Módulo 1: Orientação a Objetos

Djonathan Barros

Apresentação

- Desenvolvedor BackEnd Sênior

- Bacharel em Sistemas da Informação
- Especialista em Engenharia de Software
- Mestrando em Ciência da Computação

https://br.linkedin.com/in/djonathanbarros

Design Orientado a Objetos

Uma linguagem suporta um Estilo de Programação se ela provê funcionalidades que tornam conveniente (razoavelmente fácil, seguro e eficiente) utilizar esse estilo. Uma linguagem não suporta uma técnica se é necessário um esforço excepcional ou habilidade para escrever os determinados programas; nesse caso, a linguagem simplesmente habilita os programadores a utilizar a técnica.

[STROUSTRUP, 1988] Tradução nossa.

Conceitos Chave

Estilo procedural: Decida quais procedimentos você deseja, utilize o melhor algoritmo que você possa encontrar.

```
procedimento liberarGuia(codigoGuia)
inicio
    tuplaGuia = consultarGuia(codigoGuia) // como é armazenado ???
    atualizarGuia(tuplaGuia, .verdadeiro) // como a nova informação é mantida na estrutura de dados ???
fim
```

Conceitos Chave

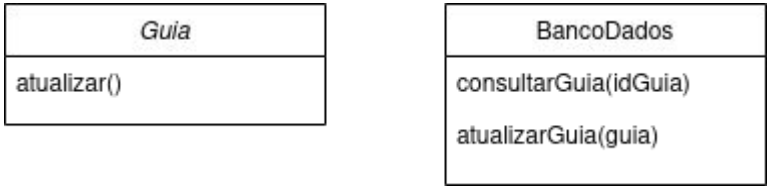
Ocultação de Dados: Decida quais procedimentos você deseja, particionar o programa de forma que os dados fiquem ocultos em módulos.

```
importar banco.dados.consultarGuia // qual o endereço do banco e senha ???
importar banco.dados.atualizarGuia


procedimento liberarGuia(codigoGuia)
inicio
        tuplaGuia = consultarGuia(codigoGuia)
        atualizarGuia(tuplaGuia, .verdadeiro)
fim
```

Conceitos Chave

Abstração de Dados: Decida quais tipos você deseja, forneça um conjunto de operações para cada tipo.

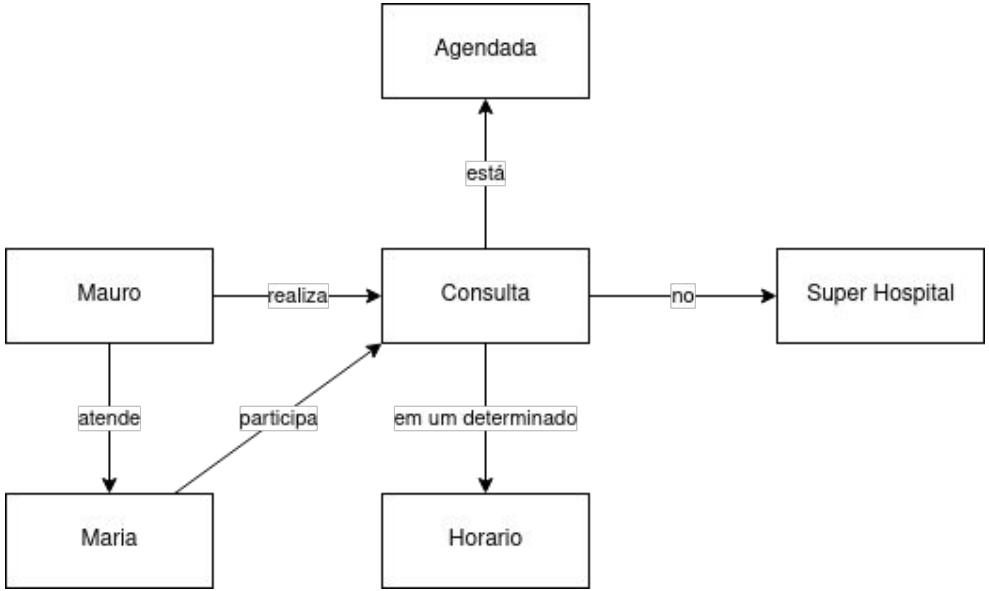


Orientação a Objetos (Conceito)

Decida quais classes você deseja; forneça um conjunto de operações para cada classe; torne as semelhanças explícitas utilizando herança.

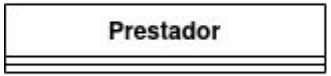
OOP (Objetos)

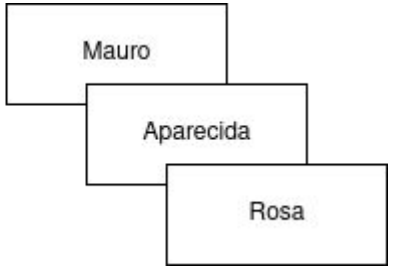
Coisas (pessoas, animais, objetos), características (cor, sabor), eventos (confirmação, cancelamento), estado (pendente, confirmado), **basicamente tudo que possamos chamar com um substantivo/nome próprio**, incluindo objetos abstratos [PECINOVSKY, 2013].



OOP (Classes)

Grupos de objetos com características muito similares, do ponto de vista técnico compõem o conjunto de tipos de uma linguagem na categoria de Tipo de Dados Abstratos. Objetos de determinada classe são denominados como **instâncias** [PECINOVSKY, 2013].

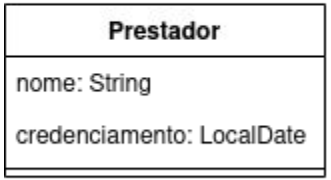


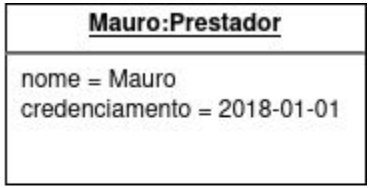


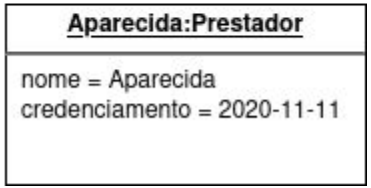
OOP (Atributos)

Descrevem as características de um objeto, fazendo parte do arquétipo da classe.

Cada instância da classe armazena os valores de seus atributos, porém atributos estáticos são compartilhados entre as instâncias.







OOP (Mensagens/Métodos)

Objetos se comunicam através de mensagens, a unidade de abstração dessas mensagens é chamada de método.

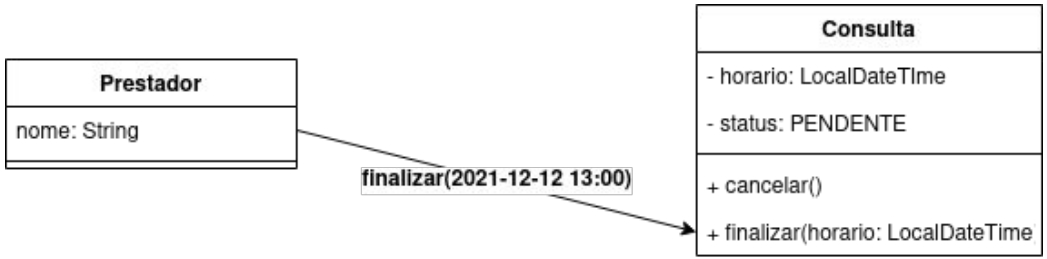
Métodos abstraem uma determinada lógica que será aplicada a uma referida instância sempre que receber uma mensagem específica.

Métodos podem conter parâmetros e retornar valores ao final de sua execução.

OOP (Encapsulamento)

Ninguém deve se preocupar com a forma em que um objeto está organizado ou o quê ele sabe.

Quanto menos comprometido uma classe tiver com o mundo exterior menor é a superfície de mudança que pode impactar a manutenabilidade do sistema.



OOP (Generalização)

Refere-se a habilidade de organizar os objetos em estruturas hierárquicas.

Classes podem implementar dois tipos de Generalização:

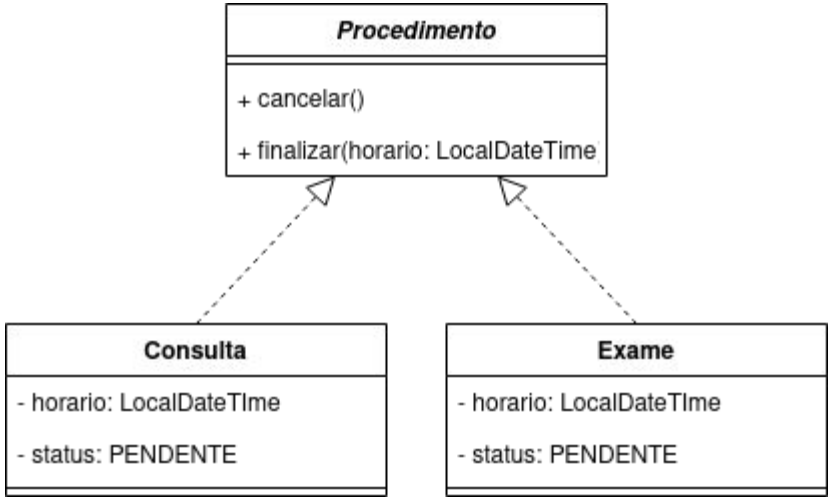
Tipo: Classes diferentes compartilham a mesma interface.

Implementação: Onde além da interface, a implementação também é compartilhada.

Generalização (Interfaces)

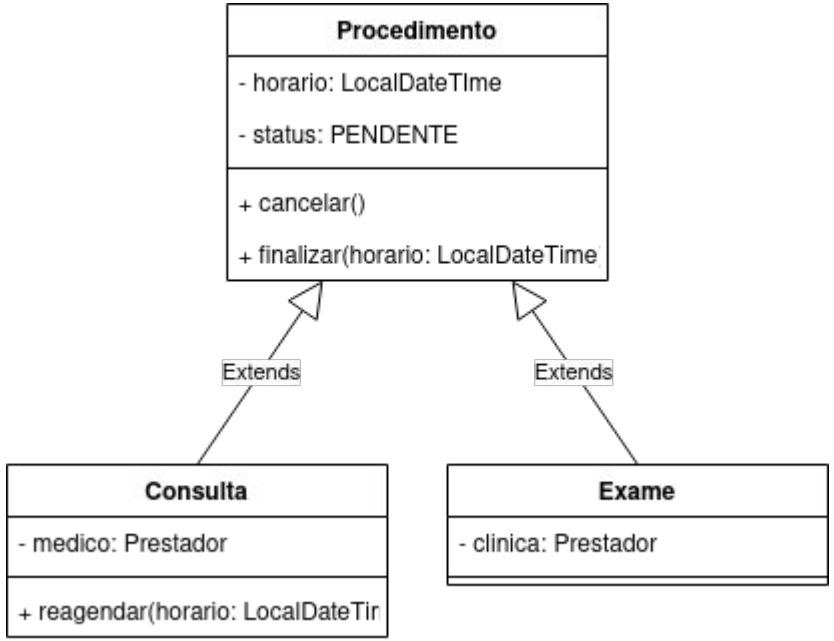
O conceito de Interface tem duas aplicações:

- O conjunto de operações que compõem o contrato de uma classe com os consumidores.
- Herança de Tipos, onde diversas classes podem compartilhar a mesma interface.



Generalização (Herança)

Refere-se diretamente a Generalização de Implementação, onde classes compartilham sua implementação além da interface.



Boas Práticas de Design

DRY/YAGNI/KISS

Don’t Repeat Yourself!

You Ain’t gonna need it!

Keep it simple, stupid.

Alta Coesão e Baixo Acoplamento

Coesão: refere-se a ligação ou harmonia entre elementos textuais.

Lembre-se de suas aulas de Português, o que define um texto coeso ???

Acoplamento: que está acoplado (ligado, conectado) / quão acoplado ???

Discussão

Como esses conceitos se aplicam a OOP???

SOLID

Acrônimo para os Cinco Princípios de um bom Design Orientado a Objeto, cunhado por Bob Martin em [MARTIN, 2000].

Single Responsibility Principle -> Princípio da Responsabilidade Única

Open-closed Principle -> Princípio do Aberto-Fechado

Liskov Substitution Principle -> Princípio da Substituição de Liskov

Interface Segregation Principle -> Princípio da Segregação de Interfaces

Dependency Inversion Principle -> Princípio da Inversão de Dependências

Princípio da Responsabilidade Única

Descrição: Todo objeto do seu sistema deveria ter uma única responsabilidade, e todos os serviços desse objeto deveriam focar em realizar essa responsabilidade única.

“Você implementou o SRP corretamente quando cada um dos seus objetos tem somente um motivo para mudar.”

[MCLAUGHLIN, 2007 “Tradução nossa”]

Princípio do Aberto-Fechado

Descrição: Classes deveriam estar abertas para extensão e fechadas para modificação.

“Você fecha classes ao não permitir a ninguém tocar seu código de funcionamento. Você abre uma classe permitindo que elas possam ser generalizadas.”

[MCLAUGHLIN, 2007 “Tradução nossa”]

Princípio da Substituição de Liskov

Descrição: Subtipos devem ser substituíveis por seus tipos base.

*“O LSP é inteiramente relacionado há **Herança Bem Planejada**. Quando você herda de sua classe base, você deve ser capaz de substituir sua subclasse pela classe base sem as coisas darem terrivelmente mau. Do contrário, você está utilizando a herança incorretamente.”*

[MCLAUGHLIN, 2007 “Tradução nossa”]

Princípio da Segregação de Interfaces

Descrição: Muitas interfaces especializadas são melhores que uma interface de propósito geral.

“Se você tem uma classe com vários clientes, ao invés de carregar a classe com todos os métodos que os cliente necessitam, crie interfaces específicas para cada cliente e herde multiplamente para a classe. ... clientes devem ser categorizados pelos seus tipos e interfaces para cada cliente devem ser criadas.”

[MARTIN, 2000 “Tradução nossa”]

Princípio da Inversão de Dependências

Descrição: Dependa de abstrações ao invés de classes concretas.

“A implicação deste princípio é simples. Toda dependência no Design deve mirar uma interface, ou uma classe abstrata. Nenhuma dependência deveria mirar uma classe concreta”

[MARTIN, 2000 “Tradução nossa”]

Padrões de Projetos

Técnica de um projetista experiente

“Uma coisa que os melhores projetistas sabem que não devem fazer é resolver cada problema a partir de princípios elementares ou do zero. Em vez disso, eles re-utilizam soluções que funcionaram no passado. Quando encontram uma boa solução, eles a utilizam repetidamente.”

[GAMMA, 1995]

Padrões

“Que serve de referência ou de modelo.”

Dicionário Aurélio

“Cada padrão descreve um problema no nosso ambiente e o cerne da sua solução, de tal forma que você possa usar essa solução mais de um milhão de vezes, sem nunca fazê-lo da mesma maneira.”

Cristopher Alexander

Padrões de Projetos OO

“Descrições de objetos e classes comunicantes que precisam ser personalizadas para resolver um problema geral de projeto num contexto particular.”

[GAMMA, 1995]

“Desenvolvedores OO experientes construíram um repertório tanto de princípios gerais quanto de soluções idiomáticas que os guiam na criação de software. Esses princípios e idiomas, se codificados em um formato estruturado, descrevendo o problema e a solução, além de serem devidamente denominados, podem ser chamados de padrões”

[LARMAN, 2012]

Estrutura básica de um Padrão

Nome do padrão: Orelhas de Coelho

Problema: Como ensinar uma criança a fazer o laço no cadarço de seu tênis?

Solução: “Orelhas de coelho, orelhas de coelho, brincando na árvore. Atravessa a árvore tentando me pegar. Orelhas de coelho, salta no buraco, saiu do outro lado bonito e ousado”

Classificação

Propósito

- **Criação:** Se preocupam com o processo de criação de objetos.
- **Estrutural:** Lidam com a composição de classes ou objetos.
- **Comportamental:** Caracterizam as maneiras pelas quais classes ou objetos interagem e distribuem responsabilidades.

Escopo

- **Classe:** Estáticos, fixados em tempo de compilação. Lidam com os relacionamentos entre classes e suas subclasses.
- **Objeto:** Dinâmicos, mudados em tempo de execução. Lidam com relacionamentos entre objetos.

Catálogo de Padrões (GOF)

		Propósito		
		De criação	**Estrutural**	**Comportamental**
Escopo	**Classe**	Factory Method	Adapter (classe)	Interpreter Template Method
	Objeto	Abstract Factory Builder Prototype Singleton	Adapter (objeto) Bridge Composite Decorator Façade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

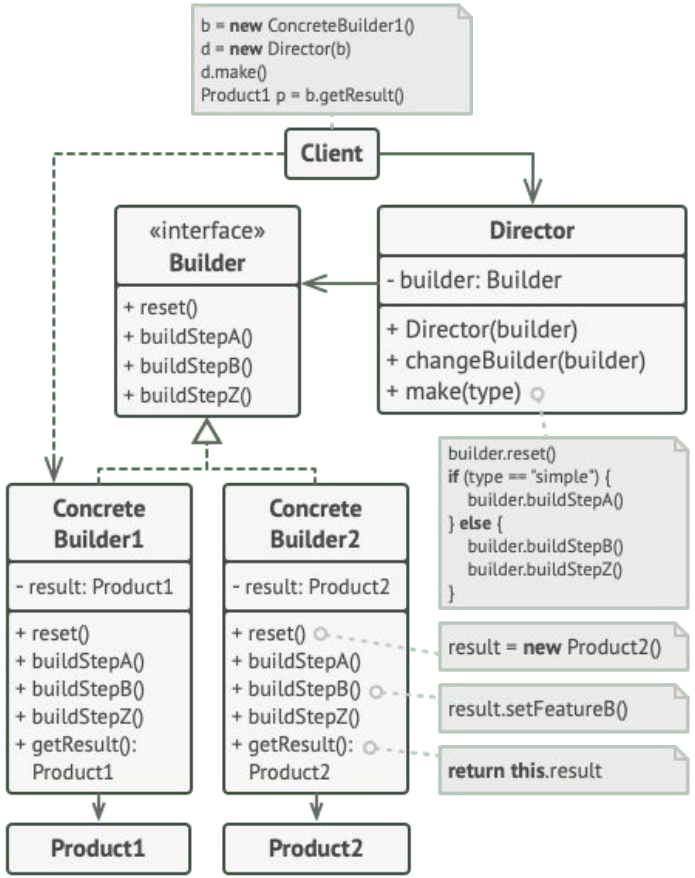
Builder (Descrição)

Nome do padrão: Builder

Problema: Como separar a construção de um objeto complexo de sua representação de forma que o mesmo processo de construção possa criar diferentes representações?

Solução: Extraia o código de criação para fora da classe e implemente os em classes “construtoras”.

Builder (Estrutura)



https://refactoring.guru/pt-br/design-patterns/builder

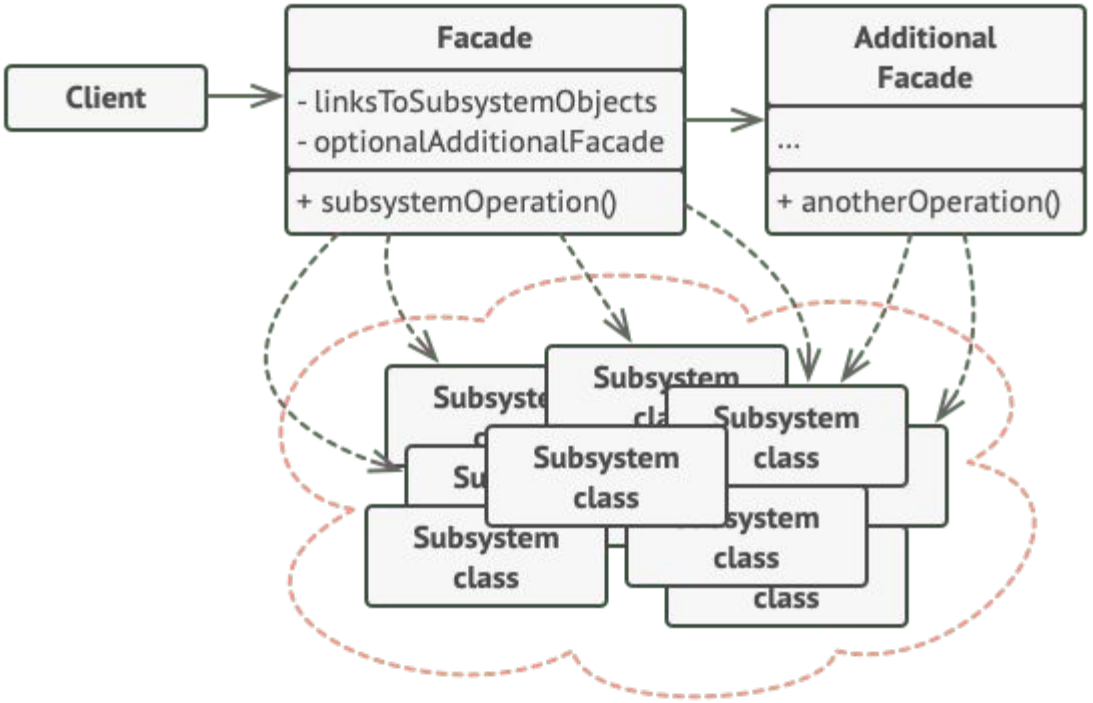
Facade (Descrição)

Nome do padrão: Facade

Problema: Como abstrair a complexidade de um sub-sistema?

Solução: Defina uma interface de alto-nível que torne o subsistema mais fácil de utilizar.

Facade (Estrutura)



https://refactoring.guru/pt-br/design-patterns/facade

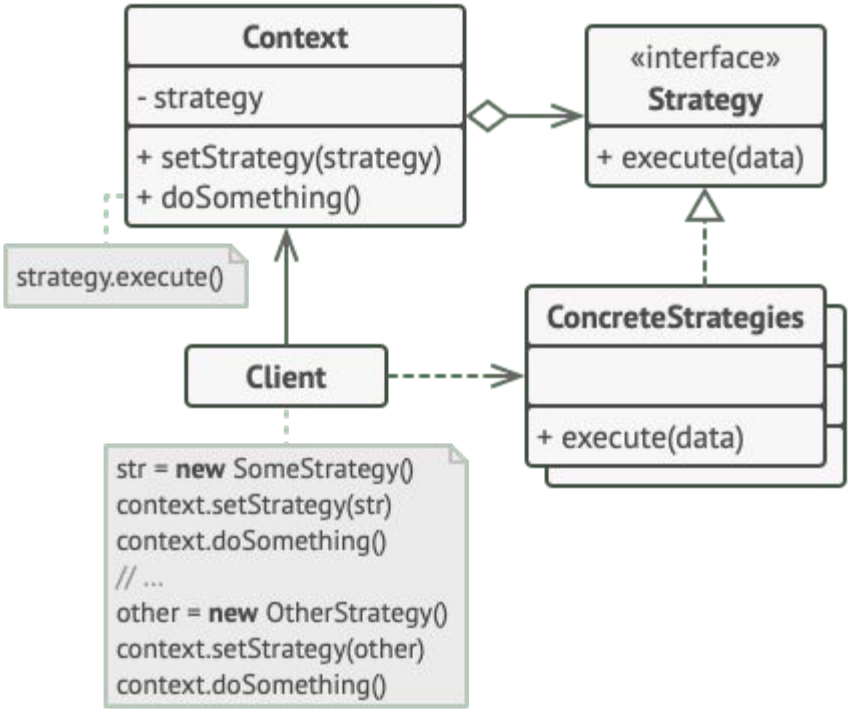
Strategy (Descrição)

Nome do padrão: Strategy

Problema: Como definir uma família de algoritmos intercambiáveis?

Solução: Defina uma interface em comum, bem como um Contexto de orquestração desse comportamento para disparar as lógicas de cada subclasse dessa família.

Strategy (Estrutura)



https://refactoring.guru/pt-br/design-patterns/strategy

Q&A

Bibliografia

GAMMA, Erich et al. **Design patterns: elements of reusable object-oriented software.** Pearson Deutschland GmbH, 1995.

LARMAN, Craig. **Applying UML and patterns: an introduction to object oriented analysis and design and interative development.** Pearson Education India, 2012.

MARTIN, Robert C. **Design principles and design patterns.** Object Mentor, v. 1, n. 34, p. 597, 2000.

MCLAUGHLIN, Brett; POLLICE, Gary; WEST, David. **Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D.** " O'Reilly Media, Inc.", 2007.

PECINOVSKY, Rudolf. **OOP-Learn Object Oriented Thinking & Programming.** Tomáš Bruckner, 2013.

STROUSTRUP, Bjarne. **What is object-oriented programming?.** IEEE software, v. 5, n. 3, p. 10-20, 1988.