Requisitos do projeto

Cadastrar os assegurados e novos seguros

Gerar gráficos

Página cadastro:

- Nome
- Data de nascimento
- Gênero
- E-mail
- Telefone
- CPF
- CEP

Página de cadastro para cotação:

- Profissão (quanto mais de risco, mais caro)
- Salário (quanto mais alto, mais caro)
- Condição de saúde (caso tenha problema de saúde o valor aumenta)
- Escolher o corretor ou corretora (pega o CEP cadastrado e mostra os corretores com CEP parecidos)

Nesse momento o sistema faz o cálculo do seguro, seguindo algumas regras e determinações (chat defina valores ficticios para termos como base o cálculo e as porcentagens)

Regras:

- Faixa etária: 18-80 anos → Quanto mais velho, mais caro o seguro
- Genêro: Mulheres pagam menos, pois estatisticamente elas vivem mais
- Profissão: trabalhos que representam risco maior a vida, encarecem o seguro
- Salário: quanto mais alto o salário, maior o valor do seguro

Página de cotação:

- Mensagem de saudação com o nome cadastrado + o nome do corretor
- Tipo do seguro (pré definido e escolhido com base nas respostas de cadastro do cliente)
- Resultado dos cálculos, resultado da soma
- Lista com os capitais segurados
    - Ex: Vida individual mulher
        - Capital segurado:
        - Básica morte → 50.000
        - Morte acidental → 50.000
        - Invalidez → 50.000
        - Doenças graves → 15.000
        - Funeral familiar → 7.000
- Opção de personalizar a cobertura, deixar opção de retirar coberturas ou adicionar coberturas
- Nos capitais assegurados, ter uma descrição explicativa do que cada capital assegura
- Deixar uma lista de assistências:
    - Ex:
      TELEMEDICINA - ORIENTAÇÃO MÉDICA ONLINE
      VIDA SAUDÁVEL PLUS

DESCONTO FARMÁCIA

REDE DE DESCONTOS

- Opção de personalizar os valores
- Valor final baseada nas escolhas prévia do sistema e nas escolhas do cliente
- Botão de escolher que direciona para uma escolha de dependentes
- Em dependentes escolher a quantidade e ter a opção de pular essa parte
- Ao pular, direcionar para o carrinho

Organização do código:

```
src/
├── main/
│   ├── java/
│   │   └── com/
│   │       └── sementedobrasil/
│   │           ├── model/
│   │           │   ├── Segurado.java
│   │           │   ├── Seguro.java
│   │           │   ├── Cotacao.java
│   │           │   └── Dependente.java
│   │           ├── view/
│   │           │   ├── CadastroSeguradoView.java
│   │           │   ├── CadastroCotacaoView.java
│   │           │   ├── CotacaoView.java
│   │           │   ├── GerenciamentoView.java
│   │           │   └── RelatoriosView.java
│   │           ├── controller/
│   │           │   ├── CadastroSeguradoController.java
│   │           │   ├── CadastroCotacaoController.java
│   │           │   ├── CotacaoController.java
│   │           │   ├── GerenciamentoController.java
│   │           │   └── RelatoriosController.java
│   │           ├── service/
│   │           │   ├── SeguradoService.java
│   │           │   ├── SeguroService.java
│   │           │   ├── CotacaoService.java
│   │           │   └── RelatorioService.java
│   │           ├── dao/
│   │           │   ├── SeguradoDAO.java
│   │           │   ├── SeguroDAO.java
│   │           │   ├── CotacaoDAO.java
│   │           │   └── DependenteDAO.java
│   │           ├── util/
│   │           │   ├── DatabaseConnection.java
│   │           │   ├── ValidationUtil.java
│   │           │   └── ChartUtil.java
│   │           └── main/
│   │               └── Application.java
│   └── resources/
│       └── database/
│           └── schema.sql
└── test/
└── java/
└── com/
└── sementedobrasil/
└── tests/
```

```
├── SeguradoServiceTest.java
├── SeguroServiceTest.java
├── CotacaoServiceTest.java
└── RelatorioServiceTest.java
```

3. Detalhamento dos Pacotes, Classes e Interfaces

3.1. Model (Modelo)

Responsável por representar as entidades do sistema.

- **Segurado.java**

```
java
Copiar código
package com.sementedobrasil.model;

public class Segurado {
    private String nome;
    private LocalDate dataNascimento;
    private String genero;
    private String email;
    private String telefone;
    private String cpf;
    private String cep;

    // Getters e Setters
    // Construtores
    // Métodos adicionais (se necessário)
}
```

- **Seguro.java**

```
java
Copiar código
package com.sementedobrasil.model;

public class Seguro {
    private String tipo;
    private double valorBase;
    private List<CapitalSeguro> capitaisSegurados;
    private List<Assistencia> assistencias;

    // Getters e Setters
    // Construtores
    // Métodos adicionais
}
```

- **Cotacao.java**

```
java
Copiar código
package com.sementedobrasil.model;

public class Cotacao {
    private Segurado segurado;
```

```
    private Seguro seguro;
    private double valorFinal;
    private Corretor corretor;
    private List<Dependente> dependentes;

    // Getters e Setters
    // Construtores
    // Métodos adicionais
}
```

- **Dependente.java**

```
java
Copiar código
package com.sementedobrasil.model;

public class Dependente {
    private String nome;
    private LocalDate dataNascimento;
    private String relacionamento;

    // Getters e Setters
    // Construtores
    // Métodos adicionais
}
```

- **CapitalSeguro.java**

```
java
Copiar código
package com.sementedobrasil.model;

public class CapitalSeguro {
    private String descricao;
    private double valor;

    // Getters e Setters
    // Construtores
}
```

- **Assistencia.java**

```
java
Copiar código
package com.sementedobrasil.model;

public class Assistencia {
    private String nome;
    private String descricao;

    // Getters e Setters
    // Construtores
}
```

3.2. View (Visão)

Responsável pela interface gráfica com o usuário, utilizando WindowBuilder.

- **CadastroSeguradoView.java**
  - Tela para cadastro de segurado com os campos: Nome, Data de Nascimento, Gênero, E-mail, Telefone, CPF, CEP.
- **CadastroCotacaoView.java**
  - Tela para cadastro de cotação com campos: Profissão, Salário, Condição de Saúde, Escolha do Corretor.
- **CotacaoView.java**
  - Tela para visualização da cotação, incluindo mensagens de saudação, tipo do seguro, resultados dos cálculos, capitais segurados, assistências, opções de personalização e dependentes.
- **GerenciamentoView.java**
  - Interface para administradores gerenciarem cotações, aprovarem/rejeitarem, e gerenciarem contas.
- **RelatoriosView.java**
  - Interface para geração e visualização de relatórios e gráficos.

Exemplo Simplificado: CadastroSeguradoView.java

```
java
Copiar código
package com.sementedobrasil.view;

import javax.swing.*;
import java.awt.event.*;

public class CadastroSeguradoView extends JFrame {
    private JTextField nomeField;
    private JTextField dataNascimentoField;
    private JComboBox<String> generoBox;
    private JTextField emailField;
    private JTextField telefoneField;
    private JTextField cpfField;
    private JTextField cepField;
    private JButton salvarButton;

    public CadastroSeguradoView() {
        setTitle("Cadastro de Segurado");
        setSize(400, 400);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new GridLayout(8, 2));

        // Inicialização dos componentes
        nomeField = new JTextField();
        dataNascimentoField = new JTextField();
        generoBox = new JComboBox<>(new String[]{"Masculino", "Feminino", "Outro"});
        emailField = new JTextField();
        telefoneField = new JTextField();
        cpfField = new JTextField();
        cepField = new JTextField();
        salvarButton = new JButton("Salvar");

        // Adicionando componentes à tela
        add(new JLabel("Nome:"));
        add(nomeField);
        add(new JLabel("Data de Nascimento (dd/mm/yyyy):"));
```

```
        add(dataNascimentoField);
        add(new JLabel("Gênero:"));
        add(generoBox);
        add(new JLabel("E-mail:"));
        add(emailField);
        add(new JLabel("Telefone:"));
        add(telefoneField);
        add(new JLabel("CPF:"));
        add(cpfField);
        add(new JLabel("CEP:"));
        add(cepField);
        add(new JLabel(""));
        add(salvarButton);

        // Adicionando ActionListener para o botão salvar
        salvarButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                // Lógica para salvar segurado
                // Validação e chamada ao Controller
            }
        });
    }

    // Métodos para acessar os campos
    public String getNome() { return nomeField.getText(); }
    public String getDataNascimento() { return dataNascimentoField.getText(); }
    public String getGenero() { return (String) generoBox.getSelectedItem(); }
    public String getEmail() { return emailField.getText(); }
    public String getTelefone() { return telefoneField.getText(); }
    public String getCpf() { return cpfField.getText(); }
    public String getCep() { return cepField.getText(); }

    // Método para adicionar ActionListener no botão salvar
    public void addSalvarListener(ActionListener listener) {
        salvarButton.addActionListener(listener);
    }
}
```

3.3. Controller (Controlador)

Gerencia as interações entre a View e o Model, manipulando eventos e atualizações.

- **CadastroSeguradoController.java**

```
java
Copiar código
package com.sementedobrasil.controller;

import com.sementedobrasil.view.CadastroSeguradoView;
import com.sementedobrasil.model.Segurado;
import com.sementedobrasil.service.SeguradoService;
import javax.swing.*;
import java.awt.event.*;

public class CadastroSeguradoController {
    private CadastroSeguradoView view;
    private SeguradoService service;
```

```
    public CadastroSeguradoController(CadastroSeguradoView view, SeguradoService servic
e) {
        this.view = view;
        this.service = service;
        initController();
    }

    private void initController() {
        view.addSalvarListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                salvarSegurado();
            }
        });
    }

    private void salvarSegurado() {
        try {
            // Validação dos dados
            String nome = view.getNome();
            String dataNascimento = view.getDataNascimento();
            String genero = view.getGenero();
            String email = view.getEmail();
            String telefone = view.getTelefone();
            String cpf = view.getCpf();
            String cep = view.getCep();

            if (nome.isEmpty() || dataNascimento.isEmpty() || email.isEmpty() || cpf.is
Empty() || cep.isEmpty()) {
                JOptionPane.showMessageDialog(view, "Todos os campos são obrigatório
s.", "Erro", JOptionPane.ERROR_MESSAGE);
                return;
            }

            // Criação do objeto Segurado
            Segurado segurado = new Segurado(nome, LocalDate.parse(dataNascimento, Date
TimeFormatter.ofPattern("dd/MM/yyyy")), genero, email, telefone, cpf, cep);

            // Chamada ao serviço para salvar
            service.salvarSegurado(segurado);

            JOptionPane.showMessageDialog(view, "Segurado cadastrado com sucesso!", "Su
cesso", JOptionPane.INFORMATION_MESSAGE);
            view.dispose(); // Fecha a janela após salvar
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(view, "Erro ao salvar segurado: " + ex.getMes
sage(), "Erro", JOptionPane.ERROR_MESSAGE);
        }
    }
}
```

- **CadastroCotacaoController.java**
  - Similar ao `CadastroSeguradoController`, gerenciando a interação na página de cadastro para cotação.

3.4. Service (Serviços)

Contém a lógica de negócio e interações com o DAO.

- **SeguradoService.java**

```
java
Copiar código
package com.sementedobrasil.service;

import com.sementedobrasil.model.Segurado;
import com.sementedobrasil.dao.SeguradoDAO;

public class SeguradoService {
    private SeguradoDAO seguradoDAO;

    public SeguradoService() {
        this.seguradoDAO = new SeguradoDAO();
    }

    public void salvarSegurado(Segurado segurado) throws Exception {
        // Validações adicionais podem ser feitas aqui
        seguradoDAO.save(segurado);
    }

    // Outros métodos, como buscar segurados, etc.
}
```

- **CotacaoService.java**

```
java
Copiar código
package com.sementedobrasil.service;

import com.sementedobrasil.model.Cotacao;
import com.sementedobrasil.model.Segurado;
import com.sementedobrasil.model.Seguro;
import com.sementedobrasil.dao.CotacaoDAO;
import com.sementedobrasil.util.ValidationUtil;

public class CotacaoService {
    private CotacaoDAO cotacaoDAO;

    public CotacaoService() {
        this.cotacaoDAO = new CotacaoDAO();
    }

    public Cotacao calcularCotacao(Segurado segurado, Seguro seguro) throws Exception {
        // Implementar lógica de cálculo baseada nas regras definidas
        double valorBase = 100.0; // Valor base fictício

        // Multiplicadores
        double multiplicadorIdade = calcularMultiplicadorIdade(segurado.getDataNascimen
to());
        double multiplicadorGenero = calcularMultiplicadorGenero(segurado.getGenero());
        double multiplicadorProfissao = calcularMultiplicadorProfissao(seguro.getProfis
sao());
        double multiplicadorSalario = calcularMultiplicadorSalario(seguro.getSalario
());
        double multiplicadorSaude = calcularMultiplicadorSaude(seguro.getCondicaoSaude
());
```

```
        // Cálculo final
        double valorFinal = valorBase * multiplicadorIdade * multiplicadorGenero * mult
iplicadorProfissao * multiplicadorSalario * multiplicadorSaude;
        seguro.setValorFinal(valorFinal);

        // Criação da cotação
        Cotacao cotacao = new Cotacao(segurado, seguro, valorFinal, null, null); // Cor
retor e Dependentes serão adicionados posteriormente

        // Salvar cotação
        cotacaoDAO.save(cotacao);

        return cotacao;
    }

    private double calcularMultiplicadorIdade(LocalDate dataNascimento) {
        int idade = Period.between(dataNascimento, LocalDate.now()).getYears();
        if (idade >= 18 && idade <= 30) return 1.0;
        if (idade >= 31 && idade <= 50) return 1.2;
        if (idade >= 51 && idade <= 65) return 1.5;
        if (idade >= 66 && idade <= 80) return 2.0;
        return 1.0; // Default
    }

    private double calcularMultiplicadorGenero(String genero) {
        if (genero.equalsIgnoreCase("Feminino")) return 0.9;
        return 1.0; // Masculino ou Outro
    }

    private double calcularMultiplicadorProfissao(String profissao) {
        switch (profissao.toLowerCase()) {
            case "administrativo":
                return 1.0;
            case "entrega":
                return 1.2;
            case "construção civil":
                return 1.5;
            default:
                return 1.0;
        }
    }

    private double calcularMultiplicadorSalario(double salario) {
        if (salario <= 3000) return 1.0;
        if (salario > 3000 && salario <= 10000) return 1.1;
        if (salario > 10000) return 1.3;
        return 1.0;
    }

    private double calcularMultiplicadorSaude(String condicaoSaude) {
        switch (condicaoSaude.toLowerCase()) {
            case "boa":
                return 1.0;
            case "leve":
                return 1.2;
            case "grave":
                return 1.5;
            default:
```

```
                return 1.0;
        }
    }

    // Outros métodos relacionados à cotação
}
```

3.5. DAO (Data Access Objects)

Responsável pela interação com o banco de dados. Utilizaremos JDBC para simplificação, mas pode-se optar por ORM como Hibernate conforme a necessidade.

- **SeguradoDAO.java**

```
java
Copiar código
package com.sementedobrasil.dao;

import com.sementedobrasil.model.Segurado;
import com.sementedobrasil.util.DatabaseConnection;
import java.sql.*;

public class SeguradoDAO {
    private Connection connection;

    public SeguradoDAO() {
        this.connection = DatabaseConnection.getConnection();
    }

    public void save(Segurado segurado) throws SQLException {
        String sql = "INSERT INTO segurados (nome, data_nascimento, genero, email, tele
fone, cpf, cep) VALUES (?, ?, ?, ?, ?, ?, ?)";
        PreparedStatement stmt = connection.prepareStatement(sql);
        stmt.setString(1, segurado.getNome());
        stmt.setDate(2, Date.valueOf(segurado.getDataNascimento()));
        stmt.setString(3, segurado.getGenero());
        stmt.setString(4, segurado.getEmail());
        stmt.setString(5, segurado.getTelefone());
        stmt.setString(6, segurado.getCpf());
        stmt.setString(7, segurado.getCep());
        stmt.executeUpdate();
        stmt.close();
    }

    // Métodos adicionais: buscar, atualizar, deletar segurados
}
```

- **CotacaoDAO.java**

```
java
Copiar código
package com.sementedobrasil.dao;

import com.sementedobrasil.model.Cotacao;
import com.sementedobrasil.util.DatabaseConnection;
import java.sql.*;
```

```
public class CotacaoDAO {
    private Connection connection;

    public CotacaoDAO() {
        this.connection = DatabaseConnection.getConnection();
    }

    public void save(Cotacao cotacao) throws SQLException {
        String sql = "INSERT INTO cotacoes (segurado_id, seguro_tipo, valor_final, corr
etor_id) VALUES (?, ?, ?, ?)";
        PreparedStatement stmt = connection.prepareStatement(sql);
        // Assumindo que Segurado e Corretor já estão salvos e possuem IDs
        stmt.setInt(1, cotacao.getSegurado().getId());
        stmt.setString(2, cotacao.getSeguro().getTipo());
        stmt.setDouble(3, cotacao.getValorFinal());
        stmt.setInt(4, cotacao.getCorretor().getId());
        stmt.executeUpdate();
        stmt.close();
    }

    // Métodos adicionais: buscar, atualizar, deletar cotacoes
}
```

3.6. Service (Relatórios e Gráficos)

- **RelatorioService.java**

```
java
Copiar código
package com.sementedobrasil.service;

import com.sementedobrasil.dao.CotacaoDAO;
import com.sementedobrasil.model.Cotacao;
import java.util.List;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.JFreeChart;
import org.jfree.data.category.DefaultCategoryDataset;

public class RelatorioService {
    private CotacaoDAO cotacaoDAO;

    public RelatorioService() {
        this.cotacaoDAO = new CotacaoDAO();
    }

    public JFreeChart gerarGraficoCotasPorIdade() throws Exception {
        List<Cotacao> cotacoes = cotacaoDAO.buscarTodasCotacoes();
        DefaultCategoryDataset dataset = new DefaultCategoryDataset();

        for (Cotacao c : cotacoes) {
            int idade = Period.between(c.getSegurado().getDataNascimento(), LocalDate.n
ow()).getYears();
            String faixaEtaria;
            if (idade <= 30) faixaEtaria = "18-30";
            else if (idade <= 50) faixaEtaria = "31-50";
            else if (idade <= 65) faixaEtaria = "51-65";
```

```
            else faixaEtaria = "66-80";

            dataset.incrementValue(1, "Cotacoes", faixaEtaria);
        }

        JFreeChart barChart = ChartFactory.createBarChart(
            "Cotações por Faixa Etária",
            "Faixa Etária",
            "Número de Cotações",
            dataset
        );

        return barChart;
    }

    // Outros métodos para gerar diferentes gráficos e relatórios
}
```

3.7. Util (Utilitários)

- **DatabaseConnection.java**

```
java
Copiar código
package com.sementedobrasil.util;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DatabaseConnection {
    private static final String URL = "jdbc:mysql://localhost:3306/seguradora";
    private static final String USER = "root";
    private static final String PASSWORD = "password";

    private static Connection connection = null;

    public static Connection getConnection() {
        if (connection == null) {
            try {
                Class.forName("com.mysql.cj.jdbc.Driver");
                connection = DriverManager.getConnection(URL, USER, PASSWORD);
            } catch (ClassNotFoundException | SQLException e) {
                e.printStackTrace();
            }
        }
        return connection;
    }
}
```

- **ValidationUtil.java**

```
java
Copiar código
package com.sementedobrasil.util;
```

```
public class ValidationUtil {
    public static boolean isCPFValid(String cpf) {
        // Implementar validação de CPF
        return true; // Placeholder
    }

    public static boolean isEmailValid(String email) {
        return email.matches("^[A-Za-z0-9+_.-]+@(.+)$");
    }

    // Outros métodos de validação
}
```

- **ChartUtil.java**

```
java
Copiar código
package com.sementedobrasil.util;

import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;

public class ChartUtil {
    public static ChartPanel criarChartPanel(JFreeChart chart) {
        return new ChartPanel(chart);
    }
}
```

3.8. Main (Ponto de Entrada)

- **Application.java**

```
java
Copiar código
package com.sementedobrasil.main;

import com.sementedobrasil.view.CadastroSeguradoView;
import com.sementedobrasil.service.SeguradoService;
import com.sementedobrasil.controller.CadastroSeguradoController;
import javax.swing.SwingUtilities;

public class Application {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                CadastroSeguradoView view = new CadastroSeguradoView();
                SeguradoService service = new SeguradoService();
                new CadastroSeguradoController(view, service);
                view.setVisible(true);
            }
        });
    }
}
```

4. Definição das Classes e Interfaces Principais

4.1. Model Classes

- **Segurado.java**
  - **Atributos:** nome, dataNascimento, genero, email, telefone, cpf, cep
  - **Métodos:** Getters, Setters, Construtores
- **Seguro.java**
  - **Atributos:** tipo, valorBase, capitaisSegurados, assistencias
  - **Métodos:** Getters, Setters, Construtores
- **Cotacao.java**
  - **Atributos:** segurado, seguro, valorFinal, corretor, dependentes
  - **Métodos:** Getters, Setters, Construtores
- **Dependente.java**
  - **Atributos:** nome, dataNascimento, relacionamento
  - **Métodos:** Getters, Setters, Construtores

4.2. DAO Interfaces e Classes

- **SeguradoDAO.java**
  - **Métodos Principais:** save(Segurado segurado), findByCPF(String cpf), findAll(), update(Segurado segurado), delete(String cpf)
- **CotacaoDAO.java**
  - **Métodos Principais:** save(Cotacao cotacao), findById(int id), findAll(), update(Cotacao cotacao), delete(int id)

4.3. Service Interfaces e Classes

- **SeguradoService.java**
  - **Métodos Principais:** salvarSegurado(Segurado segurado), buscarSegurado(String cpf), listarSegurados(), atualizarSegurado(Segurado segurado), removerSegurado(String cpf)
- **CotacaoService.java**
  - **Métodos Principais:** calcularCotacao(Segurado segurado, Seguro seguro), buscarCotacao(int id), listarCotacoes(), atualizarCotacao(Cotacao cotacao), removerCotacao(int id)
- **RelatorioService.java**
  - **Métodos Principais:** gerarGraficoCotasPorIdade(), gerarRelatorioGerencial(), etc.

4.4. Controller Classes

- **CadastroSeguradoController.java**
  - **Função:** Controlar o fluxo de cadastro do segurado, validar dados e interagir com o serviço.
- **CadastroCotacaoController.java**
  - **Função:** Controlar o fluxo de cadastro de cotação, calcular valores e interagir com o serviço.
- **CotacaoController.java**
  - **Função:** Gerenciar visualizações e interações relacionadas às cotações.
- **GerenciamentoController.java**
  - **Função:** Controlar ações de aprovação/rejeição de cotações e gerenciamento de contas.
- **RelatoriosController.java**
  - **Função:** Gerenciar a geração e exibição de relatórios e gráficos.

4.5. View Classes

Cada classe de View representará uma janela ou painel da interface gráfica, desenvolvida com o WindowBuilder. Elas serão interligadas pelos Controllers.

4.6. Util Classes

- **DatabaseConnection.java:** Gerencia a conexão com o banco de dados.
- **ValidationUtil.java:** Fornece métodos de validação de dados (e.g., CPF, email).
- **ChartUtil.java:** Facilita a criação e exibição de gráficos utilizando bibliotecas como JFreeChart.

5. Exemplos de Implementação

5.1. CadastroCotacaoView.java

```
java
Copiar código
package com.sementedobrasil.view;

import javax.swing.*;
import java.awt.event.*;

public class CadastroCotacaoView extends JFrame {
    private JComboBox<String> profissaoBox;
    private JTextField salarioField;
    private JComboBox<String> condicaoSaudeBox;
    private JComboBox<String> corretorBox;
    private JButton calcularButton;

    public CadastroCotacaoView() {
        setTitle("Cadastro para Cotação");
        setSize(400, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new GridLayout(5, 2));

        // Inicialização dos componentes
        profissaoBox = new JComboBox<>(new String[]{"Administrativo", "Entrega", "Construç
ão Civil"});
        salarioField = new JTextField();
        condicaoSaudeBox = new JComboBox<>(new String[]{"Boa", "Leve", "Grave"});
        corretorBox = new JComboBox<>();
        calcularButton = new JButton("Calcular Cotação");

        // População do corretorBox com base no CEP (simulação)
        // Em implementação real, consultar base de dados ou API
        corretorBox.addItem("Corretor A");
        corretorBox.addItem("Corretor B");
        corretorBox.addItem("Corretor C");

        // Adicionando componentes à tela
        add(new JLabel("Profissão:"));
        add(profissaoBox);
        add(new JLabel("Salário:"));
        add(salarioField);
        add(new JLabel("Condição de Saúde:"));
        add(condicaoSaudeBox);
        add(new JLabel("Escolher Corretor:"));
        add(corretorBox);
        add(new JLabel(""));
        add(calcularButton);
```

```
        // Adicionando ActionListener para o botão calcular
        calcularButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                // Lógica para calcular cotação
                // Validação e chamada ao Controller
            }
        });
    }

    // Métodos para acessar os campos
    public String getProfissao() { return (String) profissaoBox.getSelectedItem(); }
    public double getSalario() {
        try {
            return Double.parseDouble(salarioField.getText());
        } catch (NumberFormatException e) {
            return 0.0;
        }
    }
    public String getCondicaoSaude() { return (String) condicaoSaudeBox.getSelectedItem();
}
    public String getCorretor() { return (String) corretorBox.getSelectedItem(); }

    // Método para adicionar ActionListener no botão calcular
    public void addCalcularListener(ActionListener listener) {
        calcularButton.addActionListener(listener);
    }
}
```

5.2. CadastroCotacaoController.java

```
java
Copiar código
package com.sementedobrasil.controller;

import com.sementedobrasil.view.CadastroCotacaoView;
import com.sementedobrasil.model.Segurado;
import com.sementedobrasil.model.Seguro;
import com.sementedobrasil.model.Cotacao;
import com.sementedobrasil.service.CotacaoService;
import javax.swing.*;
import java.awt.event.*;

public class CadastroCotacaoController {
    private CadastroCotacaoView view;
    private CotacaoService service;
    private Segurado segurado; // Assumindo que o segurado já foi cadastrado

    public CadastroCotacaoController(CadastroCotacaoView view, CotacaoService service, Seg
urado segurado) {
        this.view = view;
        this.service = service;
        this.segurado = segurado;
        initController();
    }
```

```
    private void initController() {
        view.addCalcularListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                calcularCotacao();
            }
        });
    }

    private void calcularCotacao() {
        try {
            String profissao = view.getProfissao();
            double salario = view.getSalario();
            String condicaoSaude = view.getCondicaoSaude();
            String corretor = view.getCorretor();

            if (salario <= 0) {
                JOptionPane.showMessageDialog(view, "Salário inválido.", "Erro", JOptionPa
ne.ERROR_MESSAGE);
                return;
            }

            // Criação do objeto Seguro
            Seguro seguro = new Seguro();
            seguro.setProfissao(profissao);
            seguro.setSalario(salario);
            seguro.setCondicaoSaude(condicaoSaude);

            // Chamada ao serviço para calcular cotação
            Cotacao cotacao = service.calcularCotacao(segurado, seguro);
            cotacao.setCorretor(null); // Associe o corretor conforme implementação

            // Exibir resultados na CotacaoView
            // Pode-se abrir uma nova janela ou atualizar a atual
            CotacaoView cotacaoView = new CotacaoView(cotacao);
            cotacaoView.setVisible(true);
            view.dispose(); // Fecha a janela de cadastro de cotação
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(view, "Erro ao calcular cotação: " + ex.getMessa
ge(), "Erro", JOptionPane.ERROR_MESSAGE);
        }
    }
}
```

6. Relacionamentos Entre Classes

- **Segurado** possui múltiplos **Cotacao**.
- **Cotacao** está associada a um **Seguro** e a um **Corretor**.
- **Seguro** possui múltiplos **CapitalSeguro** e **Assistencia**.
- **Cotacao** pode ter múltiplos **Dependente**.

7. Implementação das Regras de Cálculo

No `CotacaoService`, como já exemplificado anteriormente, implementamos os multiplicadores baseados nos requisitos:

- **Faixa Etária:** Calcula a idade do segurado a partir da data de nascimento.
- **Gênero:** Aplica desconto para mulheres.

- **Profissão:** Multiplicador baseado no risco associado à profissão.
- **Salário:** Multiplicador baseado no valor do salário.
- **Condição de Saúde:** Multiplicador baseado na condição de saúde declarada.

8. Gerenciamento de Interface com WindowBuilder

Utilize o **WindowBuilder** no Eclipse para criar as interfaces gráficas de cada View. Alguns passos gerais:

1. **Criar Novos Formulários:**
   - Clique com o botão direito no pacote `view` → `New` → `Other` → `WindowBuilder` → `Swing Designer` → `JFrame`.
   - Nomeie o formulário conforme a View (e.g., `CadastroSeguradoView`).
2. **Adicionar Componentes:**
   - Arraste e solte componentes (JLabel, JTextField, JComboBox, JButton) conforme os requisitos de cada página.
   - Configure propriedades e layouts para organizar os componentes de forma intuitiva.
3. **Interligar com o Controller:**
   - No código gerado pelo WindowBuilder, adicione métodos públicos para acessar os campos e botões, permitindo que o Controller adicione `ActionListeners`.

9. Exemplos de Regras de Negócio com Valores Fictícios

Faixa Etária:

- **18-30 anos:** Multiplicador = 1.0
- **31-50 anos:** Multiplicador = 1.2
- **51-65 anos:** Multiplicador = 1.5
- **66-80 anos:** Multiplicador = 2.0

Gênero:

- **Feminino:** Multiplicador = 0.9
- **Masculino:** Multiplicador = 1.0

Profissão:

- **Baixo Risco (Administrativo):** Multiplicador = 1.0
- **Médio Risco (Entrega):** Multiplicador = 1.2
- **Alto Risco (Construção Civil):** Multiplicador = 1.5

Salário:

- **Até R$3.000:** Multiplicador = 1.0
- **R$3.001 - R$10.000:** Multiplicador = 1.1
- **Acima de R$10.000:** Multiplicador = 1.3

Condição de Saúde:

- **Boa:** Multiplicador = 1.0
- **Leve (e.g., hipertensão controlada):** Multiplicador = 1.2
- **Grave (e.g., doenças cardíacas):** Multiplicador = 1.5

Fórmula de Cálculo:

Valor Cotac¸a˜o=Valor Base×Multiplicador Idade×Multiplicador Geˆnero×Multiplicador Profissa˜o×Multiplicador Salario×Cotação} = \text{Valor Base} \times \text{Multiplicador Idade} \times \text{Multiplicador Gênero} \times \text{Multiplicador Profissão} \times \text{Multiplicador Salario} \times \text{Multiplicador Saude}

Valor Cotac¸
a˜o=Valor Base×Multiplicador Idade×Multiplicador Geˆnero×Multiplicador Profissa˜o×Multiplicador Salario×Multiplicadc

Exemplo:

- **Valor Base:** R$100
- **Segurado:** 35 anos, masculino, profissão de médio risco, salário de R$5.000, condição leve de saúde.
- **Cálculo:**

100×1.2×1.0×1.2×1.1×1.2=R$190,08100 \times 1.2 \times 1.0 \times 1.2 \times 1.1 \times 1.2 = R\$190,08

100×1.2×1.0×1.2×1.1×1.2=R$190,08

10. Considerações Adicionais

10.1. Persistência de Dados

- **Banco de Dados:** Utilize MySQL ou PostgreSQL para armazenar os dados.
- **Tabelas Principais:**
  - `segurados`
  - `cotacoes`
  - `seguros`
  - `dependentes`
  - `corretores`
  - `capitais_segurados`
  - `assistencias`

10.2. Segurança

- **Validação de Dados:** Assegure que todos os inputs do usuário sejam validados para evitar SQL Injection e outros ataques.
- **Autenticação e Autorização:** Implementar mecanismos para que apenas administradores autorizados possam acessar funcionalidades de gerenciamento e relatórios.

10.3. Relatórios e Gráficos

- **Biblioteca de Gráficos:** Utilize bibliotecas como JFreeChart para gerar gráficos dinâmicos.
- **Funções de Relatório:** Permitir que administradores gerem relatórios de cotações por faixa etária, gênero, profissão, etc.

10.4. Testes

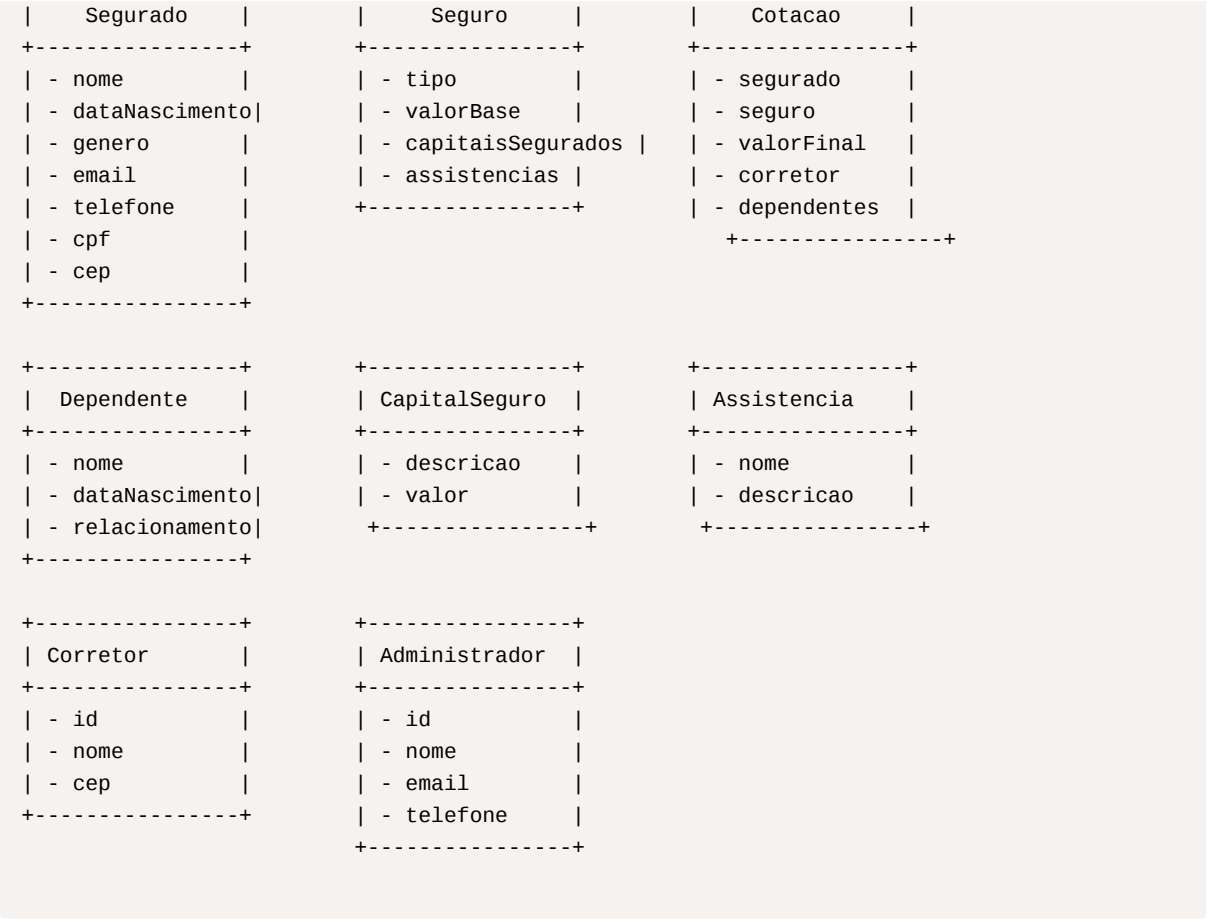
- **Unitários:** Utilize JUnit para testar as classes de serviço e DAO.
- **Integração:** Assegure que a interação entre componentes (View, Controller, Service, DAO) funcione corretamente.
- **Interface:** Teste a usabilidade das interfaces gráficas para garantir uma experiência intuitiva.

10.5. Documentação

- **Diagrama de Classe:** Crie um diagrama de classe UML para visualizar as relações entre as classes.
- **Diagrama de Caso de Uso:** Como discutido anteriormente, para representar as interações dos atores com o sistema.
- **Documentação Técnica:** Descreva a arquitetura, pacotes, classes e métodos principais.

11. Exemplo de Diagrama de Classe (Texto)

```
lua
Copiar código
+----------------+        +----------------+        +----------------+
```



12. Próximos Passos

1. **Implementação das Classes e Pacotes:**
   - Comece implementando as classes do **Model**.
   - Configure a conexão com o banco de dados no pacote **util**.
   - Implemente os DAOs para interação com o banco.
   - Desenvolva as Views usando o WindowBuilder.
   - Vincule as Views com os Controllers, que por sua vez interagem com os Serviços.
2. **Desenvolvimento Incremental:**
   - Adote sprints curtos (por exemplo, de 3 dias) para desenvolver funcionalidades específicas.
   - Realize revisões ao final de cada sprint para garantir que os requisitos estão sendo atendidos.
3. **Testes:**
   - Escreva testes unitários para as classes de serviço e DAO.
   - Realize testes manuais das interfaces gráficas.
4. **Documentação e Preparação da Apresentação:**
   - Documente o sistema com diagramas de classe e caso de uso.
   - Prepare slides que mostrem a arquitetura, funcionalidades principais e benefícios do sistema.
5. **Simulação e Refinamento:**
   - Teste o sistema com dados fictícios para garantir que os cálculos de cotação estão corretos.
   - Refine a interface gráfica para melhorar a usabilidade e aparência.

13. Exemplos de Interfaces e Métodos Adicionais

13.1. CotacaoView.java

```
java
Copiar código
package com.sementedobrasil.view;

import javax.swing.*;
import java.awt.event.*;
import com.sementedobrasil.model.Cotacao;

public class CotacaoView extends JFrame {
    private JLabel saudacaoLabel;
    private JLabel tipoSeguroLabel;
    private JTextArea resultadoCalculoArea;
    private JPanel capitaisPanel;
    private JPanel assistenciasPanel;
    private JButton personalizarCoberturaButton;
    private JButton escolherDependentesButton;

    public CotacaoView(Cotacao cotacao) {
        setTitle("Cotação de Seguro de Vida");
        setSize(600, 600);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());

        // Saudação
        saudacaoLabel = new JLabel("Bem-vindo, " + cotacao.getSegurado().getNome() + "! Se
u corretor: " + cotacao.getCorretor().getNome());
        add(saudacaoLabel, BorderLayout.NORTH);

        // Tipo do Seguro e Resultado dos Cálculos
        JPanel centralPanel = new JPanel(new GridLayout(2, 1));
        tipoSeguroLabel = new JLabel("Tipo do Seguro: " + cotacao.getSeguro().getTipo());
        resultadoCalculoArea = new JTextArea("Valor Final da Cotação: R$ " + cotacao.getVa
lorFinal());
        centralPanel.add(tipoSeguroLabel);
        centralPanel.add(resultadoCalculoArea);
        add(centralPanel, BorderLayout.CENTER);

        // Capitais Segurados
        capitaisPanel = new JPanel();
        capitaisPanel.setLayout(new BoxLayout(capitaisPanel, BoxLayout.Y_AXIS));
        capitaisPanel.add(new JLabel("Capitais Segurados:"));
        for (CapitalSeguro cs : cotacao.getSeguro().getCapitaisSegurados()) {
            capitaisPanel.add(new JLabel(cs.getDescricao() + ": R$ " + cs.getValor()));
        }
        add(capitaisPanel, BorderLayout.WEST);

        // Assistências
        assistenciasPanel = new JPanel();
        assistenciasPanel.setLayout(new BoxLayout(assistenciasPanel, BoxLayout.Y_AXIS));
        assistenciasPanel.add(new JLabel("Assistências:"));
        for (Assistencia a : cotacao.getSeguro().getAssistencias()) {
            assistenciasPanel.add(new JLabel(a.getNome() + " - " + a.getDescricao()));
        }
        add(assistenciasPanel, BorderLayout.EAST);

        // Botões
```

```
        JPanel botoesPanel = new JPanel();
        personalizarCoberturaButton = new JButton("Personalizar Cobertura");
        escolherDependentesButton = new JButton("Escolher Dependentes");
        botoesPanel.add(personalizarCoberturaButton);
        botoesPanel.add(escolherDependentesButton);
        add(botoesPanel, BorderLayout.SOUTH);

        // ActionListeners para os botões
        personalizarCoberturaButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                // Implementar personalização de cobertura
            }
        });

        escolherDependentesButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                // Implementar escolha de dependentes
            }
        });
    }

    // Métodos para atualizar a View conforme necessário
}
```

14. Banco de Dados (Schema Básico)

schema.sql

```
sql
Copiar código
CREATE DATABASE seguradora;

USE seguradora;

CREATE TABLE segurados (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(100) NOT NULL,
    data_nascimento DATE NOT NULL,
    genero VARCHAR(10) NOT NULL,
    email VARCHAR(100) NOT NULL,
    telefone VARCHAR(20),
    cpf VARCHAR(14) UNIQUE NOT NULL,
    cep VARCHAR(8) NOT NULL
);

CREATE TABLE corretores (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(100) NOT NULL,
    cep VARCHAR(8) NOT NULL,
    email VARCHAR(100),
    telefone VARCHAR(20)
);

CREATE TABLE segurros (
    id INT AUTO_INCREMENT PRIMARY KEY,
    tipo VARCHAR(50) NOT NULL,
    valor_base DOUBLE NOT NULL,
```

```
    profissao VARCHAR(50) NOT NULL,
    salario DOUBLE NOT NULL,
    condicao_saude VARCHAR(50) NOT NULL
);

CREATE TABLE cotacoes (
    id INT AUTO_INCREMENT PRIMARY KEY,
    segurado_id INT,
    seguro_id INT,
    valor_final DOUBLE NOT NULL,
    corretor_id INT,
    FOREIGN KEY (segurado_id) REFERENCES segurados(id),
    FOREIGN KEY (seguro_id) REFERENCES segurros(id),
    FOREIGN KEY (corretor_id) REFERENCES corretores(id)
);

CREATE TABLE dependentes (
    id INT AUTO_INCREMENT PRIMARY KEY,
    cotacao_id INT,
    nome VARCHAR(100) NOT NULL,
    data_nascimento DATE NOT NULL,
    relacionamento VARCHAR(50),
    FOREIGN KEY (cotacao_id) REFERENCES cotacoes(id)
);

CREATE TABLE capitais_segurados (
    id INT AUTO_INCREMENT PRIMARY KEY,
    seguro_id INT,
    descricao VARCHAR(100) NOT NULL,
    valor DOUBLE NOT NULL,
    FOREIGN KEY (seguro_id) REFERENCES segurros(id)
);

CREATE TABLE assistencias (
    id INT AUTO_INCREMENT PRIMARY KEY,
    seguro_id INT,
    nome VARCHAR(100) NOT NULL,
    descricao TEXT,
    FOREIGN KEY (seguro_id) REFERENCES segurros(id)
);
```

15. Conclusão

Com essa estrutura, seu sistema estará bem organizado, facilitando a manutenção e escalabilidade. A separação de pacotes e a clara definição de responsabilidades das classes e interfaces garantem que cada parte do sistema funcione de maneira coesa e eficiente.

Próximos Passos:

1. **Configuração do Ambiente:**
   - Configure o banco de dados utilizando o `schema.sql`.
   - Assegure que as dependências necessárias (e.g., bibliotecas JDBC, JFreeChart) estejam incluídas no projeto.
2. **Implementação Iterativa:**
   - Siga as etapas definidas para desenvolver funcionalidades por sprints, priorizando os requisitos essenciais.
   - Realize reuniões rápidas diárias (Daily Standups) para monitorar o progresso e ajustar o plano conforme necessário.

3. **Testes e Validações:**
    - Escreva e execute testes unitários e de integração para garantir que cada componente funcione corretamente.
    - Teste a interface gráfica para assegurar uma experiência de usuário intuitiva e sem erros.
4. **Documentação e Apresentação:**
    - Complete a documentação técnica com diagramas UML e descrições detalhadas das funcionalidades.
    - Prepare uma apresentação destacando a arquitetura do sistema, funcionalidades principais, e os resultados obtidos.