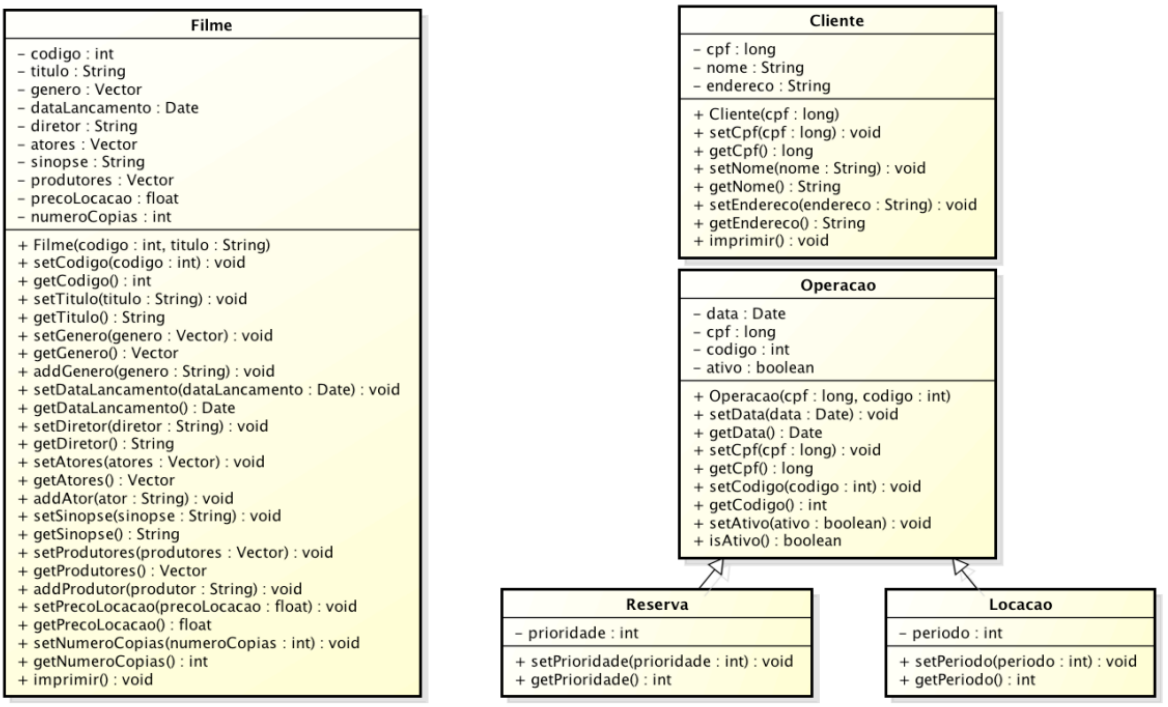
Universidade Federal do Rio de Janeiro
Centro de Ciências Matemáticas e da Natureza
INSTITUTO DE MATEMÁTICA
Departamento de Ciência da Computação

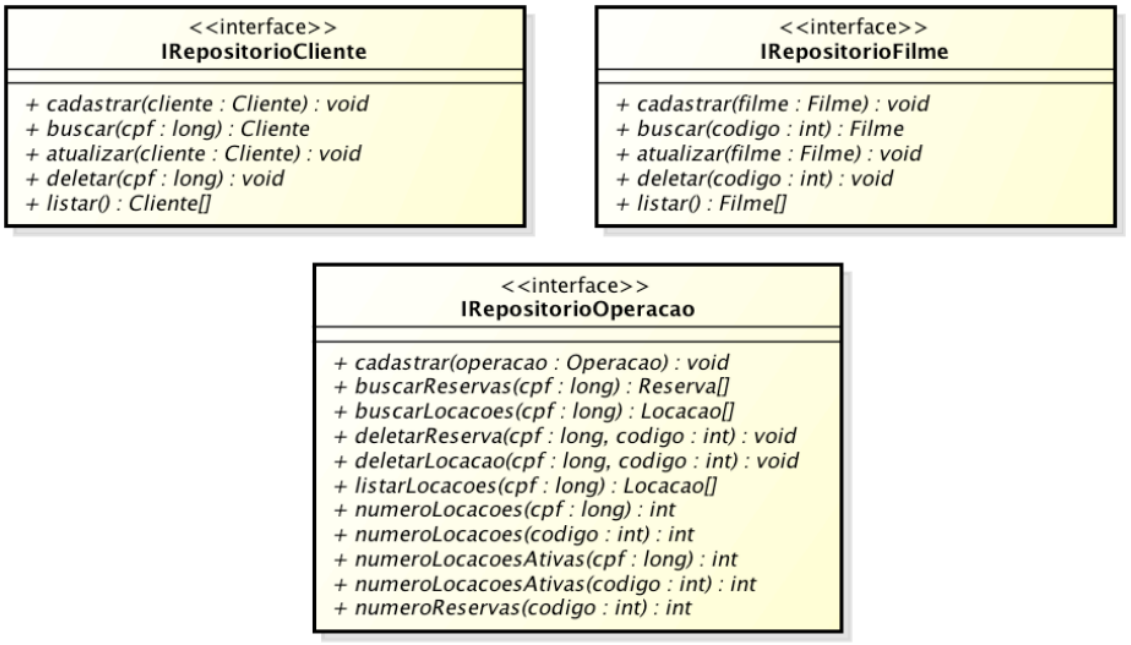
Disciplina	**MAB240 - Computação II**		**Semestre**	**2021.1**	
Professor	Anderson Gonçalves Uchôa				
Aluno			DRE		
Trabalho Prático Final - Individual		**Data**	30/08/2021	**Nota**	

Descrição do Programa: Sistema de Locadora de Vídeo - SisLoc
O SisLoc é um sistema responsável pelo controle de operações de reservas e locações de filmes em pequenas Locadoras de Vídeo de uma cidade perdida no tempo chamada Lugar Remoto. Além de controlar essas operações, o SisLoc é responsável por gerenciar o cadastro de clientes e filmes da locadora. Adicionalmente, o SisLoc fornece relatórios imprimindo o histórico de locações de um dado cliente e uma lista de top filmes mais locados. Mais detalhes sobre o que é requerido para a implementação do sistema é dado a seguir.

Forneça uma implementação para as classes `Cliente`, `Filme`, `Operação`, `Reserva` e `Locacao` como descrito na figura abaixo. O método `imprimir()` da classe `Filme` deve imprimir um resumo do filme, contendo seu código, título, gêneros, data de lançamento, atores, diretor e sinopse. Na classe `Cliente` o método `imprimir()` deve imprimir uma descrição do cliente contendo seu nome e endereço. A classe `Operacao` representa as possíveis operações que podem ocorrer no sistema, sendo as classes `Reserva` e `Locacao` especializações de `Operacao`. Em todas as classes, os métodos `setXXX()` e `getXXX()/isXXX()`, onde `XXX` é o nome de um atributo, são, respectivamente, métodos de atribuição direta e de acesso direto aos atributos `XXX`.



Para que o sistema funcione corretamente, é preciso criar repositórios que permitam armazenar, atualizar e recuperar clientes, filmes e operações. Dessa forma, você deve fornecer implementações para as interfaces `IRepositorioCliente`, `IRepositorioFilme` e `IRepositorioOperacao` como descrito na figura abaixo. Essas implementações podem utilizar como estruturas de armazenamento *arrays* (e.g., `Filme[]`, `Cliente[]` ou `Operacao[]`) ou bibliotecas de classe do Java, como `Vector` ou `ArrayList`.

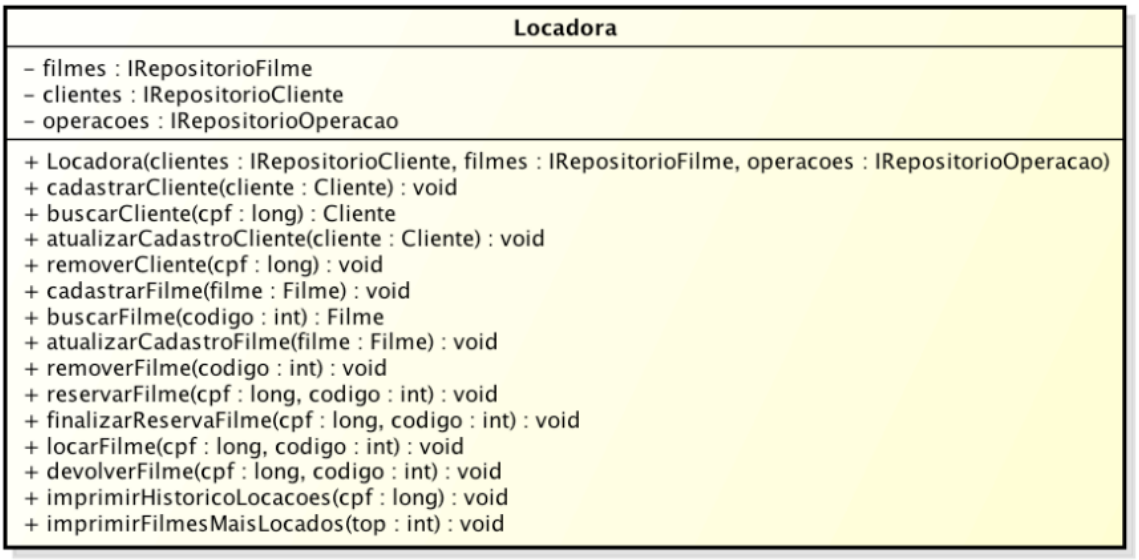


- A implementação da interface `IRepositorioCliente` deve prover os seguintes comportamentos para seus métodos:
    - Método `cadastrar()`**:** é responsável por cadastrar clientes. Restrição, não devem ser cadastrados clientes com o mesmo CPF, isto é o CPF é um identificador único;
    - Método `buscar()`: é responsável por procurar cliente pelo seu CPF. Deve retornar o cliente solicitado ou null em caso contrário;
    - Método `atualizar()`: é responsável por fazer a atualização do cliente. Restrição, a atualização só pode ocorrer se o usuário já estiver cadastrado;
    - Método `listar()`: deve retornar a lista de todos os clientes cadastrados.

- A implementação da interface `IRepositorioFilme` deve prover os seguintes comportamentos para seus métodos:
    - Método `cadastrar()`: é responsável por cadastrar filmes. Restrição, não devem ser cadastrados filmes com o mesmo código, isto é o código é um identificador único;
    - Método `buscar()`: é responsável por procurar filmes pelo seu código. Deve retornar o filme solicitado ou null em caso contrário;
    - Método `atualizar()`: é responsável por fazer a atualização do filme. Restrição, a atualização só pode ocorrer se o filme já estiver cadastrado;
    - Método `listar()`: deve retornar a lista de todos os filmes cadastrados.

- A implementação da interface `IRepositorioOperacao` deve prover os seguintes comportamentos para seus métodos:
    - Método `cadastrar():` é responsável por cadastrar operações de reserva ou locação e alterar status da locação para ativo (i.e., `setAtivo(true)`);
    - Método `buscarReservas(long cpf)`: é responsável por procurar reservas ativas realizadas pelo cliente que possui um determinado CPF;
    - Método `buscarLocacoes(long cpf)`: é responsável por procurar as locações ativas realizadas pelo cliente que possui um determinado CPF;
    - Método `deletarReservas(long cpf, int codigo)`: é responsável por alterar o status da reserva do filme de código específico por um cliente de CPF particular como inativa (i.e., `setAtivo(false)`);
    - Método `deletarLocacao(long cpf, int codigo)`: é responsável por alterar o status da locação do filme de código específico por um cliente de CPF particular como inativa (i.e., `setAtivo(false)`);
    - Método `listarLocacoes(long cpf)`: é responsável por listar todas as locações (ativas e inativas) realizadas pelo cliente que possui um determinado CPF;
    - Método `numeroLocacoes(long cpf)`: retorna o número total de locações (ativas e inativas) realizadas por um cliente de CPF específico;
    - Método `numeroLocacoes(int codigo):` retorna o número total de locações (ativas e inativas) que foram feitas do filme de código particular;
    - Método `numeroLocacoesAtivas(long cpf)`: retorna o número total de locações ativas realizadas por um cliente de CPF específico;
    - Método `numeroLocacoesAtivas(int codigo)`: retorna o número total de locações ativas que foram feitas do filme de código particular;
    - Método `numeroReservas(int codigo)`: retorna o número total de reservas ativas para o filme de código particular.

O SisLoc é acessado a partir da classe `Locadora`, descrita na figura abaixo. Esta classe possui três repositórios como atributos, filmes (`IRepositorioFilme`), clientes `(IRepositorioCliente)` e operacoes `(IRepositorioOperacoes)`, que são passados como parâmetros no momento da instanciação e utilizados na implementação do comportamento da `Locadora`. Cada um dos métodos da classe são explicados em detalhes a seguir:



- Método `cadastrarCliente()`: é responsável pelo cadastro de clientes. Restrição: o sistema não deve permitir o cadastro de clientes com o mesmo CPF;
- Método `buscarCliente()`: é responsável por recuperar um cliente do cadastro ou retornar null em caso não exista;
- Método `atualizarCadastroCliente()`: é responsável por atualizar o cadastro de um cliente, caso este já esteja cadastrado;
- Método `removerCliente()`: é responsável por excluir o cadastro de um cliente particular. Restrição: só podem ser removidos do cadastro os clientes que não possuem locações ativas;
- Método `cadastrarFilme()`: é responsável pelo cadastro de filmes. Restrição: o sistema não deve permitir o cadastro de filmes com o mesmo código;
- Método `buscarFilme():` é responsável por recuperar um filme do cadastro ou retornar `null` em caso contrário;
- Método `atualizarCadastroFilme()`: é responsável por atualizar o cadastro de um filme, caso este já esteja cadastrado;
- Método `removerFilme()`: é responsável por excluir o cadastro de um filme particular. Restrição: só podem ser removidos do cadastro os filmes que não possuem locações ativas;
- Método `reservarFilme()`: é responsável por realizar a reserva de um determinado filme para um cliente específico. Restrição: só podem ser feitas reservas de filmes cadastrados para clientes cadastrados e se não houver cópias do filme disponível, isto é, o número de locações do filme for igual ao número de cópias do filme (descrita pelo atributo `numeroCopias` da classe `Filme`);
- Método `finalizarReservarFilme()`: é responsável por alterar o status da reserva de ativa para inativa. Restrição: só podem ser finalizadas reservas existentes, isto é,

reservas previamente cadastrada de um determinado filme para um cliente específico;

- Método `locarFilme()`: é responsável por realizar a Locação de um filme para um determinado cliente. Restrições: a locação só pode ser feita se, ambos, o filme e o cliente forem cadastrados e se o filme estiver disponível (i.e., o número de locações ativas for menor que o número de cópias do filme) e se não existir reservas ou, no caso de existir reservas, a reserva mais antiga ser a do cliente que deseja realizar a locação;
- Método `devolverFilme():` é responsável por realizar a devolução de um filme feita por um determinado cliente (i.e., modificar o status da locação para inativo). Restrições: a devolução só pode ser feita se, ambos, o filme e o cliente forem cadastrados e se a locação existir;
- Método `imprimirHistoricoLocacoes()`: é responsável por imprimir o histórico de locações realizadas por um determinado cliente. Restrição: o cliente precisa estar cadastrado;
- Método `imprimirFilmesMaisLocados()`: é responsável por imprimir a lista dos filmes mais locados pela locadora, onde o tamanho da lista é indicado pelo argumento `int top`. A lista deve ser impressa em ordem decrescente, dos filmes mais locados para os menos locados.

Critérios para correção do trabalho:

- **Funcionamento:** este critério será utilizado para avaliar se o sistema está ou não funcional, isto é, se ele pode ser compilado e executado;
- **Cobertura de funcionalidades:** esse critério será utilizado para medir qual a porcentagem de funcionalidades foram implementadas pelo aluno;
- **Domínio do código fonte pelo aluno:** este critério será utilizado para avaliar qual é o domínio do aluno com respeito ao código fonte do sistema;
- **Boas práticas de POO:** este critério será utilizado para avaliar se o aluno aplicou os princípios da OO (encapsulamento, herança, etc) e boas práticas de programação para diminuir o acoplamento e aumentar a coesão;
- **Legibilidade do código:** este critério será utilizado para avaliar se o código fonte é legível, isto é, segue a formação padrão, adota a nomenclatura e estilo de programação recomendados para a linguagem Java;

Bonificações. Será concedido pontos extra para os alunos que utilizarem técnicas avançadas de programação em Java na implementação do trabalho: arquivos e/ou tratamento de exceção e/ou Swing. Dependendo do quanto de cada técnica for utilizada dentro do sistema, a bonificação pode chegar até a 2,0 pontos.