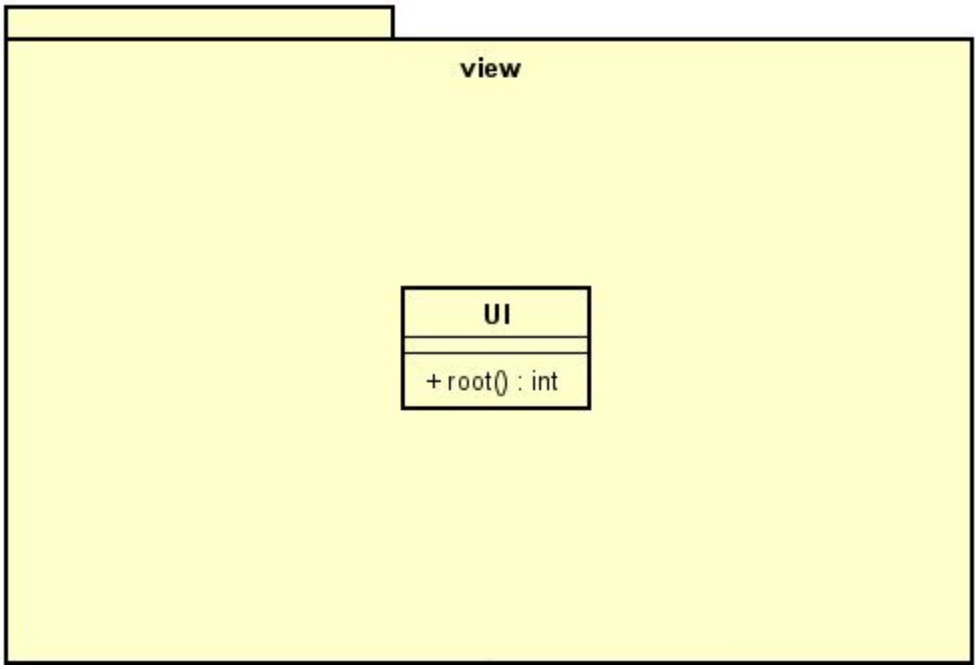
Sistemas de Receitas

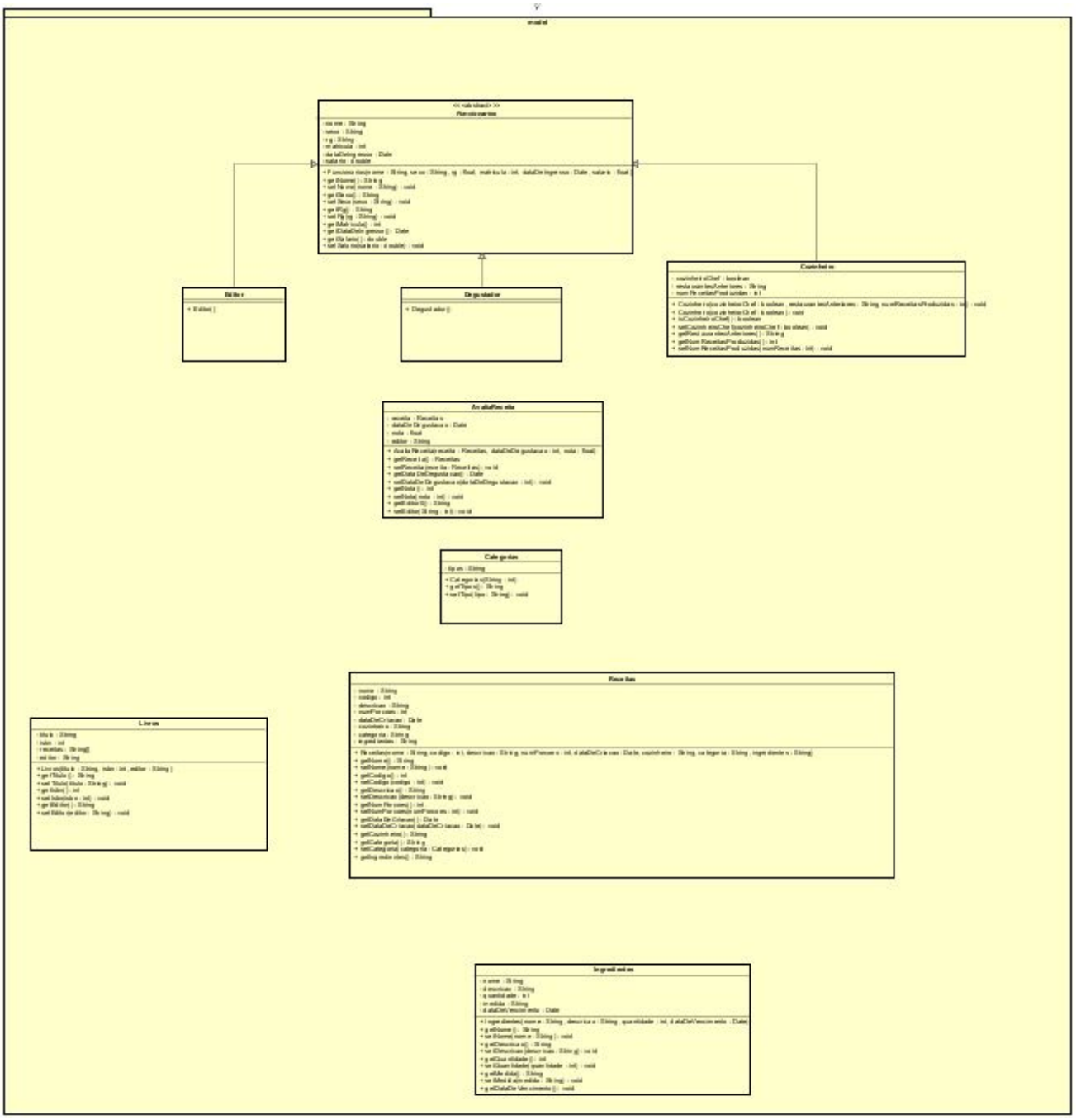
Pacotes e Padrões de Projeto

Para a solução do problema, foi adotada a arquitetura *MVC* (Model-View-Controller) a fim de isolar a lógica de negócio com a interface do usuário, promover a reutilização, maior legibilidade e manutenção no código.

Na primeira camada ou Pacote *view*, é onde está localizada a classe UI, responsável por gerar o menu iniciar possibilitando a interação com o usuário de forma dinâmica.



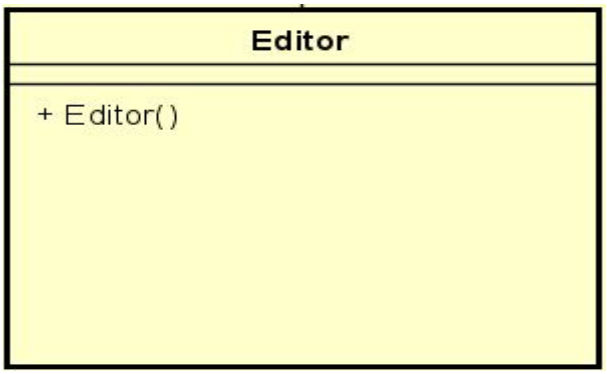
Na segunda camada ou Pacote *model*, é onde estão localizados os principais conceitos do cenário além das regras de negócio e onde serão armazenados os dados. É constituído pelas classes Cozinheiro, Degustador, Editor, Livros, Receitas, Categorias, Ingredientes, AvaliaReceita e também a classe abstrata Funcionario.

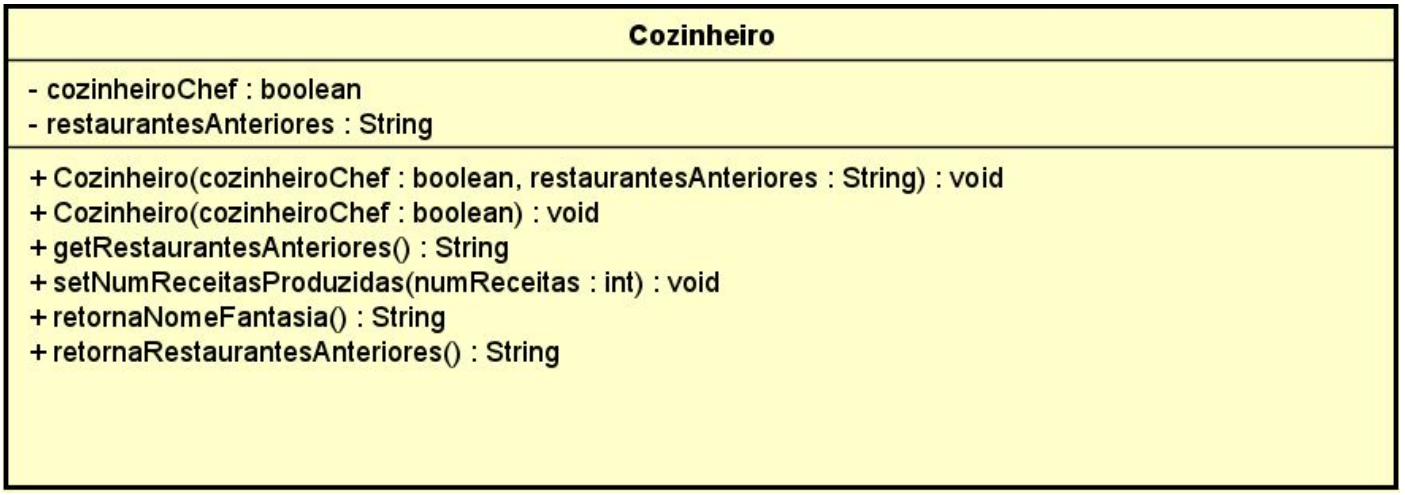


<<<abstract>>>
Funcionarios

- nome : String
- sexo : String
- rg : String
- matricula : int
- dataDeIngresso : Date
- salario : double

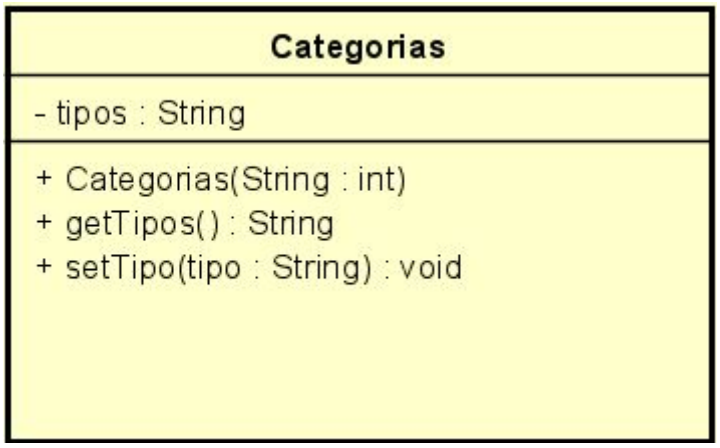
+ Funcionarios(nome : String, sexo : String, rg : float, matricula : int, dataDeIngresso : Date, salario : float)
+ getNome() : String
+ setNome(nome : String) : void
+ getSexo() : String
+ setSexo(sexo : String) : void
+ getRg() : String
+ setRg(rg : String) : void
+ getMatricula() : int
+ getDataDeIngresso() : Date
+ getSalario() : double
+ setSalario(salario : double) : void

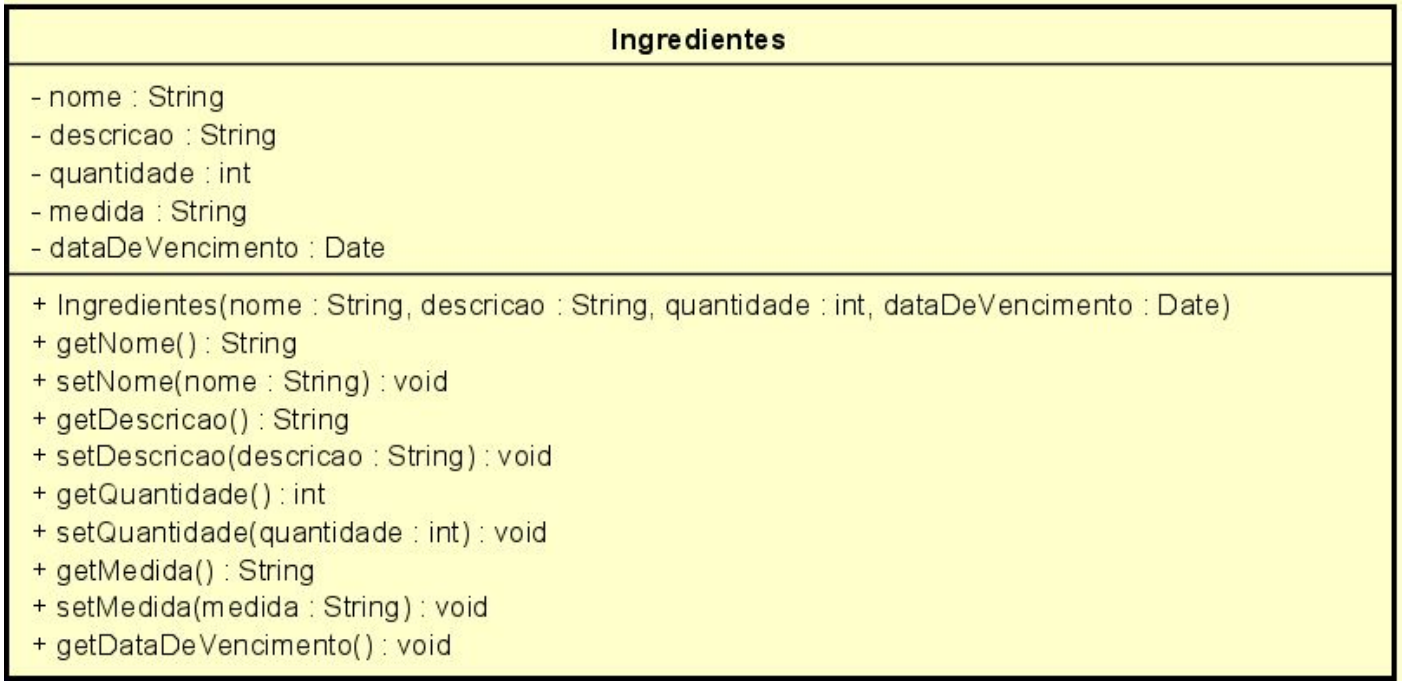

Editor
+ Editor()

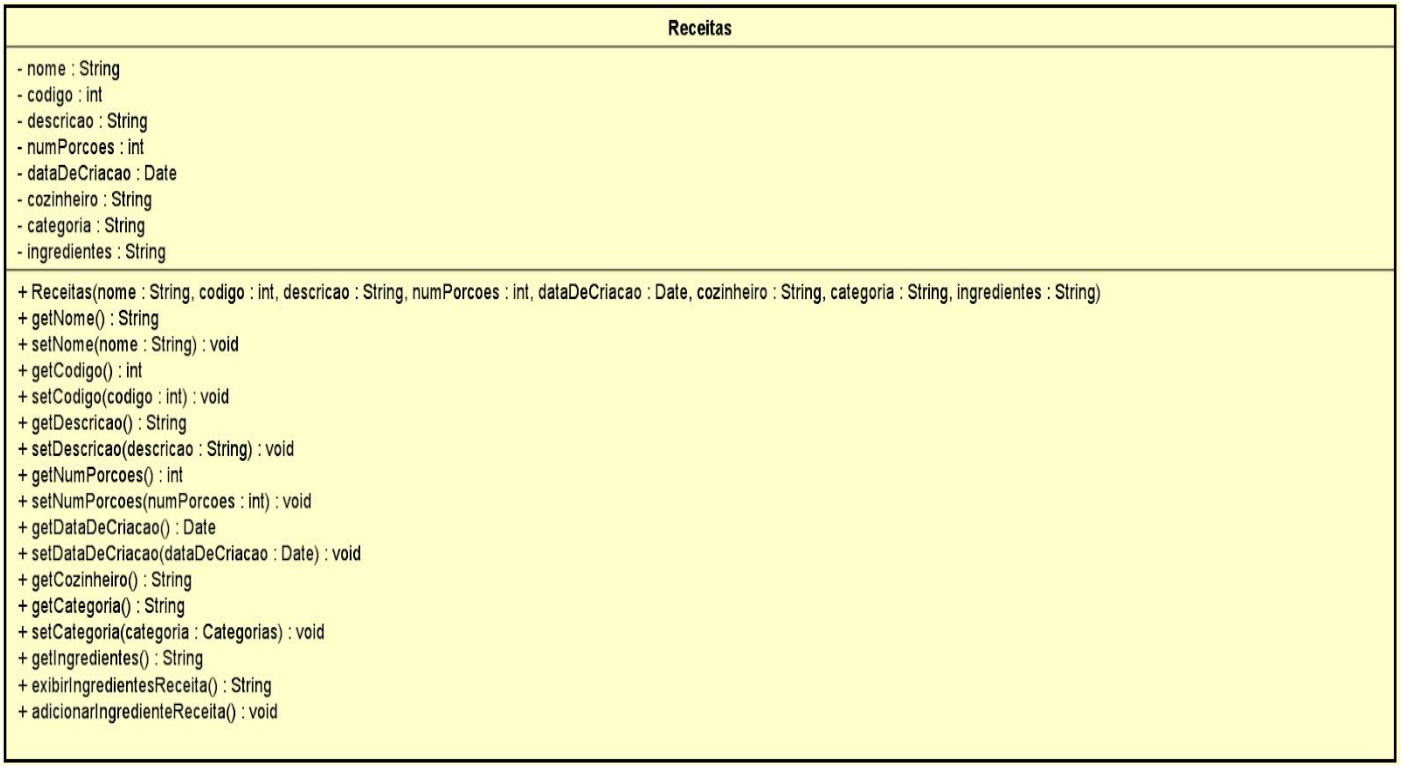

Cozinheiro
- cozinheiroChef : boolean
- restaurantesAnteriores : String
+ Cozinheiro(cozinheiroChef : boolean, restaurantesAnteriores : String) : void
+ Cozinheiro(cozinheiroChef : boolean) : void
+ getRestaurantesAnteriores() : String
+ setNumReceitasProduzidas(numReceitas : int) : void
+ retornaNomeFantasia() : String
+ retornaRestaurantesAnteriores() : String


Degustador
+ Degustador()

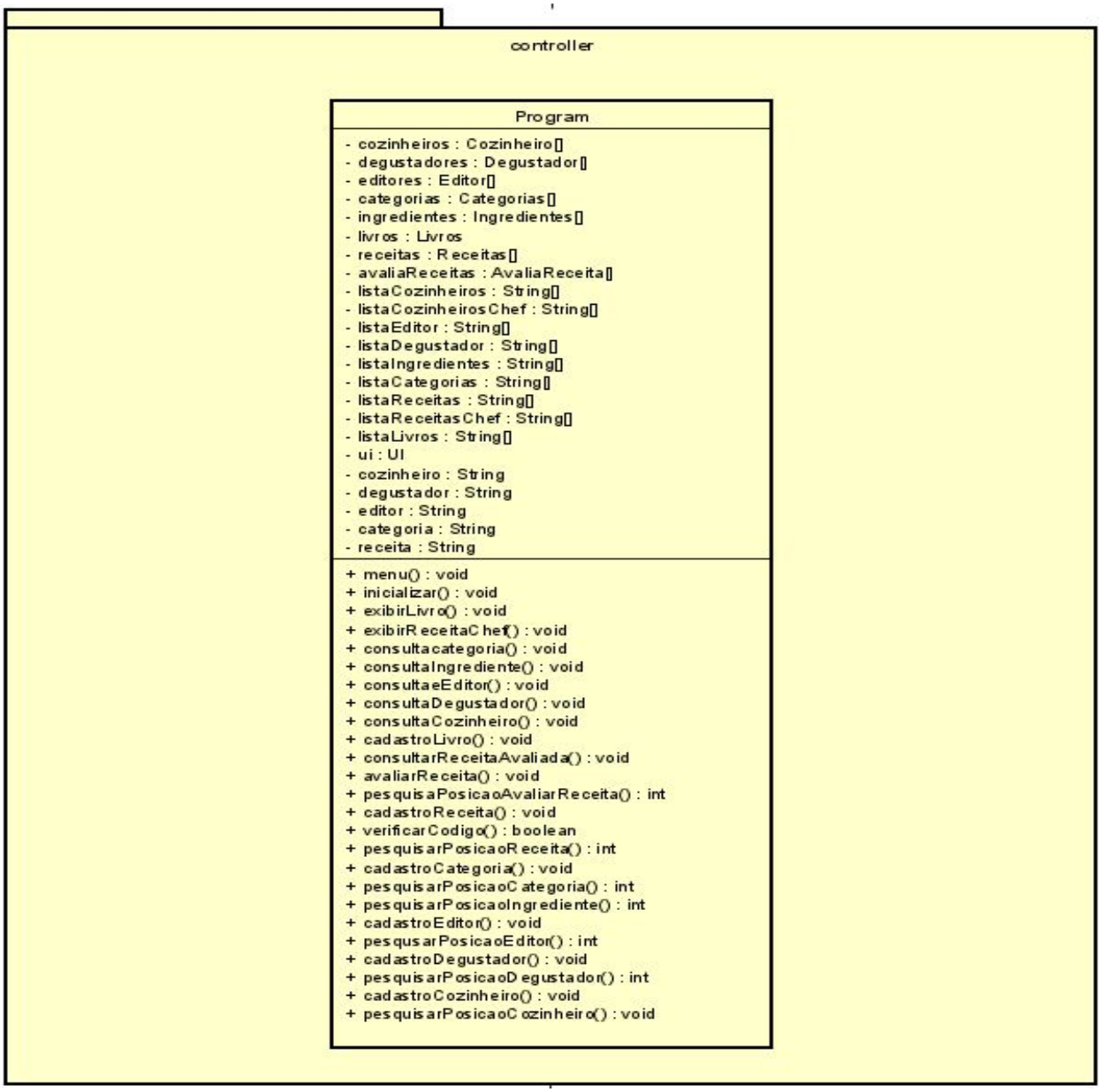

AvaliaReceita
- receita : Receitas
- dataDeDegustacao : Date
- nota : float
- editor : String
+ AvaliaReceita(receita : Receitas, dataDeDegustacao : int, nota : float)
+ getReceita() : Receitas
+ setReceita(receita : Receitas) : void
+ getDataDeDegustacao() : Date
+ setDataDeDegustacao(dataDeDegustacao : int) : void
+ getNota() : int
+ setNota(nota : int) : void
+ getEditor0() : String
+ setEditor(String : int) : void


Categorias
- tipos : String
+ Categorias(String : int)
+ getTipos( ) : String
+ setTipo(tipo : String) : void

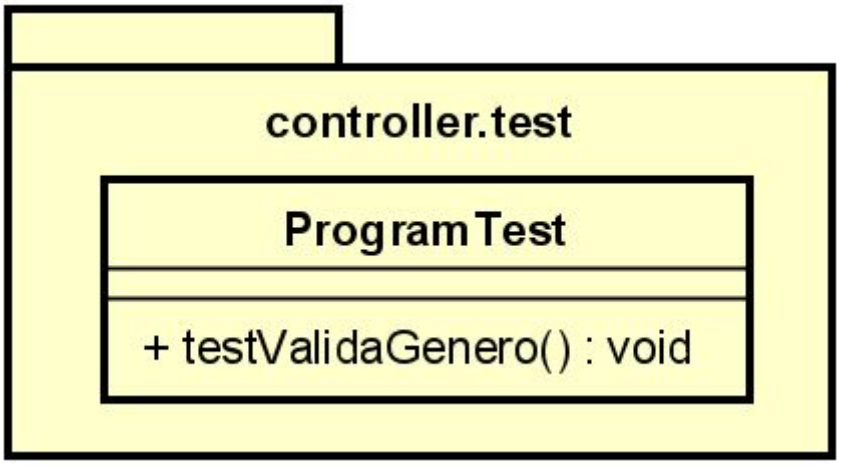

Ingredientes
- nome : String
- descricao : String
- quantidade : int
- medida : String
- dataDeVencimento : Date
+ Ingredientes(nome : String, descricao : String, quantidade : int, dataDeVencimento : Date)
+ getNome( ) : String
+ setNome(nome : String) : void
+ getDescricao( ) : String
+ setDescricao(descricao : String) : void
+ getQuantidade( ) : int
+ setQuantidade(quantidade : int) : void
+ getMedida( ) : String
+ setMedida(medida : String) : void
+ getDataDeVencimento( ) : void


Receitas
- nome : String
- codigo : int
- descricao : String
- numPorcoes : int
- dataDeCriacao : Date
- cozinheiro : String
- categoria : String
- ingredientes : String
+ Receitas(nome : String, codigo : int, descricao : String, numPorcoes : int, dataDeCriacao : Date, cozinheiro : String, categoria : String, ingredientes : String)
+ getNome() : String
+ setNome(nome : String) : void
+ getCodigo() : int
+ setCodigo(codigo : int) : void
+ getDescricao() : String
+ setDescricao(descricao : String) : void
+ getNumPorcoes() : int
+ setNumPorcoes(numPorcoes : int) : void
+ getDataDeCriacao() : Date
+ setDataDeCriacao(dataDeCriacao : Date) : void
+ getCozinheiro() : String
+ getCategoria() : String
+ setCategoria(categoria : Categorias) : void
+ getIngredientes() : String
+ exibirIngredientesReceita() : String
+ adicionarIngredienteReceita() : void

Na terceira camada ou Pacote *controller*, é onde está a classe principal Program responsável fazer o controle do sistema, solicitando e armazenando dados informados pelo usuário além do controle e verificação de informações por meio de validações. Esta classe possui uma dependência com os Pacotes view para gerar a tela de interação e model para fazer o armazenamento de dados.

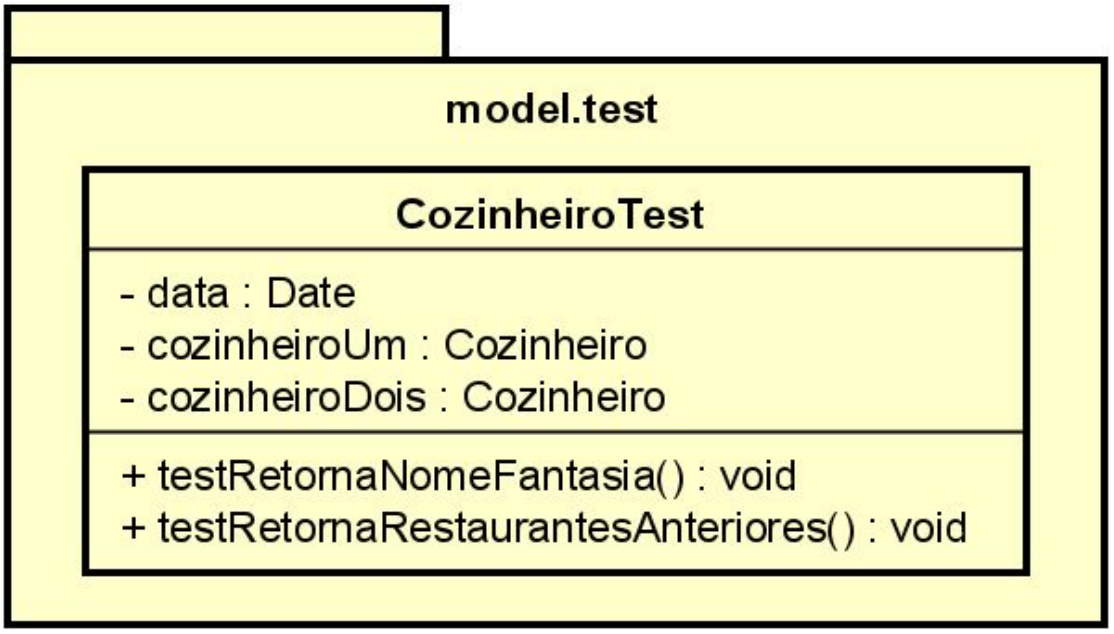


Os pacotes *controller.test* e *model.test* é onde estão localizadas as Classes de teste unitários utilizando JUnit para os pacotes *controller* e *model* respectivamente.



O método testValidaGenero é responsável por verificar se o método validaGenero da classe Program possui o retorno esperado com base nos seguintes casos de teste:

ID	Entrada ou ação	Resultado Esperado	Status
1	masculino	true	passou
2	masc	false	passou
3	feminino	true	passou
4	fem	false	passou



O método testRetornaNomeFantasia é responsável por verificar se o método retornaNomeFantasia da Classe Cozinheiro possui o retorno esperado com base nos seguintes casos de teste:

ID	Entrada ou ação	Resultado Esperado	Status
1	false	Não	passou
2	true	Sim	passou

O método testRetornaRestaurantesAnteriores é responsável por verificar se o método retornaRestaurantesAnteriores da Classe Cozinheiro possui o retorno esperado com base nos seguintes casos de teste:

ID	Entrada ou ação	Resultado Esperado	Status
1	Null	Não se aplica	passou