Notas sobre Conceitos Fundamentais

Type	Info
Topics	📚 Studies
Notebooks	Programação Profissional com Clean Code
Related Notes	Ementa do Curso

Princípios SOLID

- Princípios: Single Responsability Principle (SRP); Open/Close Principle (OCP); Liskov Substituion Principle (LSP); Interface Segregation Principle (ISP); Dependency Inversion Principle (DIP)
- Práticas devem ser unidas no código para melhor escrita de código limpo
- Tudo é basicamente sobre o SRP; os demais são usados para que o SRP seja possível; o SRP é a base do SOLID

Princípio DRY (Don’t Repeat Yourself)

- Cada parte do conhecimento da aplicação deve ter uma única representação
- Evitar duplicar código; promove a reutilização; reduz a redundância; facilita a manutenção; evolução da aplicação

Princípio KISS (Keep It Simple, Stupid)

- Enfatiza simplicidade no design; facilita a compreensão, manutenção e modificação no código; menor a probabilidade de erros

Princípios YAGNI (You Ain’t Gonna Need It)

- Enfatiza a importância de não implementar funcionalidades que não são necessárias no momento; evitar adicionar modificações no código baseado em suposições sobre futuras necessidades
- Foco apenas no que realmente é necessário agora

Acoplamento e Coesão

- “Classes devem ter alta coesão e baixo acoplamento”
- Acoplamento
  - Refere-se ao grau de dependência entre os módulos
  - Acoplamento fraco: utilização de interfaces ou abstrações para interagir; facilita a manutenção; promove a reusabilidade e testabilidade

```
// Acoplamento Forte
public class Order
{
    public void ProcessOrder()
    {
        Inventory inventory = new Inventory();
        inventory.UpdateStock();
    }
}
```

```
// Acoplamento Fraco
public class Order
{
    private IInventory _inventory;

    public Order(IInventory inventory)
    {
        _inventory = inventory;
    }

    public void ProcessOrder()
    {
        _inventory.UpdateStock();
    }
}
```

- Coesão
  - Elementos relacionados para trabalhar juntos em uma única tarefa ou objetivo; medida usada para definir o quão focado e unificado é um módulo
  - Coesão alta: torna o código mais fácil de entender; promove a clareza, manutenção (alterações são locais) e reusabilidade (definidos para uma única responsabilidade)

```
// Coesão Baixa
public class Employee
{
    public void CalculateSalary() { /* ... */ }
    public void GenerateReport() { /* ... */ }
    public void AttendMeeting() { /* ... */ }
}
```

```
// Coesão Alta
public class SalaryCalculator
{
    public void CalculateSalary() { /* ... */ }
}

public class ReportGenerator
{
    public void GenerateReport() { /* ... */ }
}

public class MeetingScheduler
{
    public void AttendMeeting() { /* ... */ }
}
```

Funções Puras

- Determinismo: mesmos argumentos, mesmos resultados
- Sem efeitos colaterais: não altera estado fora do seu escopo (não modifica variáveis globais); não realiza operações de I/O; não depende de estados fora do escopo
- Facilidade de testes; compreensão; paralelismo

```
public class CalculadoraDeImpostos
{
    public decimal CalcularImposto(decimal renda)
    {
        return renda * 0.15m; // Função pura
    }
}
```

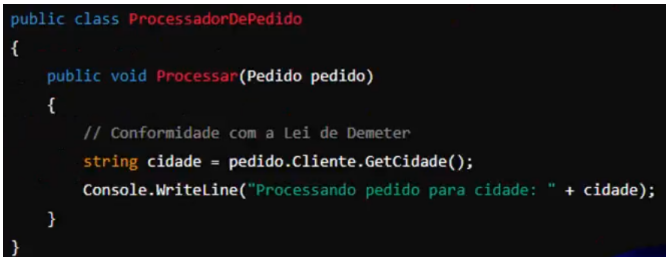
```
public class ServicoDeImpostos
{
    private CalculadoraDeImpostos _calculadora = new CalculadoraDeImpostos();

    public void ProcessarImposto(decimal renda)
    {
        var imposto = _calculadora.CalcularImposto(renda);
        SalvarImposto(imposto); // Função impura
    }

    private void SalvarImposto(decimal imposto)
    {
        // Código para salvar o imposto no banco de dados
    }
}
```

- Utilize funções puras para a lógica de negócios; deixe as impuras para as interações com o ambiente externo

Lei de Demeter (LoD)

- Também conhecida como “Principío do Mínimo Conhecimento”
- Não fale com estrandos: objeto deve ter conhecimento limitado sobre outros objetos
- Fale apenas com amigos: comunicar com objetos imediatos; evitar acessar objetos profundos ou realizar chamadas encadeadas
    - Métodos: próprio objeto; passados como parâmetro; criados; atributos próprios; variavéis locais
- Manutenibilidade: facilita a modificação

```
public class ProcessadorDePedido
{
    public void Processar(Pedido pedido)
    {
        // Violação da Lei de Demeter
        string cidade = pedido.Cliente.Endereco.Cidade;
        Console.WriteLine("Processando pedido para cidade: " + cidade);
    }
}
```



```
public class ProcessadorDePedido
{
    public void Processar(Pedido pedido)
    {
        // Conformidade com a Lei de Demeter
        string cidade = pedido.Cliente.GetCidade();
        Console.WriteLine("Processando pedido para cidade: " + cidade);
    }
}
```

LoD vs SRP

- Possuem focos e objetivos parecidos; reduzir o acoplamento
- São conceitos que caminham juntos durante a implementação do código

4 Regras da Simplicidade

- Rodar todos os testes
  - Código deve passar em todos os testes; mudanças não introduzem novos bugs
  - Mantém a confiabilidade e detecção de regressão
- Revelar a intenção
  - Código claro; sem a necessidade de documentação
  - Facilita a manutenção
- Não duplicar código
  - Pedaço de conhecimento deve ser único, inequívoco e autoritativo
  - Dificuldade a manutenção se precisar ser replicado
- Minimizar o número de entidades (classes, métodos, variáveis, etc)
  - Evitar a criação de classes, métodos ou variáveis desnecessárias
  - Reduz a complexidade do sistema

Linguagem Onipresente

- Uso de terminologia em comum durante o desenvolvimento; mesma comunicação entre membros, documentação, código-fonte, testes
- Elimina mal-entendidos; entendimento claro dos conceitos e funcionalidades
  - Consistência: mesma terminologia usada em reuniões, documentação, código, testes, etc

- Clareza: definição de termos de forma clara e específica; evitar jargões técnicos ou ambiguidade
- Compartilhada: toda a equipe (desenvolvedores, analistas, testers, stakeholders, etc) usa a mesma linguagem
- Baseada no domínio: termos derivados do problema; reflete a realidade do negócio

- Benefícios: comunicação melhorada; código mais claro; documentação coerente

```
// Classe usando um termo inconsistente "OcorrenciaSeguradora"
public class GeradorDeRelatorio
{
    public void CriarRelatorioDeOcorrencias()
    {
        // Lógica para gerar relatório de sinistros
        Console.WriteLine("Gerando relatório de ocorrências seguradoras...");
    }
}
```

```
// Classe usando o termo correto "Sinistro"
public class RelatorioDeSinistros
{
    public void GerarRelatorio()
    {
        // Lógica para gerar relatório de sinistros
        Console.WriteLine("Gerando relatório de sinistros...");
    }
}
```