama de Classes



ThesisMan- Relatório Técnico

Construção de Sistemas de Software

Projeto Prático Fase #1

Daniela Camarinha fc58199

Gonçalo Pinto fc58178

José Brás fc55449

ThesisMan: Gestão de Teses de Mestrado

A Faculdade de Ciências da Universidade de Lisboa, tal como muitas outras universidades que lecionam o segundo ciclo, tem a necessidade de gerir o estado dos mestrados dos seus alunos, desde a submissão de temas por parte de docentes e das empresas, até à marcação das salas a usar.

Descrição da arquitetura em camadas.

A arquitetura em camadas na Engenharia de Software serve para estruturar sistemas complexos em camadas distintas, de forma a gerir essa mesma complexidade dividindo responsabilidades bem definidas pelas camadas. A comunicação entre camadas faz-se de forma que os pedidos fluam para baixo e as respostas a esses pedidos para cima.

Torna-se mais fácil de escrever software pois sabemos onde cada parte de uma funcionalidade deve estar. As camadas da aplicação ThesisMan vão estar divididas da seguinte forma: Apresentação - Lógica de Negócio - Persistência - Acesso aos Dados

Neste 1º Trabalho, interessa-nos falar das Camadas de Lógica de Negócio, Persistência e Acesso aos Dados.

A camada de Negócio utilizará o padrão Domain Model que organiza o código baseado em conceitos do domínio, suas características e associações.

A Camada de Persistência é uma camada intermédia entre o Acesso aos Dados e Lógica de Negócio que serve para abstrair os conceitos OO e os tipos de dados representados na base de dados. Aqui são utilizados conceitos ORM e Java Persistence Annotation para automaticamente criar as relações entre as entidades na Camada de Dados.

Por sua vez, a Camada de Dados é uma camada baseada numa base de dados em PostgreSQL.

Modelo de Domínio.

Desenhar o modelo de domínio que modelo o problema e colocar na pasta docs

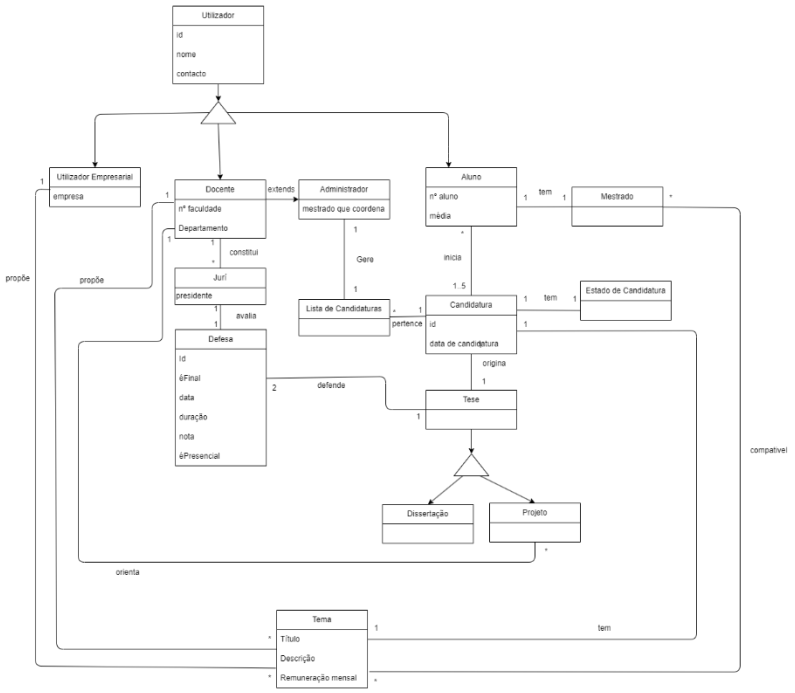


Figura 1: Modelo de domínio

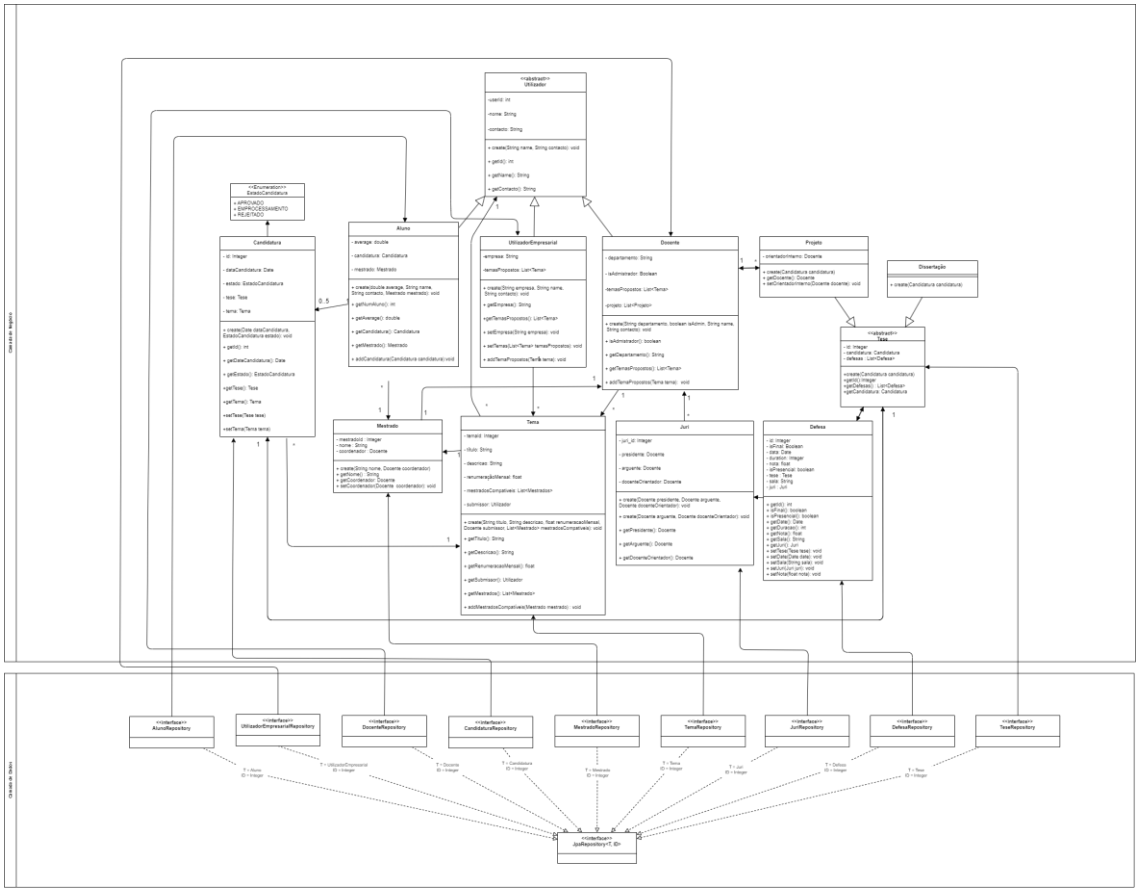
Numa primeira fase do desenvolvimento do modelo de domínio, identificamos todos os conceitos chaves da aplicação e de seguida definimos as entidades que a compõem, respetivos atributos e ligações entre as mesmas de modo a representar estes conceitos num formato de modelo de domínio.

Decidimos usar uma herança para representar os diferentes tipos de utilizadores uma vez que todos estes apresentavam atributos que definimos em comum: id, nome e contacto. Para além dos utilizadores, recorremos ao uso da herança para representar também os dois diferentes tipos de modalidades que uma tese pode tomar: Projeto e Dissertação.

Foi estabelecido para o problema de o aluno escolher até 5 temas para a sua tese que, para cada tema escolhido, é criada uma candidatura independente das anteriores. De modo a representar se o tema foi aceite criamos uma entidade, Estado de Candidatura, com o intuito de apresentar o estado da atribuição dos temas ao aluno: aprovado, em processamento ou rejeitado.

No processo da criação do modelo de domínio, foi definido uma Lista de Candidaturas, para demonstrar os diferentes processos de candidatura que um Administrador gere.

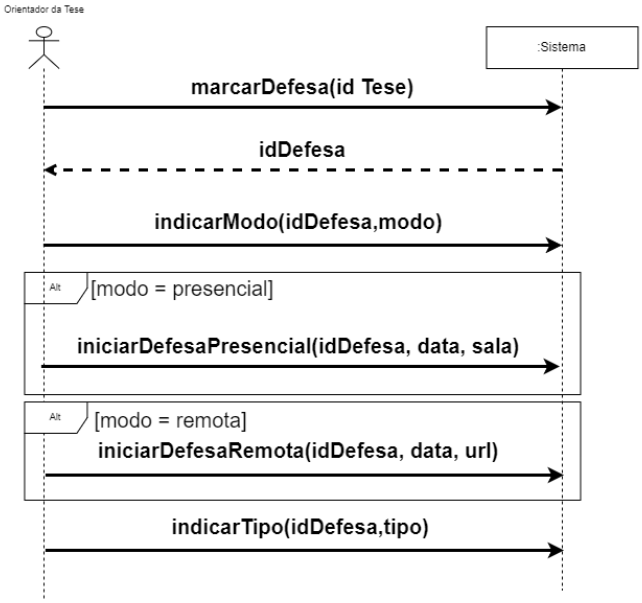
Diagrama de Classes.



Qualidade do SSD pedido.

Desenhar o SSD para o caso de uso K.

K. Marcação da Defesa da proposta de tese, por parte do orientador da tese



Nota:
modo refere-se a se a defesa é presencial ou remota
tipo refere-se é uma defesa final ou de proposta (que neste SSD é 'proposta';
Omite-se a escolha de opções de tipo, pois o SSD indica-nos que é uma defesa de proposta

Mapeamento JPA e justificação das decisões de mapeamento, de acordo com os padrões descritos na aula.

Classe abstrata Tese

A classe abstrata Tese contém a anotação **@Entity** tornando-a numa entidade e por sua vez fazendo com que o JPA crie uma tabela para a mesma.

Utilizamos a anotação **@Inheritance** com o tipo de estratégia **SINGLE_TABLE** de modo a existir apenas uma tabela a representar todas as classes que estendem a classe abstrata Tese. Usamos este tipo de abordagem uma vez que as classes que estendem Tese - Dissertação e Projeto - não apresentam um elevado número de atributos que diferem da sua superclasse e por isso não se justifica a existência de tabelas separadas para as classes Dissertação e Projeto.

A anotação **@DiscriminatorColumn** explicita a existência de uma coluna (type) a indicar o subtipo do objeto - Dissertação ou Projeto - que tem como valor a *String* correspondente ao nome da subclasse usada.

Para chave primária da tabela usamos um Integer (sendo que estamos ao nível universitário, não há necessidade de escalar para valores do tipo Long), que é atribuído por ordem sequencial a todas as entradas da mesma. Utilizamos a estratégia **GenerationType.SEQUENCE** para atingir tal fim.

Como cada tese está associada a uma só candidatura, utilizamos uma associação **@OneToOne** para mostrar esta ligação através de uma nova coluna na tabela representando o id da candidatura em questão. Para propagar possíveis mudanças na candidatura, recorremos ao tipo **CascadeType.MERGE**.

Tendo em conta que a cada tese estão ligadas mais que uma defesa, utilizamos uma associação **@OneToMany** para representar a associação, juntamente com o tipo **CascadeType.MERGE** para propagar qualquer mudança.

Classe Dissertação

A classe Dissertação estende a classe abstrata Tese. Como não existe nenhuma diferença entre as duas, esta não tem nenhum atributo ou método e por isso o seu construtor chama o construtor da sua superclasse.

Classe Projeto

A classe Projeto estende a classe abstrata Tese. Foi acrescentado um atributo “docente” que representa o orientador interno escolhido para auxiliar no Projeto. Tendo em conta que um orientador interno pode orientar mais que uma Tese, utilizamos uma associação **@ManyToOne** para transmitir esta associação. É adicionada uma coluna a tabela da Tese que contem o id do docente em questão.

Classe Mestrado

A classe Mestrado contém a anotação **@Entity** tornando-a numa entidade e por sua vez fazendo com que o JPA crie uma tabela para a mesma.

Para chave primária da tabela usamos um ***Integer*** que é atribuído por ordem sequencial a todas as entradas da mesma. Utilizamos a estratégia **GenerationType.SEQUENCE** para atingir tal fim.

Para representar o nome do mestrado usamos um atributo do tipo ***String*** que contém a anotação **@NonNull** fazendo com que este atributo não possa ser inicializado sem qualquer valor.

Para representar o coordenador do mestrado usamos um atributo do tipo **Docente**. Como cada mestrado contém um coordenador e esse coordenador apenas coordena um mestrado, utilizamos uma associação **@OneToOne** para mostrar esta ligação através de uma nova coluna na tabela, representando o id do docente.

Classe Juri

A Classe Juri contém a anotação **@Entity** tornando-a numa entidade e por sua vez fazendo com que o JPA crie uma tabela para a mesma.

Para a chave primária da tabela usamos um ***Integer*** que é atribuído por sequencial a todas as entradas da mesma. Utilizamos a estratégia **GenerationType.SEQUENCE** para atingir tal fim.

Dependendo do tipo de Defesa que o Juri está a avaliar – Final ou Proposta - este pode ser constituido de forma diferente – para uma Defesa proposta o Juri é adicionalmente constituído por um presidente que dá a nota final da Tese. Este é representado por um Docente, com uma coluna para o seu Id com a relação **@ManyToOne**, sendo que o mesmo Docente pode estar em vários Júris mas um Júri só é constituído por um único presidente.

De seguida o Juri também é sempre constituído por um arguente e por um docenteOrientador, estes têm a mesma relação com a classe Juri - **@ManyToOne –** com a mesma justificação dada para o atributo presidente.

Classe Enumerado EstadoCandidatura

Esta classe representa um enumerado com os possíveis estados de uma candidatura, estes sendo: APROVADO, EMPROCESSAMENTO e REJEITADO.

Classe Defesa

A Classe Defesa contém a anotação **@Entity** tornando-a numa entidade e por sua vez fazendo com que o JPA crie uma tabela para a mesma.

Para a chave primária da tabela usamos um Integer que é atribuído por sequencial a todas as entradas da mesma. Utilizamos a estratégia **GenerationType.SEQUENCE** para atingir tal fim.

De modo a identificar se uma defesa é final ou não, usamos um atributo do tipo ***Boolean***

Para representar se uma defesa é final ou não usamos um atributo do tipo ***Boolean*** que contém a anotação **@NonNull** fazendo com que este atributo não possa ser inicializado sem qualquer valor.

Para representar se uma defesa é presencial ou não usamos um atributo do tipo ***Boolean*** que contém a anotação **@NonNull** fazendo com que este atributo não possa ser inicializado sem qualquer valor.

Para representar a data de uma defesa é presencial ou não usamos um atributo do tipo ***Date***.

Para representar a duração de uma defesa usamos um atributo do tipo ***Integer*** que contém a anotação **@NonNull** fazendo com que este atributo não possa ser inicializado sem qualquer valor.

Para representar a nota de uma defesa usamos um atributo do tipo ***float***.

Para representar a sala onde uma defesa vai ocorrer, usamos um atributo do tipo ***String***.

Tendo em conta que uma defesa é composta por um júri e que um júri pode estar relacionado com mais do que uma defesa, utilizamos a anotação **@ManyToOne** com o tipo **CascadeType.ALL** para representar a associação e propagar qualquer alteração do júri. Na tabela é criada uma coluna com o id do júri, com o nome "juri_id", através da anotação **@JoinColumn**.

Como uma tese tem mais do que uma defesa e uma defesa refere-se a apenas uma tese, decidimos mostrar esta relação através da anotação **@ManyToOne**. Na tabela é criada uma coluna com o id da tese, com o nome "tese_id", através da anotação **@JoinColumn**.

Classe Utilizador

A classe abstrata Utilizador contém a anotação **@Entity** tornando-a numa entidade e por sua vez fazendo com que o JPA crie uma tabela para a mesma. Utilizamos a anotação **@Inheritance** com o tipo de estratégia **InheritanceType.TABLE_PER_CLASS** de modo a existir uma tabela por cada subclasse de Utilizador. Usamos esta abordagem pois, no contexto do problema, cada tipo de utilizador tem um propósito diferente.

As 3 subclasses são: Aluno, Docente e UtilizadorEmpresarial. O aluno é o elemento avaliado que terá de submeter uma candidatura, enquanto o docente orienta e avalia o aluno e o utilizador empresarial define um tema para a tese.

Cada utilizador tem um identificador representado por um número inteiro que é gerado automaticamente com a anotação **@GeneratedValue**. Usamos a strategy **GenerationType.Sequence** para termos uma ordem sequencial sempre que adicionamos um novo utilizador.

Classe Aluno

A classe Aluno é uma entidade que estende Utilizador, é colocada a anotação **@Entity**, para que seja contabilizada aquando da criação da tabela global das classes que estendem Utilizador.

Na altura da candidatura, um aluno tem de ter obrigatoriamente uma média de curso. Isso é atingido com a anotação **@NonNull**.

Na solução proposta no modelo de domínio, o aluno tem até 5 candidaturas, por isso terá a relação **@OneToMany** mapeada pela variável "aluno" na classe Candidatura. Contudo, isto simplesmente diz que um aluno tem várias candidaturas. A limitação de 5 candidaturas terá de ser implementada num método da camada de negócio. Colocamos a estratégia de **fetch = FetchType.EAGER** para automáticamente retornar com a lista de candidaturas as suas entradas, e o atributo **cascade = CascadeType.ALL** que opera sobre as classes filhas sempre que há uma operação nesta classe.

Para além das candidaturas, o aluno tem de pertencer a um Mestrado. Como vários alunos podem pertencer ao mesmo mestrado, usamos a anotação **@ManyToOne**. Usamos o **@JoinColumn** para guardar o id do mestrado com o atributo **nullable = false**, pois, cada aluno tem de pertencer a um mestrado.

Classe Docente

A classe Docente é uma entidade que estende Utilizador, é colocada a anotação **@Entity**, para que seja contabilizada aquando da criação da tabela global das classes que estendem Utilizador.

Um docente tem de pertencer obrigatoriamente a um departamento e o sistema terá de saber se ele tem permissões de administrador, para que esse valor seja obrigatório de atribuir, utilizamos a anotação **@NonNull**.

Um docente pode propor vários temas, por isso há uma relação **@OneToMany** entre docente e temas. Cada Tema terá o atributo "submissor" do tipo Utilizador, por isso usamos o atributo **mappedBy = "submissor"**.

Um docente pode orientar vários projetos, por isso há uma relação **@OneToMany** entre docente e temas. Cada Tema terá o atributo "docente" do tipo Docente, por isso usamos o atributo **mappedBy = "docente"**.

Utilizador Empresarial

A classe Utilizador Empresarial é uma entidade que estende Utilizador, é colocada a anotação **@Entity**, para que seja contabilizada aquando da criação da tabela global das classes que estendem Utilizador.

Um utilizador empresarial tem de pertencer obrigatoriamente a uma empresa, para que esse valor seja obrigatório de atribuir, utilizamos a anotação **@NonNull**.

Um utilizador empresarial pode propor vários temas, por isso há uma relação **@OneToMany** entre utilizadores empresariais e temas. Cada Tema terá o atributo "submissor" do tipo Utilizador, por isso usamos o atributo **mappedBy = "submissor"**.

Classe Candidatura

A classe Candidatura representa as Candidaturas efetuadas por cada Aluno.

O seu Id é do tipo ***Integer***, gerado a partir de uma sequência com a estratégia **@GenerationType.SEQUENCE.** A Data de candidatura é representada com o tipo ***Date***.

O Estado da Candidatura é representado pelo enumerado do tipo **EstadoCandidatura**

A Candidatura refere-se a uma Tese, sendo assim tem uma associação **@OneToOne**, pois uma candidatura refere-se a uma única tese, e uma Tese corresponde a só uma Candidatura.

A Candidatura é efetuada por um Aluno, sendo assim, o aluno pode ter mais que uma Candidatura (limite de 5 é verificado na camada de lógica de negócio). Assim sendo, a anotação utilizada é de **@ManyToOne**

Classe Tema

A classe Tema corresponde aos temas propostos pelos Docentes e pelos Utilizadores Empresarias.

O Tema é representado pelo seu *Título*, uma ***String***; descrição, também um ***String*** e pela remuneração Mensal, um número do tipo ***float***. Um tema tem que ter compatibilidade com pelo menos 1 mestrado (associação **@ManyToMany)**, para ser escolhido pelos alunos desse mesmo Mestrado (verificação passada para a camada de lógica de negócio). Para isso é criada uma tabela separada para fazer a associação entre o tema e os seus mestrados compatíveis. Isto é possível com a anotação **@JoinTable** onde referimos as colunas de cada uma das entidades que queremos associar. Por último, é também necessário representar o submissor do Tema, que pode ser um Utilizador com as subclasses – Docente ou Utilizador Empresarial- cada um deste Utilizadores pode propôr mais que um tema, e por isso a anotação utilizada é de **@ManyToOne.**