# DIP - Dependency Inversion Principle

O DIP afirma que módulos de alto nível não devem depender de módulos de baixo nível, mas sim de abstrações. Abstrações não devem depender de detalhes; os detalhes é que devem depender das abstrações.

**Benefícios:**

- **Desacoplamento:** Facilita a troca de implementações sem impacta o código de alto nível.
- **Facilidade de teste:** Módulos podem ser testados isoladamente usando abstrações.

**Exemplo de Má-Prática:** Um módulo de alvo nível depende diretamente de uma classe concreta de baixo nível.

## DIP - Mau Exemplo

**Problema:** A classe `ServicoRelatorio` depende diretamente da implementação concreta `Repositorio`. Isso dificulta a troca da implementação do repositório e aumenta o acoplamento.

```
public class Repositorio
{
    public void Salvar(string dados)
    {
        Console.WriteLine("Dados salvos no banco de dados.");
    }
}

public class ServicoRelatorio
{
    private Repositorio _repositorio;

    public ServicoRelatorio()
    {
        _repositorio = new Repositorio();
    }

    public void Processar()
    {
```

```
        _repositorio.Salvar("Relatorio processado");
    }
}
```

## DIP - Bom Exemplo

**Solução:** A classe `ServicoRelatorio` depende da abstração `IRepositorio`, permitindo trocar a implementação concreta `RepositorioBanco` por qualquer outra que implemente `IRepositorio`.

```
public interface IRepositorio
{
    void Salvar(string dados);
}

public class RepositorioBanco : IRepositorio
{
    public void Salvar(string dados)
    {
        Console.WriteLine("Dados salvos no banco de dados.");
    }
}

public class ServicoRelatorio
{
    private readonly IRepositorio _repositorio;

    public ServicoRelatorio(IRepositorio repositorio)
    {
        _repositorio = repositorio;
    }

    public void Processar()
    {
        _repositorio.Salvar("Relatorio processado.");
    }
}
```