LISTA DE EXERCÍCIOS

DISCIPLINA
Desenvolvimento de Software
PROFESSOR
Geucimar Briatore

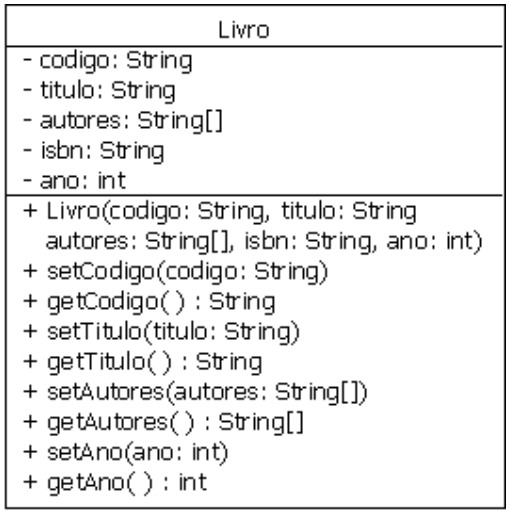
LISTA	TEMA
04	**Orientação a objetos**

OBJETIVOS

- Praticar o uso de classes e relacionamento entre objetos.
- Utilizar Encapsulamento, Associação e Herança na linguagem Java.

EXERCÍCIOS

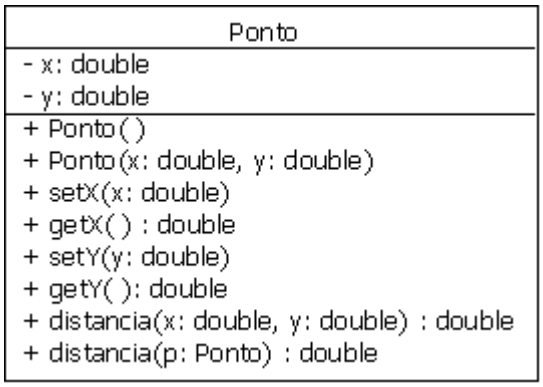
Cenário 1: Crie uma classe que represente um livro em uma biblioteca. Em outra classe (Programa), instancie os seguintes livros e apresente seus dados na tela.



Livro 01:
Código: 1598FHK
Título: Core Java 2
Autores: Cay S. Horstmann e Gary Cornell
ISBN: 0130819336
Ano: 2005

Livro 02:
Código: 9865PLO
Título: Java, How to Program
Autores: Harvey Deitel
ISBN: 0130341517
Ano: 2015

Cenário 2: Crie uma classe que represente um ponto no espaço bidimensional. Na classe Programa, efetue os seguintes procedimentos.



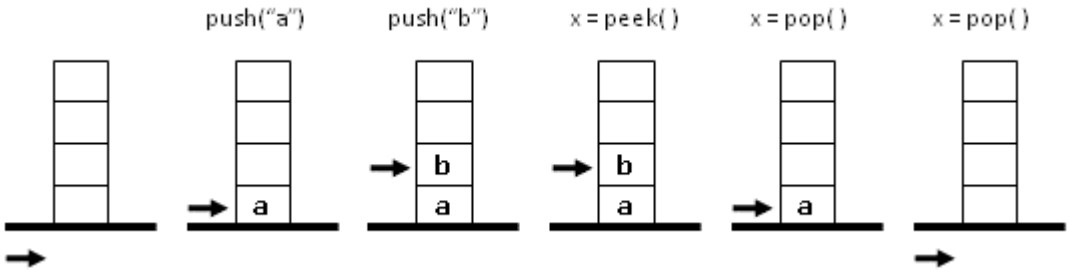
1. Crie um objeto ponto1 na origem usando o primeiro construtor;
2. Crie um objeto ponto2 na posição (2,5);
3. Calcule a distância do ponto1 ao ponto2;
4. Calcule a distância do ponto2 às coordenadas (7,2);
5. Altere o valor de x para 10 no ponto1;
6. Altere o valor de y para 3 no ponto1;

Obs.:
- Construtor Ponto( ): cria um ponto na origem (0,0);
- Construtor Ponto(x: double, y: double): cria um ponto nas coordenadas passadas;

- Cálculo da distância entre dois pontos:

$$d = \sqrt{(x1 - x2)^2 + (y1 - y2)^2}$$

Cenário 3: Crie as classes indicadas e implemente suas operações.



Livro
- codigo: String - titulo: String - autores: String[] - isbn: String - ano: int
+ Livro(codigo: String, titulo: String autores: String[], isbn: String, ano: int) + setCodigo(codigo: String) + getCodigo( ) : String + setTitulo(titulo: String) + getTitulo( ) : String + setAutores(autores: String[]) + getAutores( ) : String[] + setAno(ano: int) + getAno( ) : int

Pilha
- pilha: Livro[] - topo: int
+ Pilha( ) + cheia( ) : boolean + vazia( ) : boolean + push(liv : Livro) + pop( ) : Livro + peek( ) : Livro + toString( ) : String

Implemente uma classe Pilha para representar uma pilha estática de até 5 livros. Os métodos desta classe devem ser:

1. **push(...)**: inserir um livro no topo da pilha, caso não esteja cheia.
2. **pop ( )**: remover e retornar o livro do topo da pilha, caso não esteja vazia. Se vazia, retornar null.
3. **peek( )**: retornar o livro do topo da pilha, caso não esteja vazia. Se vazia, retornar null.
4. **toString( )**: listar o título dos livros da pilha, do primeiro livro empilhado ao último, caso não esteja vazia. Se vazia, retornar “”
5. **cheia( )**: retornar verdadeiro ou falso indicando se a pilha está cheia
6. **vazia( )**: retornar verdadeiro ou falso indicando se a pilha está vazia

Implemente uma classe Programa com o seguinte menu:

1. **Push** – inserir livro
2. **Pop** – remover livro
3. **Peek** – visualizar livro
4. **Listar** – listar todos os livros
5. **Sair**

Para cada opção de menu, implemente as seguintes regras:

1. **Push**: ler do teclado os dados de um livro, criar um objeto da classe Livro e passá-lo ao método push para que seja inserido na pilha.
2. **Pop**: receber o livro retornado do método pop e apresentá-lo na tela. Caso o retorno seja null, apresentar na tela a mensagem “Pilha vazia”
3. **Peek**: receber o livro retornado do método pop e apresentá-lo na tela. Caso o retorno seja null, apresentar na tela a mensagem “Pilha vazia”
4. **Listar**: apresentar na tela os livros empilhados
5. **Sair**: abandonar o programa.

Cenário 4: Controle de tráfego aéreo

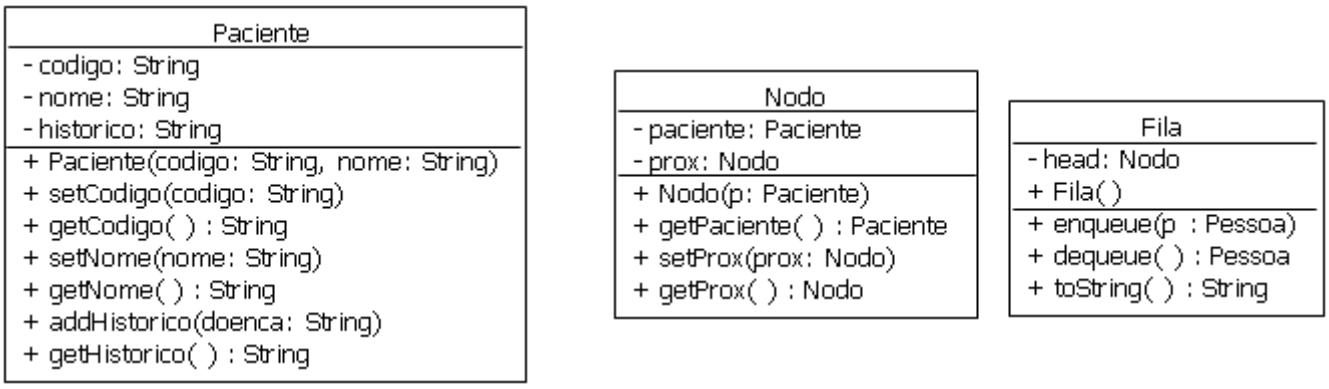
Um aeroporto precisa-se registrar informações sobre as diferentes pessoas que nele trafegam.
Todas as pessoas possuem nome e rg. As pessoas se dividem entre passageiros e tripulação. Os passageiros possuem um identificador de bagagem e a sua passagem. Sobre a passagem, armazena-se o número do acento, a classe do acento e a data do voo, contendo dia, mês, ano, hora e minuto de partida.
Sobre a tripulação, sabe-se a sua identificação aeronáutica e matrícula do funcionário. Dos comandantes, registra-se o seu total de horas de voo e dos comissários os idiomas em que possuem fluência.
Todas as pessoas possuem ainda informações sobre a aeronave em que farão o voo. Sobre ela, armazena-se o seu código, tipo e quantidade de assentos.

Cenário 5: Controle acadêmico

Em uma instituição de ensino, devem ser registradas informações sobre professores, alunos e seus relacionamentos entre disciplinas.
Todas as pessoas representadas no sistema possuem nome, rg e matrícula. Os professores possuem número de identificação do seu currículo Lattes e titulação, envolvendo nome da instituição, ano de conclusão, nome do título obtido e título do trabalho de conclusão. Os alunos, por sua vez, possuem o ano de ingresso na instituição, nome do curso e turno.
Todas as disciplinas possuem um nome, identificador, currículo a que pertencem e um conjunto de competências, classificadas como Necessárias e Complementares. Na disciplina também estão registrados o professor que a ministra e os alunos nela matriculados.
Para cada aluno é registrada a situação acerca de suas competências, sendo ela Atingida ou Não-Atingida. A partir da sua situação, pode-se avaliar a situação do aluno como:

- Aprovado: 100% das competências Necessárias, pelo menos 50% das competências complementares;
- Reprovado: menos de 50% das competências Necessárias ou menos de 50% das competências complementares;
- Pendente: nenhuma das duas situações anteriores.

Cenário 6: Controle de pacientes



Implemente uma classe Fila para representar uma fila dinâmica de pacientes. Os métodos desta classe devem ser:

1. **enqueue(...)**: inserir um paciente no final da fila.
2. **dequeue( )**: remover e retornar o paciente do começo da fila caso não esteja vazia. Se vazia, retorne null.
3. **toString( )**: listar o nome dos pacientes da fila, do primeiro ao último, caso não esteja vazia. Se vazia, retornar “”

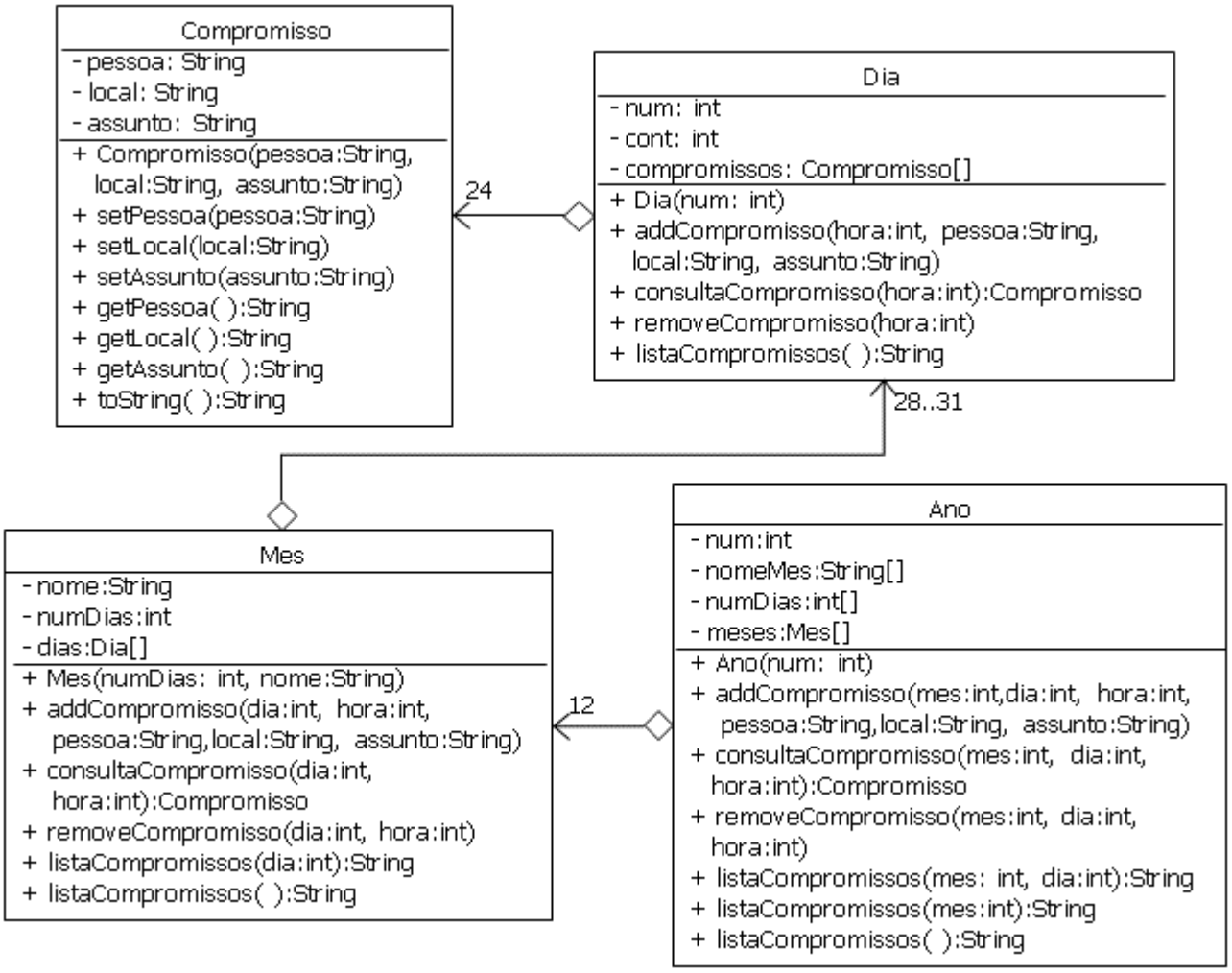
Implemente uma classe Programa com o seguinte menu:

1. **Enqueue** – inserir paciente
2. **Dequeue** – remover paciente
3. **Listar** – listar todos os pacientes
4. **Sair**

Para cada opção de menu, implemente as seguintes regras:

1. **Enqueue**: ler do teclado os dados de um paciente, criar um objeto da classe Paciente e passá-lo ao método enqueue para que seja inserido na fila.
2. **Dequeue**: receber o paciente retornado do método dequeue e apresentá-lo na tela. Caso o retorno seja null, apresentar na tela a mensagem “Fila vazia”
3. **Listar**: apresentar na tela os pacientes da fila
4. **Sair**: abandonar o programa.

Cenário 7: Agenda de Compromissos



Classe Compromisso

- Representa um compromisso com uma pessoa, em um determinado local, para tratar um assunto específico.

Classe Dia

- Possui um vetor de compromissos de 24 elementos representando as 24 horas de um dia. O atributo **hora** armazena a hora do dia, **num** armazena o dia da semana e o atributo **cont** a quantidade de compromissos agendados no dia.
- **Construtor**: recebe o número do dia e inicializa o vetor de compromissos.
- **adicionarCompromisso(dia: int, hora: int, pessoa: String, local: String, assunto: String)**: agendar um compromisso na hora informada.
- **consultarCompromisso(hora: int)**: retornar o compromisso agendado da hora especificada. Caso não haja compromisso agendado, retornar null.

- **removerCompromisso(hora: int)**: remover (ou anular) o compromisso agendado para a hora especificada.
- **listarCompromissos( )**: retornar uma lista com todos os compromissos agendados no dia.

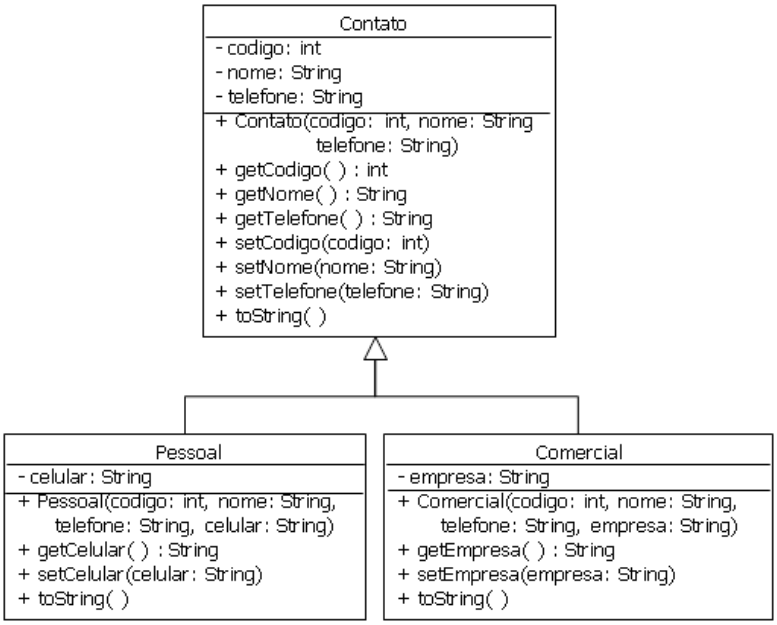
Classe Mes

- Possui um vetor de dias de **numDias** elementos representando os dias de um mês.
- **Construtor**: recebe o número de dias que mês possui e o nome do mês. Inicializa o atributo numDias e o vetor de dias.
- **adicionarCompromisso(dia: Dia, pessoa: String, local: String, assunto: String)**: no dia informado, solicita ao objeto da classe Dia o agendamento do compromisso na hora especificada.
- **consultarCompromisso(dia: Dia, hora: int)**: no dia informado, solicita ao objeto da classe Dia o compromisso agendado na hora especificada.
- **removerCompromisso(dia: Dia, hora: int)**: no dia informado, solicita ao objeto da classe Dia a remoção do compromisso agendado para a hora especificada.
- **listarCompromissos(dia: Dia)**: no dia informado, solicita ao objeto da classe Dia a lista de compromissos agendados.
- **listarCompromissos( )**: para cada dia do mês, solicita ao objeto da classe Dia a lista de compromissos agendados.

Classe Ano

- Para um ano específico, possui o nome dos 12 meses do ano, o número de dias de cada mês e um vetor de 12 meses representando cada mês do ano.
- **Construtor**: recebe o número do ano e inicializa o vetor meses. Para cada posição do vetor, criar um objeto da classe Mes com o nome e o número de dias específico.
- **adicionarCompromisso(mes: Mes, dia: Dia, hora: int, pessoa: String, local: String, assunto: String)**: no mês informado, solicita ao objeto da classe Mes o agendamento do compromisso no dia e na hora especificadas.
- **consultarCompromisso(mes: Mes, dia: Dia, hora: int)**: no mês informado, solicita ao objeto da classe Mes o compromisso agendado no dia e na hora especificada.
- **removerCompromisso(mes: Mes, dia: Dia, hora: int)**: no mês informado, solicita ao objeto da classe Mes a remoção do compromisso agendado para o dia e a hora especificada.
- **listarCompromissos(mes: Mes, dia: Dia, hora: int)**: para o mês informado, solicita ao objeto da classe Mês a lista de compromissos agendados para o dia especificado
- **listarCompromissos(mes: Mes)**: para o mês informado, solicita ao objeto da classe Mês a lista de compromissos agendados
- **listarCompromissos( )**: para cada mês do ano, solicita ao objeto da classe Mês a lista de compromissos agendados

Cenário 8: Controle de contatos com herança



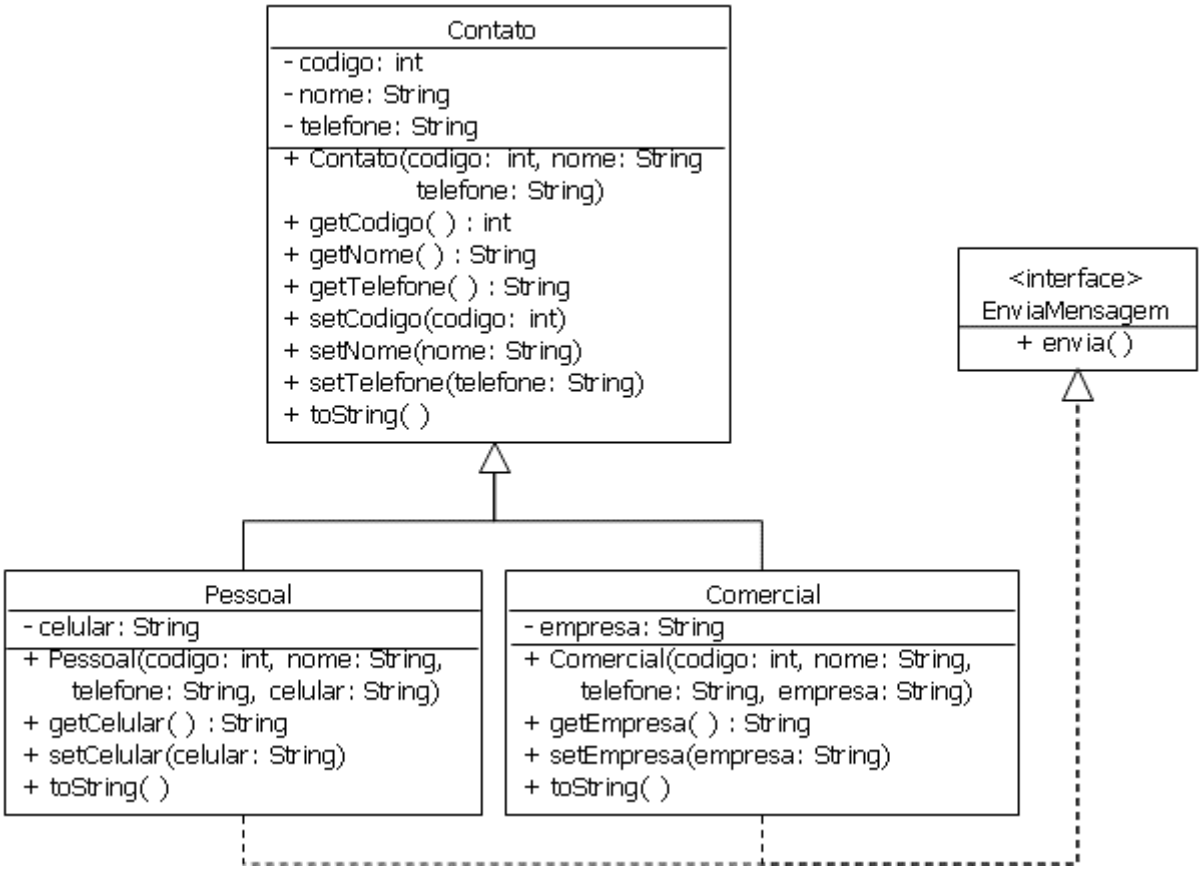
Classe Contato
- código não pode ser inferior a 1
- nome não pode ser nulo ou branco
- telefone não pode ser nulo ou conter menos de 8 caracteres

Classe Pessoal
- celular não pode ser nulo ou conter menos de 8 caracteres

Classe Empresa
- empresa não pode ser nula ou branca

Observações: classe Programa
Crie uma outra classe com o método main e faça instanciações de objetos das classes Pessoal e Comercial. Os valores passados aos objetos devem ser lidos do teclado.

Cenário 9: Controle de contatos com abstração



Torne a classe Contato abstrata.

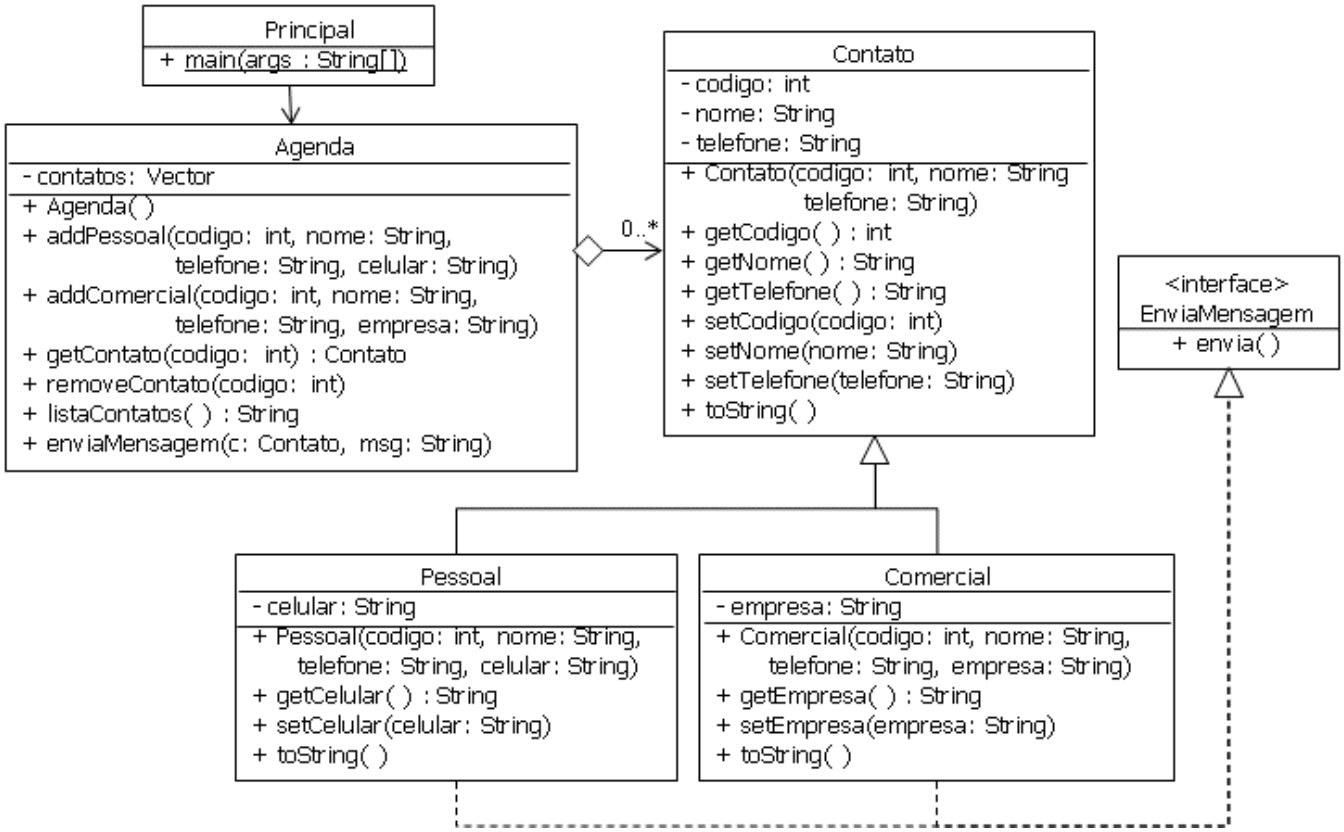
Para um contato pessoal, a mensagem é enviada por SMS ao celular da pessoa.

Para um contato comercial, a mensagem é enviada por SEDEX para a empresa em questão.

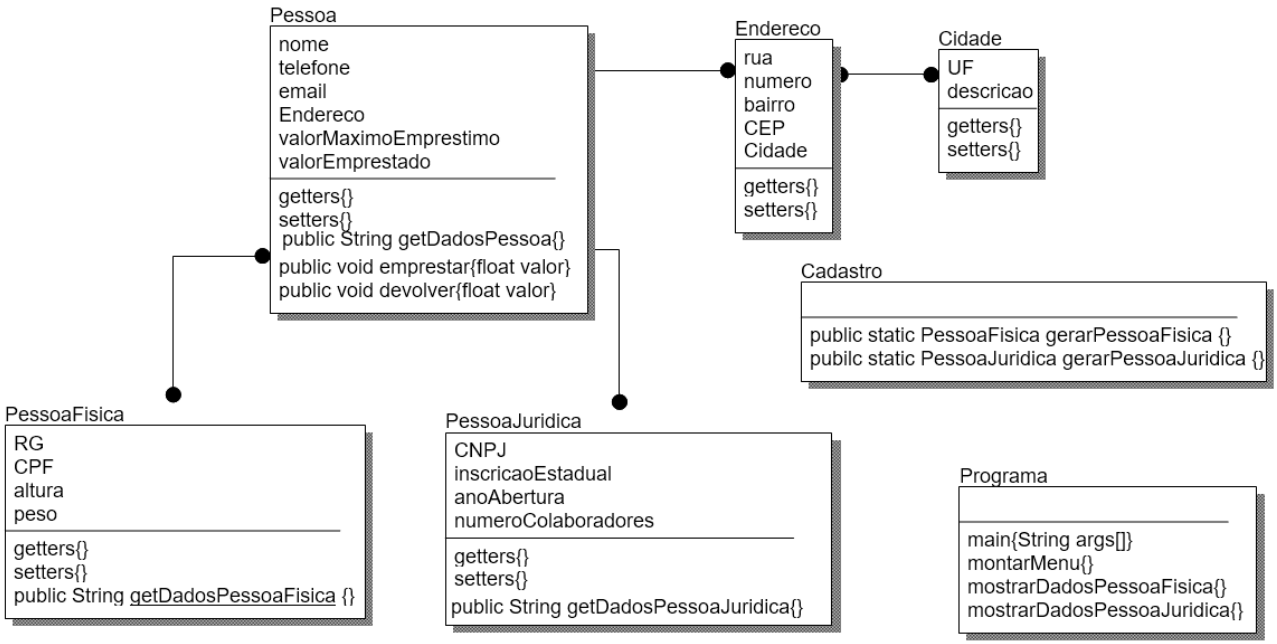
De acordo com as classes implementadas, implemente uma agenda de contatos onde é permitido:

1. inserir um contato pessoal
2. inserir um contato comercial
3. excluir um contato em função do código
4. consultar um contato em função do código
5. listar todos os contatos
6. enviar uma mensagem a um contato

Crie uma classe Agenda com estas funcionalidades e em uma classe Programa habilite um menu para as mesmas.



Cenário 10: Cadastro de pessoas com herança



Tendo em vista o “Diagrama de Classes” apresentado, implemente uma aplicação, considerando os seguintes tópicos:

- As classes PessoaFisica e PessoaJuridica herdam da classe Pessoa

- As classes devem possuir os getters e setters para seus atributos assim como os demais métodos identificados para cada uma delas, respeitando as assinaturas.

- A classe Cadastro deve possuir somente os métodos apresentados na mesma

- A classe Programa deve possuir em seu método main() somente chamadas para métodos

- O atributo valorEmprestado da classe Pessoa, deve ser manipulados pelos métodos emprestar() e devolver() considerando que os métodos devem ser responsáveis por impedir que o valor emprestado exceda o valor máximo de empréstimo ou fique negativo, mostrando uma mensagem.

- As validações para os atributos (não necessariamente para todos) devem ser feitas a seu critério.

O menu apresentado ao usuário deve conter os seguintes itens (encerrar o programa no Sair):

- Incluir Pessoa Física

- Incluir Pessoa Jurídica

- Mostrar Dados Pessoa Física

- Mostrar Dados Pessoa Jurídica

- Realizar Empréstimo para Pessoa Física

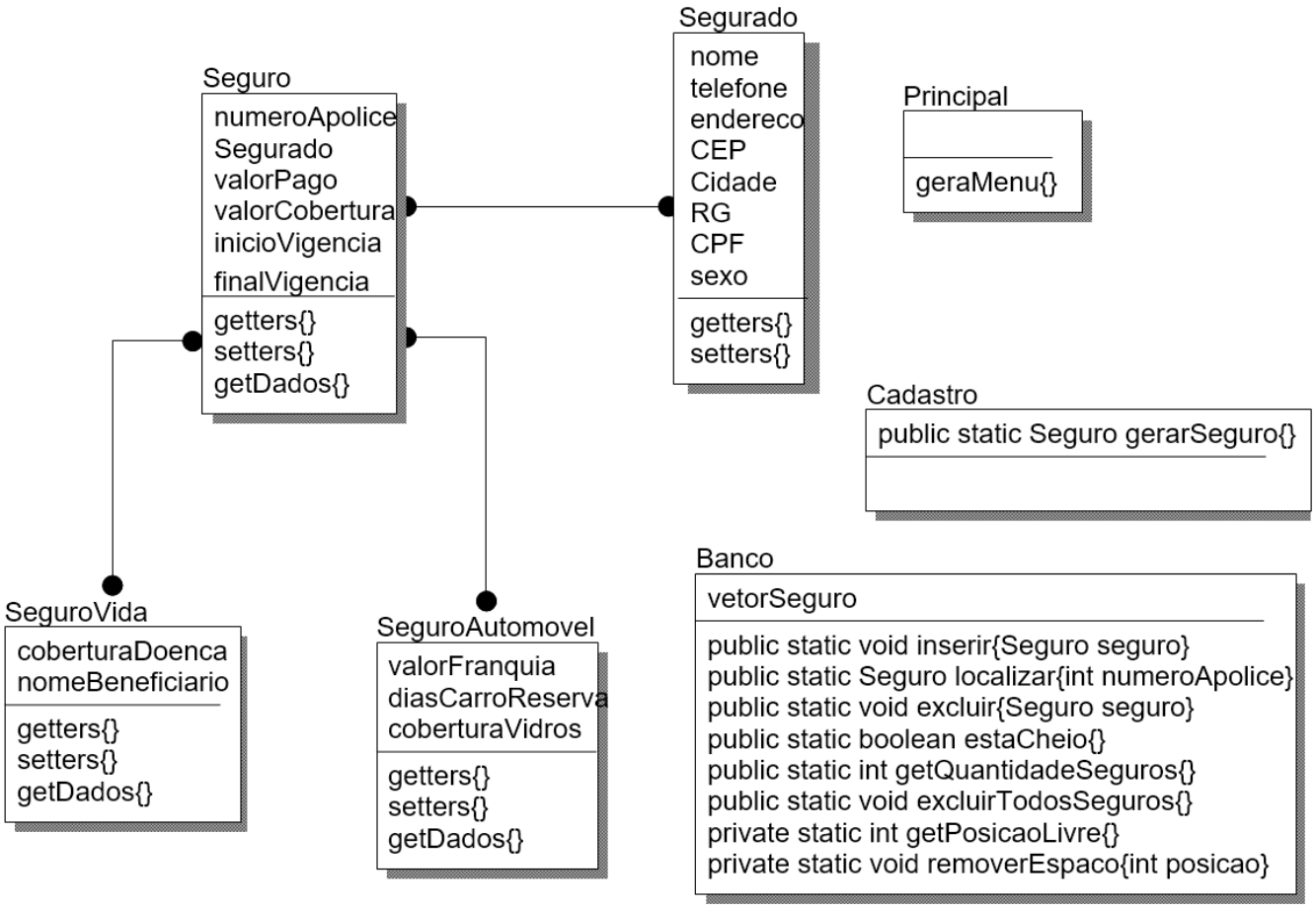
- Realizar Empréstimo para Pessoa Jurídica

- Realizar Devolução de Pessoa Física

- Realizar Devolução de Pessoa Jurídica

- Sair

Cenário 11: Apólice de seguros



Implemente o Diagrama de Classes acima, considerando os seguintes tópicos:

- Deve ser criado um Projeto separado para a implementação
- As classes SeguroVida e SeguroAutomovel herdam da Classe Seguro
- As classes devem possuir os getters e setters para seus atributos assim como os demais métodos identificados para cada uma delas, respeitando as assinaturas (quando estiver completa).
- A classe GeraSeguros deve possuir somente o método apresentado na mesma.(um tipo de Seguro)
- A classe principal deve possuir em seu método main() somente chamadas para métodos
- Podem ser implementados métodos e atributos para as classes além dos especificados no diagrama.
- Quando o usuário informar o “numeroApolice” do seguro, o sistema deve verificar se ele já não foi inserido, avisando o usuário imediatamente, evitando que sejam digitados todos os dados.
- O método “excluirTodosSeguros” deve pedir confirmação e excluir todos os seguros do vetor.
- O método “quantidadeSeguros” deve retornar quantos seguros estão inseridos.
- As validações para os atributos (não necessariamente para todos) devem ser feitas a seu critério.
- O menu apresentado ao usuário deve conter os seguintes itens (encerrar o programa no Sair):
  - Inserir Seguro
  - Localizar Seguro
  - Excluir Seguro
  - Excluir Todos os Seguros
  - Ver Seguros
  - Ver Todos Seguros
  - Ver Quantidade de Seguros
  - Sair