

## ***Herança***

A herança é o relacionamento onde uma classe (subclasse) herda os atributos e métodos de outra classe (superclasse). No código Medico estende a classe Paciente, ou seja, Medico é uma subclasse de Paciente. Isso é indicado pela palavra-chave `extends`. Assim, Medico herda os atributos `nome`, `cpf`, `dataNascimento`, `historicoConsultas`, e receitas de Paciente.

## ***Polimorfismo***

O polimorfismo permite que classes diferentes possam ser tratadas de maneira uniforme através de interfaces ou superclasses. A classe Medico implementa as interfaces `IMedico` e `IPaciente`. Isso significa que Medico deve fornecer implementações para todos os métodos definidos nas interfaces, permitindo que objetos do tipo Medico sejam tratados como objetos de qualquer uma dessas interfaces. A classe Paciente implementa a interface `IPaciente`. Isso significa que Paciente deve fornecer implementações para todos os métodos definidos na interface `IPaciente`. Qualquer objeto do tipo Paciente pode ser tratado como um objeto do tipo `IPaciente`.

## ***Associações***

A associação entre as classes acontece quando uma classe possui uma referência a outra classe. `historicoMedico` é uma lista de objetos `Consulta`, representando um relacionamento de agregação, onde a classe Medico possui um histórico de consultas médicas.

Paciente e Medico: `Consulta` contém referências às classes Paciente e Medico, representando as pessoas envolvidas na consulta. Exame e Medicamento: `Consulta` contém listas de objetos Exame e Medicamento, que representam os exames e medicamentos prescritos durante a consulta.

Exame, não temos associações diretas, pois ela não contém referências a outras classes. No entanto, Exame pode ser associada a outras classes no contexto do sistema maior, como a classe `Consulta`, onde uma lista de Exame é mantida.

Medicamento, não temos associações diretas, pois ela não contém referências a outras classes. No entanto, Medicamento pode ser associada a outras classes no contexto do sistema maior, como a classe `Consulta`, onde uma lista de Medicamento é mantida.

Paciente, temos as seguintes associações: `historicoMedico` é uma lista de objetos `Consulta`, representando o histórico médico do paciente. `Pagamentos` é uma lista de objetos `Pagamento`, representando os pagamentos realizados pelo paciente.

Pagamento, não temos associações diretas, pois ela não contém referências a outras classes. No entanto, Pagamento pode ser associada a outras classes no contexto do sistema maior, como a classe Paciente, onde uma lista de Pagamento é mantida.

### ***Exceções customizadas***

Uma exceção customizada em Java é usada para definir e tratar condições de erro específicas da aplicação, fornecendo uma maneira mais clara e precisa de lidar com situações excepcionais que podem ocorrer durante a execução do programa.

Especificidade e Clareza:

As exceções customizadas permitem definir erros específicos de domínio que são mais significativos e fáceis de entender. Por exemplo, `PagamentoPendenteException` indica claramente que o erro está relacionado a um pagamento pendente, facilitando a identificação da causa do problema.

Melhoria na Legibilidade do Código:

Ao usar exceções customizadas, o código fica mais legível e autoexplicativo. Em vez de lançar uma exceção genérica, como `Exception` ou `RuntimeException`, o uso de `PagamentoPendenteException` torna o propósito do código mais claro para outros desenvolvedores.

Controle de Fluxo:

Exceções customizadas permitem um controle de fluxo mais refinado. No caso de um pagamento pendente, lançar uma `PagamentoPendenteException` permite que o programa tome ações específicas para resolver essa situação, como notificar o usuário ou registrar o incidente para futuras análises.

Manutenção e Escalabilidade:

À medida que a aplicação cresce, a utilização de exceções customizadas facilita a manutenção e a escalabilidade do código. Novos tipos de erros específicos podem ser facilmente adicionados e tratados de forma consistente.