

Avaliação do Projeto


Assiduidade
Participação
Desempenho

| | | | |
|--------------|--------------|-----|-----------|
| 0 | 1 | 2 | 3 |
| NF (ass<50%) | Insuficiente | Bom | Muito Bom |

Avaliação do Projeto

- Processo de software baseado em metodologias ágeis
 - Design orientado pelas funcionalidades
 - Usando artefactos: UC,SSD, SD, CD, ...
- **Desenvolvimento cooperativo suportado pelo GIT numa perspetiva de integração contínua**
- **Competências técnicas associadas**
 - Análise e Design
 - Padrões de software
 - Criação de artefactos
 - Implementação e Teste de Software
 - Testes Unitários
 - Persistência em Base de Dados Relacional - ORM
 - JPA 2.0

Funcionalidades Prioritárias

- 
- Registrar / Listar Tipo de Despesa
 - Registrar / Listar Meios de Pagamento
 - Registrar Despesa - C/ Tipo de Despesa e Meios de Pagamento (Cash, DebitCard, CreditCard, Check)
 - Visualizar sempre total de gastos da semana e total gastos do mês
 - Consulta de gastos mensais
 - Consulta de gastos mensais agrupados por Tipo de Despesa
 - Consultar despesas
 - Consulta despesas mensais agrupados por mês
 - Registrar Tipo de Rendimento
 - Registrar Rendimento com Tipo de Rendimento
 - Listar Rendimento
 - Classe Conta -substituir p/ AccountServices -(Balance)

- Criar os artefactos relevantes associados às funcionalidades
- Repositórios
 - Em Memória
 - Base Dados - H2 Data Base
 - ORM - Hibernate-JAP
- Testes Unitários
- Utilização dos Padrões de Design - soluções padrão para problemas recorrentes e fornecem um vocabulário comum a todos os que os aplicam:
 - Information Expert, Singleton, Abstract Factory, Strategy, Factory

Git - Best practices

- Pull changes/Update *before* editing
- Commit often
- One commit - one issue
- Write meaningful commit messages
- Don't commit broken code
- Review the merge before commit
- Read Diffs from other developers
- Don't delete files through filesystem

Modelação do Negócio

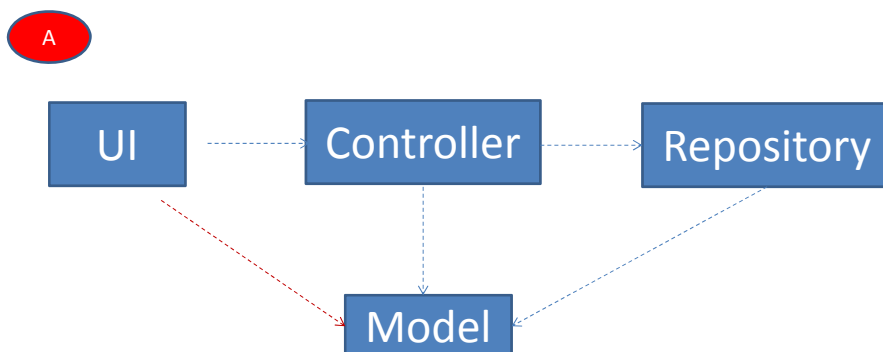
- Modelo do Domínio

Análise e Discussão

- O desenvolvimento é orientado pelos casos de uso
- Arquitetura Lógica
 - Que camadas e como interagem?
 - Um dos objetivos é isolar o modelo do domínio- que tb guarda as regras de negócio - da Interface e do Repositório.
Os controladores do caso de uso guardam o acesso ao modelo e ao repositório
 - Diagrama de packages

Diagramas de packages - Varias alternativas

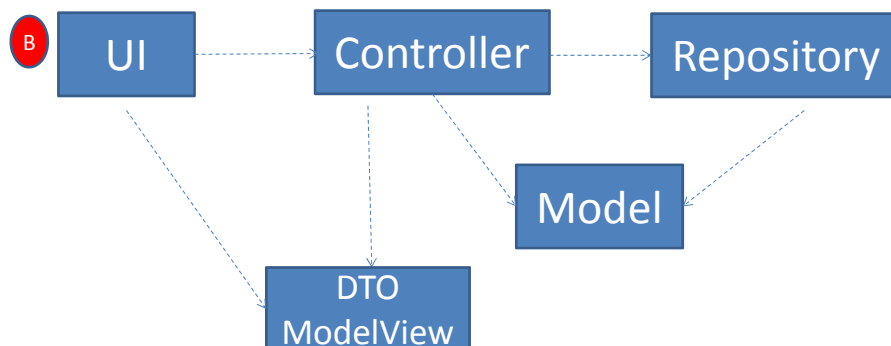
- O modelo deve estar o mais isolado possível



Análise e Discussão

- Passar objetos do domínio para a UI?
- Realizar operações em memória ou diretamente nos repositório?

Diagramas de packages - Vários modelos



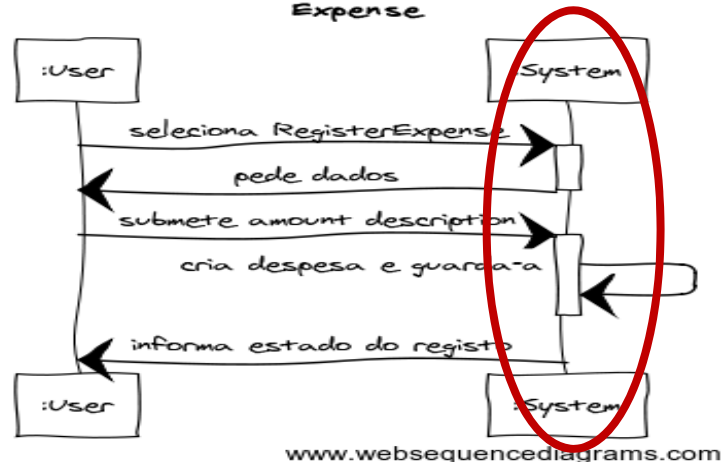
Caso de uso-Registro de Despesa - RegisterExpense Versão inicial

- O utilizador acede à aplicação e seleciona a opção "register expense".
- O sistema apresenta uma interface para introdução dos dados da despesa description, amount and date.
- O utilizador fornece a informação e submete.
- O sistema guarda essa despesa num repositório e informa o utilizador do sucesso da operação.

Quem é o System?

O System são várias classes/objetos com diversas responsabilidades

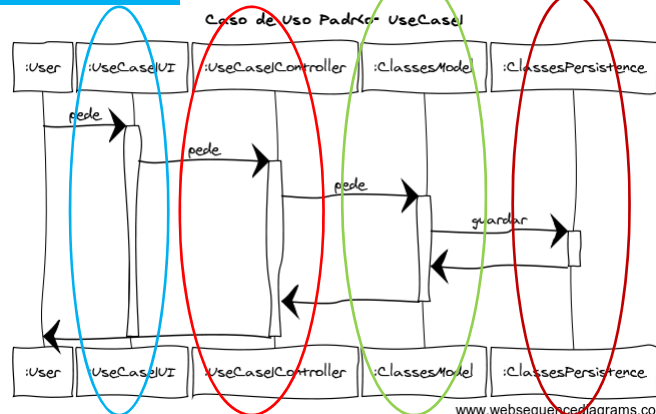
Sequence System Diagram - Register Expense



Sequence Diagram de um Caso de Uso Padrão – UseCase1

Quem recebe as ações do ator

Quem processa

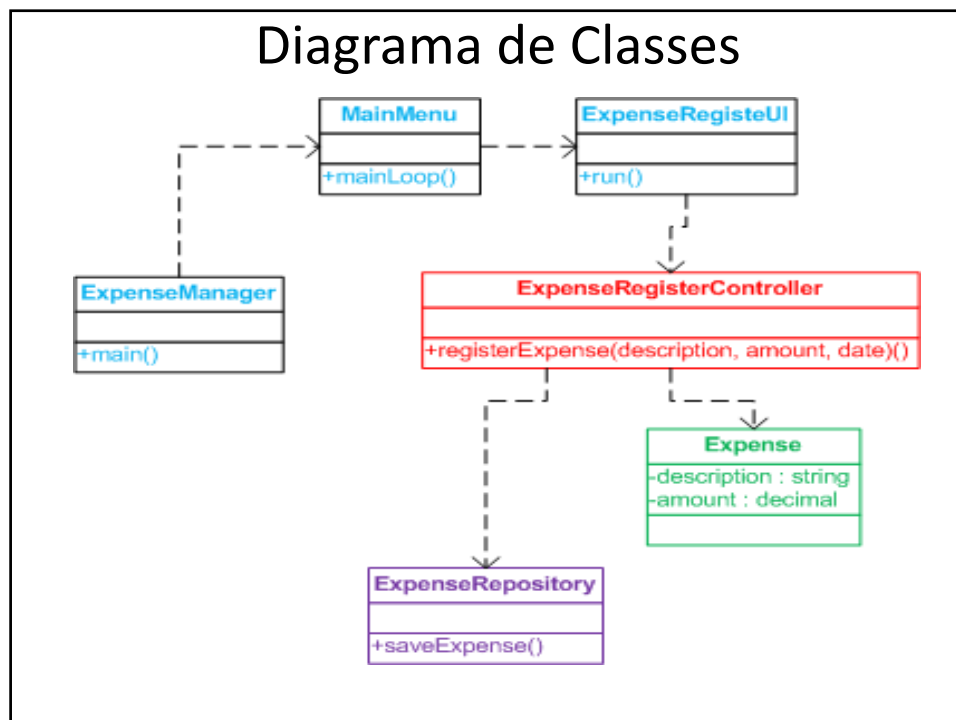
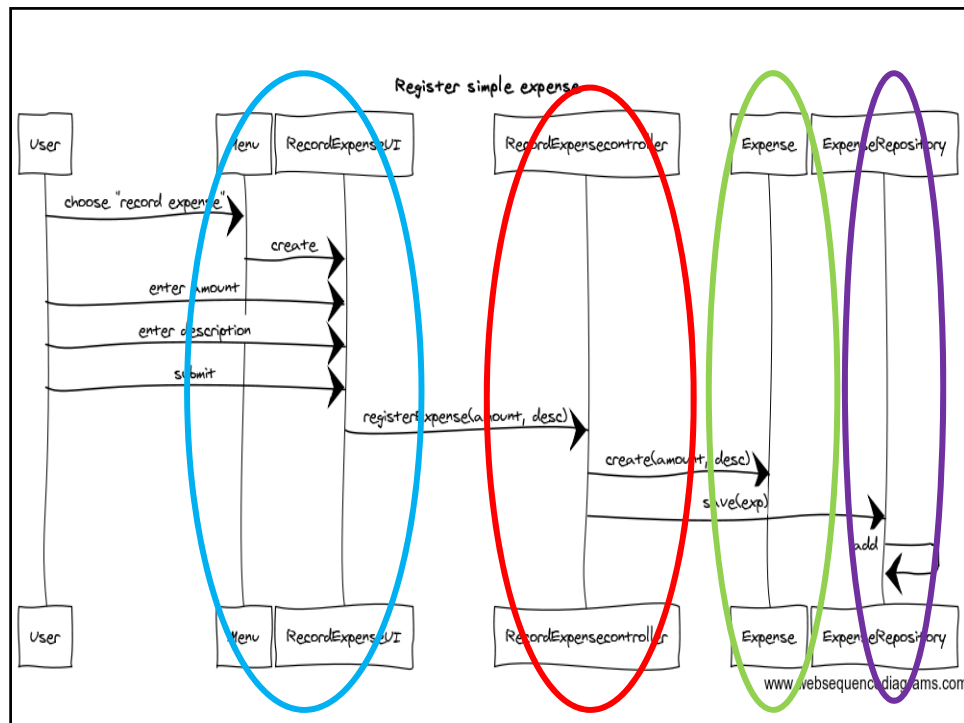


Quem coordena

Quem guarda

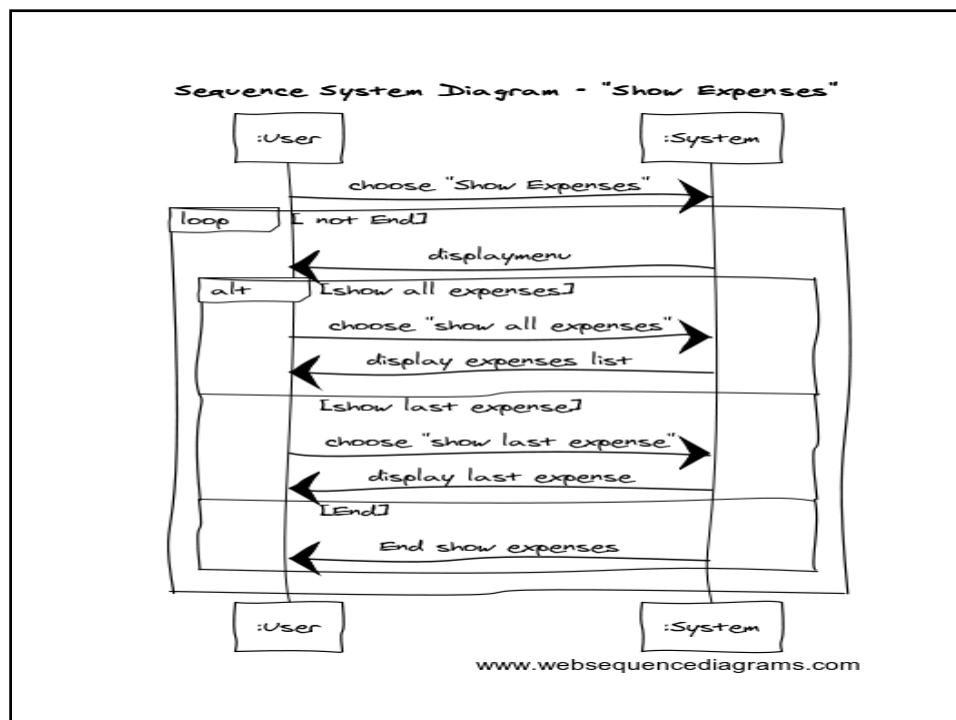
Arquitetura Lógica do software

- Organizada em 4 camadas (packages)
 - Presentation (UIs)
 - Controllers
 - Model
 - Persistence



Caso de uso-Mostra Despesas Versão inicial

1. O utilizador acede à aplicação e seleciona a opção "mostrar despesas".
2. O sistema apresenta um menu disponibilizando os tipos de consulta
3. O utilizador escolhe o tipo de consulta.
4. Se a escolha for
 1. "listagem de todas as despesas" o sistema acede à lista de todas as despesas e mostra no ecrã
 2. "mostra última despesa" o sistema acede à última despesa e mostra no ecrã
 3. "Fim"
5. Enquanto a escolha do utilizador for diferente de "Fim" volta a realizar os pontos 2. , 3. e 4.
6. O sistema informa da finalização da operação



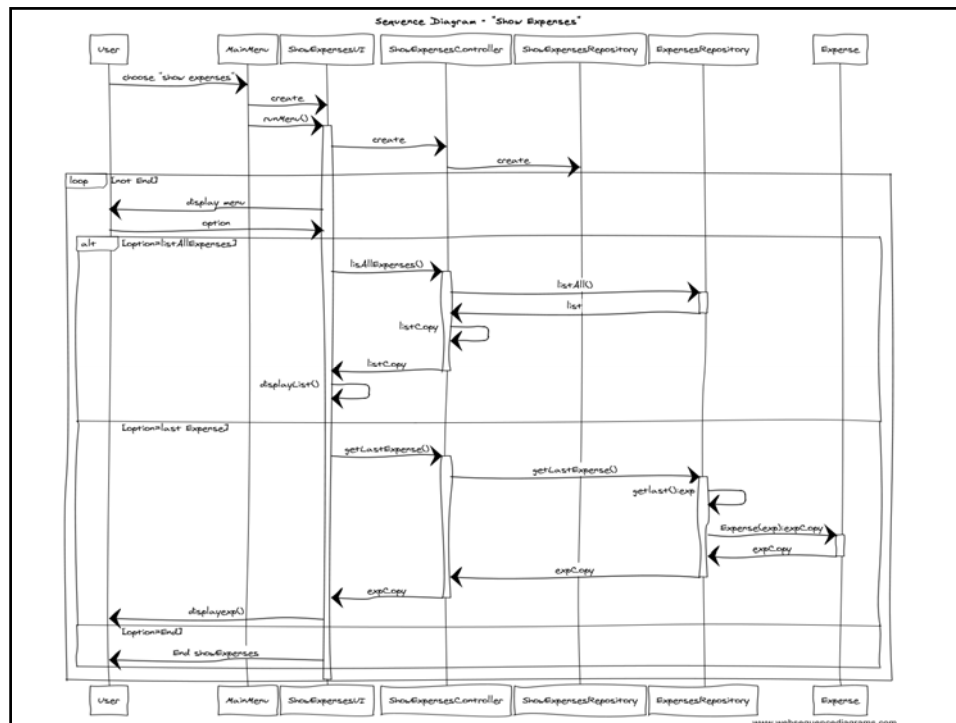
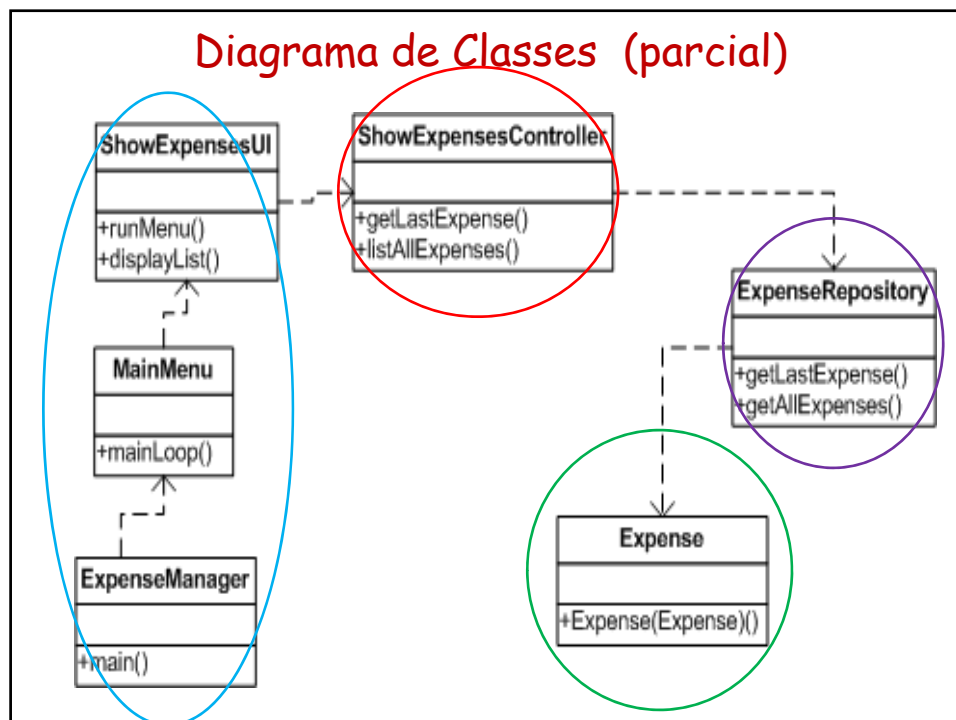


Diagrama de Classes (parcial)

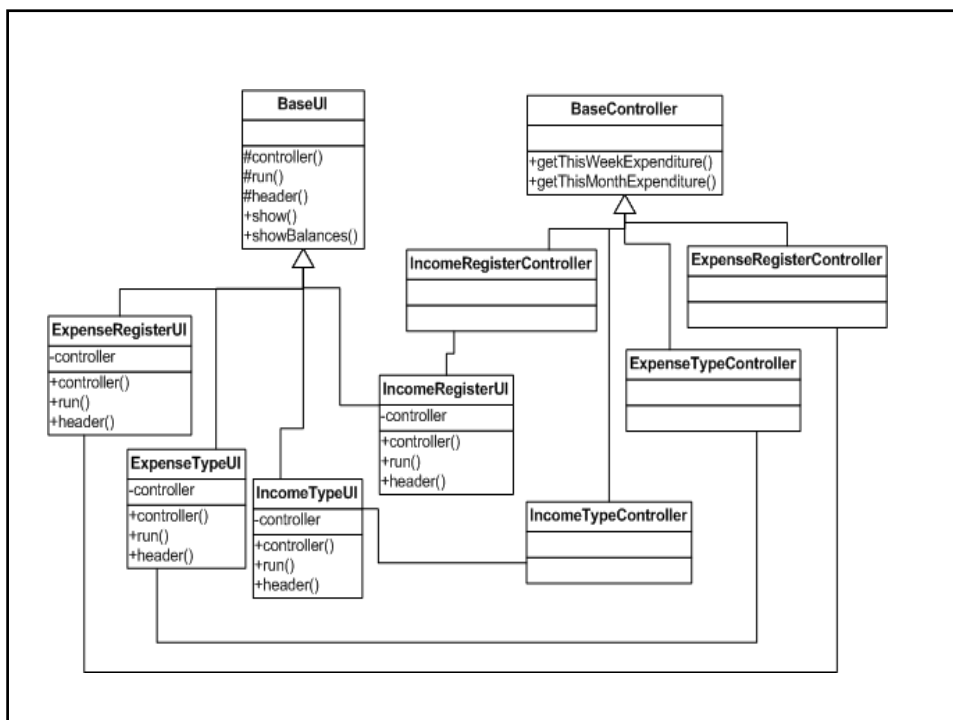


Objetivos a atender no desenvolvimento do projeto:

- Factorização de comportamentos Comuns

As Super Classes BaseUi e BaseController

A Super Classe Movement pai de Expense e Income



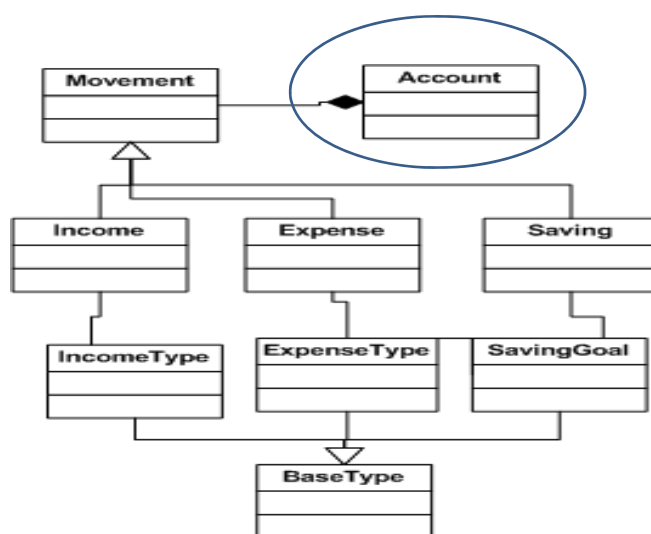
Outras funcionalidades

- As classes - Despesa, Rendimento, ... têm Comportamentos Comuns

Factoring out common behavior

Extrair comportamentos comuns de Despesa, Rendimento, ... e criar uma super classe abstrata Movimento passando as classes Despesa, Rendimento a ser suas subclasses que herdam informação

- Classe Conta
- Inicializar o balanço da conta

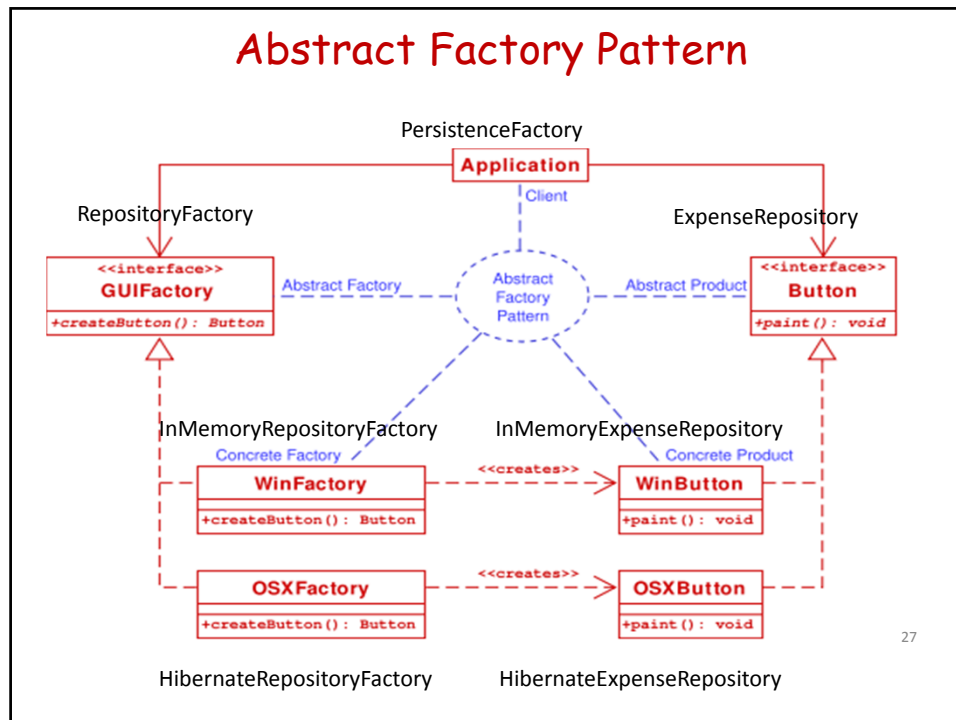


Análise e Discussão

- **Objetos do domínio com conhecimento de persistência ou não**
 1. Puros objetos sem qualquer conhecimento da sua persistência
 2. Objetos do domínio que se podem "save na load" a eles próprios (**Active Record**)
- No caso 1 o controlador é responsável por obter os objetos de um repositório, pedir a objetos do domínio para realizar as regras de negócio e passá-los de novo ao repositório. Neste caso os objetos do domínio são facilmente testáveis pois não dependem de outros packages.
- No caso 2 o controlador pede a uma entidade do domínio para carregar uma dada instância, pede ao objeto que realiza uma operação do domínio e pede ao objeto para se gravar na base de dados

Análise e Discussão

- **Camada de persistência**
 - É organizada em torno do conceito de Repository
 - Começou por ser uma classe atuando como lista em memória que "cuida dos objetos"
 - Para acomodar repositórios em memória com repositórios em bases de dados usou-se o padrão Abstract Factory. A programação foi baseada em interfaces (IRepositoryFactory (, IExpenseRepository, ...)) e criado uma PersistenceFactory (Singleton) que atua como uma factory de factories de repositórios (a escolha da factory de repositórios foi baseado no padrão Strategy)



27

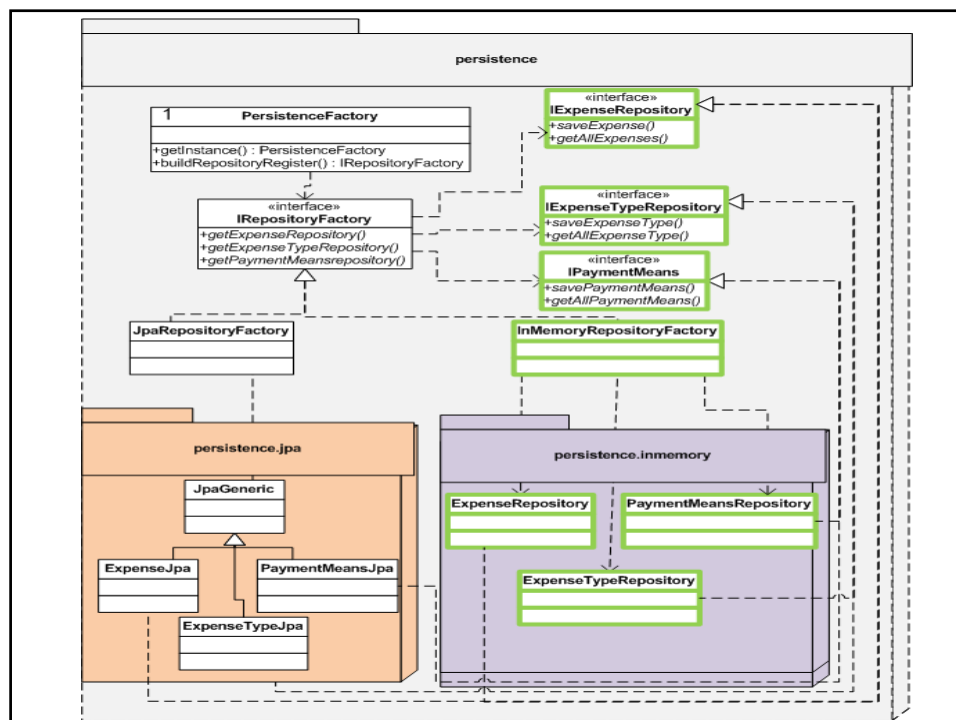


Diagrama de Sequência

- Nos diagramas de sequência anteriores ainda não se usava a camada de persistência implementada segundo o padrão abstract factory.
- Analise o diagrama seguinte onde já é utilizada a Persistence Factory
- [Diagrama de Sequência de RegisterExpenseType](#)

Testes Unitários usando JUnit Exemplo: classe a testar Expense

1. Criar testes para a classe Expense

1. Seleciona Classe e com botão direito rato Tools >Create Tests
2. Des-selecionar TestInitializer e TestFinalizer
3. Seleccionar Junit 4._

O Netbeans cria uma **classe ExpenseTest** no diretório **TestPackages** no package crrespondente ao da classe a testar (neste caso Model).

Esta classe tem um esqueleto de testes. Os testes gerados estão anotados com **@Test** e têm nome baseado no método a testar. Todos eles falham, têm que ser reescritos