INSTITUTO FEDERAL DE CIÊNCIA E TECNOLOGIA DO PIAUÍ

CAMPUS - TERESINA CENTRAL

**Felipe Resende Gomes**

**Programação Corporativa: Sistema de Eventos Acadêmicos**

Teresina, 2016

**RESUMO**

Este trabalho apresenta a descrição detalhada de todo o desenvolvimento do Sistema de Eventos Acadêmicos, proposto em sala de aula pelo professor Rogério da Silva na Disciplina de Programação Corporativa do módulo 3 do curso de Análise e Desenvolvimento de Sistemas do Instituto Federal do Piauí, Campus Teresina Central. O mesmo projeto foi desenvolvido em sala de aula por todos os alunos, mas estes tinham liberdade de escolher as ferramentas de suas preferência. É preferível que as características do sistema abranja todo o conteúdo visto durante as aulas.

**SUMÁRIO**

1 INTRODUÇÃO.............................................................................................. 4

2 DESENVOLVIMENTO.................................................................................. 4

2.1 OBJETIVO GERAL....................................................................................... 4

2.1.1 OBJETIVO ESPECIFICO................................................................................. 4

2.2 METODOLOGIA............................................................................................ 5

2.4 RESULTADOS.............................................................................................. 13

3 CONCLUSÃO....................................................... 14

4 REFERENCIA BIBLIOGRAFICA ....................................................... 15

**1. INTRODUÇÃO**

A atividade proposta em sala de aula descrevia um sistema para controle de eventos acadêmicos. Durante o desenvolvimento, houveram várias discussões sobre como seria a utilização de orientação à objetos para a construção do sistema. Vários padrões foram citados bem como metodologias de desenvolvimento, boas práticas e exercícios práticos. Em um primeira aula foi discutido um artigo que debatia sobre Engenharia de Software e questionava o real produto que é manufaturado pelo desenvolvedor.

**2. DESENVOLVIMENTO**

Muito discutiu-se sobre Padrões de Projeto. Este são basicamente soluções encontradas para os problemas mais comuns de arquitetura de código que pudessem permitir a utilização de conceitos básicos com Aberto/Fechado e Coesão. Os padrões não são regras a serem seguidas, eles apenas definem uma solução entre várias que pode ser utilizada. Por isso, para o desenvolvimento do projeto, alguns padrões foram utilizados como base apenas, nenhum deles foi seguido a risca pois precisavam de leves adaptações para o código existente.

**2.1. OBJETIVO GERAL**

Desenvolvimento de um sistema para gerenciar eventos acadêmicos tais como: Semanas Científicas, Simpósio, Ciclo de Palestras, Jornadas, Congressos etc. O design deve permitir controlar/gerenciar/adicionar as atividades como: Palestras e Minicursos. Inicialmente o sistema deve gerencia apenas divulgação e inscrição de participantes. Posteriormente, o sistema deve gerenciar também as atividades desenvolvidas bem como seus horário e preço.

**2.1.1. OBJETIVO ESPECIFICO**

O desenvolvimento do sistema deve utilizar do conteúdo previsto em sala de aula: Conceitos fundamentais de S.O.L.I.D, refatoração, expressões regulares, aplicação de mapeamento-objeto-relacional para a construção de banco de dados, utilização de GIT para versionamento do projeto e aplicação de testes unitários para validação de diversas implementações.

**2.2. METODOLOGIA**

Para desenvolvimento do sistema, a linguagem orientada a objetos utilizada foi C# juntamente a IDE Visual Studio Express 2015 for Windows Desktop, versão 14.0 Update2. Para a construção dos testes unitários, foi utilizado uma ferramenta do próprio Visual Studio onde se é possível criar uma solução que importa as classes modelos da solução principal e realizar os testes. O código está hospedado em no servidor GitHub e pode ser encontrado pelo do endereço: <https://github.com/FelipeRes/Sistema-de-Eventos/>.

Durante o desenvolvimento do sistema, diversos padrões de projeto foram utilizados para a construção das classes, sempre seguindo o princípios de S.O.L.I.D. e as metodologias de refatoração. Alguns padrões foram misturados e reutilizados em outras situações para reduzir a repetição de código.

**A) Atividade e Evento (Padrão Composite)**

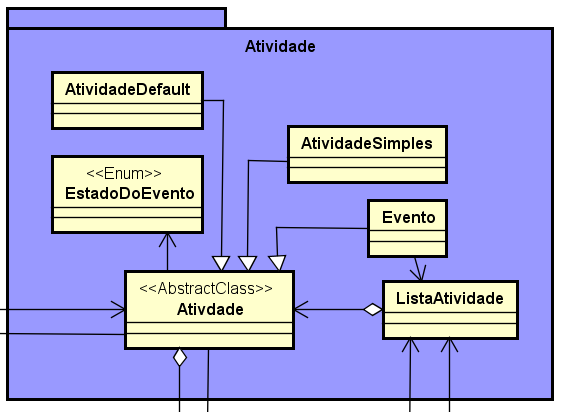


Figura 1. Pacote Atividade

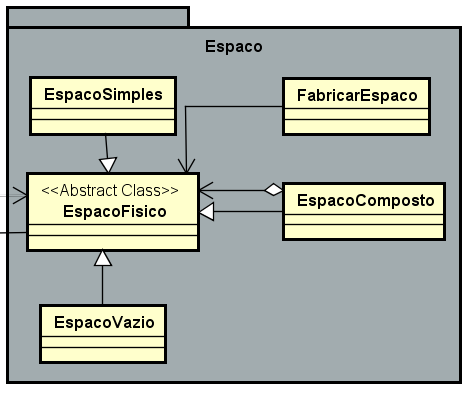


Figura 2. Pacote Espaco.

O pacote atividade possui as principais classes do sistema(Figura 1). Elas controlam os inscritos das atividades. A classe *ListaAtividade* é uma abstração de uma lista de atividades, contudo com mais algumas funcionalidades. Ela foi criada devido a repetição das implementações de inserir e remover atividades de uma lista que estavam presentes em *Evento*, *Inscricao* e *EspacoFisico.* Através dessa classe, é possível gerenciar listas de atividades de forma mais pratica, sem poluir o código.

Outro detalhe importante é que *Evento* possui uma *ListaAtividade,* semelhante ao padrão Composite. Ainda dentro de Atividade, há a presença de *Hook* *Method*, padrão que diz que partes de funcionalidades de *Atividade* devem ser implementadas unicamente em suas herdeiras.

Espaços são os locais onde as atividades acontecem, eles possuem um nome e uma quantidade de pessoas. Para a construção de espaços flexíveis, também foi utilizado o padrão *Composite* onde uma das implementações de *EspacoFisico* possui uma lista de *EspacoFisico.*

**B) Criação de Cupons (Strategy)**

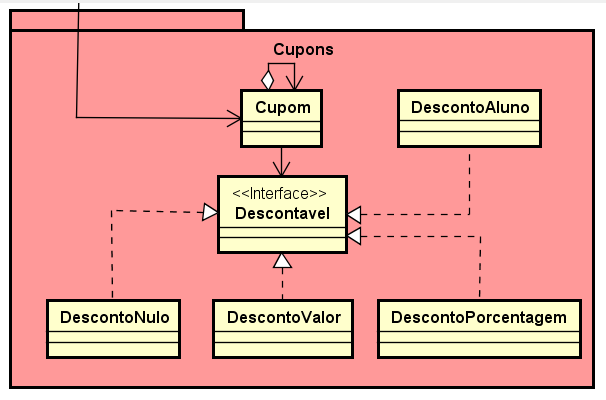


Figura 3. Pacote Cupons.

Um dos requisitos do sistema era que fosse possível criar cupons de desconto para as inscrições.

Afim de atender a todas as possíveis variações de desconto, foi feita uma interface *Descontavel* que pode receber diversas implementações de desconto diferente. Um cupom ainda pode ser uma composição de diversos outros cupons, podendo assim se formar um combo de cupons.

A grande vantagem do Padrão Strategy é a possibilidade de criar diversas implementações para uma funcionalidade e alterá-las em tempos de execução.

**D) Notificações. (Padrão Observer)**

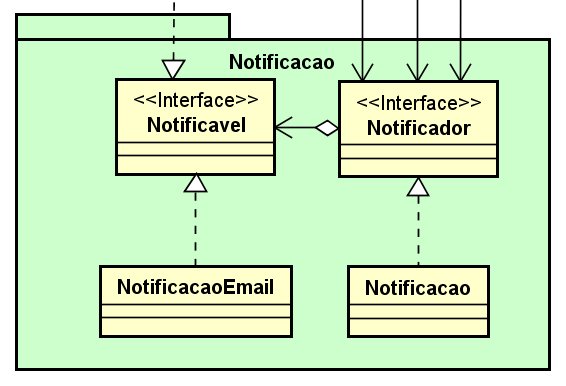
****

Figura 4. Sistema de Notificação.

O sistema utiliza do padrão *Observer* para enviar notificações aos inscritos com informações como confirmação de inscrição e alteração de horários do evento.

Esta arquitetura está sendo aproveitada de duas maneiras. Na primeira, a *Atividade* possui diversos usuários como *Notificavel* e os atualiza sobre alterações no evento. Contudo, os usuários utilizam a mesma arquitetura para selecionar por qual veículo eles querem ser notificados, em outras palavras, a classe *Usuario* também possui um *Notificador* que possui uma lista de veículos, como *NotificaoEmail* por exemplo*,* que podem ser cadastrados.

Em implementações futuras, pode ser colocado outros notificáveis para o usuário como SMS ou redes sociais. O padrão Observer oferece um design de objeto onde os sujeitos e os observadores são levemente ligados, reduzindo as dependências das classes concretas que utilizam dessa arquitetura.

**E) Usuários e pessoas.**

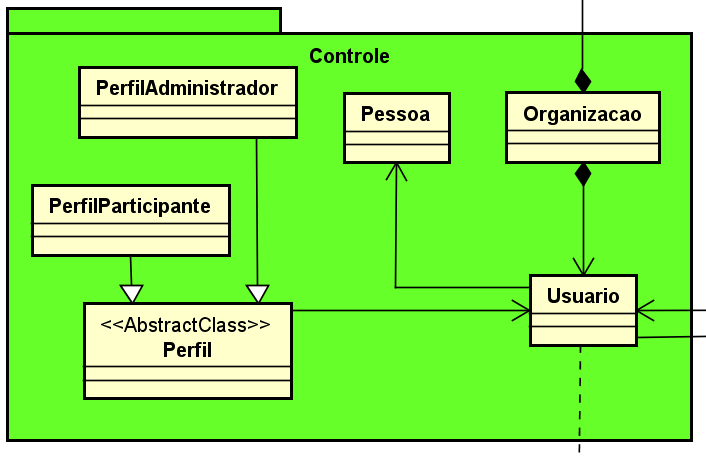
****

Figura 5. Controle de Usuários.

Organização é uma classe feita apenas para associar *Usuario* com *Atividade.* Ela define quem são os responsáveis pela(a) atividade(s). A classe *Pessoa* guarda informações pessoais do usuário como seu nome, data de nascimento, currículo, endereço etc. Ela utiliza do padrão *Builder* com interface fluente, muito recomendada para a criação de formulários.

Para *Usuario*, foi utilizado Regex para validação de e-mail e para composição da senha. A senha nesse caso deve possuir mais de 6 dígitos, apenas letras e números. A parte de Perfil para administrador e participante não foram implementadas completamente, apenas há o conceito de seu design de classes e funcionalidades.

**E) Encapsulamento e Fábricas.**

A criação de alguns objetos pode ser complicado durante a construção da aplicação que utiliza esse sistema como base, podendo ocorrer erros na integridade dos dados.

Para solucionar este problema, todas as abstrações receberam fábricas. Estas fábricas são classes com funções estáticas que recebem os valores pertinentes aos seus respectivos objetos e os constroem de forma mais prática e rápida.

Para que fosse assegurado de que qualquer desenvolvedor não criasse objetos de maneira errada, os construtores dos objetos foram definidos como *internal*. Esta é a especificação do C# para visibilidade por Assembly. As classes em C# podem ser agrupadas por um Assembly em comum quando o código é compilado. As fábricas estão no mesmo assembly que o modelo e por isso podem construí-lo, mas a aplicação que está fora fica dependente das fábricas.

Algumas das fábricas implementadas utilizam de interface fluente para a construção de objetos de maneira mais customizada. Exemplo:

*EspacoFisico local = FabricarEspaco.Composto("IFPI")*

*.CriarEspaco("PredioA")*

*.CriarEspaco("PredioB")*

*.CriarEspaco("PredioC")*

*.build();*

*Pessoa pessoa = Pessoa*

*.BuildNome("Maria")*

*.CPF(0402234012)*

*.DataNascimento(DateTime.Now)*

*.build();*

**F) Fluent NHibernate**

Na fase inicial de desenvolvimento do projeto, utilizava-se a Entity Framework Code First, desenvolvida pela Microsoft e que possui diversos utilitários no Visual Studio. Contudo, esta Framework possui problemas para mapear associações polifórmicas pois suas classes DAO não são capazes de identificar as sub-classes de forma automática.

Então, para a persistência de dados, utilizou-se da Framework NHibernate. Esta Framework faz o mapeamento objeto relacional da classe modelo e persiste no banco de dados. Ela utiliza de arquivos XML customizados para cada classe do modelo e é possível configurar os relacionamentos entre as tabelas.

De forma a se ganhar tempo e praticidade na construção do banco de dados, utilizou-se de uma API chama Fluent NHibernate. Ela permite que o mapeamento do modelo para o banco de dados seja feito através de uma outra classe que irá gerar XML. Exemplo:

*public class InscricaoMap : ClassMap<Inscricao> {*

*public InscricaoMap() {*

*Id(x => x.Id);*

*HasMany(x => x.listaDeCupons).Cascade.All();*

*Map(x => x.Pagamento);*

*Map(x => x.Participacao).CustomType<TipoInscricao>();*

*References(x => x.User);*

*References(x => x.Atividades);*

*}*

*}*

O Banco de dados é configurado através da API que possui “drivers” para diversos tipos de bancos. O tipo de banco escolhido foi o SQLite, mas ele pode ser facilmente trocado por outro banco através da alteração do driver. A API também é capaz de criar o banco em uma primeira execução e depois só atualizá-lo conforme a execução do software.

**2.4. RESULTADOS**

O resultado final apresenta a camada de modelo de dados do sistema e a maioria de suas implementações. O projeto se encontra em código livre e disponível em: <https://github.com/FelipeRes/Sistema-de-Eventos/>.

O conjunto de classe se encontra bem pequeno e com poucas implementações, atende a quase todo o PDF. Assim o padrão o código se encontra simples e pratico.

O modelo de classes se encontra ainda incompleto por faltar implementações, mas toda a base já está funcional. Ele possui atualmente 51 teste unitários, 8 branches e mais de 96 commits feitos. A cada nova etapa do processo sempre era um feito um branche no GIT, preservando assim o código original e organizando melhor as etapas do projeto.

**3. CONCLUSOES**

Apesar do trabalho possuir diversos elementos de várias áreas da informática como web e banco de dados, houve um foco maior na construção do modelo para que este seja flexível e enxuto. Então estudou-se diversos padrões de projeto e suas principais características além de manter os princípios de S.O.L.I.D e construir uma boa base para uma futura aplicação.

Notou-se que algumas das funcionalidades requeridas no PDF enviado pelo professor poderiam ser implementadas de maneira bem simples utilizando um banco de dados para consulta de informações.

**4. BIBLIOGRAFIA**

FREEMAN E., FREEMAN E.Use a cabeça. Padrões de Projeto. Editora O’REILLY. 2º Edição. 2009.

Fowler, Martin. *Refactoring: improving the design of existing code*. Pearson Education India, 2009.

FLUENT Nhibernate. Disponivel em:<http://www.fluentnhibernate.org/> Acessado em: 21/09/2016.