

**UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ**



**RELATÓRIO TRABALHO FINAL - BANCO IMOBILIÁRIO
DA DISCIPLINA DE PROJETO DETALHADO DE SOFTWARE**

Rafael Gonçalves Lima, 399509
Victor Lucas da Silva Monteiro, 397307

Quixadá, 05 de dezembro de 2018

1. Introdução

O trabalho de implementação do Banco Imobiliário, proposto pela professora Paulyne Matthews Jucá, fora realizado conforme os requisitos elicitados, atendendo todas as exigências presentes no documento fornecido no dia 22 (vinte e dois) de novembro de 2018. Para correta execução do projeto, houve necessidade de adotar parte dos Padrões de Projeto GoF, uma porção destes apresentados ou comentados em sala. As respectivas explicações e motivações de escolha para cada padrão, presente no diagrama de classes, estão presentes nos tópicos abaixo.

2. Padrões de projeto

2.1 Singleton

Para garantir a existência de apenas uma instância das classes **ControlBancoImobiliario**, **ControlSorteOuReves**, **FactoryCampoEspecial** e **FactoryCartas**, mantendo um ponto de acesso global e único ao objeto, fora usado o padrão de projeto Singleton. A implementação deste padrão fora realizado da seguinte forma:

ControlBancoImobiliario

```
private static ControlBancoImobiliario controlBancoImobiliario = null;

public static synchronized ControlBancoImobiliario getInstance(){
    if(controlBancoImobiliario == null) controlBancoImobiliario = new ControlBancoImobiliario();
    return controlBancoImobiliario;
}
```

ControlSorteOuReves

```
private static ControlSorteOuReves controlSorteOuReves = null;

public static synchronized ControlSorteOuReves getInstance() {
    if(controlSorteOuReves == null) controlSorteOuReves = new ControlSorteOuReves();
    return controlSorteOuReves;
}
```

FactoryCartas

```
private static FactoryCartas factoryCartas = new FactoryCartas();

private FactoryCartas() {}

public static synchronized FactoryCartas getInstance() {
    if(factoryCartas == null) factoryCartas = new FactoryCartas();
    return factoryCartas;
}
```

FactoryCampoEspecial

```
private static FactoryCampoEspecial factoryCampoEspecial = new FactoryCampoEspecial();

private FactoryCampoEspecial() {}

public static synchronized FactoryCampoEspecial getInstance() {
    if(factoryCampoEspecial == null) factoryCampoEspecial = new FactoryCampoEspecial();
    return factoryCampoEspecial;
}
```

2.2 Iterator

Para encapsular a iteração realizada nas cartas de sorte ou revés, obtendo conhecimento da primeira carta ou da próxima a ser chamada, fora implementado o padrão Iterator na classe **ControlSorteOuReves**. A implementação deste padrão depende que a classe implemente a interface **Iterator**, realizando sobrescrita de métodos presentes nesta classe, pré-determinados na criação deste último.

Iterator

```
public interface Iterator {
    Object first();
    Object next();
    boolean isDone();
}
```

ControlSorteOuReves

```
public class ControlSorteOuReves implements Iterator {
```

```

@Override
public Object first() { return this.cartas.get(0); }

@Override
public Object next() {
    if (cartas.isEmpty()) {
        this.criarCartas();
    }
    Random random = new Random();
    int i = random.nextInt(this.cartas.size());
    return this.cartas.remove(i);
}

@Override
public boolean isDone() { return false; }

```

2.3 Observer

Para saber quando o jogador vai a falência tivemos de implementar o padrão de projeto observer, que escutará quando uma determinada pessoa falir e irá tomar ações para que o mesmo (usuário) seja notificado disto. Para implementar um observer as interfaces **ObserverJogador** e **SubjectObserver** foram construídas e elas são implementadas por **JogadorHumano** (que é o sujeito observado) e **ControlBancoImobiliario** (que é o observador).

ObserverJogador

```

public interface ObserverJogador {
    void update(JogadorHumano jogador, float valor);
}

```

SubjectObserver

```

public interface SubjectObserver {
    void addObserver(ObserverJogador o);
    void removeObserver(ObserverJogador o);
    void notifyObserver(float valor);
}

```

JogadorHumano

```
public class JogadorHumano extends Jogador implements SubjectObserver {

    @Override
    public void addObserver(ObserverJogador o) {
        this.observerCollection.add(o);
        System.out.println(this.observerCollection.size());
    }

    @Override
    public void removeObserver(ObserverJogador o) { this.observerCollection.remove(o); }

    @Override
    public void notifyObserver(float valor) {
        for (ObserverJogador object: this.observerCollection) {
            object.update(jogador this, valor);
        }
    }
}
```

ControlBancoImobiliario

```
public class ControlBancoImobiliario implements ObserverJogador {
```

```
    @Override
    public void update(JogadorHumano jogador, float valor) {
        System.out.println(jogador.getNome()+" está devendo R$ "+valor);
        float propriedadesVendidas = 0;

        if (jogador.getPropriedades().isEmpty()) {
            JOptionPane.showMessageDialog( parentComponent: null, message: jogador.getNome() + " foi à falência (Sem dinheiro para pagar os credores).");
            this.jogadoresAtivos.remove(jogador.getId());
            jogador.getPeca().obterLocalizacao().removerJogador(jogador);
        } else {
            for (Propriedade propriedade: jogador.getPropriedades()) {
                if (propriedade instanceof Terreno) {
                    if (((Terreno) propriedade).isHasHotel()) {
                        propriedadesVendidas += ((4 * ((Terreno) propriedade).getPrecoCasa() + ((Terreno) propriedade).getPrecoHotel())/2);
                        propriedadesVendidas += propriedade.getPreco();
                    } else {
                        propriedadesVendidas += (((Terreno) propriedade).getNumCasas() * ((Terreno) propriedade).getPrecoCasa())/2);
                        propriedadesVendidas += propriedade.getPreco();
                        System.out.println("Sem Hotel: "+propriedadesVendidas);
                    }
                } else {
                    propriedadesVendidas += propriedade.getPreco();
                }
            }
            JOptionPane.showMessageDialog( parentComponent: null, message: jogador.getNome()+" vendeu "+propriedade.getNome()+" por "+propriedadesVendidas+" para pagar suas divi");
            if (propriedadesVendidas >= valor) {
                Banco.getInstance().pagar(propriedadesVendidas);
                jogador.receber(propriedadesVendidas);
                JOptionPane.showMessageDialog( parentComponent: null, message: jogador.getNome()+" pagou sua dívida e continua no jogo!");
                break;
            } else {
                JOptionPane.showMessageDialog( parentComponent: null, message: "Dinheiro Insuficiente Para Pagar O Credor");
            }
        }
    }

    if (this.jogadoresAtivos.size() == 1) {
        JOptionPane.showMessageDialog( parentComponent: null, message: "<Alguém> ganhou essa partida");
        TelaPrincipal.getInstance().closeJanela();
    }
}
```

2.4 Strategy

Para executar comportamentos diferentes relacionados a ação de cada carta, tivemos que implementar o padrão strategy. Para implementar esse padrão utilizamos de uma classe abstrata **Carta** que é estendida por **CartaAposta** e **CartaSaiaDaPrisão**. Conforme pode ser observado abaixo o método **acao(...)** é implementado de forma

diferente em cada uma das classes que herdam de **Carta**. Pode ser observada cada classe implementada abaixo:

Carta

```
public abstract class Carta {  
  
    protected String titulo;  
    protected String descricao;  
    protected boolean isCoringa;  
  
    public abstract void acao(JogadorHumano jogador);  
  
    public String getTitulo() { return this.titulo; }  
  
    public void showCarta(String titulo, String mensagem, String acao) {  
        MensagemSorteReves ms = new MensagemSorteReves(titulo, mensagem, acao);  
        ms.setVisible(true);  
    }  
}
```

CartaAposta

```
public class CartaAposta extends Carta {  
  
    private float valor;  
  
    public CartaAposta(String titulo, String descricao, float valor) {  
        this.titulo = titulo;  
        this.descricao = descricao;  
        this.valor = valor;  
        this.isCoringa = false;  
    }  
  
    public void acao(JogadorHumano jogador) {  
        Map<Integer, JogadorHumano> listaJogadores = ControlBancoImobiliario.getInstance().getJogadoresAtivos();  
        listaJogadores.forEach((key, value) -> {  
            if (jogador != value) {  
                if (value.pagarCredor(this.valor)) {  
                    jogador.receber(this.valor);  
                }  
            }  
        });  
        this.showCarta(this.titulo, this.descricao, acao: jogador.getNome() + " Recebeu R$ 50 de cada Jogador.");  
    }  
}
```

CartaSaiaDaPrisao


```

public class CartaSaiaDaPrisao extends Carta {

    public CartaSaiaDaPrisao(String titulo, String descricao) {
        this.titulo = titulo;
        this.descricao = descricao;
        this.isCoringa = true;
    }

    public void acao(JogadorHumano jogador) {
        if(ControlBancoImobiliario.getInstance().getJogadoresPresos().containsValue(jogador)) {
            ControlBancoImobiliario.getInstance().soltarJogador(jogador);
            this.showCarta(this.titulo, this.descricao, acao: jogador.getNome() + " foi solto da Prisão.");
        } else {
            jogador.setCartaPrisao(this);
            this.showCarta(this.titulo, this.descricao, acao: jogador.getNome() + " ganhou: 'Saida Livre da Prisão.'");
        }
    }
}

```

2.5 Factory

Para evitar vários casos condicionais para produção de um campo no tabuleiro criamos uma fábrica. Utilizamos duas classes para realizar a produção dos campos previamente citados, são elas: **FactoryCampoEspecial** (que contém os condicionais para campos especiais) e **ControlCampos** (que “instancia” por meio da fábrica os campos especiais). As implementações destas classes pode ser observadas abaixo:

FactoryCampoEspecial

```

public class FactoryCampoEspecial {
    private static FactoryCampoEspecial factoryCampoEspecial = null;

    private FactoryCampoEspecial() {}

    public static synchronized FactoryCampoEspecial getInstance() {
        if(factoryCampoEspecial == null) factoryCampoEspecial = new FactoryCampoEspecial();
        return factoryCampoEspecial;
    }

    public Campo criar(String tipoCampo, int indice, int eixoX, int eixoY) {
        if (tipoCampo.equals("PontoDePartida")) {
            return new PontoDePartida(indice, eixoX, eixoY);
        } else if (tipoCampo.equals("ImpostoDeRenda")) {
            return new ImpostoDeRenda(indice, eixoX, eixoY);
        } else if (tipoCampo.equals("LucrosDividendos")) {
            return new LucrosDividendos(indice, eixoX, eixoY);
        } else if (tipoCampo.equals("ParadaLivre")) {
            return new ParadaLivre(indice, eixoX, eixoY);
        } else if (tipoCampo.equals("VaParaPrisao")) {
            return new VaParaPrisao(indice, eixoX, eixoY);
        } else if (tipoCampo.equals("Prisao")) {
            return new Prisao(indice, eixoX, eixoY);
        } else if (tipoCampo.equals("SorteOuReves")) {
            return new SorteOuReves(indice, eixoX, eixoY);
        }
        return null;
    }
}

```

ControlCampos (apenas parte do uso, o restante pode ser observado no código)

```
ControlCampos > criarCampos()
campos.put(1, FactoryCampoEspecial.getInstance().criar(tipoCampo: "SorteioDeFartida", indice: 1, exoK: 639, exoK: 631));
campos.put(2, new Terreno( nome: "LESLION", indice: 2, dono, preco: 100, cor: "ROSA", aluguel: 6, precoCasa: 50, precoHotel: 50, aluguel1Casas: 30, aluguel2Casas: 90, aluguel3Casas: 270, alu...
campos.put(3, FactoryCampoEspecial.getInstance().criar(tipoCampo: "SorteioReves", indice: 3, exoK: 519, exoK: 631));
campos.put(4, new Terreno( nome: "AV. PRESIDENTE VARGAS", indice: 4, dono, preco: 60, cor: "ROSA", aluguel: 2, precoCasa: 50, precoHotel: 50, aluguel1Casas: 10, aluguel2Casas: 30, alugu...
campos.put(5, new Terreno( nome: "AV. NOSSA SENHORA DE COPACABANA", indice: 5, dono, preco: 60, cor: "ROSA", aluguel: 4, precoCasa: 50, precoHotel: 50, aluguel1Casas: 20, aluguel2Casas: 60, alugu...
campos.put(6, new Companhia( nome: "COMPANHIA FERROVIARIA", indice: 6, preco: 200, dono, taxa: 50, exoK: 335, exoK: 631 ));
campos.put(7, new Terreno( nome: "AV. BRIGADEIRO FARIA LIMA", indice: 7, dono, preco: 240, cor: "AZUL", aluguel: 20, precoCasa: 150, precoHotel: 150, aluguel1Casas: 100, aluguel2Casas: 300, alugu...
campos.put(8, new Companhia( nome: "COMPANHIA DE VIACAO", indice: 8, preco: 200, dono, taxa: 50, exoK: 337, exoK: 631));
campos.put(9, new Terreno( nome: "AV. REBOUCAS", indice: 9, dono, preco: 220, cor: "AZUL", aluguel: 18, precoCasa: 150, precoHotel: 150, aluguel1Casas: 90, aluguel2Casas: 250, aluguel3Casas: 750, alugu...
campos.put(10, new Terreno( nome: "AV. 9 DE JULHO", indice: 10, dono, preco: 220, cor: "AZUL", aluguel: 18, precoCasa: 150, precoHotel: 150, aluguel1Casas: 90, aluguel2Casas: 250, aluguel3Casas: 750, alugu...
campos.put(11, FactoryCampoEspecial.getInstance().criar(tipoCampo: "Frisas", indice: 11, exoK: 23, exoK: 638));
campos.put(12, new Terreno( nome: "AV. EUROPA", indice: 12, dono, preco: 200, cor: "VIOLETA", aluguel: 16, precoCasa: 100, precoHotel: 100, aluguel1Casas: 80, aluguel2Casas: 220, aluguel3Casas: 660, alugu...
campos.put(13, FactoryCampoEspecial.getInstance().criar(tipoCampo: "SorteioReves", indice: 13, exoK: 23, exoK: 512));
campos.put(14, new Terreno( nome: "RUA AUGUSTA", indice: 14, dono, preco: 180, cor: "VIOLETA", aluguel: 14, precoCasa: 100, precoHotel: 100, aluguel1Casas: 70, aluguel2Casas: 200, aluguel3Casas: 600, alugu...
campos.put(15, new Terreno( nome: "AV. BACAMBU", indice: 15, dono, preco: 180, cor: "VIOLETA", aluguel: 14, precoCasa: 100, precoHotel: 100, aluguel1Casas: 70, aluguel2Casas: 200, aluguel3Casas: 600, alugu...
campos.put(16, new Companhia( nome: "COMPANHIA DE TAXI", indice: 16, preco: 150, dono, taxa: 40, exoK: 23, exoK: 387));
campos.put(17, FactoryCampoEspecial.getInstance().criar(tipoCampo: "SorteioReves", indice: 17, exoK: 23, exoK: 277));
campos.put(18, new Terreno( nome: "INTERLAGOS", indice: 18, dono, preco: 350, cor: "LARANJA", aluguel: 35, precoCasa: 200, precoHotel: 200, aluguel1Casas: 175, aluguel2Casas: 500, aluguel3Casas: 1500, alugu...
campos.put(19, FactoryCampoEspecial.getInstance().criar(tipoCampo: "LucrosDevidendos", indice: 19, exoK: 23, exoK: 161));
campos.put(20, new Terreno( nome: "MORUMBI", indice: 20, dono, preco: 400, cor: "LARANJA", aluguel: 50, precoCasa: 200, precoHotel: 200, aluguel1Casas: 200, aluguel2Casas: 600, aluguel3Casas: 1800, alugu...
campos.put(21, FactoryCampoEspecial.getInstance().criar(tipoCampo: "ParadaLivre", indice: 21, exoK: 23, exoK: 15));
campos.put(22, new Terreno( nome: "FLAMENGO", indice: 22, dono, preco: 120, cor: "VERMELHO", aluguel: 8, precoCasa: 50, precoHotel: 50, aluguel1Casas: 40, aluguel2Casas: 100, aluguel3Casas: 300, alugu...
campos.put(23, FactoryCampoEspecial.getInstance().criar(tipoCampo: "SorteioReves", indice: 22, exoK: 164, exoK: 15));
campos.put(24, new Terreno( nome: "BOTAFOGO", indice: 24, dono, preco: 100, cor: "VERMELHO", aluguel: 6, precoCasa: 50, precoHotel: 50, aluguel1Casas: 30, aluguel2Casas: 90, aluguel3Casas: 270, alugu...
campos.put(25, FactoryCampoEspecial.getInstance().criar(tipoCampo: "ImpostoDeRenda", indice: 25, exoK: 280, exoK: 15));
campos.put(26, new Companhia( nome: "COMPANHIA DE NAVEGACAO", indice: 26, preco: 150, dono, taxa: 40, exoK: 340, exoK: 15));
campos.put(27, new Terreno( nome: "AV. BRASIL", indice: 27, dono, preco: 160, cor: "AMARELO", aluguel: 12, precoCasa: 100, precoHotel: 100, aluguel1Casas: 60, aluguel2Casas: 180, aluguel3Casas: 540, alugu...
campos.put(28, FactoryCampoEspecial.getInstance().criar(tipoCampo: "SorteioReves", indice: 28, exoK: 454, exoK: 15));
campos.put(29, new Terreno( nome: "AV. PAULISTA", indice: 29, dono, preco: 140, cor: "AMARELO", aluguel: 10, precoCasa: 100, precoHotel: 100, aluguel1Casas: 50, aluguel2Casas: 150, aluguel3Casas: 450, alugu...
campos.put(30, new Terreno( nome: "JARDIM EUROPA", indice: 30, dono, preco: 140, cor: "AMARELO", aluguel: 10, precoCasa: 100, precoHotel: 100, aluguel1Casas: 50, aluguel2Casas: 150, aluguel3Casas: 450, alugu...
campos.put(31, FactoryCampoEspecial.getInstance().criar(tipoCampo: "ValeParaFrisas", indice: 31, exoK: 640, exoK: 45));
ControlCampos > criarCampos()
```

3. Especificação de responsabilidades

a. ControlBancoImobiliario

- Inicia um novo jogo
- Pede renderização das telas de solicitação de quantidade de jogadores, nomes e cores dos jogadores e tela principal, onde o jogo ocorre
- Inicia uma nova rodada
- Mapeia e pede alteração de posição do jogador no tabuleiro
- Mapeia e pede execução das ações realizadas nos campos
- Observa se algum jogador veio a falência

b. Tabuleiro

- Pede ao *ControlCampos* para instanciar todos os campos;
- Devolve a quem solicitar o campo que o jogador deve avançar

c. Terreno

- Executa o *EfeitoEspecial* referente a ele
- Constrói casa ou hotel
- Mantém informações sobre o dono
- Realiza ações para os visitantes do terreno (*como pagar o aluguel*)

d. Companhia

- i. Executa o *EfeitoEspecial* referente a ele
- ii. Mantém informações sobre o dono
- iii. Realiza ações para os visitantes do terreno (*como pagar o taxa*)

e. Carta

- i. Aplica o *EfeitoEspecial* ao jogador solicitado

f. Jogador

- i. Compra propriedade
- ii. Vende propriedade
- iii. Recebe dinheiro e realiza pagamentos
- iv. Lança dados
- v. Guarda sua peça

g. Peça

- i. Guarda um campo
- ii. É renderizada pela *TelaPrincipal* conforme suas coordenadas

h. FichaCriminal

- i. Controla os delitos do jogador (quantas vezes ele tirou o mesmo conjunto de faces iguais seguidos) para saber a quantas rodas o mesmo está preso

4. Conclusão

Durante as atividades de planejamento e execução pudemos identificar quais padrões poderiam ser mais adequados ao conjunto de objetos do sistema. De forma geral, tivemos algumas dúvidas sobre as responsabilidades do controlador geral do sistema e sobre a comunicação entre os objetos e as views. As implementações mais críticas foram: o gerenciamento dos jogadores presos, as validações para construção de casas e hotéis, a lógica de verificação dos jogadores falidos e finalização do jogo com um vencedor. Por fim, nossos maiores desafios foram gerenciar o tempo e as atividades acumuladas no final do semestre, estamos felizes por ter concluído a atividade. Até uma próxima oportunidade.