UNICAMP

Universidade Estadual de Campinas
Instituto de Computação

INSTITUTO DE COMPUTAÇÃO

Laboratório 3

Seguradora - Cadastro de Clientes

MC322 - Programação Orientada a Objetos

---

1 Descrição Geral

No Laboratório anterior (02), foi desenvolvido a base de um Sistema de Seguradora de Veículos automotores. Nesse laboratório, por meio de 4 classes, Seguradora, Sinistro, Cliente e Veículo, um cadastro inicial de Clientes na Seguradora com seus respectivos Veículos foi possível. Ademais, a geração de IDs para os sinistros e o método para verificação de CPFs trouxeram um grau de complexidade ao laboratório. Contudo, as classes mencionadas acima não possuíam nenhum relacionamento entre si.

Sendo assim, para este laboratório (03), iremos explorar novos conceitos de Orientação Objetos vistos em classe, tais como: Coleção de objetos, variáveis estáticas e finais, alguns tipos de relacionamento entre em classes, em especial o conceito de Herança Simples. Tais conceitos implementados proporcionarão uma maior robustez ao Sistema de Cadastro de Clientes na Seguradora. Para ilustrar as novas implementações para este laboratório, a Figura 1 apresenta um diagrama de classe com o relacionamento entre as classes.

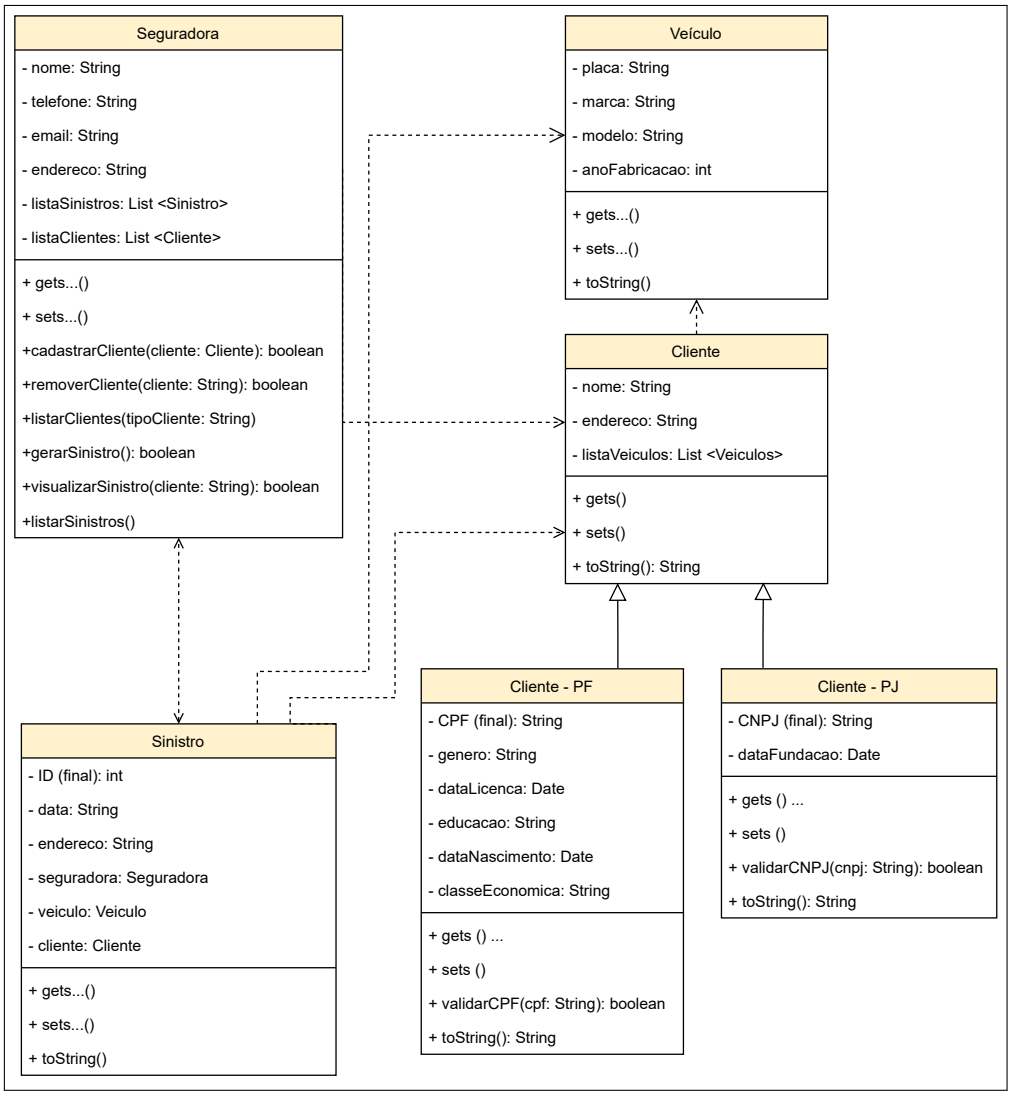

Figura 1: Diagrama de Classe - Cadastro de Clientes na Seguradora 2.0

Com base na Figura 1, duas novas classes foram adicionadas ao Sistema, ClientePF e ClientePJ. Essas classes abstraem o conceito de Pessoa Física (PF) e Jurídica (PJ). Ambas são classes herdeiras da classe Cliente, compartilhando assim atributos comuns e possuindo atributos próprios, bem como métodos próprios. Por exemplo, na Classe PF temos um método de validaCPF()[1], e por sua vez na Classe PJ teremos o método validaCNPJ(). Além das duas novas classes herdeiras da classe Cliente, temos relacionamentos de dependência entre as demais classes.

Por fim, há a adição de Coleção de Objetos. Na Classe Cliente há o *listaVeiculos*, ou seja, agora um cliente pode possuir n-veículos associados ao seu seguro. Na Classe Seguradora podemos armazenar também agora uma lista de Clientes (*listaClientes*) e uma lista de Sinistros (listaSinistros). A seguir apresentaremos os objetivos do laboratório e as atividades a serem desenvolvidas.

2 Objetivos

Os objetivos principais do Laboratório 3 são os seguintes:

- Aplicação do conceito de Herança Simples;
- Utilização de Coleção de Dados (ArrayList e LinkedList);
- Utilização de constantes (finals);
- Entrada de dados via teclado (System.in);
- Consolidação dos conteúdos abordados no Laboratório 02 (visibilidade de variáveis e métodos; instanciação de objetos; impressão de objetos utilizando *toString()*)

3 Atividades

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

- Atualização das classes Seguradora, Veículo, Sinistro e Cliente com os novos métodos e atributos apresentados na Figura 1;
- Implementação do conceito de Herança simples através da criação das classes ClientePF e ClientePJ, que devem herdar os atributos e métodos da classe Cliente;
- Implementação de um método na classe Main que seja capaz de ler dados do teclado (utilizando a classe Scanner no pacote System.in), e, de acordo com esse dados de entrada, exiba novo conteúdo na tela (um exemplo seria a implementação de um menu textual que liste opções para visualizar certos dados da Seguradora);

Na classe Main:

- Cadastrar e remover pelo menos 1 Cliente (ClientePF ou ClientePJ);
- Chamar os métodos *validarCPF(cpf: String)* (ClientePF) e *validarCNPJ(cnpj: String)* (ClientePJ);
- Adicionar pelo menos 1 Veiculo em cada Cliente instanciado;
- Instanciar pelo menos 1 objeto de Seguradora;
- Cadastrar pelo menos 2 clientes em Seguradora (sem remover), sendo 1 do tipo ClientePF e 1 do tipo ClientePJ;
- Gerar pelo menos 1 Sinistro;
- Chamar o método *toString()* de cada classe;
- Chamar os métodos *listarClientes (tipoCliente: String)*, *visualizarSinistro(cliente: String)* e *listarSinistros()* da classe Seguradora;
- Implementar e chamar um método que faça leitura de dados usando a função *System.In*.

[1] Esse método foi implementado no Laboratório 02

4 Exemplo de Herança Simples

Um exemplo de implementação da classe ClientePF pode ser observado no código abaixo. Repare que é utilizada a palavra reservada **extends** para indicar a relação de herança entre as classes. Além disso, as subclasses não herdam os construtores da superclasse (classe Cliente), então é necessário chamar explicitamente o construtor da superclasse como a primeira ação a ser realizada no construtor da subclasse. Isso acontece através do comando **super**.

```
1  public class ClientePF extends Cliente {
2      private String cpf;
3      private Date dataNascimento;
4
5      public ClientePF(String nome, String endereco, Date dataLicenca,
6                           String educacao, String genero, String classeEconomica,
7                           List<Veiculo> listaVeiculos, String cpf, Date dataNascimento) {
8           // chama o construtor da superclasse
9          super(nome, endereco, dataLicenca, educacao, genero, classeEconomica, listaVeiculos);
10         this.cpf = cpf;
11         this.dataNascimento = dataNascimento;
12     }
13
14     // TO DO:
15     // metodos getters e setters para cpf e dataNascimento
16     // ...
17
18     @Override
19     public String toString(){
20     // ...
21     }
22     public boolean validarCPF(String cpf){
23     // ...
24     }
25
26 }
```

Listing 1: ClientePF.java

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;
- Qualidade do código desenvolvido (tabulação, comentários);
- Instanciação dos objetos, e principais métodos das classes implementadas, na classe Main;
- Desenvolvimento correto dos métodos de acesso.

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 <lab03-RA>. Por exemplo, para o aluno com RA 123456, a tag será: lab03-123456.
- 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:** 18/04 - 14h

6.1 Organização das pastas do repositório

É esperado que seu repositório do Github contenha a seguinte estrutura de pastas: