6/13/2016

Tiago F CARDOSO/Wellington Alves/Fernando Medina

FACULDADES oPET

Projeto Interdisciplinar

HELP DESK SYSTEM

**FACULDADES OPET**

**PROJETO INTERDISCIPLINAR**

**CURITIBA 2016**

**Fernando Medina**

**Tiago Fabricio Cardoso**

**Wellington Alves**

***HELP DESK SYSTEM***

Este projeto de tem o objetivo de integrar as disciplinas Engenharia de Software, Tópicos avançados Banco de Dados, Desenvolvimento de Sistemas Orientados a Objetos e Web apresentadas no curso de Analise e Desenvolvimento de Sistemas 2º Período – TDS152.

**Curitiba 2016**

Sumário

**Objetivos2**

**Objetivos Específicos2**

**Especificação3**

**Modelagem de Banco de Banco de Dados6**

**Padrão MVC12**

**Estrutura do Projeto13**

**Modelagem do Sistema14**

**Chamadas do Sistema14**

**Diagrama de Use Case15**

**Diagrama de Classe17**

**Diagrama de Atividades19**

**Diagrama de Sequência20**

**Objetivos**

Desenvolver um sistema para auxiliar na organização do atendimento de Help Desk do departamento de TI, onde podem ser gerados chamados (Ticket) para atendimento a demandas de suporte técnico de informática, sendo elas, Hardware, Software, Redes, etc.

1. Cadastro de Funcionários/usuários

* NOME
* DEPARTAMENTO
* SEXO
* EMAIL
* LOGIN
* SENHA

1. Cadastro de Analista

* NOME
* LOGIN
* SENHA
* PERMISSÃO DE ACESSO

1. Gerar/alterar/excluir TICKET

* USUARIO
* TIPO
* CATEGORIA
* ASSUNTO
* DESCRIÇÃO
* SOLUÇÃO
* NIVEL DO ANALISTA
* ANALISTA
* DATA INICIAL
* DATA FINAL
* STATUS

**Objetivos Específicos**

Compreender e aplicar os conceitos fundamentais do **Paradigma Orientado a Objeto – POO**:

1. Encapsulamento

1.1. Classes e Objetos

1.2. Overload – Sobrecarga de Métodos

2. Herança

2.1. Classe Abstrata

2.2. Interface

2.3. Override – Sobrescrita de Métodos

3. Polimorfismo

3.1. Método Polimórfico

Compreender e aplicar os conceitos de **Padrões de Projeto Orientados a Objetos – PPOO.**

Desenvolver o sistema de acordo com o padrão de projeto **Model, Controller e View – MVC**.

Acrescentar a camada de **Acesso a Dados**, a camada de **Values Objects – VO** e a camada **Utilitária**.

Compreender e aplicar os conceitos de **Java Database Connectivity – JDBC**, conectividade com Banco de Dados do **Java**.

Conhecer e utilizar o **Sistema de Tratamento de Exceção do Java**.

**Especificação**

Desenvolver um sistema na **Linguagem de Programação Java** utilizando o **Ambiente de Desenvolvimento Integrado (IDE) eclipse**.

Uma classe deve ser definida de acordo com as especificações denominada **POJO – Plain Old Java Objects (Objetos Comuns do Java)**:

1. Nome da classe

1.1. A primeira palavra deve ser escrita com a primeira letra em maiúscula.

1.2. Se houver mais palavras, as demais palavras devem ser escritas com a primeira letra em maiúscula.

1.3. Todas as palavras são escritas por extenso para documentar o propósito da classe.

1.4. Não existe nenhum caractere separando as palavras, nem mesmo o caractere sublinhado.

2. Estrutura da classe

2.1. Variáveis de instância – variáveis que pertencem à classe. A primeira palavra deve ser escrita em letra minúscula e se houver mais palavras, as demais palavras devem ser escritas com a primeira letra em maiúscula. Todas as palavras são escritas por extenso para documentar o propósito da variável. Não existe nenhum caractere separando as palavras, nem mesmo o caractere sublinhado.

2.2. Métodos da classe. Todo e qualquer método da classe deve ser declarado com a primeira letra em minúscula e se houver mais palavras, as demais palavras devem ser escritas com a primeira letra em maiúscula. Todas as palavras são escritas por extenso para documentar o propósito do método. Não existe nenhum caractere separando as palavras, nem mesmo o caractere sublinhado. Os métodos de uma classe são definidos e classificados pelos seguintes tipos de métodos:

2.2.1. Função construtora da classe – utilizado para auxiliar a construção de um objeto da classe.

2.2.2. Métodos de acesso – **getters** e **setters**. Formam a interface entre as variáveis da classe e o usuário da classe.

2.2.3. Operações da classe – métodos para operar os atributos da classe.

As classes que fazem parte da regra de negócio devem implementar as seguintes funcionalidades:

1. Redefinir o método **toString()** da classe **Object** para mostrar o estado da classe.

2. Redefinir o método **clone()** da classe **Object** para permitir a clonagem de um objeto da classe.

3. Implementar a interface **Serializable** do pacote **java.io** que gera a variável ***serialVersionUID*** que representa um valor identificador da classe.

4. Implementar os métodos **hashCode()** e **equals()** que geram um valor identificador para cada objeto da classe e um método de comparação entre objetos da classe, respectivamente.

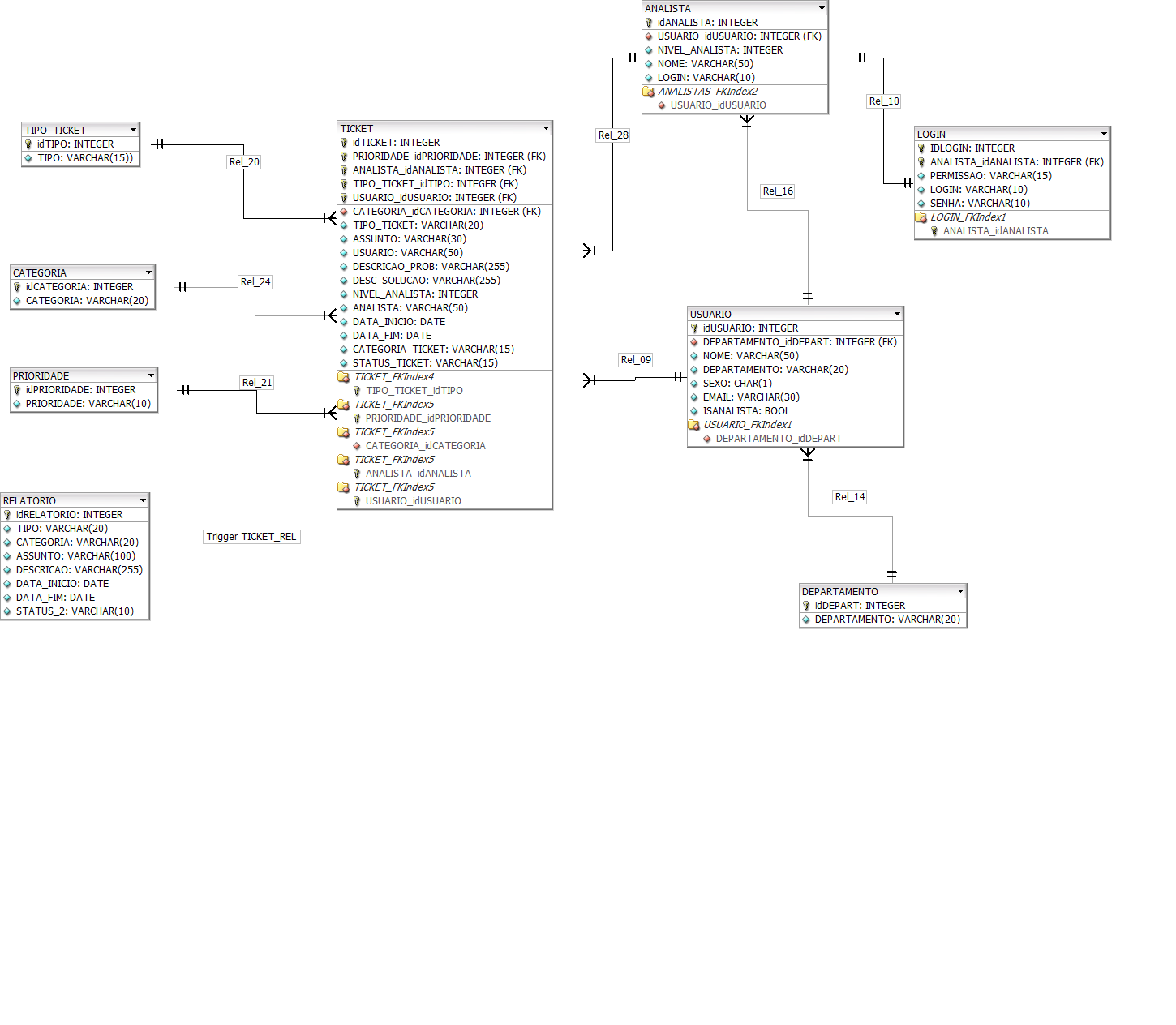
O sistema deve implementar a **lógica da apresentação** e fazer a **validação de entrada de dados**.

O sistema deve **persistir os dados no Banco de Dados Oracle XE**. O sistema deve gerar um script para o Banco de Dados criando os objetos de banco de dados.

**Modelagem de Banco de Dados**

O modelo de Banco de dados mostra que Ticket é a principal entidade que possui relacionamento com as entidades Tipo, Categoria, Prioridade, Usuário, Analista que também se relaciona com Usuário.

O SGBD utilizado para esse projeto é o Oracle11g.



**Script SQL**

CREATE TABLE ANALISTA (

idANALISTA INTEGER NOT NULL,

idUSUARIO INTEGER NOT NULL,

NIVEL\_ANALISTA INTEGER not NULL,

NOME VARCHAR2(50) not NULL,

LOGIN VARCHAR2(10),

SENHA VARCHAR2(15),

PERMISSAO VARCHAR2(20),

PRIMARY KEY(idANALISTA));

CREATE TABLE CATEGORIA (

idCATEGORIA INTEGER NOT NULL ,

CATEGORIA VARCHAR2(20) NULL,

PRIMARY KEY(idCATEGORIA));

COMMIT;

CREATE TABLE DEPARTAMENTO (

idDEPART INTEGER NOT NULL ,

DEPARTAMENTO VARCHAR2(20) NULL,

PRIMARY KEY(idDEPART));

COMMIT;

CREATE TABLE PRIORIDADE (

idPRIORIDADE INTEGER NOT NULL ,

PRIORIDADE VARCHAR2(10) NULL,

PRIMARY KEY(idPRIORIDADE));

COMMIT;

CREATE TABLE TICKET (

idTICKET INTEGER NOT NULL ,

idTIPO INTEGER NOT NULL,

idPRIORIDADE INTEGER NOT NULL,

idUSUARIO INTEGER NOT NULL,

idANALISTA INTEGER NOT NULL,

idCATEGORIA INTEGER NOT NULL,

USUARIO VARCHAR2(50),

TIPO VARCHAR2(20),

CATEGORIA VARCHAR2(15),

ASSUNTO VARCHAR2(100),

DESCRICAO VARCHAR2(255),

SOLUCAO VARCHAR2(255),

NIVEL\_ANALISTA INTEGER,

ANALISTA VARCHAR2(50),

DATA\_INICIO DATE,

DATA\_FIM DATE,

STATUS VARCHAR2(15),

PRIMARY KEY(idTICKET));

COMMIT;

CREATE TABLE TIPO\_TICKET (

idTIPO INTEGER NOT NULL ,

TIPO VARCHAR2(15),

PRIMARY KEY(idTIPO));

COMMIT;

CREATE TABLE USUARIO (

idUSUARIO INTEGER NOT NULL ,

idDEPART INTEGER NOT NULL,

PERMISSAO VARCHAR2(20) NOT NULL,

NOME VARCHAR2(50) NOT NULL,

DEPARTAMENTO VARCHAR2(20) NOT NULL,

SEXO VARCHAR2(1),

EMAIL VARCHAR2(50),

ISANALISTA CHAR(1) NOT NULL,

NIVEL\_ANALISTA INTEGER,

LOGIN VARCHAR2(15),

SENHA VARCHAR2(10),

PRIMARY KEY(idUSUARIO));

COMMIT;

CREATE TABLE LOGIN(

IDLOGIN INTEGER,

IDANALISTA INTEGER,

LOGIN VARCHAR(10),

SENHA VARCHAR(10),

PERMISSAO VARCHAR2(20),

PRIMARY KEY (IDLOGIN));

COMMIT;

CREATE TABLE RELATORIO(

IDRELATORIO INTEGER ,

IDTICKET INTEGER ,

TIPO VARCHAR2(20) ,

CATEGORIA VARCHAR2(20),

ASSUNTO VARCHAR2(100),

DESCRICAO VARCHAR2(255) ,

SOLUCAO VARCHAR2(255) ,

DATA\_INICIO DATE,

DATA\_FIM DATE ,

STATUS VARCHAR2(10));

-- ALTER TABLES PARA ADICIONAR AS FOREING KEYS

ALTER TABLE TICKET

ADD CONSTRAINT FK\_TIPOO

FOREIGN KEY (idTIPO)

REFERENCES TIPO\_TICKET(idTIPO)

ON DELETE CASCADE ;

ALTER TABLE TICKET

ADD CONSTRAINT FK\_PRIORIDADE

FOREIGN KEY (idPRIORIDADE)

REFERENCES PRIORIDADE(idPRIORIDADE)

ON DELETE CASCADE ;

ALTER TABLE TICKET

ADD CONSTRAINT FK\_USUARIO

FOREIGN KEY (IDUSUARIO)

REFERENCES USUARIO(IDUSUARIO)

ON DELETE CASCADE ;

ALTER TABLE TICKET

ADD CONSTRAINT FK\_ANALISTA

FOREIGN KEY (IDANALISTA)

REFERENCES ANALISTA(IDANALISTA)

ON DELETE CASCADE ;

ALTER TABLE TICKET

ADD CONSTRAINT FK\_CATEGORIA

FOREIGN KEY (idCATEGORIA)

REFERENCES CATEGORIA(idCATEGORIA)

ON DELETE CASCADE;

ALTER TABLE LOGIN

ADD CONSTRAINT FK\_LOGIN1

FOREIGN KEY (IDANALISTA)

REFERENCES ANALISTA(IDANALISTA)

ON DELETE CASCADE ;

ALTER TABLE ANALISTA

ADD CONSTRAINT FK\_USER\_ANALISTA

FOREIGN KEY (IDUSUARIO)

REFERENCES USUARIO(IDUSUARIO)

ON DELETE CASCADE ;

COMMIT;

--SEQUENCES

CREATE SEQUENCE CONT\_IDANALISTA NOCACHE;

CREATE SEQUENCE CONT\_IDLOGIN NOCACHE;

CREATE SEQUENCE CONT\_IDTICKET NOCACHE;

create sequence CONT\_IDUSUARIO

increment by 1

start with 1

maxvalue 999

nocycle;

COMMIT;

--TRIGGER PARA SALVAR NA TABELA RELATORIO

create or replace TRIGGER TICKET\_REL AFTER INSERT ON TICKET

REFERENCING NEW AS NEW OLD AS OLD

FOR EACH ROW

BEGIN

INSERT INTO RELATORIO (IDTICKET,TIPO, CATEGORIA, ASSUNTO, DESCRICAO,SOLUCAO, DATA\_INICIO, DATA\_FIM, STATUS)

VALUES (:NEW.IDTICKET,:NEW.TIPO, :NEW.CATEGORIA, :NEW.ASSUNTO, :NEW.DESCRICAO, :NEW.SOLUCAO , :NEW.DATA\_INICIO, :NEW.DATA\_FIM, :NEW.STATUS);

END;

--EXECUTE UMA TRIGGER DE CADA VEZ SE NÃO AO CRIAR A PRIMEIRA TRIGGER A TRIGGER SEGUINTE IRA PARA DENTRO DA MESMA OCASIONANDO O ERRO

create or replace TRIGGER UPDATE\_REL AFTER UPDATE ON TICKET

REFERENCING NEW AS NEW OLD AS OLD

FOR EACH ROW

BEGIN

UPDATE RELATORIO SET TIPO = :NEW.TIPO, CATEGORIA = :NEW.CATEGORIA, ASSUNTO = :NEW.ASSUNTO,DESCRICAO = :NEW.DESCRICAO,SOLUCAO = :NEW.SOLUCAO,DATA\_FIM = :NEW.DATA\_FIM ,STATUS = :NEW.STATUS WHERE IDTICKET = :NEW.IDTICKET;

END;

--EXECUTE UMA TRIGGER DE CADA VEZ SE NÃO AO CRIAR A PRIMEIRA TRIGGER A TRIGGER SEGUINTE IRA PARA DENTRO DA MESMA OCASIONANDO O ERRO

--EXECUTE UMA TRIGGER DE CADA VEZ SE NÃO AO CRIAR A PRIMEIRA TRIGGER A TRIGGER SEGUINTE IRA PARA DENTRO DA MESMA OCASIONANDO O ERRO

CREATE OR REPLACE TRIGGER ADD\_ANALISTA AFTER INSERT ON USUARIO

REFERENCING NEW AS NEW OLD AS OLD

FOR EACH ROW

BEGIN

IF :NEW.ISANALISTA = 'S' THEN

INSERT INTO ANALISTA (IDANALISTA,IDUSUARIO,NIVEL\_ANALISTA,NOME,LOGIN,SENHA,PERMISSAO)

VALUES(CONT\_IDANALISTA.NEXTVAL,:NEW.IDUSUARIO,:NEW.NIVEL\_ANALISTA,:NEW.NOME,:NEW.LOGIN,:NEW.SENHA,:NEW.PERMISSAO);

end if;

END ADD\_ANALISTA;

--EXECUTE UMA TRIGGER DE CADA VEZ SE NÃO AO CRIAR A PRIMEIRA TRIGGER A TRIGGER SEGUINTE IRA PARA DENTRO DA MESMA OCASIONANDO O ERRO

CREATE OR REPLACE TRIGGER ADD\_LOGIN AFTER INSERT ON ANALISTA

REFERENCING NEW AS NEW OLD AS OLD

FOR EACH ROW

BEGIN

INSERT INTO LOGIN (IDLOGIN,IDANALISTA,LOGIN,SENHA,PERMISSAO)

VALUES(CONT\_IDLOGIN.NEXTVAL,:NEW.IDANALISTA,:NEW.LOGIN,:NEW.SENHA,:NEW.PERMISSAO);

END ADD\_LOGIN;

--EXECUTE A INSTRUÇÃO DA TRIGGER SOZINHA

--EXECUTE A INSTRUÇÃO DA TRIGGER SOZINHA

--EXECUTE A INSTRUÇÃO DA TRIGGER SOZINHA

create or replace TRIGGER UPDATE\_ANALISTA AFTER update ON USUARIO

REFERENCING NEW AS NEW OLD AS OLD

FOR EACH ROW

BEGIN

UPDATE ANALISTA SET NIVEL\_ANALISTA = :NEW.NIVEL\_ANALISTA, LOGIN = :NEW.LOGIN, SENHA = :NEW.SENHA ,PERMISSAO = :NEW.PERMISSAO WHERE IDUSUARIO = :NEW.IDUSUARIO;

END UPDATE\_ANALISTA;

--EXECUTE A INSTRUÇÃO DA TRIGGER SOZINHA

--EXECUTE A INSTRUÇÃO DA TRIGGER SOZINHA

--EXECUTE A INSTRUÇÃO DA TRIGGER SOZINHA

create or replace TRIGGER UPDATE\_LOGIN AFTER update ON ANALISTA

REFERENCING NEW AS NEW OLD AS OLD

FOR EACH ROW

BEGIN

UPDATE LOGIN SET LOGIN = :NEW.LOGIN, SENHA = :NEW.SENHA ,PERMISSAO = :NEW.PERMISSAO WHERE IDANALISTA = :NEW.IDANALISTA;

END UPDATE\_LOGIN;

-- EXECUTE A TRIGGER ACIMA SOZINHA SE NÃO VAI DAR ERRO

-- STORED PROCEDURES

create or replace PROCEDURE GERA\_REL

(

status\_rel IN VARCHAR2,

CURSOR\_OUT OUT sys\_refcursor

)

IS

BEGIN

OPEN CURSOR\_OUT FOR

SELECT TIPO, CATEGORIA, ASSUNTO, DESCRICAO ,SOLUCAO, DATA\_INICIO , DATA\_FIM FROM RELATORIO WHERE STATUS = status\_rel ;

END;

-- função

create or replace FUNCTION RETORNA\_DIAS

RETURN sys\_refcursor

AS

MYCURSOR sys\_refcursor;

BEGIN

OPEN MYCURSOR FOR

select IDTICKET,USUARIO,TIPO,CATEGORIA,ASSUNTO,DESCRICAO,DATA\_INICIO, sysdate - DATA\_INICIO as DIFERENCA\_DATAS from TICKET WHERE STATUS = 'ABERTO' ORDER BY DIFERENCA\_DATAS ASC;

RETURN MYCURSOR;

END;

**Padrão MVC**

O Model**, View e Controller – MVC** determina que o desenvolvimento do sistema seja realizado **separando as responsabilidades do sistema** em três camadas: **Model**, **View** e **Controller.**

A camada de **Model** é responsável pela implementação da **Regra de Negócio do Sistema**. Regra de negócio significa a principal atividade ou objetivo para a qual se está implementando o sistema.

A camada **View** tem como responsabilidade implementar a **interface entre o usuário do sistema e o sistema.** A forma na qual o usuário do sistema utiliza o sistema é estabelecida por meio da **View**, ou seja, o usuário tem acesso somente as classes de fronteira as quais implementam a **View** e não deve interagir diretamente com o Model do sistema.

A camada de apresentação implementar a **lógica da apresentação**, que tem como objetivo principal verificar a consistência dos dados. Por exemplo, verificar se um valor de preenchimento obrigatório foi informado. A camada de apresentação também realiza a validação de entrada de dados. Validar a entrada de dados consiste em duas ações. A primeira de refere a validar o tipo do dado. A segunda consistem em validar o valor informado. A camada de apresentação não deve permitir que dados inválidos ou inconsistentes sejam enviados para o sistema.

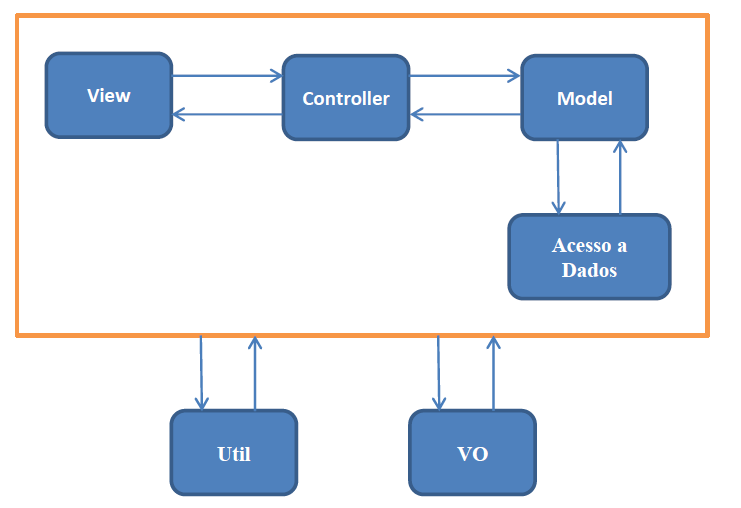
A camada de **Controller** tem a responsabilidade de implementar o **controle do fluxo de execução do sistema**, gerenciando as solicitações e respostas do usuário.

O padrão **MVC** determina como as camadas devem interagir entre si. A camada **View** interagem somente com a camada **Controller**. A camada de **Controller** interagem com a camada **View** e com a camada **Model**. A camada **Model** interage somente com a camada **View**. A camada **View** não tem acesso a camada **Model**.

A camada **Model**, para implementar a regra de negócio do sistema, precisa de dados. A camada de **Acesso de Dados** tem como responsabilidade fornecer os dados para o sistema. A camada **Model** solicita uma cópia dos dados para a camada de **Acesso de Dados**.

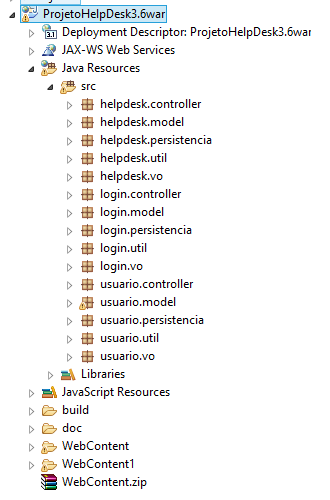
Um sistema **Orientado a Objetos** é formado por um **conjunto de objetos** que interagem entre si e como um todo formam a funcionalidade do sistema. No desenvolvimento de sistemas existem diversos objetos que fazem parte da regra de negócio e são utilizados por todo o sistema. Para organizar a utilização destes **objetos específicos** é criada uma camada denominada de **Value Objects – VO (Objetos de Valores)** que fornece as funcionalidades relacionadas à regra de negócio para todo o sistema.

A camada **Util** implementa as **classes utilitárias** que irão fornecer funcionalidades de implementação para o sistema como um todo.



# Estrutura do Projeto

A estrutura do projeto será desenvolvida utilizando o Padrão de Projeto Orientado a Objetos (PPOO) – MVC (Model, View e Controle).

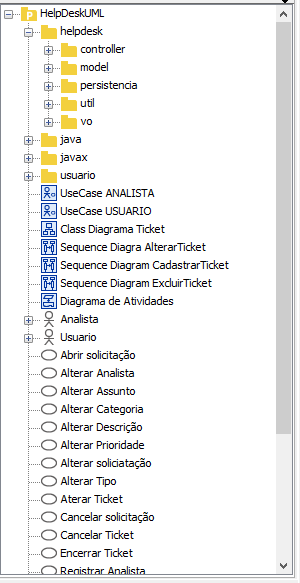


# Modelagem do Sistema

O desenvolvimento de um sistema Orientado a Objetos é feito por meio de criação de classes. Em torno de aproximadamente 70% do tempo de desenvolvimento do sistema é gasto na elaboração das classes para o sistema.

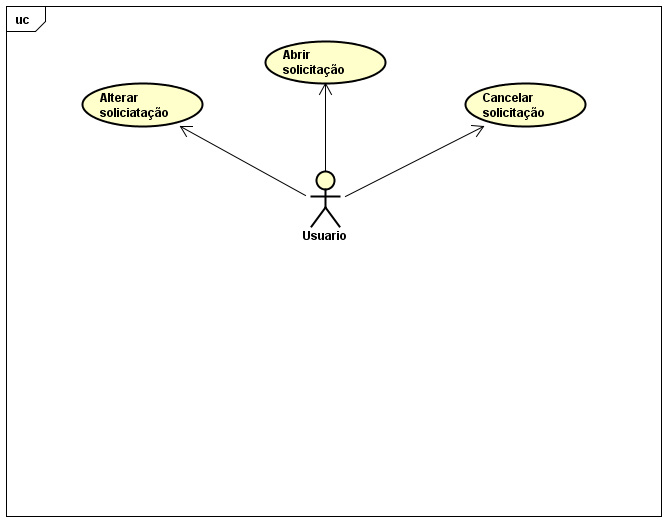
## **Camadas do Sistema**

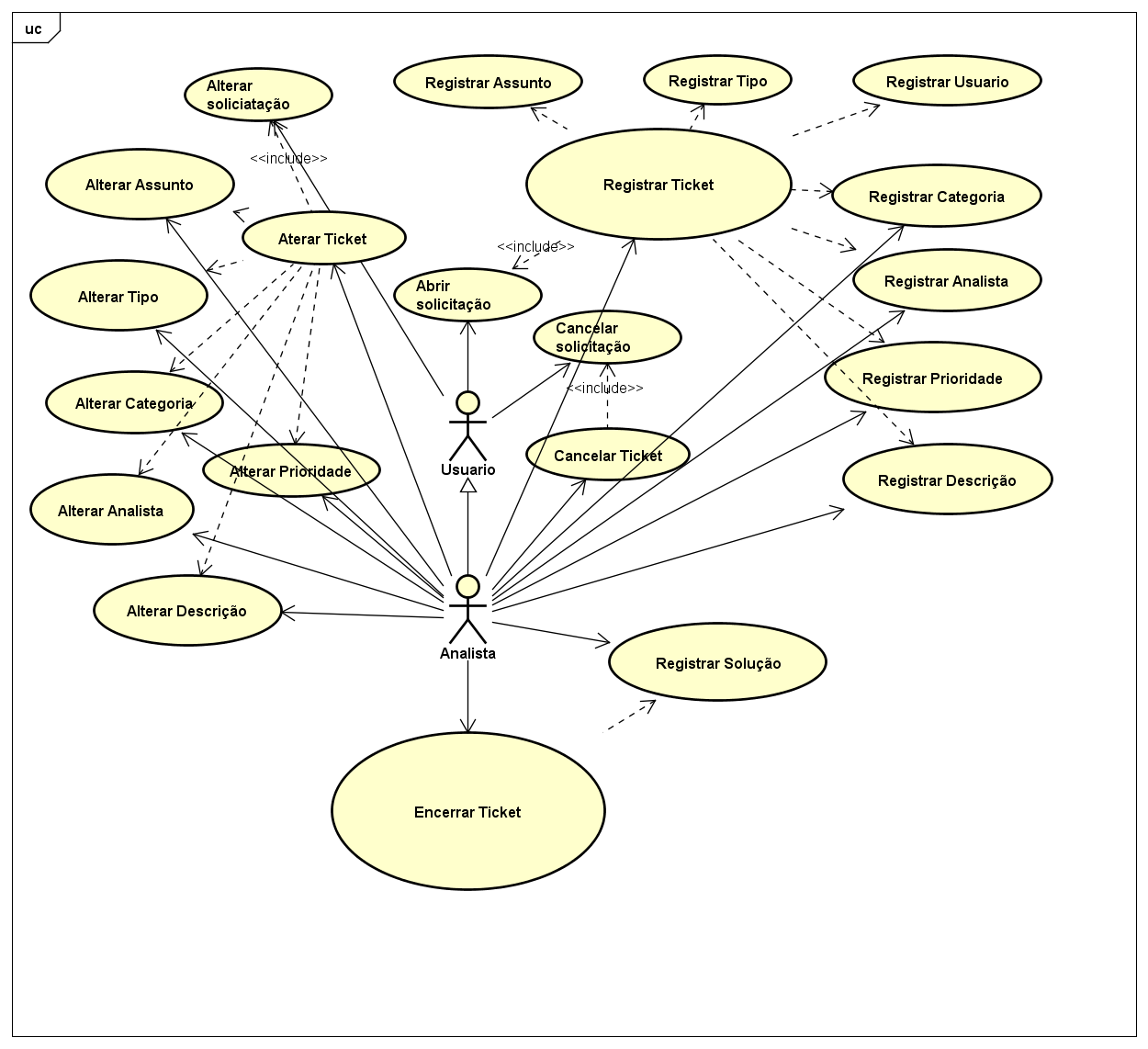
Para este tipo de desenvolvimento é necessário realizar a modelagem do sistema para que seja possível a sua compreensão e elaboração. Utilizando a linguagem de modelagem Unified Modeling Language – UML (Linguagem Unificada de Modelagem), o sistema é modelado de acordo com as camadas do MVC adicionando as camadas de Value Objects, Acesso a Dados e Util.



## **Diagrama de Use Case**

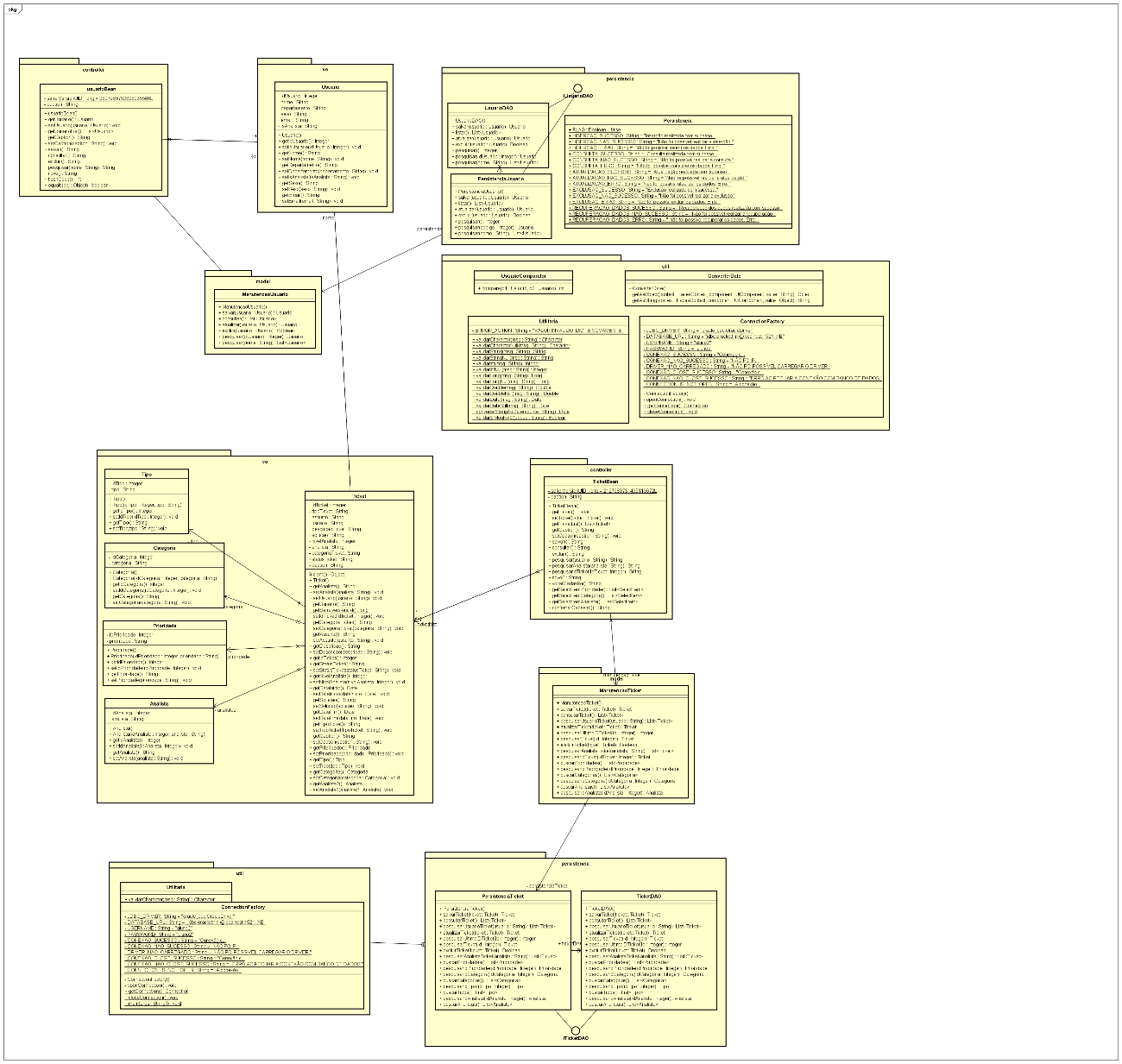
O diagrama de Use Case é utilizado para capturar os requisitos do sistema. Para o sistema Help Desk as funcionalidades básicas são o cadastro de Funcionário, Analista criação de Tickets o qual se inicia a partir da solicitação de um funcionário feita a um analista que por sua vez deverá registrar um ticket (Solicitação, Duvida ou Problema) diretamente no sistema Help Desk.

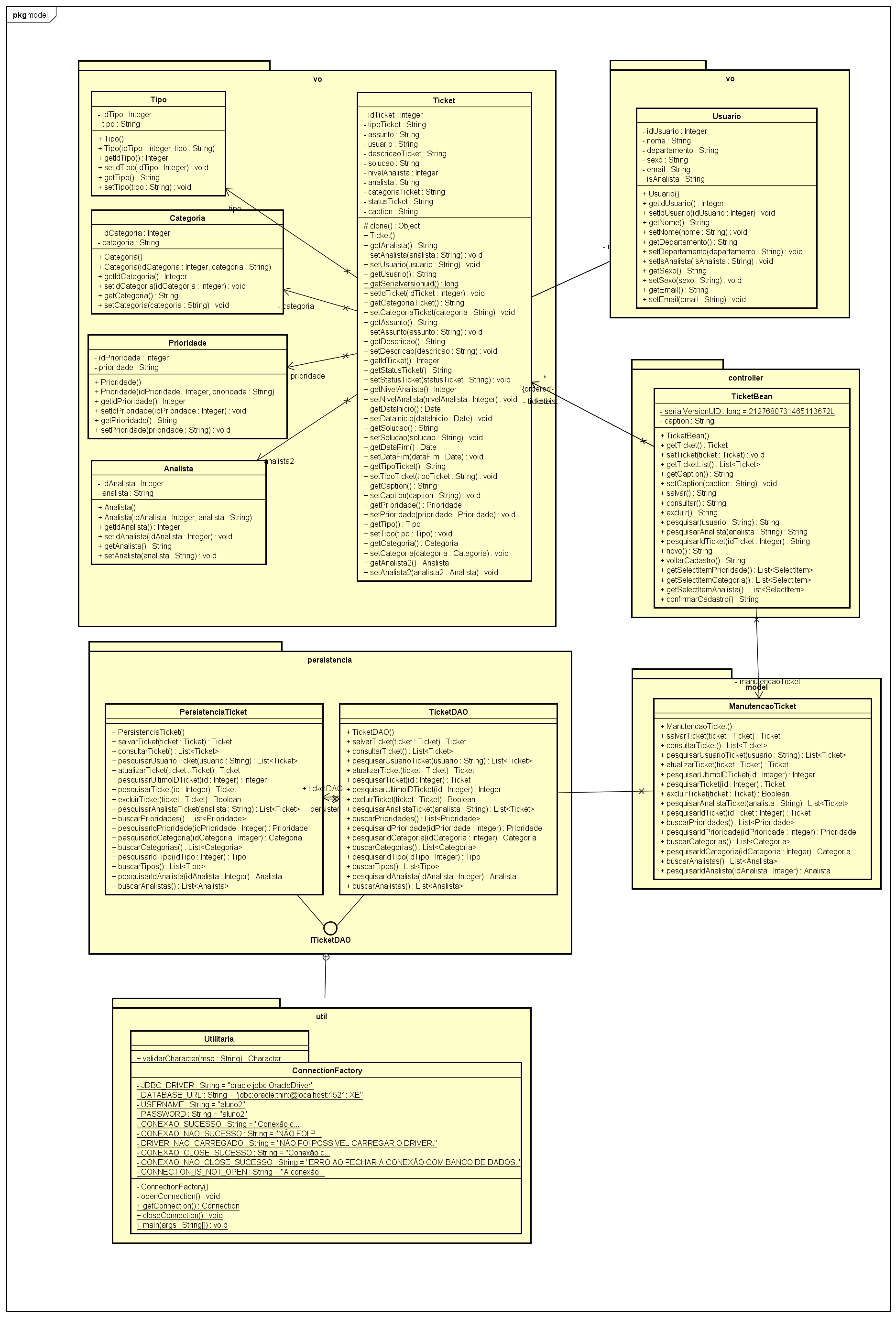




## **Diagrama de Classes**

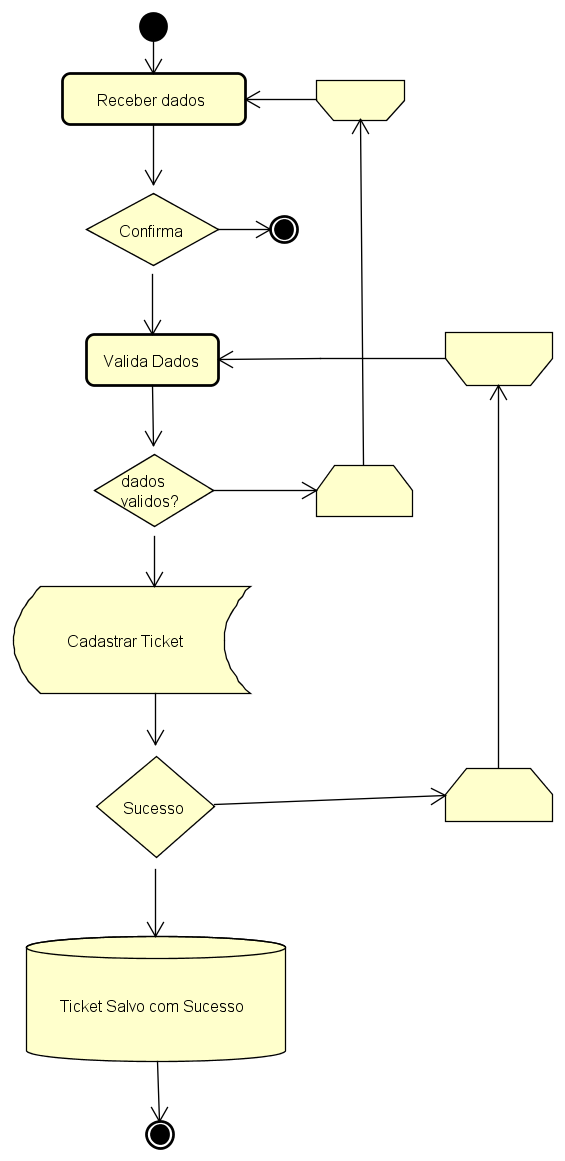
## O diagrama de classe é utilizado para realizar a modelagem do sistema mostrando as classes envolvidas. Nesta representação é possível mostrar o relacionamento entre as classes, assim pode se ter ideia de como o comportamento interno do sistema irá ocorrer. Abaixo está representado o fluxo completo e o fluxo para cadastro de um Ticket.





## **Diagrama de Atividades**

## O diagrama de Atividades é utilizado para mostrar o fluxo das atividades que o sistema e ou o negócio principal. Neste fluxo o negócio principal está sendo demonstrado através do Fluxograma abaixo.



## **Diagrama de Sequência**

## O diagrama de Sequência é utilizado para demonstrar a sequência exata das atividades que o sistema executa durante seu processamento. Neste diagrama o negócio principal está sendo demonstrado através do abaixo.

