
EMBARCADERO CONFERENCE
RAD
C++
embarcadero®

Programação Defensiva

Técnicas para um código mais robusto

Julio Cesar de Andrade Amorim

Programação Defensiva

Todo sistema é construído esperando que os dados sigam um fluxo que foi previamente desenhado e definido como o "caminho certo". Para que esse caminho seja respeitado, existem certas condições a serem consideradas.

Programação Defensiva

Uma das principais é a entrada de dados (inputs). Ela precisa estar nas mesmas condições em que os arquitetos, designers e desenvolvedores pensaram. Dessa forma, os resultados (outputs) do sistema serão sempre os esperados.

Programação Defensiva

No dia a dia é assim que funciona?

Programação Defensiva

A pessoa ou o programa que utilizará o seu código fará entrada de dados que nem sempre estão da forma que se espera. O que gera erros em tempo de execução ou pior, executa o código de forma não planejada gerando resultados inesperados, podendo violar a integridade dos dados.

Programação Defensiva

Para evitar esse tipo de problema, foi desenvolvido um conjunto de práticas que garantem que o código saiba gerenciar situações não esperadas, garantindo a disponibilidade do sistema, a integridade dos dados e a saúde da aplicação (e seus envolvidos). A esse conjunto de práticas foi dado o nome de **Programação Defensiva**.

Programação Defensiva

"Direção Defensiva"

Direção defensiva é o ato de conduzir de modo a evitar acidentes, apesar das ações (erradas) os outros e das condições adversas (contrárias), que encontramos nas vias de transito.

Programação Defensiva

Similar à direção defensiva, na programação defensiva codificamos de modo a manter a aplicação funcionando apesar das ações incorretas dos outros e das condições adversas que encontramos no dia a dia.

Programação Defensiva

Robustez

- Significa sempre tentar fazer algo que permita que o software continue operando, mesmo que isso às vezes leve a resultados imprecisos

Corretude

- Significa nunca retornar um resultado impreciso
- Não retornar nenhum resultado será melhor do que retornar um resultado incorreto

Qual característica deve ser priorizada?

Programação Defensiva

- Validação de entradas
- Programação por contrato
- Barricadas

Programação Defensiva

Validação de Entradas

Regra de ouro

"Nunca confie em dados vindos de fontes externas"

Programação Defensiva

Validação de Entradas

Devemos definir um conjunto de valores válidos

Ao receber os valores, validar com esse conjunto

Definir um comportamento caso os valores não sejam válidos, priorizando a robustez ou a corretude.

Programação Defensiva

Validação de Entradas – Exemplo

```
procedure TConta.Depositar(Value: Currency);
begin
  FSaldo := FSaldo + Value;
end;
```

E se o usuário informasse um valor negativo?

Validação de Entradas – Exemplo

Nesse caso deveríamos verificar se o valor recebido é maior do que zero e dependendo da situação verificar a quantidade de casas decimais.

Programação Defensiva

Validação de Entradas – Exemplo

```
procedure TConta.Depositar(Value: Currency);
begin
  if Value < 0.01 then
    Exception.Create('O valor não pode ser inferior a 0,01');

  if Decimais(Value) > 2 then
    Exception.Create('O valor não pode ter mais de 2 casas decimais');

  FSaldo := FSaldo + Value;
end;
```

Programação Defensiva

Programação por Contrato

A programação por contrato vem do termo em inglês *Design by Contract (DbC)* e foi criada por Bertrand Meyer em 1986.

Programação Defensiva

Programação por Contrato

A ideia central segue o mesmo princípio dos contratos celebrados por duas ou mais partes em que de um lado temo o **cliente** e do outro o **fornecedor**

Programação Defensiva

Programação por Contrato

O contrato vai reger as regras, obrigações e benefícios de ambos os lados, como:

- As obrigações que o cliente tem que cumprir para receber os benefícios do fornecedor
- As obrigações que o fornecedor tem de cumprir na entrega do benefício ao cliente.
- As regras imutáveis do contrato

Programação Defensiva

Programação por Contrato

Os termos citados anteriormente pode ser descritos como:

- Pré-Condições
- Pós-Condições
- Invariantes

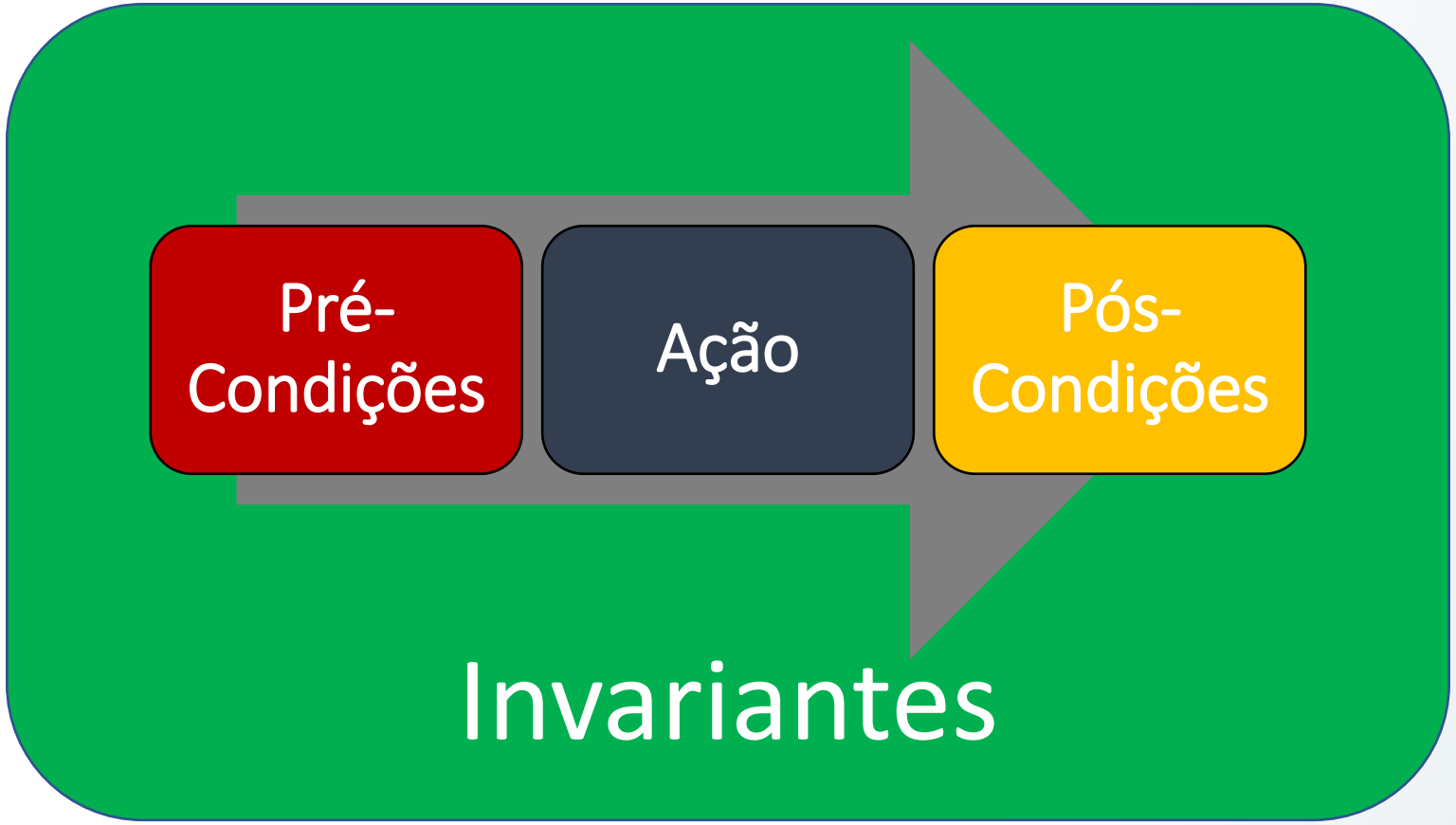
Programação Defensiva

Programação por Contrato

A definição do contrato faz parte do design da solução e podemos obtê-la por meio de três perguntas:

- O que o contrato espera (obrigações do cliente)
- O que o contrato garante (obrigações do fornecedor)
- O que o contrato mantém (regras imutáveis)

Programação por Contrato



Programação Defensiva

Programação por Contrato

Pré-Condições

- O que deve ser verdadeiro para a rotina poder utilizada (requisitos mínimos necessários no início da chamada)

Programação Defensiva

Programação por Contrato

Pós-Condições

- O que deve ser verdadeiro ao término da execução da rotina

Programação Defensiva

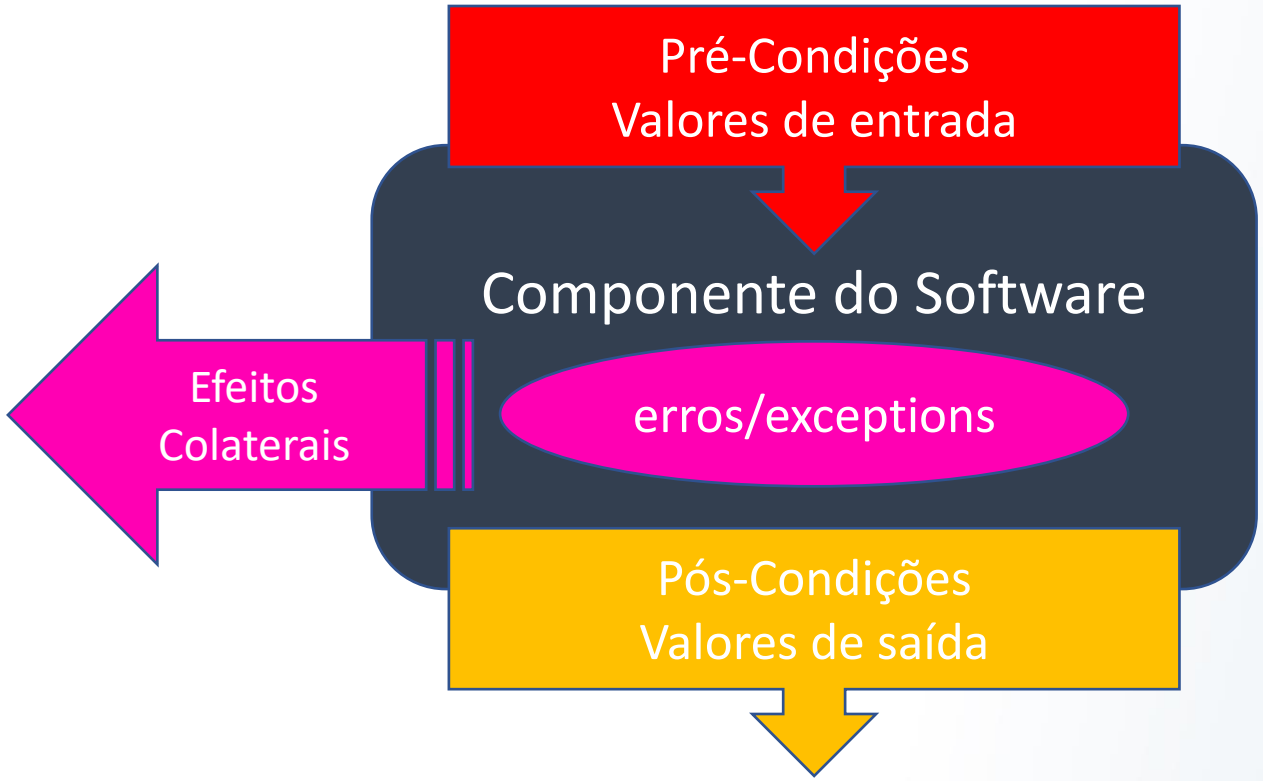
Programação por Contrato

Invariantes

- Condições que devem sempre ser verdade, antes, durante e após a execução da rotina

Programação Defensiva

Programação por Contrato



Se não for possível cumprir o contrato, geramos exceção

Programação Defensiva

Programação por Contrato – Exemplo

```
procedure TConta.Sacar(Value: Currency);
begin
  FSaldo := FSaldo - Value;
end;

end.
```

Programação por Contrato – Exemplo

O que o contrato espera?

Um valor acima de zero

O que o contrato garante?

Que o valor informado seja subtraído do saldo

O que o contrato mantém?

O saldo não pode ser inferior a zero.

Programação Defensiva

Programação por Contrato – Exemplo

```
procedure TConta.Sacar(Value: Currency);
begin
  if Value < 0.01 then
    Exception.Create('O valor não pode ser inferior a 0,01');

  if not ValidarSaldo(Value) then
    Exception.Create('Saldo insuficiente');

  FSaldo := FSaldo - Value;
end;
```

Programação Defensiva

Barricadas

As barricadas são trechos de código dentro do sistema que atuam como barreiras visando impedir/minimizar a propagação de problemas causados por dados incorretos para outras partes do sistema

Programação Defensiva

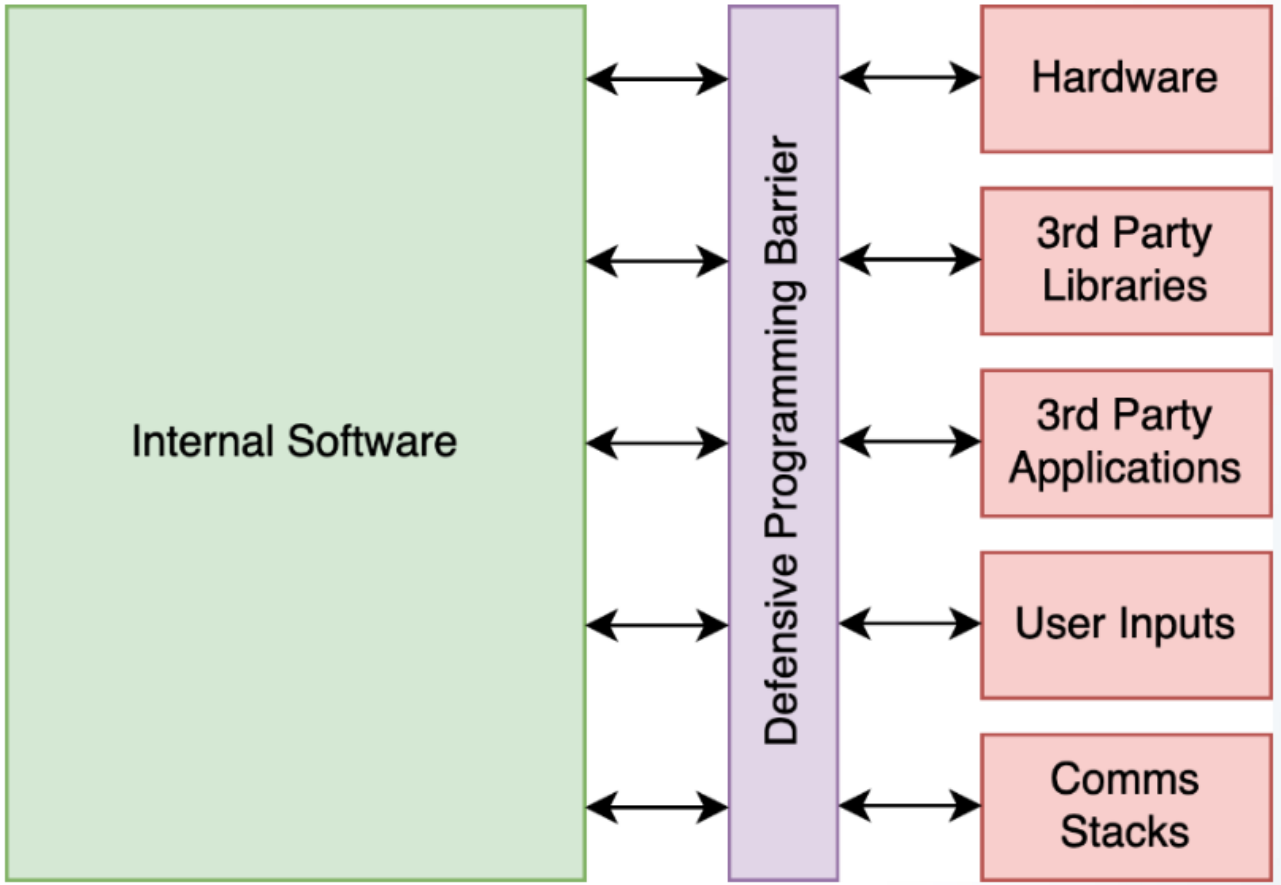
Barricadas

Na barreira vamos ter classes e métodos que vão verificar todos os dados que passem por ela, garantindo a sua integridade ou informando se forem inválidos.

Criando barreiras, vamos restringir e centralizar a responsabilidade da verificação de dados, separando a parte do código que trabalha com dados limpos, liberando-os da verificação

Programação Defensiva

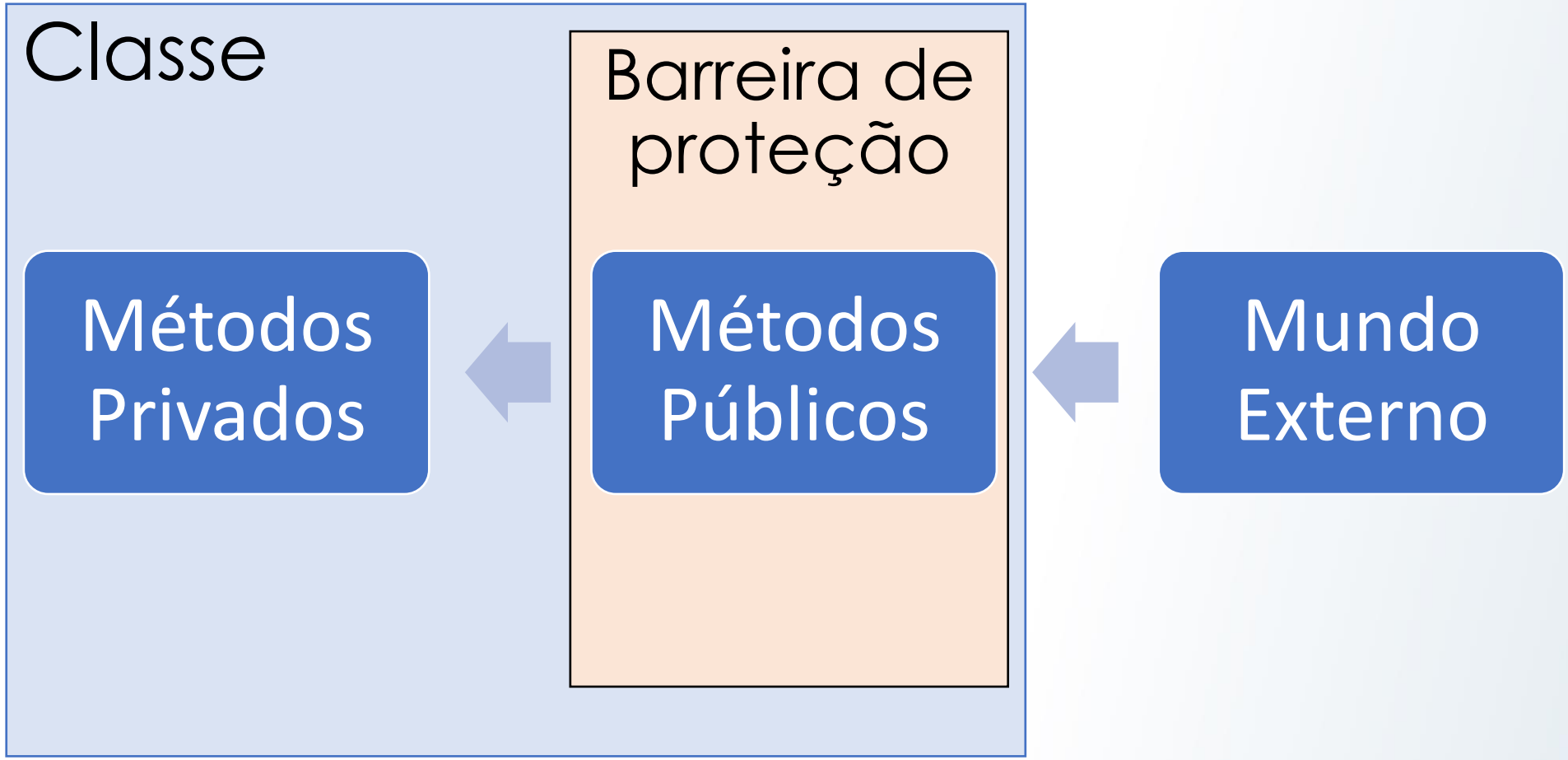
Barricadas



Programação Defensiva

Aplicando esse conceito às classes , os métodos públicos se encarregariam de criar essa barreira, validando todos os dados vindo do mundo externo, enquanto os métodos privados não precisarão fazer outras validações, pois os dados que serão repassados a eles são limpos e confiáveis.

Programação Defensiva



Programação Defensiva

Nos exemplos anteriores, optei pela corretude, mas poderíamos optar pela robustez, assumindo valores para os parâmetros recebidos que não atenderam a regra de negócio.

Programação Defensiva

Quando criamos um if defensivo, estamos assumindo uma responsabilidade que não deveríamos, primeiramente devemos sempre nos perguntar o quanto essa ação vai impactar na regra de negócio.

Programação Defensiva

Problemas técnicos se resolvem com soluções técnicas, problemas de negócios precisam ser resolvidos com o especialista do negócio. Antes de sair escrevendo código, criando validações, tente entender a regra de negócio e por que tem de ser daquele jeito, sempre faça perguntas.

Juliocandrade@gmail.com

facebook.com/juliocesar.andrade.982

Julio.c.andrade

linkedin.com/in/juliocandrade