**01Apresentação**

**Transcrição**

Olá, meu nome é Paulo Silveira. Neste curso iremos desvendar a orientação a objetos. Veremos, dentre outros tópicos, quais são as boas práticas, o que é o encapsulamento, como se dá a comunicação entre os objetos e qual é a diferença específica entre **classe** e **instância**.

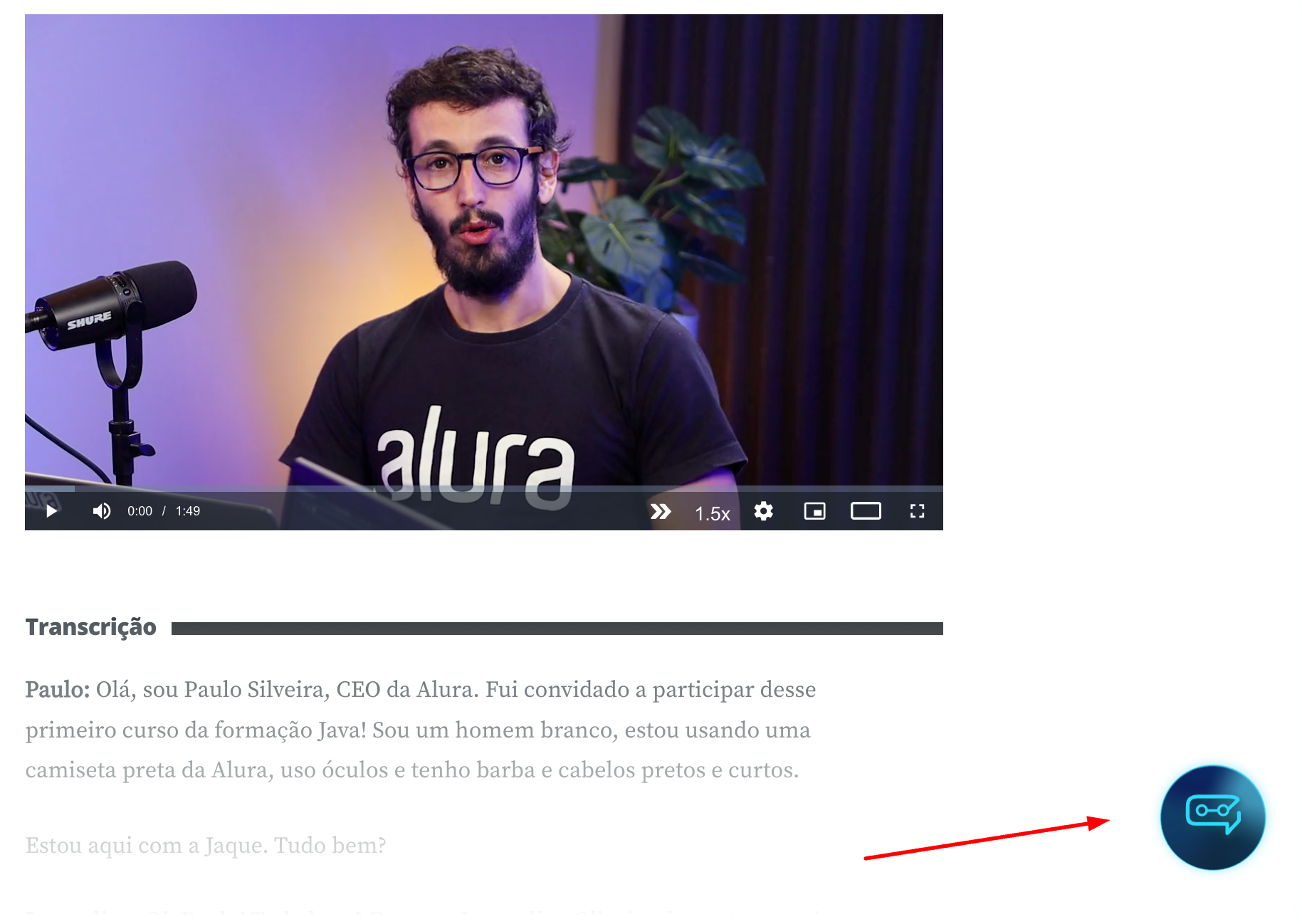
Agora que você já conhece a sintaxe básica da linguagem Java, teve contato com o Eclipse, configurou seu primeiro programa, testou interações de laços, utilizou if e for e compreendeu como funciona a declaração de variável, neste curso terá toda a base para dar o próximo passo.

O fundamento da orientação a objetos é o pilar da linguagem e trouxe a popularidade para o Java, pois há mais de vinte anos representava uma novidade no mercado. Muito embora o conceito de orientação a objetos já existisse, é no Java que ele será difundido nos espaços de corporação, agências e até mesmo no governo.

O que veremos com especial ênfase, é como funciona o encapsulamento, a orientação ao objeto, os gets e sets e como utilizados da melhor forma. Veremos, ainda, como utilizar os *construtores*, a palavra static e as *arrays*`s.

Com a base fornecida por este curso, você terá ferramentas para lidar com recursos mais avançados da orientação a objetos, e com isso, poder consumir classes de outras bibliotecas.

**01Conheça a Luri, nossa IA**

Oi, estudante! Sabia que você possui acesso à Luri, nossa IA para te ajudar em todos os momentos com suas dúvidas? Ela encontra-se disponível em todos os cursos da plataforma, presente na parte inferior da tela.

Aqui seguem algumas dicas para que a Luri possa te ajudar ainda mais em sua jornada. Bom proveito!

**🌟 Principais usos da Luri**

| **Finalidade** | **Como Perguntar** |
| --- | --- |
| Resumir Aulas | "Faça um resumo dessa aula." |
| Sugerir Exercícios | "Me sugira exercícios com base nessa aula." |
| Auxílio com Códigos | "O seguinte código está com um bug: [seu código aqui]." |
| Dúvidas Detalhadas | "Me explique novamente o que é [conceito], dando outros exemplos." |
| Revisão de Exercício | "Corrija minha resolução desse exercício: [sua solução aqui]." |

**❌ Quando NÃO usar a Luri?**

* **Suporte da Plataforma**: A Luri não é treinada para te ajudar com o suporte da plataforma. Para esses casos, indicamos entrar em contato com nossa equipe de suporte que está sempre disponível para te ajudar.
* **Assuntos Fora de Contexto**: Suas dúvidas devem ter a ver com o assunto do curso no qual está sendo feita. Ex.: Fazer uma pergunta de PHP em um curso de React pode não gerar um bom resultado.
* **Dúvidas Menos Técnicas**: Para questões de carreira, por exemplo, opte por outros canais como fóruns ou Discord.

**💡Como obter boas respostas?**

* **Seja claro. Sua pergunta deve ser detalhada:**
  + ❌ "Erro IndexOutOfBounds"
  + ✅ "Luri, segui o código da aula e tive um erro de IndexOutOfBoundsException."
* **Guie a Resposta. Indique à Luri o tipo de resposta que você espera:** Se gostar de respostas mais curtas, adicione um “Seja conciso". Se gostar de analogias, insira um “Use analogias do dia a dia”. A ideia é que a Luri possa se adequar à sua forma de aprender favorita.

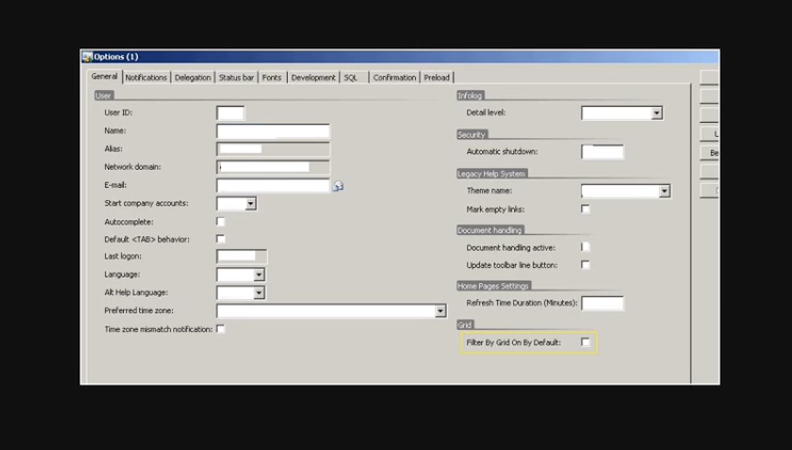
**02Paradigma procedural vs Objetos**

**Transcrição**

Para entendermos a grande vantagem da orientação aos objetos, veremos quais são as dificuldades que e impulsionaram a criação desse paradigma.

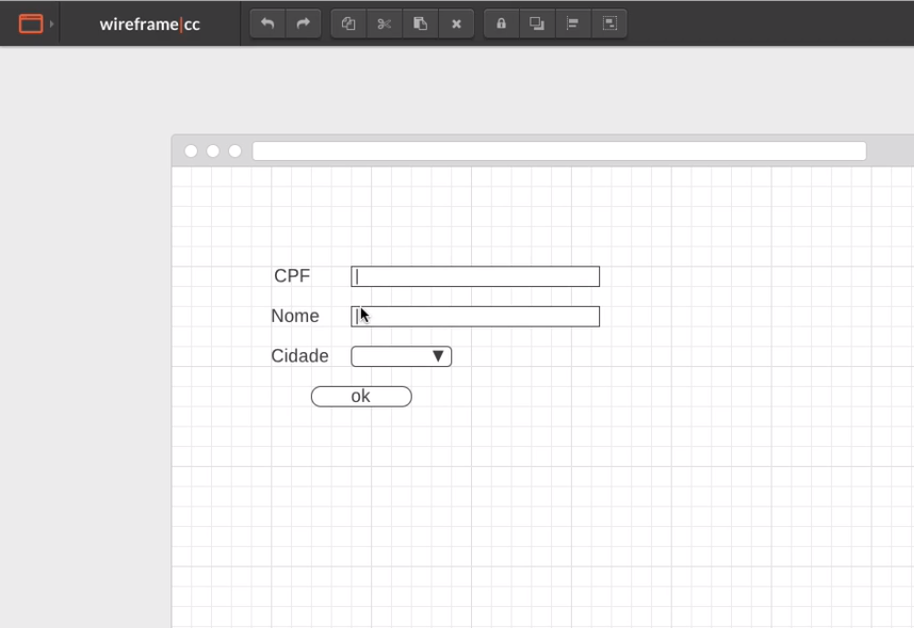
Antigamente tudo era procedural, não havia o conceito de programar voltado para um objeto. Mesmo hoje, com o desenvolvimento de tantas linguagens, alguns programadores ainda utilizam técnicas arcaicas da programação procedural. Esse método ainda faz sentido em alguns ambientes, mas na maioria dos casos, isso não se aplica ao Java.

No começo da década de noventa, os programadores trabalhavam com formulários longos e times enxutos, não havia uma equipe grande de desenvolvedores trabalhando em um projeto. Quem escrevia todo o formulário, cuidava de sua validação e de seu banco de dados, muitas vezes era um único programador ou programadora. Portanto, não havia como se atentar para todos os detalhes. Hoje em dia temos programas muito complexos, e o sistema de formulário se tornou insustentável.



Iremos verificar como trabalhávamos com os formulários para entendermos a diferença de paradigma gerada pela orientação aos objetos.

Usaremos o site [Wire Frame](https://wireframe.cc/), muito utilizado para quem quer esboçar UX Design, para analisarmos um exemplo de como eram criados os formulários antigamente. Usaremos uma linguagem e sintaxe hipotéticas.Temos um formulário simples de cadastro que contém **CPF**, **Nome** e **Cidade**.



Veremos como seria o código fonte desse formulário nessa linguagem imaginária. No campo CPF, poderíamos escrever que a variável cpf recebe do formulario1 o campo denominado CPF.

var cpf := formulario1->CPFCopiar código

Repetir o mesmo procedimento par todos os campos do formulário.

var cpf := formulario1->CPF

var nome := formulario1->Nome

var cidade := formulario1->CidadeCopiar código

Suponhamos que este seja um formulário utilizado no sistema de uma padaria. Neste ponto, poderíamos acionar uma função denominada gravar, que salvaria no banco dados as informações de cadastro do cliente.

var cpf := formulario1->CPF

var nome := formulario1->Nome

var cidade := formulario1->Cidade

gravarCliente(cpf, nome, cidade)Copiar código

Segundo a linguagem hipotética, o código está funcional.

Imaginemos a seguinte situação: seu chefe pede uma validação para o campo "CPF". Da forma como o código está organizado no momento, o campo "CPF" aceita qualquer tipo de dado, como letras e números aleatórios. Em outras palavras, o que o código faz é somente guardar o *input text* no banco de dados, independente do conteúdo.

Poderíamos solucionar esse problema adicionando outras funções ao código. Antes da gravação no banco de dados, adicionaremos uma função que valida o CPF - existe um cálculo específico para gerar CPFs - e passa uma variável. Teríamos como retorno um dado *booleano*; caso seja um CPF válido (if) a informação será gravada no banco de dados.

Caso contrário (else) será emitida uma mensagem de erro.

var cpf := formulario1->CPF

var nome := formulario1->Nome

var cidade := formulario1->Cidade

var sucesso = validaCpf(cpf)

if(sucesso)

gravarCliente(cpf,nome,cidade)

else

mostraErro()Copiar código

Não existe nenhum problema estrutural no código, e muitas vezes é dessa forma que solucionaremos questões na programação.

Mas imaginem a seguinte situação: no sistema padaria não existe apenas o formulário de cadastro, mas também um formulário de busca de clientes através do CPF, e esse CPF precisa ser validado antes da busca ser realizada.

Poderíamos começar o nosso código fonte da seguinte maneira:

var cpf := formularioBusca->CPF

buscaNoBanco(cpf)Copiar código

Poderíamos utilizar a função de validação que conhecemos, basta copiá-la do código de cadastro e colá-la, fazendo pequenas alterações. Ao invés de salvar no banco de dados, iremos procurar no banco um CPF específico.

var cpf := formularioBusca->CPF

var sucesso = validaCpf(cpf)

if(sucesso)

buscaNoBanco(cpf)

else

mostrarErro()Copiar código

Conseguimos atender as novas demandas da empresa, e o nosso código está funcional. Mas existem problemas nesse tipo de abordagem. Caso tenhamos trinta e seis formulários diferentes que articulam a informação "CPF", a nova demanda da empresa é que cada CPF seja validado, e caso não, o texto ficará em vermelho e surgirá uma mensagem de erro.

Com uma quantidade grande de formulários para configurar, teremos dificuldade em descobrir o trecho adequado do código.

Acionar o atalho "Ctrl + F" e procurar cada trecho de código que contenha a palavra "CPF" seria muito trabalhoso.

Um problema mais grave: caso entre um novo integrante na equipe e sua primeira tarefa é lidar com um novo sistema que cadastre receitas sugeridas pelos clientes. E esse novo sistema da empresa faz uso da informação do CPF dos usuários. O novo integrante terá dificuldade em validar o CPF.

Poderíamos, por exemplo, criar um manual do sistema da empresa para os novos funcionários, mas essa não é a alternativa mais simples.

O ideal é que possamos fazer uma alteração em um único local do sistema, e assim, os CPFs em todas as interfaces de usuário precisariam ser validados.

Com a orientação a objetos, a ideia de dados e funcionalidades - ou "comportamentos" - estarão interligados, gerando uma enorme facilidade na organização e manutenção de um determinado sistema.

**03Idéia central do paradigma OO**

Com base no que você ouviu no vídeo, selecione a alternativa que expressa a ideia central do paradigma da Orientação a Objetos.

Parte superior do formulário

* Todo código deve estar dentro de uma classe.
* Alternativa correta

Dado e funcionalidade sobre ele andam juntos.

No exemplo do vídeo, se o CPF e sua função de validação andassem juntos, teríamos aplicado OO.

* Alternativa correta

Getters e setters devem ser usados em suas classes.

Parte inferior do formulário

Parabéns, você acertou!

Este conteúdo foi útil para o seu aprendizado?

Sim

Não muito

**04Cheiro procedural**

De acordo com as situações citadas no vídeo, escolha a única alternativa que **não é** sinal de programação procedural.

Parte superior do formulário

* Formulário com inúmeros campos distribuídos em várias abas, tudo sendo gerenciado por um único programador.
* Alternativa correta

**Copy & Paste** e Ctrl + F como prática regular do desenvolvedor para propagar mudanças no projeto.

* Alternativa correta

Várias equipes trabalhando em um único projeto.

Para que várias equipes consigam trabalhar em um mesmo projeto, é necessário que as responsabilidades de cada código estejam bem definidas e claras, evitando conflitos na hora de realizar mudanças e evoluções. Código com responsabilidades coesas é sinal do paradigma OO.

Parte inferior do formulário

Parabéns, você acertou!

Este conteúdo foi útil para o seu aprendizado?

Sim

Não muito

**05Seja bem-vindo, Fulano!**

Fulano é novo na equipe da empresa KWZ. Ficou responsável por uma tela de cadastro, onde existe um campo que deve ser validado conforme a regra de negócio XYZ. Ele pergunta a seu gerente onde pode obter informações sobre a tal regra. Marque as alternativas que podem ser identificadas como respostas de quem usa o paradigma procedural em seus projetos.

Parte superior do formulário

* Dependendo do módulo, Fulano deve perguntar ao responsável técnico do mesmo.

Essa resposta denota que a regra está implementada em vários lugares. Quem é o responsável por ela? Como diz o ditado popular "Cachorro que tem muitos donos acaba morrendo de fome". Tipicamente procedural.

* Alternativa correta

"Copie o código de validação que está no formulário ABCD."

Resposta típica de quem usa o paradigma procedural para não arriscar inserir erros em códigos que já estão funcionando ou em produção.

* Alternativa correta

Basta que Fulano consuma uma classe que representa a informação a ser cadastrada pelo campo.

* Alternativa correta

Fulano deve entrar em contato com alguém da equipe de analistas de negócio da empresa, para ele entender a regra e implementá-la.

Essa resposta indica que Fulano deve implementar novamente a regra, gerando mais um local caso a regra precise ser alterada. Tipicamente procedural.

Parte inferior do formulário

Parabéns, você acertou!

**06O que aprendemos?**

Nessa aula conhecemos o paradigma procedural, que era utilizado como prática de programação antes da introdução das linguagens orientadas a objetos.

Foi usada como exemplo a necessidade de validar o CPF em um formulário para discutirmos os principais problemas que podem aparecer neste paradigma.

Em especial, à medida que outros formulários e desenvolvedores precisam da mesma validação de CPF, não era fácil de perceber que já havia procedimentos e funções que faziam esse trabalho, pois dados e funções não possuíam um vínculo tão forte. Isso poderia resultar em mais uma nova função ou trecho de código com responsabilidade parecida.

Além disso, conhecemos a ideia central do paradigma da orientação a objetos, que é criar unidades de código que combinem os dados associados a determinada informação com as funcionalidades aplicadas aqueles dados (por exemplo, CPF + validação). São os atributos e métodos.

Este conteúdo foi útil para o seu aprendizado?

Sim

Não muito