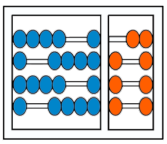
Universidade Estadual de Campinas
Instituto de Computação



Laboratório 5

Seguradora - Relacionamentos e Abstração

MC322 - Programação Orientada a Objetos

---

1 Descrição Geral

Nas atividades deste laboratório, iremos focar em conceitos de Orientação a Objetos vistos em classe, como: Classes abstratas e relacionamentos entre classes (Herança, Associação, Agregação e Composição). Esses conceitos, junto com uma refatoração das classes, proporcionarão maior robustez ao Sistema de Seguradora. Para ilustrar as novas implementações que estão inclusas neste laboratório, a Figura 1 apresenta o diagrama de classes[1].

Observe que no diagrama, novas classes foram inseridas: Frota; Condutor; Seguro, bem como Seguro-PF e Seguro-PJ. Nesse novo contexto Cliente-PJ faz Seguro-PJ de Frota para uma lista de Condutor, e Cliente-PF faz Seguro-PF de Veiculo para uma lista de Condutor. A nova classe Seguro fica responsável por armazenar e gerenciar a relação entre Condutor e Sinistro. Quando houver necessidade, um sinistro deve ser criado a partir da classe Seguro e o objeto condutor envolvido deve atualizar sua lista de sinistros. A classe Seguro também deve ficar responsável por calcular o valor do Seguro e deve fazer isso de acordo com os atributos das classes envolvidas. As classes Seguro e Cliente devem ser implementadas como classes abstratas.

Neste Laboratório (05), utilizaremos uma versão refatorada das classes do Laboratório anterior (04). Esse laboratório visa fixar os conceitos vistos até então e apresentar novos conceitos como:

1. **Relacionamentos entre Classes:** Tipos de relacionamentos (Herança, Associação, Agregação e Composição) assim como Cardinalidade dos relacionamentos;
2. **Classes Abstratas:** As classes Cliente e Seguro devem ser implementadas como classes abstratas;
3. **Operações entre classes:** Operações como cadastrar Cliente, Veículo e Gerar Sinistro devem automaticamente atualizar os atributos das classes envolvidas. Por exemplo, ao se gerar um seguro, o atributo listaSeguros da classe Seguradora deve ser atualizado automaticamente.

2 Objetivos

Os objetivos principais do Laboratório 5 são os seguintes:

- Consolidação dos conteúdos vistos nos labs anteriores;
- Capacidade de refatorar um projeto;
- Aplicação de diferentes relacionamentos entre classes;
- Aplicação do conceito de abstração;
- Capacidade de abstração de como aplicar os conceitos de orientação objetos em face das funcionalidades requeridas.

3 Atividades

As atividades a serem desenvolvidas para este Laboratório são as seguintes:

- Refatoração do Sistema de Seguros;
  - Reimplementar classes vistas nos laboratórios anteriores e implementar novas classes propostas;

[1] Note que algumas das funcionalidades requeridas não estão explicitamente demonstradas no diagrama de classes. Tal abordagem visa um dos objetivos desse laboratório, o qual é a ampliação da capacidade de abstração por parte dos alunos em como aplicar os conceitos visto em aula para a resolução de problemas.

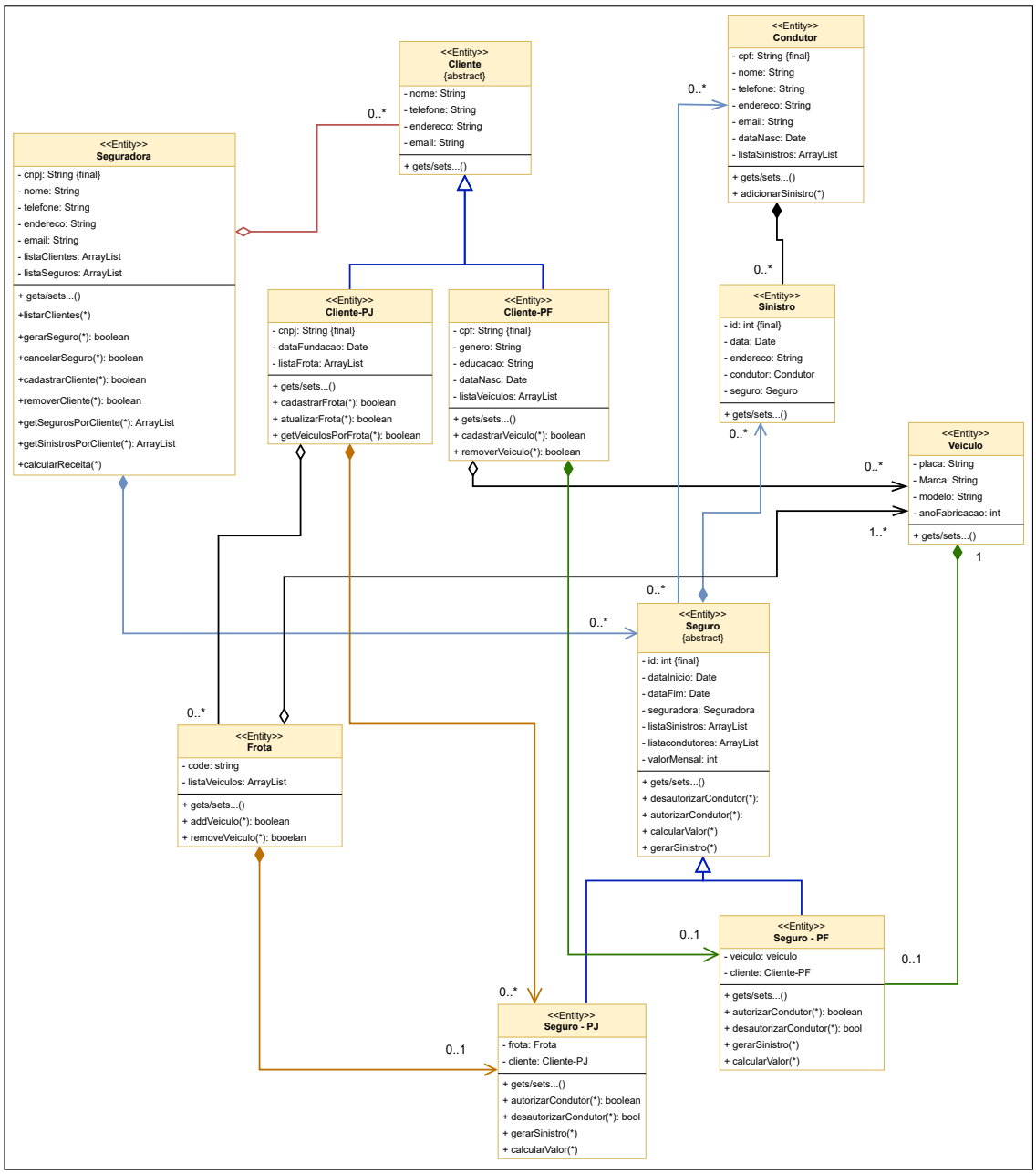


Figura 1: Diagrama de Classe - Sistema da Seguradora - Relacionamentos e Abstração

- Abstração das funcionalidades exigidas: para esse laboratório muitas funcionalidades necessárias não estarão explicitas neste PDF, porém é exigido:
  - Que todas as classes contenham um método ToString();
  - Devem ser implementados métodos Gets e Setters para todos os atributos das classes que convenham ter Gets e Sets;
  - Os parâmetros dos métodos exigidos devem ser abstraídos pelo aluno;
  - Criação de classes auxiliares para o funcionamento do projeto são altamente encorajadas. Ex: Classe para armazenar métodos estáticos de validar CPF e CNPJ;
  - Criação de um menu iterativo que seja capaz de realizar as principais operações do Projeto[2];

Na classe AppMain:

- Instanciar pelo menos 1 objeto de cada classe na ordem mais conveniente (é fortemente recomendado que se instancie múltiplos objetos para cada classe);

[2]As operações recomendadas são aquelas necessárias para instanciar as classes requeridas, assim como gerar seguro e sinistro e exibir detalhes dos objetos das classes

  - Seguros devem ser instanciados a partir do método apropriado da classe Seguradora;
  - Sinistros devem ser criados a partir do método apropriado da classe Seguro;

- Apresentar os detalhes de pelo menos 1 objeto de cada classe com o respectivo método ToString().
- Apresentar exemplos da utilização dos principais métodos das classes do Sistema de Seguradora.
- Apenas ao final da execução do projeto: chamar o menu de operações.

4 Observações

4.1 Atributo *valorMensal* e método *calcularValor()* em Seguro

O atributo *valorMensal* armazena o valor mensal do seguro. Esse valor deve ser calculado chamando o método *calculaValor()* no construtor do objeto da classe Seguro correspondente. Além disso, sempre que o *valorMensal* precisar ser atualizado, o método *calculaValor()* deve ser chamado.

Para clientes do tipo PF o valor do *calcularValor()* será dado por:

```
(VALOR_BASE * FATOR_IDADE * (1 + 1/(quantidadeVeiculos+2)) *
            (2 + quantidadeSinistrosCliente/10) *
            (5 + quantidadeSinistrosCondutor/10))

  sendo

  VALOR_BASE = 10
  FATOR_IDADE =
                [< 30 anos] = 1.25
                [30-60 anos] = 1.0
                [> 60 anos] = 1.5
```

,

Sendo *quantidadeVeiculos* a quantidade de veículos segurados que aquele cliente possui anteriormente cadastrados na seguradora. E *quantidadeSinistrosCliente* e *quantidadeSinistrosCondutor* deve ser respectivamente o número de sinistros que aquele cliente e condutores possuem registrados naquela Seguradora.

Para clientes do tipo PJ o valor do *calcularValor()* será dado por:

```
(VALOR_BASE * (10 + (quantidadeFunc)/10) *
            (1 + 1/(quantidadeVeiculos+2))  *
            (1 + 1/(AnosPosFundacao+2)) *
            (2 + quantidadeSinistrosCliente/10) *
            (5 + quantidadeSinistrosCondutor/10)
            )
  sendo

  VALOR_BASE = 10
```

,

Sendo *quantidadeVeiculos* a quantidade de veículos que aquele cliente possui cadastrados na frota e *AnosPosFundacao* a *idade* do cliente PJ (calculado com base no atributo dataFundacao). E *quantidadeSinistrosCliente* e *quantidadeSinistrosCondutor* deve ser respectivamente o número de sinistros que aquele cliente e condutores possuem registrados previamente naquela seguradora.

4.2 Método *calcularReceita()* em Seguradora

O método *calcularReceita()* será utilizado para mostrar o balanço de seguros de todos os clientes da Seguradora.

4.3 Método *atualizarFrota()* em Cliente-PJ

Além de adicionar e remover veículos em uma frota, este método também deve ser capaz de remover a frota inteira.

4.4 Listar atributos

Recomenda-se a criação de métodos que exibam na tela os itens de cada lista presente nas classes. Por exemplo um método que liste todos os Seguros que uma seguradora possuí.

4.5 Gerar lista de Seguros por Cliente em Seguradora

O código a seguir tem como objetivo demonstrar um método da classe Seguradora que retorna os seguros que um cliente específico possui. Não é recomendado que ele seja copiado, mas sim que seja usado como inspiração para criação dos métodos exigidos no diagrama de classes.

```
public List<Seguro> getSegurosPorCliente(Cliente cliente) {
    List<Seguro> segurosCliente = new ArrayList<>();
    for (Seguro seguro : seguros) {
        if (seguro.getCliente().equals(cliente)) {
            segurosCliente.add(seguro);
        }
    }
    return segurosCliente;
}
```

4.6 Classes que não estão no diagrama

Como pode-se notar o diagrama não contém certas classes exigidas em laboratórios anteriores. As funcionalidades dessas classes ainda estão sendo exigidas neste laboratório, porém a forma como elas estarão sendo implementadas fica a critério do aluno.

5 Avaliação

Além da correta execução do laboratório, os seguintes critérios serão utilizados para a composição da nota do laboratório:

- Entrega realizada dentro do prazo estipulado;
- Execução do código;
- Qualidade do código desenvolvido (saída dos dados na tela, tabulação, comentários);
- Instanciação dos objetos, e principais métodos das classes implementadas, na classe AppMain;
- Desenvolvimento correto dos métodos e classes requisitadas;

6 Entrega

- **A entrega do Laboratório é realizada exclusivamente via Github**. Para a submissão no Github, gere um release (tag) com a identificação do laboratório no estilo <lab05-RA>. Por exemplo, para o aluno com RA 123456, a tag será: lab05-123456.
- Observação: Evite criar releases enquanto não tiver certeza que seu código está funcionando como esperado.
- Utilize os horários de laboratório e atendimentos para tirar eventuais dúvidas de submissão e também relacionadas ao desenvolvimento do laboratório.
- **Prazo de Entrega:** 30/05 - 14h

6.1 Organização das pastas do repositório

É esperado que seu repositório do Github contenha a mesma estrutura de pastas dos Laboratórios anteriores.