Sistema de Apólices de Seguros

Programação Orientada a Objetos

Nome: Giulianno Raphael Sbrugnera RA: 408093

Nome: Gustavo Rodrigues RA: 489999

Nome: Matheus Casarin Paez RA: 438308

Nome: Rafael Paschoal Giordano RA: 408298

Turma: Bacharelado em Ciência da Computação

UFSCAR – Sorocaba

BCC 011 & BCC 012

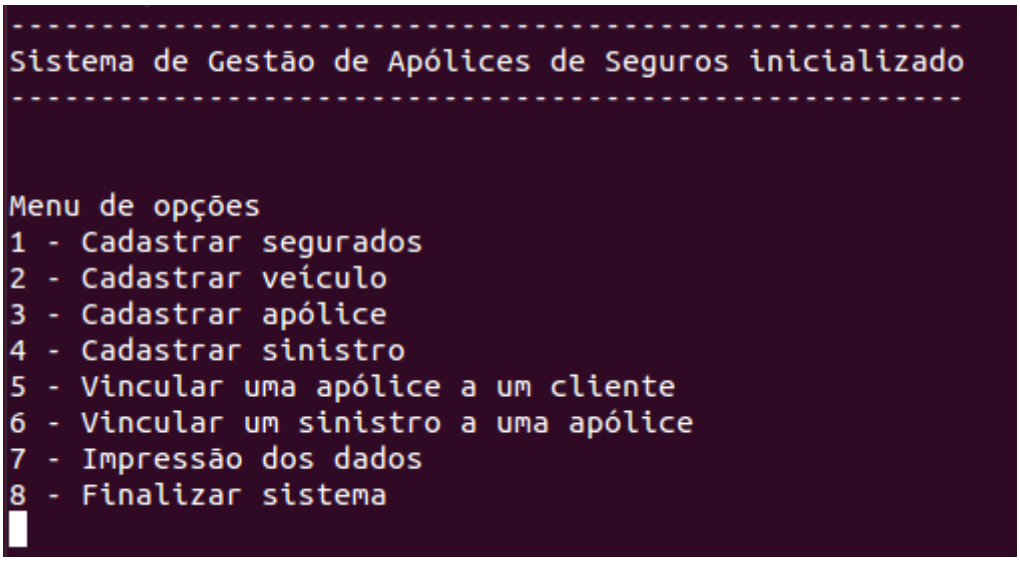
Descrição do Programa

O programa realizado é para desenvolver um sistema que dará suporte a gestão de apólices de seguros. Nele permite o cadastro de clientes, cadastro de veículos, cadastro de segurados, controle de sinistros, controle dos pagamentos, geração de faturas de pagamento, geração de apólices, vincula um cliente a uma apólice, faz o relacionamento entre o segurado a uma ou mais apólices, geração de orçamentos, além da pesquisa por número da apólice, CPF ou pela placa do veículo. Para o cadastro de cada cliente é necessário à solicitação do nome, telefone, endereço e CPF. Já para cadastrar o veículo é preciso solicitar a placa, o ano e o código do RENAVAM, além disso, é selecionada a marca do veículo (Fiat, Volkswagen, Ford, Honda) e o seu tipo (passeio ou transporte). Na geração da apólice é inserido o seu número, a vigência e a classificação da apólice (classe1 ou classe2).

Foi criada uma classe 'Data', onde se faz a atribuição e o retorno (*set* e *get*) do dia, mês e ano. Esta classe foi definida como um atributo das classes 'BO', 'Apolice' e 'Sinistro'.

Forma de utilização do programa

Para utilizar o programa é inicializado um "menu" contendo 8 opções:



1- Cadastra um segurado no sistema;

2- Cadastra um veículo no sistema;

3- Cadastra uma apólice no sistema;

4- Cadastra um sinistro no sistema;

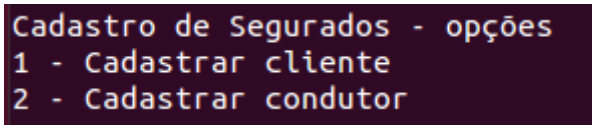
5- Vincula uma apólice específica à um cliente específico;

6- Vincula um sinistro específico à uma apólice específica;

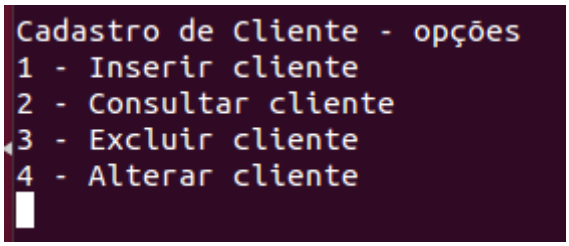
7- Imprimem todos os dados do sistema;

8- Finaliza o programa;

Para a inserção de segurados, o usuário deve escolher se o cadastrado será um cliente ou um condutor.

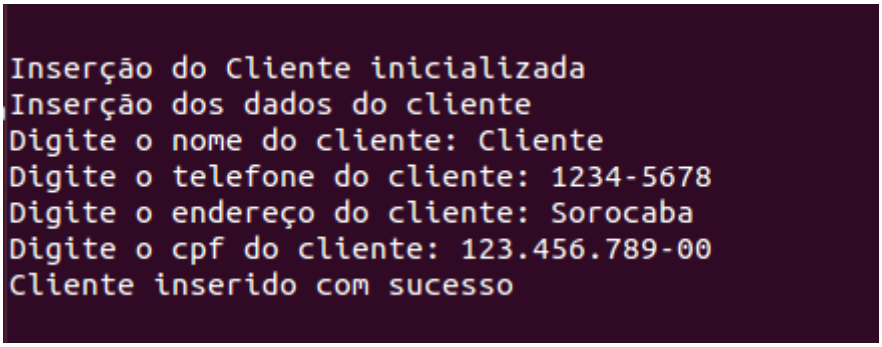


Caso o usuário escolha o cadastro de cliente, o seguinte “menu” de manutenção do objeto cliente aparecerá:



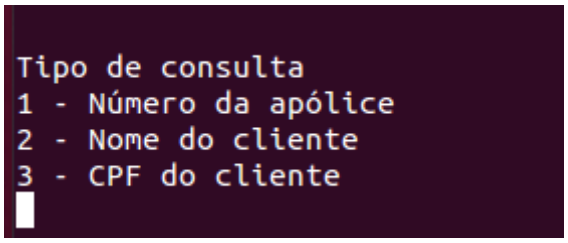
Neste “menu” o usuário pode manter o controle de todos os clientes, ou seja, poderá escolher se deseja inserir um novo cliente, consultar se um certo cliente já existe, editar um cliente já existente ou também poderá excluir um cliente inutilizado.

Caso o usuário escolha inserir um novo cliente, deverá informar o nome, telefone, endereço e o CPF do mesmo.

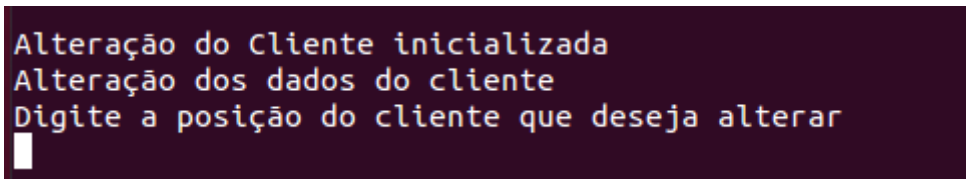


O sistema informará ao usuário caso o cliente seja inserido com sucesso.

Caso o usuário escolha o modo de consulta, ele terá as opções de consulta por número de apólice, por nome do cliente ou por CPF do cliente, como mostrado abaixo:

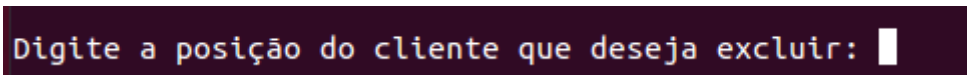


Caso a intenção do usuário seja alterar os dados de um certo cliente, ele deverá informar a posição do cliente dentro dos dados do sistema:



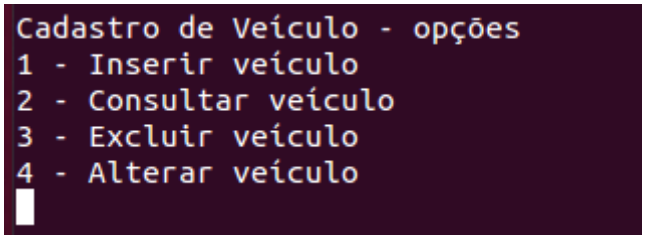
Após isso basta o usuário modificar os dados do usuário selecionado.

Assim como a alteração de dados, a exclusão do cliente é feita pela posição do cliente no sistema:



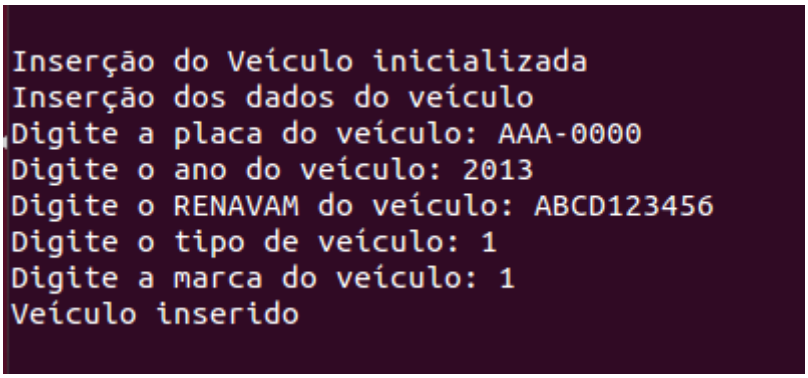
O sistema informará ao usuário caso o código seja inexistente.

Caso o usuário escolha o cadastro de veículos, o seguinte “menu” de manutenção aparecerá:

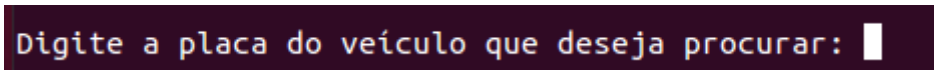


Neste caso, o “menu” servirá para manter o controle dos veículos no sistema.

Escolhendo inserir, o usuário deverá preencher os seguintes campos com as informações do veículo:

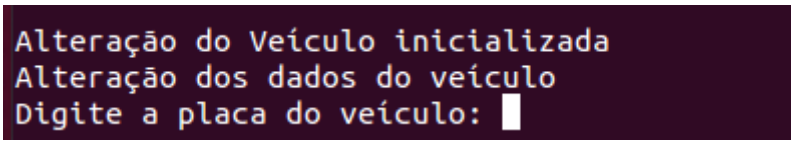


Escolhendo consultar um veículo, o usuário deve informar a placa do veículo que deseja pesquisar:



Após isso, o sistema retornará os dados caso o veículo seja encontrado.

Caso o usuário queira alterar os dados de um certo veículo, bastará apenas informar novamente os dados do veículo:



Caso o usuário escolha a opção Cadastrar Sinistro, o sistema solicitará todas as informações necessárias para o cadastro do mesmo, como: informações sobre o terceiro, boletim de ocorrência e sinistro.

```
Inserção do Sinistro inicializada
Inserção dos dados do terceiro
Digite a idade do terceiro: 30
Digite o nome do terceiro: Terceiro
Digite o telefone do terceiro: 1234-5678
Digite o endereco do terceiro: Sorocaba
Digite o nome da seguradora: Seguradora
Digite o CPF do terceiro: 123.456.789-00

Inserção dos dados do BO
Digite o número do BO: 1
Digite a data do BO
Dia: 1
Mês: 7
Ano: 2013
Digite o tipo do BO: Furto

Inserção dos dados do sinistro
Digite o número do sinistro: 1
Digite a data do sinistro
Dia: 1
Mês: 7
Ano: 2013
Digite o local do sinistro: Sorocaba
Digite o tipo do sinistro: 1

Sinistro inserido
```

O cliente também tem a opção de vincular uma apólice a um cliente. Caso o usuário escolha essa opção, o programa pedirá a posição do cliente desejado dentro dos dados do programa, para depois vincular, caso o cliente exista.

```
Digite a posição do cliente que deseja vincular a apólice:
```

O cliente pode optar por vincular um sinistro a uma apólice também. O sistema não irá requisitar nada.

Caso o sinistro seja vinculado, será mostrada uma mensagem informando o usuário.

```
Sinistro vinculado com sucesso
```

Caso contrário, será mostrada a mensagem de aviso:

```
Quantidade máxima de sinistros atingida!
```

Descrição dos algoritmos utilizados na resolução do problema proposto

Segue abaixo os métodos de cada classe com uma explicação sucinta sobre o que cada uma faz ou representa.

1. Apólice

1.1 Apolice::Apolice()

Construtor da classe zera os contadores e atribui *NULL* a todas as posições dos vetores de ponteiros.

1.2 Apolice::~Apolice()

Destrutor da classe devolve a memória alocada por cada ponteiro dos vetores de ponteiros, depois atribui *NULL*.

1.3 void Apolice::criarApolice(int x, int c, Data d, Veiculo *veic, Orcamento *orcam)

Chama as funções *set* da classe para atribuir valor aos atributos.

1.4 void Apolice::setNumApolice(int x)

Função *set* que modifica o atributo 'num'.

1.5 void Apolice::setClassificacaoApolice(int c)

Função *set* que modifica o atributo 'classificacao'.

1.6 void Apolice::setVigenciaApolice(Data d)

Função *set* que modifica o atributo 'vigencia'.

1.7 void Apolice::setVeiculoApolice(Veiculo *veic)

Função *set* para o atributo ‘veiculo’.

1.8 bool Apolice::setSinistroApolice(Sinistro *snt)

Função *set* para o atributo sinistro utiliza o contador para que cada índice do vetor de ponteiros aponte para o objeto corretamente.

1.9 bool Apolice::setOrcamentoApolice(Orcamento *orcam)

Função *set* para o atributo ‘orcamento’, utiliza o contador para que cada índice do vetor de ponteiros aponte para o objeto corretamente.

1.10 int Apolice::getNumApolice() const

Função *get* que retorna o número da apólice.

1.11 int Apolice::getClassificacaoApolice() const

Função *get* que retorna a classificação da apólice.

1.12 int Apolice::getQtdeSinistro() const

Função *get* que retorna a quantidade de sinistros da apólice.

1.13 int Apolice::getQtdeOrcamento() const

Função *get* que retorna a quantidade de orçamentos da apólice.

1.14 Data Apolice::getVigenciaApolice() const

Função *get* que retorna a data de vigência da apólice.

1.15 Veiculo* Apolice::getVeiculoApolice() const

Função *get* que retorna o veículo da apólice.

1.16 bool Apolice::gerarOrcamentoApolice()

Se não houver orçamentos na apólice, o método retorna *false*. Caso contrário, é impresso o acumulado do valor de cada orçamento.

1.17 void Apolice::gerarBoletoApolice()

Imprime as informações referentes ao boleto da apólice.

1.18 bool Apolice::pesquisarVeiculo(string placa)

Chama o método da classe 'Veiculo', que retorna a placa do veículo. A função retorna *true* se as placas forem iguais.

1.19 void Apolice::imprimirDadosApolice()

Função que imprime os atributos da classe 'Apolice'.

1.20 void Apolice::imprimirDadosOrcamentoApolice()

Função que imprime os atributos da classe 'Orcamento'.

1.21 void Apolice::imprimirDadosSinistroApolice()

Função que imprime os atributos da classe 'Sinistro'.

2. BO

2.1 void BO::criarBO(int x, Data d, string n)

Chama as funções *set* da classe para atribuir valores aos atributos.

2.2 void BO::setNumBO(int x)

Função *set* que modifica o atributo 'num'.

2.3 void BO::setDateBO(Data d)

Função *set* que modifica o atributo 'dateBO'.

2.4 void BO::setTipoBO(string n)

Função *set* que modifica o atributo 'tipo'.

2.5 int BO::getNumBO() const

Função *get* que retorna o número da apólice.

2.6 Data BO::getDateBO() const

Função *get* que retorna a data do BO.

2.7 string BO::getTipoBO() const

Função *get* que retorna o tipo do BO.

2.8 void BO::imprimirDadosBO()

Método que imprime os atributos referentes à classe.

3. Cliente

3.1 Cliente::Cliente()

Construtor da classe zera o contador e atribui *NULL* a todas as posições do vetor de ponteiros.

3.2 Cliente::~Cliente()

Destrutor da classe devolve a memória alocada por cada ponteiro do vetor de ponteiros, depois atribui *NULL*.

3.3 void Cliente::setNomeCliente(string n)

Função *set* que modifica o atributo 'nome'.

3.4 void Cliente::setTelefoneCliente(string t)

Função *set* que modifica o atributo 'telefone'.

3.5 void Cliente::setEnderecoCliente(string e)

Função *set* que modifica o atributo 'endereco'.

3.6 bool Cliente::setApoliceCliente(Apolice *ap)

Função *set* para o atributo ‘apolice’, utiliza o contador para que cada índice do vetor de ponteiros aponte para o objeto corretamente.

3.7 void Cliente::setCPFCliente(string cpf)

Função *set* que modifica o atributo 'CPF'.

3.8 string Cliente::getNomeCliente() const

Função *get* que retorna o nome do cliente.

3.9 string Cliente::getTelefoneCliente() const

Função *get* que retorna o telefone do cliente.

3.10 string Cliente::getEnderecoCliente() const

Função *get* que retorna o endereço do cliente.

3.11 string Cliente::getCPFCliente() const

Função *get* que retorna o CPF do cliente.

3.12 int Cliente::getQtdeApoliceCliente() const

Função *get* que retorna a quantidade de apólices do cliente.

3.13 bool Cliente::procurarVeiculoCliente(string placa)

Chama o método da classe ‘Apolice’, passando como parâmetro a placa. Retorna *true* se a placa for igual à placa passada como parâmetro.

3.14 bool Cliente::excluirApoliceCliente(int pos)

Dada a posição passada por parâmetro, um laço puxa os elementos logo à frente, sobrescrevendo a apólice da posição.

3.15 bool Cliente::alterarApoliceCliente(int pos, Apolice *ap)

Dada a posição passada por parâmetro e um ponteiro para a classe ‘Apolice’, na posição o vetor de ponteiro aponta agora para o ponteiro passado por referência.

3.16 void Cliente::imprimirDadosApoliceCliente()

Imprime os dados das apólices do cliente.

4. Condutor

4.1 Condutor::Condutor()

Construtor da classe zera o contador e atribui *NULL* a todas as posições do vetor de ponteiros.

4.2 Condutor::~Condutor()

Destrutor da classe devolve a memória alocada por cada ponteiro do vetor de ponteiros, depois atribui *NULL*.

4.3 void Condutor::setNomeCondutor(std::string n)

Função *set* que modifica o atributo 'nome'.

4.4 void Condutor::setTelefoneCondutor(std::string t)

Função *set* que modifica o atributo 'telefone'.

4.5 void Condutor::setEnderecoCondutor(std::string e)

Função *set* que modifica o atributo 'endereco'.

4.6 bool Condutor::setApoliceCondutor(Apolice *ap)

Função *set* para o atributo ‘apolice’, utiliza o contador para que cada índice do vetor de ponteiros aponte para o objeto corretamente.

4.7 void Condutor::setCNHCondutor(std::string cnh)

Função *set* que modifica o atributo 'numCNH'.

4.8 void Condutor::setIdadeCondutor(int x)

Função *set* que modifica o atributo 'idade'.

4.9 void Condutor::setCPFCondutor(std::string cpf)

Função *set* que modifica o atributo 'CPF'.

4.10 std::string Condutor::getNomeCondutor()

Função *get* que retorna o nome do condutor.

4.11 std::string Condutor::getTelefoneCondutor()

Função *get* que retorna o telefone do condutor.

4.12 std::string Condutor::getEnderecoCondutor()

Função *get* que retorna o endereço do condutor.

4.13 std::string Condutor::getCNHCondutor()

Função *get* que retorna o número da CNH do condutor.

4.14 int Condutor::getIdadeCondutor()

Função *get* que retorna a idade do condutor.

4.15 std::string Condutor::getCPFCondutor()

Função *get* que retorna o CPF do condutor.

5. CRUDTerceiro

5.1 CRUDTerceiro::CRUDTerceiro()

Construtor da classe aloca espaço suficiente para um objeto da classe Terceiro.

5.2 CRUDTerceiro::~CRUDTerceiro()

Destrutor da classe devolve a memória que foi alocada.

5.3 void CRUDTerceiro::criarTerceiro(int anos, std::string nomeTerceiro, std::string tel, std::string end, std::string seguroTerceiro, std::string cpf)

Chama as funções *set* da classe para modificar os atributos.

5.4 void CRUDTerceiro::alterarTerceiro(int anos, std::string nomeTerceiro, std::string tel, std::string end, std::string seguroTerceiro, std::string cpf)

Chama o método 'criarTerceiro', passando os parâmetros recebidos na chamada.

5.5 void CRUDTerceiro::excluirTerceiro()

Devolve a memória que foi alocada.

5.6 bool CRUDTerceiro::consultarTerceiro()

Possui seis opções de pesquisa de um determinado terceiro. Retorna *true* e chama a função 'imprimirDadosTerceiro' se achou e *false* caso contrário.

5.7 void CRUDTerceiro::imprimirDadosTerceiro()

Método que imprime os atributos da classe.

5.8 Terceiro* CRUDTerceiro::getTerceiro() const

Função *get* para o atributo terceiro.

6. Data

6.1 void Data::inserirData(int d, int m, int a)

Chama as funções *set* da classe para atribuir valores aos atributos.

6.2 void Data::setDiaData(int d)

Função *set* que modifica o atributo 'dia'.

6.3 void Data::setMesData(int m)

Função *set* que modifica o atributo 'mes'.

6.4 void Data::setAnoData(int a)

Função *set* que modifica o atributo 'ano'.

6.5 int Data::getDiaData() const

Função *get* que retorna o dia.

6.6 int Data::getMesData() const

Função *get* que retorna o mês.

6.7 int Data::getAnoData() const

Função *get* que retorna o ano.

7. GerenciarOrcamento

7.1 GerenciarOrcamento::GerenciarOrcamento()

Construtor da classe aloca memória necessária pra um objeto da classe ‘Orcamento’.

7.2 GerenciarOrcamento::~GerenciarOrcamento()

Destrutor da classe devolve a memória alocada.

7.3 void GerenciarOrcamento::inserirOrcamento(string t, float x)

Chama as funções *set* da classe para atribuir valores aos atributos, além de incrementar o atributo estático.

7.4 void GerenciarOrcamento::imprimirDadosOrcamento()

Método que imprime os atributos da classe.

7.5 float GerenciarOrcamento::getValorTotal()

Função *get* que retorna o valor total dos orçamentos.

7.6 Orcamento* GerenciarOrcamento::getOrcamento() const

Função *get* que retorna um ponteiro do tipo 'Orcamento'.

8. GerenciarSegurados

8.1 GerenciarSegurados::GerenciarSegurados()

Construtor da classe zera os contadores e atribui *NULL* a todas as posições dos vetores de ponteiros.

8.2 GerenciarSegurados::~GerenciarSegurados()

Destrutor da classe devolve a memória alocada por cada ponteiro dos vetores de ponteiros, depois atribui *NULL*.

8.3 bool GerenciarSegurados::criarCliente(string n, string t, string e, int pos, string cpf)

Aloca a memória necessária para um objeto da classe ‘Cliente’, e em seguida chama as funções *set* da classe para atribuir aos atributos da classe os parâmetros passados.

8.4 bool GerenciarSegurados::alterarCliente(string n, string t, string e, int pos, string cpf)

Se a posição passada por parâmetro for válida, chama a função ‘criarCliente’ para que o objeto na posição dada seja alterado.

8.5 bool GerenciarSegurados::excluirCliente(int pos)

Se a posição passada por parâmetro for válida, há um laço que puxa todos os objetos da posição dada para frente, garantindo que o objeto seja excluído.

8.6 bool GerenciarSegurados::consultarCliente()

A consulta por um cliente pode ser feita pelo número da apólice, nome do cliente ou CPF do cliente.

8.7 bool GerenciarSegurados::consultarVeiculo(string placa)

Chama o método da classe 'Veiculo' que verifica se a placa passada como parâmetro é igual ao atributo 'placa'.

8.8 bool GerenciarSegurados::imprimirDadosCliente()

Realiza a impressão dos dados de cada objeto da classe 'Cliente', em cada posição do vetor de ponteiros.

8.9 bool GerenciarSegurados::removerApolice(int posicao, int pos)

Se a posição for válida, a função chama o método 'excluirApolice' da classe 'Cliente'.

8.10 bool GerenciarSegurados::alterarApolice(int posicao, int pos, Apolice *ap)

Se a posição for válida, a função chama o método 'alterarApoliceCliente' da classe 'Cliente'.

8.11 bool GerenciarSegurados::vincularApoliceCliente(Apolice *ap, int pos)

Se a posição for válida, a função chama o método 'setApoliceCliente' da classe 'Cliente'.

9. GerenciarSinistro

9.1 GerenciarSinistro::GerenciarSinistro()

Construtor da classe aloca espaço suficiente para um objeto da classe Sinistro.

9.2 GerenciarSinistro::~GerenciarSinistro()

Destrutor da classe devolve à memória que foi alocada.

9.3 void GerenciarSinistro::criarSinistro(int x, Data d, string place, Terceiro *t, int tipoSinistro, BO *b)

Chama as funções *set* da classe para atribuir valores aos atributos.

9.4 void GerenciarSinistro::alterarSinistro(int x, Data d, string place, Terceiro t, int tipoSinistro, BO b)

Chama o método 'criarSinistro' passando os parâmetros recebidos na chamada.

9.5 void GerenciarSinistro::excluirSinistro()

Devolve para a memória que foi alocada.

9.6 bool GerenciarSinistro::consultarSinistro()

Possui duas opções de pesquisa de um determinado sinistro. Retorna *true* e chama a função ‘imprimirDadosSinistro’ se achou e *false* caso contrário.

9.7 Sinistro* GerenciarSinistro::getSinistro() const

Função *get* que retorna um ponteiro do tipo 'Sinistro'.

10. GerenciarVeiculos

10.1 GerenciarVeiculos::GerenciarVeiculos()

Construtor da classe aloca espaço suficiente para um objeto da classe ‘Veiculo’.

10.2 GerenciarVeiculos::~GerenciarVeiculos()

Destrutor da classe devolve a memória que foi alocada.

10.3 void GerenciarVeiculos::inserirVeiculo(string p, int x, string r, int tv, int m)

Chama as funções *set* da classe para atribuir valores aos atributos.

10.4 void GerenciarVeiculos::imprimirDadosVeiculo()

Método que imprime os atributos da classe.

10.5 Veiculo* GerenciarVeiculos::getVeiculo() const

Função *get* que retorna um ponteiro do tipo 'Veiculo'.

11. Orçamento

11.1 void Orcamento::setTipoOrcamento(string t)

Função *set* que modifica o atributo 'tipo'.

11.2 void Orcamento::setValorRessarcOrcamento(float x)

Função *set* que modifica o atributo 'valorRessarc'.

11.3 string Orcamento::getTipoOrcamento()const.

Função *get* que retorna o tipo do orçamento.

11.4 float Orcamento::getValorRessarcOrcamento() const

Função *get* que retorna o valor de ressarcimento do orçamento.

12. Sinistro

12.1 Sinistro::~Sinistro()

Destrutor da classe devolve a memória alocada por cada ponteiro.

12.2 void Sinistro::incluirBO()

Aloca espaço suficiente para um objeto da classe ‘BO’.

12.3 void Sinistro::incluirTerceiro()

Aloca espaço suficiente para um objeto da classe ‘Terceiro’.

12.4 void Sinistro::imprimirDadosSinistro()

Imprime os dados do sinistro e do terceiro.

12.5 void Sinistro::setNumSinistro(int x)

Função *set* que modifica o atributo 'num'.

12.6 void Sinistro::setDateSinistro(Data d)

Função *set* que modifica o atributo 'dateSinistro' .

12.7 void Sinistro::setLocalSinistro(string place)

Função *set* que modifica o atributo 'local'.

12.8 void Sinistro::setTerceiroSinistro(Terceiro *t)

Função *set* que modifica o atributo 'terceiro'.

12.9 void Sinistro::setBOSinistro(BO *b)

Função *set* que modifica o atributo 'bo'.

12.10 void Sinistro::setTipoSinistroSinistro(int x)

Função *set* que modifica o atributo 'tiposinistro'.

12.11 int Sinistro::getNumSinistro() const

Função *get* que retorna o número do sinistro.

12.12 Data Sinistro::getDateSinistro() const

Função *get* que retorna a data do sinistro.

12.13 string Sinistro::getLocalSinistro() const.

Função *get* que retorna o local do sinistro.

12.14 Terceiro* Sinistro::getTerceiroSinistro() const

Função *get* que retorna um ponteiro para o terceiro do sinistro.

12.15 int Sinistro::getTipoSinistro() const

Função *get* que retorna o tipo do sinistro.

12.16 BO* Sinistro::getBOSinistro() const

Função *get* que retorna um ponteiro para o BO do sinistro.

13. Terceiro

13.1 void Terceiro::setIdadeTerceiro(int anos)

Função *set* que modifica o atributo 'idade'.

13.2 void Terceiro::setNomeTerceiro(string nomeTerceiro)

Função *set* que modifica o atributo 'nome'.

13.3 void Terceiro::setTelefoneTerceiro(string tel)

Função *set* que modifica o atributo 'telefone'.

13.4 void Terceiro::setEnderecoTerceiro(string end)

Função *set* que modifica o atributo 'endereco'.

13.5 void Terceiro::setSeguradoraTerceiro(string seguroTerceiro)

Função *set* que modifica o atributo 'seguradora'.

13.6 void Terceiro::setCPFTerceiro(string cpf)

Função *set* que modifica o atributo 'CPF'.

13.7 int Terceiro::getIdadeTerceiro() const

Função *get* que retorna a idade do terceiro.

13.8 string Terceiro::getNomeTerceiro() const

Função *get* que retorna o nome do terceiro.

13.9 string Terceiro::getTelefoneTerceiro() const

Função *get* que retorna o telefone do terceiro.

13.10 string Terceiro::getEnderecoTerceiro() const

Função *get* que retorna o endereço do terceiro.

13.11 string Terceiro::getSeguradoraTerceiro() const

Função *get* que retorna o nome da seguradora do terceiro.

13.12 string Terceiro::getCPFTerceiro() const

Função *get* que retorna o CPF do terceiro.

14. Veículo

14.1 void Veiculo::setPlacaVeiculo(string p)

Função *set* que modifica o atributo 'placa'.

14.2 void Veiculo::setAnoVeiculo(int x)

Função *set* que modifica o atributo 'ano'.

14.3 void Veiculo::setRenavamVeiculo(string r)

Função *set* que modifica o atributo 'renavam' .

14.4 void Veiculo::setTipoVeiculoVeiculo(int tv)

Função *set* que modifica o atributo 'tipoveiculo'.

14.5 void Veiculo::setMarcaVeiculo(int m)

Função s*et* que modifica o atributo 'marca'.

14.6 string Veiculo::getPlacaVeiculo() const

Função *get* que retorna a placa do veículo.

14.7 int Veiculo::getAnoVeiculo() const

Função *get* que retorna o ano do veículo.

14.8 string Veiculo::getRenavamVeiculo() const

Função *get* que retorna o RENAVAM do veículo.

14.9 int Veiculo::getTipoVeiculoVeiculo() const

Função *get* que retorna o tipo do veículo.

14.10 int Veiculo::getMarcaVeiculo() const

Função *get* que retorna a marca do veículo.

Descrição das condições de contorno

Em toda alocação de memória é testada se a alocação foi realizada com sucesso, com o tratador de exceções *try*, que lança um *bad alloc* caso a alocação tenha falhado. O *catch* faz o tratamento para o *bad alloc*.

Em todo vetor de ponteiros foi estipulado um tamanho máximo de 100 (cem) posições, após a inserção do centésimo elemento não são mais possíveis inserções.

Em todo "menu" de opções é tratado valores incorretos.

A impressão de dados só ocorre se houver clientes cadastrados. A impressão de apólices de um determinado cliente só ocorre se esse cliente as possuir. Consequentemente, a impressão dos sinistros referentes a uma apólice também depende da sua existência.

Descrição dos testes realizados

Inserir, alterar, remover e consultar um cliente funcionam corretamente.

Ao inserir um carácter quando o programa espera um inteiro, o programa falhará e entrará em loop infinito.

Impressão de dados funciona corretamente.

Cadastrar veículo, apólice e sinistro funcionam corretamente, exceto em remover sinistro e alterar sinistro.

Dificuldades encontradas

Houve dificuldade para a compreensão do Diagrama de Classes, com isso fizemos diversos tipos de implementações até encontrarmos uma condizente com o que nos foi proposto.

Não conseguimos implementar corretamente a classe Condutor. O motivo essencial para tal foi porque o Condutor herda do Cliente um vetor de ponteiros do tipo Apólice.

Possíveis extensões e melhorias do programa

Para melhorar o programa deve-se consertar pequenos erros no que se refere à funcionalidade do programa e ajustar alguns métodos incompletos. Uma melhoria que pode ser feita é a alteração do tipo de saída que o programa deve fazer, ou seja, guardar em um arquivo a saída do programa ao invés de imprimir no terminal apenas e enviar um e-mail lembrando o cliente sobre o vencimento do seguro.

Comentários pessoais

A dinâmica do programa e o cenário onde se encontra são muito interessantes, pois englobam situações plausíveis aonde poderíamos nos inserir. Toda a estruturação criada para a implementação do sistema abrange boa parte do conteúdo visto em aula, nos mostrando que o paradigma de programação orientada a objetos é muito útil ao lidar com inúmeros arquivos e classes.