Padrão para criação de projetos - códigos e algoritmos

[**Entendendo e Dominando o Algoritmo: 2**](#_j0t51q63oox4)

[**Criando os Casos de Uso: 2**](#_tirmxjsgjjn3)

[**Documentação com Javadoc: 2**](#_s8ih7wsc35dc)

[**Documentação com código 3**](#_tid9bml48yv0)

[Escrever um código legível, atribuindo bons nomes para classes, variáveis e métodos é fundamental para a velocidade de entendimento e desenvolvimento. 3](#_hpq5qqm7l8wt)

[Jamais deixe de documentar métodos com alto nível matemático. 3](#_bb3j0ogcdxdn)

[**Otimização de Desempenho: 3**](#_ngokelb94jnp)

[**Organização do projeto: 3**](#_3c0798jbu5as)

[Resumo por tópicos: 4](#_b253lmk2rvkr)

[Criando entidades 4](#_utjopzirlb6h)

[Modelagem de tabelas 4](#_xftrw7q41rko)

[Criação de classes java (em composição) 4](#_dq8xtb2430gn)

[Interface 5](#_xi6gdilx2bsm)

[1. Suporta Herança Múltipla: 5](#_g4k9dipoxbeg)

[2. Permite apenas métodos abstratos: 5](#_292h5gj9dmot)

[3. Não contém atributos: 6](#_57wl6ebfzptx)

[4. Não contém construtor 6](#_jkw19vru20p0)

[5. Quando utilizar? 6](#_yb5we3y3v8ss)

[Classe abstrata 6](#_2xuye6yr0ipt)

[1. Não suporta Herança Múltipla 6](#_dfqbyeve20sp)

[2. Pode conter métodos concretos ou abstratos: 6](#_v2fb9wr7jt57)

[3. Pode conter atributos de todos os tipos 6](#_tl91l3ymfmh)

[4. Contém construtor 6](#_c7efwt22lhi1)

[5. Quando utilizar? 6](#_zf9u7jwopcrf)

[Criando views em jsf(padrão indentar) 7](#_xr1n9mjtfty3)

[Geração de relatórios 7](#_d9zeowrr0mh)

[**Entendimento da metodologia Lean para desenvolvimento de códigos 8**](#_209iqqvbofr)

[1. Eliminate Waste (Elimine o desperdício) 8](#_8zkfdvr0hf9o)

[2. Build Quality In (Construa qualidade desde o início) 8](#_jn2z72kga9vv)

[3. Amplify Learning (Amplifique o aprendizado) 8](#_lnytcz379x6n)

[4. Decide as Late as Possible (Decida o mais tarde possível) 8](#_aqbplzm48p6s)

[5. Deliver as Fast as Possible (Entregue o mais rápido possível) 9](#_qccjimafu5y7)

[6. Empower the Team (Empodere o time) 9](#_i28uss0d2mv)

[7. Optimize the Whole (Otimize o todo) 9](#_fy4xspseo84q)

# Entendendo e Dominando o Algoritmo:

Antes de começar a escrever código, é crucial entender o problema e o algoritmo necessário.

**Isso envolve:**

**Analisar o problema:** Compreenda os requisitos, entradas e saídas esperadas.

**Visualizar a solução:** Pense na abordagem geral antes de se preocupar com a implementação detalhada.

# Criando os Casos de Uso:

Os casos de uso descrevem como o sistema será usado em diferentes cenários. Aqui estão alguns passos para criar casos de uso:

**Identifique os atores:** Quem interage com o sistema? (por exemplo, usuários, administradores, etc.)

Descreva os cenários: Para cada ator, liste as ações que eles podem realizar e as respostas do sistema.

**Trate exceções:** Considere situações excepcionais e como o sistema deve se comportar.

# **Documentação com Javadoc:**

O Javadoc é uma ferramenta de documentação para Java. Ele permite que você documente seu código por meio de comentários específicos.   
Aqui estão algumas dicas:

Use /\*\* ... \*/ para comentários Javadoc.

Documente classes, métodos e atributos.

Descreva o propósito, parâmetros e retorno de cada método.

# Documentação com código

## 

## Escrever um código legível, atribuindo bons nomes para classes, variáveis e métodos é fundamental para a velocidade de entendimento e desenvolvimento.

## Jamais deixe de documentar métodos com alto nível matemático.

# Otimização de Desempenho:

Algumas dicas para otimizar o desempenho do código Java:

Evite escrever métodos longos.

Evite várias declarações If-else.

Use tipos primitivos sempre que possível.

Evite criar objetos grandes com frequência, o objeto pode ser grande, mas nem sempre é necessário carregar ele por completo. Nessa questão pode ser trabalhado o padrão DTO, ou criar Fábricas próprias para situações específicas.

# Organização do projeto:

Ter padrão para nome de variáveis e padrões de organização de classes é fundamental para que se evite perda de tempo.

É obrigatório o bom uso de interfaces, classes abstratas para organização dos códigos.   
Se ainda não sabe como usar, estude e aprenda.

<https://www.w3schools.com/java/default.asp>

Más práticas de organização serão analisadas e punidas.

Padrão indentar de codificação

## Resumo por tópicos:

Não faça nada local,siga o padrão do disco G:

Se direto e não seja desorganizado

Faça o protótipo com requisitos mínimos de acordo com o levantamento de casos de uso e faça os testes de mesa

Crie seu algoritmo

Crie uma entidade model > Test sua instância

Crie um entidade repositório DAO > Teste sua consulta

Crie uma entidade regra de negócio > Teste suas regras

Identifique necessidade de criar relatório, se sim crie uma listagem e um documento

Depois de todos testes locais, faça uma view MB + xhtml

Verifique sempre que se sentir perdido, esse documento do [canvas](https://www.canva.com/design/DAE200vFU5Y/Diftffj4AJOR_ZwykfP2Nw/edit?utm_content=DAE200vFU5Y&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton)

Use nomes comuns, em caso de dúvida acesse padrão de [variáveis](https://docs.google.com/spreadsheets/d/1ZJsVMIqOSrfHJXxE9D-qt6LlMCq-5MH8C1sB6Dif8Jc/edit?gid=0#gid=0)

## Criando entidades

Toda entidade é uma tabela, logo deve passar por algumas etapas:

### [Modelagem de tabelas](https://www.devmedia.com.br/modelagem-de-dados-tutorial/20398)

Baseado nos casos de usos, definir quais campos são necessários e quantas tabelas serão necessários, verificando sempre a possibilidade de indexação.

### Criação de classes java (em composição)

Após modelar as tabelas, criar as classes java. Atentando sempre por atribuir padrões em tabelas, usando a anotação de columndefinition.

Trabalhar sempre a possibilidade de orientação à objeto, priorizando sempre que necessário o uso de [enums](https://tomee.apache.org/tomee-8.0/pt/examples/jpa-enumerated.html#:~:text=%C3%80s%20vezes%20pode%20ser%20desej%C3%A1vel,persistence.) e [embed](https://www.youtube.com/watch?v=OZ-z0HlrzNw) para campos quando necessários.

Trabalhar sempre com o uso de interfaces , herança e abstração.

Dentro de uma boa arquitetura , sempre é importante o uso de recursos de padronização.

Exemplo de arquitetura:

EntidadeAbstrata para receber campos comuns

EntidadeInterface para receber metodos comuns

EntidadeAbstrata implementa EntidadeInterface

EntidadeQualquer extends EntidadeAbastrata

Software CRM  
Entidade :Cadastro de cliente

Entidade : Lançamento de avisos

Como criar um software?

Imaginando o cenário

Primeiro cenário a ser tratado, sempre será o crud básico

Salvar,Listar ou Pesquisar,Alterar,Atualizar e Deletar (Create,Read,Update,Delete)

Logo criamos um contrato básico

CrudInterface.java

Esse contrato é o mínimo para tratar uma entidade persistent qualquer.

À medida que o software vai se desenvolvendo, definimos mais contratos.Por exemplo para o usuário desse sistema, poderia ter dashboards padrões.

Ex:NotificacaoDashInterface.java

Que poderia ter os seguintes métodos:  
carregaNotificacoes()

exibeNotificacoes()

detalhaNotificacao()

#### Interface

##### 1. Suporta Herança Múltipla:

Uma classe pode implementar múltiplas interfaces;

##### 2. Permite apenas métodos abstratos:

Nenhum método com implementação. Temos algumas exceções que chegaram com o Java 8, mas essa é a ideia principal;

##### 3. Não contém atributos:

Essa é outra regra que possui uma exceção. Enquanto é comum ouvirmos que interfaces não guardam o estado da classe por não suportarem a instância de atributos, elas ainda podem conter atributos constantes de classe (public static final);

##### 4. Não contém construtor

##### 5. Quando utilizar?

O ideal é utilizar a interface quando várias classes diferentes compartilham apenas a assinatura de seus métodos. Em outras palavras, podemos ter várias classes sem qualquer relação entre si, mas se elas compartilharem as funcionalidades oferecidas pela interface que estamos criando, então faz total sentido que essas classes a implementem. Conclusão: interfaces estão aí para ditar o que uma classe deve fazer, ajudando a definir quais habilidades as classes que assinarem esse "contrato" devem possuir.

#### Classe abstrata

##### 1. Não suporta Herança Múltipla

##### 2. Pode conter métodos concretos ou abstratos:

Isso quer dizer que todos os métodos de uma classe abstrata podem ser tanto concretos como, também, todos podem ser abstratos;

##### 3. Pode conter atributos de todos os tipos

##### 4. Contém construtor

##### 5. Quando utilizar?

Diferente da interface que pode estar envolvida com diversas classes sem qualquer relação entre si, uma classe abstrata continua sendo uma classe. E sabemos que quando uma classe X herda de uma classe Y é o mesmo que dizer que X é um Y. Ou seja, quando queremos criar várias classes que irão compartilhar um mesmo comportamento, uma classe abstrata é o componente ideal para ser a base para criação de todas elas, servindo como um molde para as futuras classes que irão derivar dela. Conclusão: classes abstratas definem a identidade de suas classes derivadas ditando o que e como uma classe deve se comportar, o que aumenta o acoplamento entre classes, porém faz total sentido em algumas situações.

No geral, esses são os pontos que eu considero mais importantes a serem lembrados. Existem muito mais detalhes relacionados à forma correta de se utilizar esses dois componentes, mas praticando e lendo o código de devs mais experientes, acredito que tudo ficará mais fácil com o tempo. Se tiver alguma dúvida é só avisar

## Criando views em jsf(padrão indentar)

**Requisitos**

Precisa entender a árvore de componentes

Entender noções de responsividade(tela dividida em 12 partes)

Padrão classes tema última:

*p-dispositivo-quantidadeColunas*

col=celular

lg=desktop

md=tablet

<sgw-app:grid>

<sgw-app:gridItem columnClasses=”p-lg-2 p-md-6 p-col-12”>

conteudo

</sgw-app:gridItem/>

</sgw-app:grid>

O exemplo acima:

Para celular o conteúdo ocupará toda a tela

Para tablet a metade

Para desktop ocupará 2 partes

### Geração de relatórios

Verificar se há padrão de interface/abstração

Testar quebra de linha para todos campos

Testar caracteres especiais

Data e hora de impressão

Data de criação do registro

Uso de json/model

# Entendimento da metodologia Lean para desenvolvimento de códigos

## 1. Eliminate Waste (Elimine o desperdício)

Este princípio foca em eliminar atividades que não agregam valor ao produto ou serviço, incluindo processos desnecessários, tempo de espera e features (características) ainda pouco relevantes.

A sua equipe pode, por exemplo, eliminar documentação ou reuniões desnecessárias que não agregam valor ao projeto ou remover recursos que não são necessários para o cliente.

## 2. Build Quality In (Construa qualidade desde o início)

O objetivo é construir qualidade no produto ou serviço desde o início, em vez de depender de testes para encontrar erros. Isso pode ser feito implementando testes automatizados para detectar erros precocemente ou realizando revisões de código para identificar possíveis problemas.

## 3. Amplify Learning (Amplifique o aprendizado)

Aqui queremos enfatizar a importância de aprender e melhorar continuamente, buscando feedback e usando-o para fazer ajustes. Para isso, é possível realizar entrevistas com usuários para entender as necessidades e preferências ou realizar retrospectivas após cada ciclo de desenvolvimento para identificar oportunidades de melhoria.

## 4. Decide as Late as Possible (Decida o mais tarde possível)

Parece estranho, mas o adiamento de decisões até o último momento possível permite mais informações e flexibilidade na tomada de uma decisão. Isso pode ser útil, por exemplo, quando estamos realizando experimentos para testar diferentes opções antes de tomar uma decisão final.

## 5. Deliver as Fast as Possible (Entregue o mais rápido possível)

Este princípio concentra-se em fornecer valor ao cliente o mais rapidamente possível para reduzir o tempo de lançamento e aumentar a capacidade de resposta às exigências por mudança. Uma maneira bem interessante de trabalhar com este princípio é entregando um software que esteja funcionando em incrementos menores para obter feedback dos usuários mais rapidamente.

## 6. Empower the Team (Empodere o time)

Aqui estamos enfatizando a importância de capacitar as equipes para tomar decisões e assumir a responsabilidade por seu trabalho, em vez de depender da gestão de cima para baixo. Para isso, permita que os membros da equipe tomem decisões sobre como realizarão suas tarefas e forneça a eles as ferramentas e recursos necessários para ter sucesso.

## 7. Optimize the Whole (Otimize o todo)

Por último, este princípio envolve otimizar todo o sistema, em vez de apenas partes individuais, para maximizar a eficiência e eficácia geral. Lembre-se de focar em toda a cadeia de valor, não apenas em um aspecto dela, e identifique quais são os gargalos no processo.

Algumas fontes escolhidas a dedo de como programar de forma inteligente:

<https://medium.com/@ti.lucasfraga/padr%C3%B5es-de-projeto-design-patterns-na-pr%C3%A1tica-com-java-1a9cf5131d47>

Entendendo funcionamento de IDE métodos e construtores

Erros comuns

Hibernate

### More than one row with the given identifier was found:

Mapeamento bi-direcional com mapped by no lugar errado

Realmente tem mais de um registro para relacionamento que não podia, exemplo: copiar algum registro que todos objetos deveriam ter novo ID