



A <PROGRAMAÇÃO> PRECISA DE TI

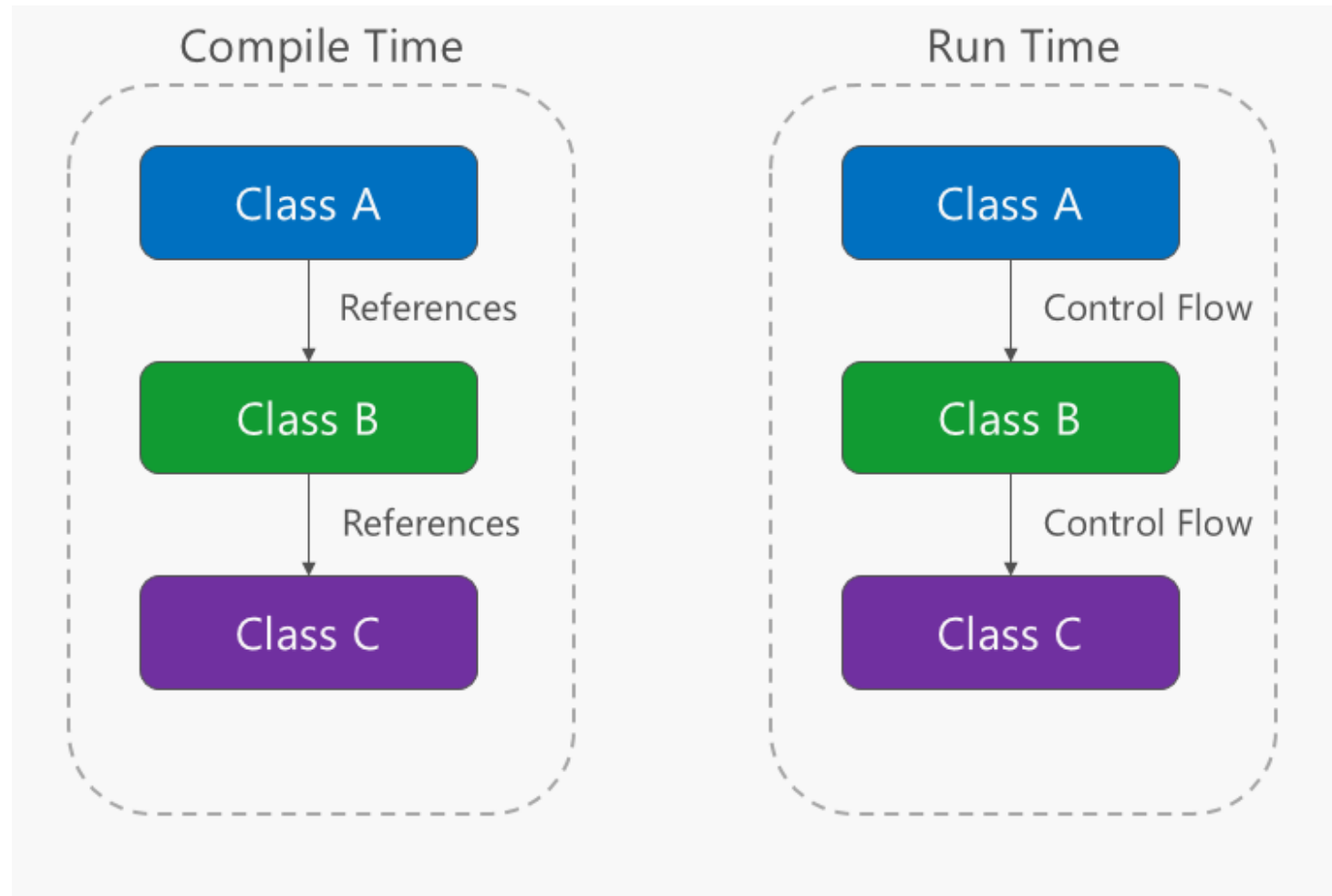
# Dependency Injection

Laboratório Web

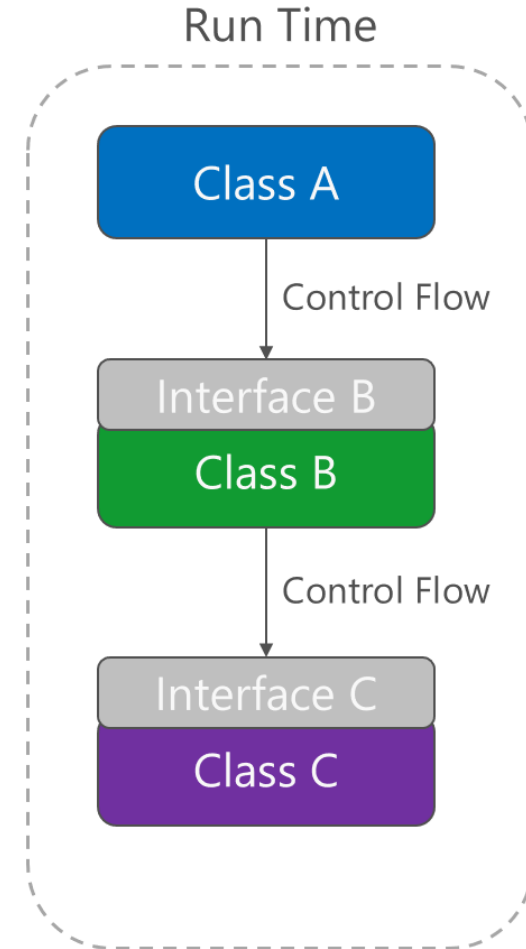
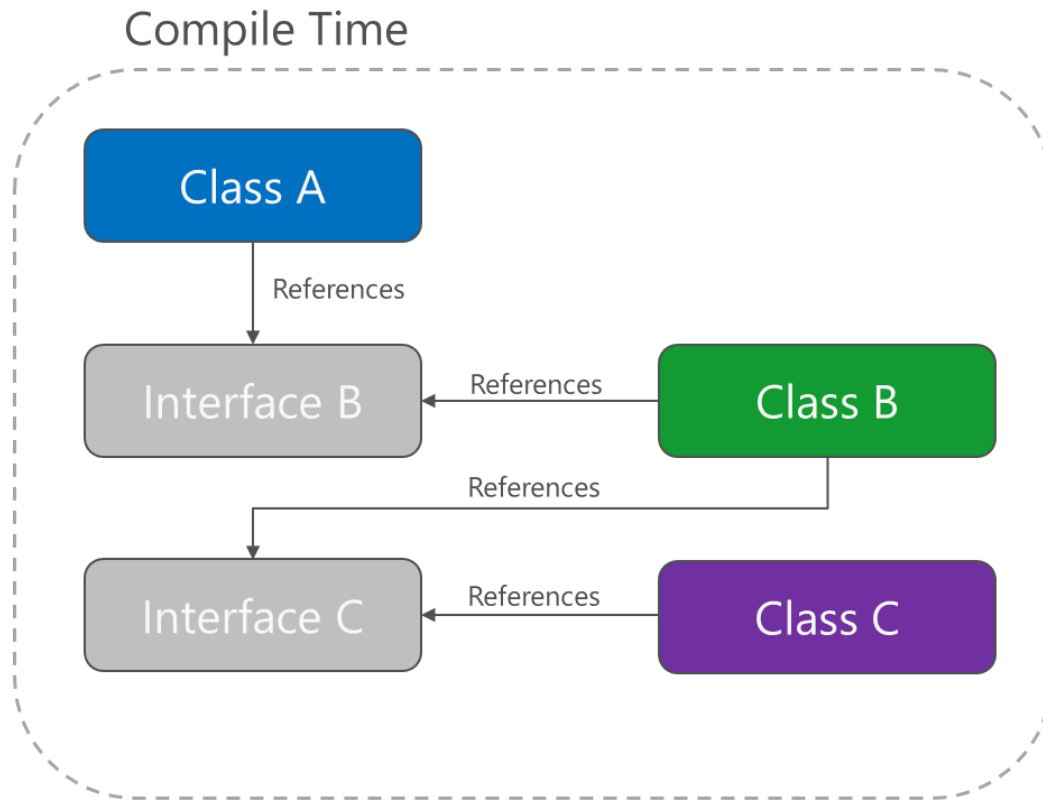
# Dependency Injection (DI)

- É um padrão em desenho de software utilizado para implementar inversão de controlo (IoC) entre as classes e as suas dependências
- Uma dependência é quando um objeto depende de outro para ser criado
- O objetivo deste padrão de desenho de software é reduzir as dependências a classes concretas através da utilização de interfaces
- A utilização de interfaces permite criar aplicações “fracamente” ligadas (loosely coupled)

# Fortemente ligadas (Tight coupling)



# Fracionamento ligadas (Loosely coupling)



```
public class Employee
{
    0 references
    public int UserId { get; set; }
    0 references
    public string JobTitle { get; set; }
    1 reference
    public string FirstName { get; set; }
    1 reference
    public string LastName { get; set; }
    0 references
    public string EmployeeCode { get; set; }
    0 references
    public string Region { get; set; }
    0 references
    public string PhoneNumber { get; set; }
    0 references
    public string EmailAddress { get; set; }
    0 references
    public string FullName()
    {
        return FirstName + " " + LastName;
    }
}
```

```

internal class Program
{
    private Employee employee;
    1 reference
    public Employee MyEmployee { get { return employee; } }
    1 reference
    public Program(Employee employee)
    {
        this.employee = employee;
    }
    0 references
    static void Main(string[] args)
    {
        Employee employee = new Employee()
        {
            FirstName = "Pedro",
            LastName = "Matos"
        };

        Program p = new Program(employee);
        Console.WriteLine(p.MyEmployee.FullName());
    }
}

```

Agora em vez da classe Employee  
queremos a classe Manager, o que  
temos que fazer?

---

# Agora em vez da classe Employee queremos a classe Manager, o que temos que fazer?

---

Alterar todas as dependências da classe Employee para a classe manager. Será que não existe uma forma melhor de fazer isto?



```

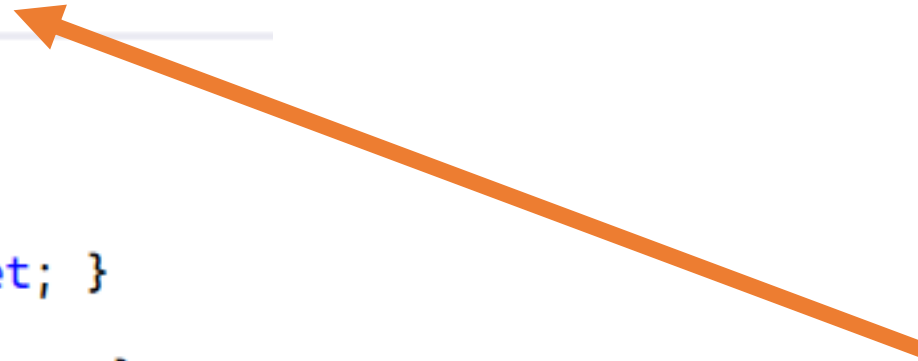
public class Employee : IEmployee
{
    0 references
    public int UserId { get; set; }
    0 references
    public string JobTitle { get; set; }
    2 references
    public string FirstName { get; set; }
    2 references
    public string LastName { get; set; }
    0 references
    public string EmployeeCode { get; set; }
    0 references
    public string Region { get; set; }
    0 references
    public string PhoneNumber { get; set; }
    0 references
    public string EmailAddress { get; set; }
    2 references
    public string FullName()
    {
        return FirstName + " " + LastName;
    }
}

```

```

internal interface IEmployee
{
    2 references
    string FullName();
}

```



```
internal class Manager : IEmployee
{
    2 references
    public string FirstName { get; internal set; }
    2 references
    public string LastName { get; internal set; }
    1 reference
    public int ManagerId { get; set; }

    2 references
    public string FullName()
    {
        return ManagerId + ": " + FirstName + " " + LastName;
    }
}
```

```

internal class Program
{
    private IEmployee employee;
    1 reference
    public IEmployee MyEmployee { get { return employee; } }
    1 reference
    public Program(IEmployee employee)
    {
        this.employee = employee;
    }
    0 references
    static void Main(string[] args)
    {
        Manager manager = new Manager()
        {
            FirstName = "Ana",
            LastName = "Silva"
        };

        Program p = new Program(manager);
        Console.WriteLine(p.MyEmployee.FullName());
    }
}

```

# Dependency Injection em Controllers MVC

- Os Controllers MVC recebem as suas dependências explicitamente através dos seus construtores
- A framework ASP.NET Core suporta o padrão de desenho DI
- DI torna as aplicações mais fáceis de testar e de realizar manutenção
- As dependências são adicionadas como serviços, que são definidos como interfaces

# Tempo de vida dos serviços (DI Service Lifetime)

- Existem 3 tipos de tempo de vida para os serviços
  - Singleton – apenas uma única instância da classe de serviço é criada, armazenada em memória e utilizada para todas as injeções (o objeto é o mesmo para todos os pedidos)
  - Scoped – é criada uma nova instância da classe de serviço por cada pedido efetuado (os objetos são os mesmos para o mesmo pedido, mas diferem para novos pedidos)
  - Transient – é criada uma instância da classe de serviço em todos os pedidos (os objetos são sempre diferentes em todos os pedidos)

# Tempo de vida dos serviços (DI Service Lifetime)

- `builder.Services.AddSingleton<IEmployees, Employees>();`
- `builder.Services.AddScoped<IEmployees, Employees>();`
- `builder.Services.AddTransient<IEmployees, Employees>();`

# Singleton

- Vamos injetar um serviço com tempo de vida Scoped ao nosso Controller
- A dependência é injetada através do construtor
- No Controller a dependência é sempre relativa à interface

```
var builder = WebApplication.CreateBuilder(args);  
  
// Add services to the container.  
  
builder.Services.AddControllers();  
// Learn more about configuring Swagger/OpenAPI at http:  
builder.Services.AddEndpointsApiExplorer();  
builder.Services.AddSwaggerGen();
```

```
builder.Services.AddSingleton<IEmployees, Employees>();
```

```
public class EmployeesController : ControllerBase  
{  
    private readonly IEmployees employees;  
  
    0 references  
    public EmployeesController(IEmployees employees)  
    {  
        this.employees = employees;  
    }  
}
```

# Como escolher um ServiceLifetime?

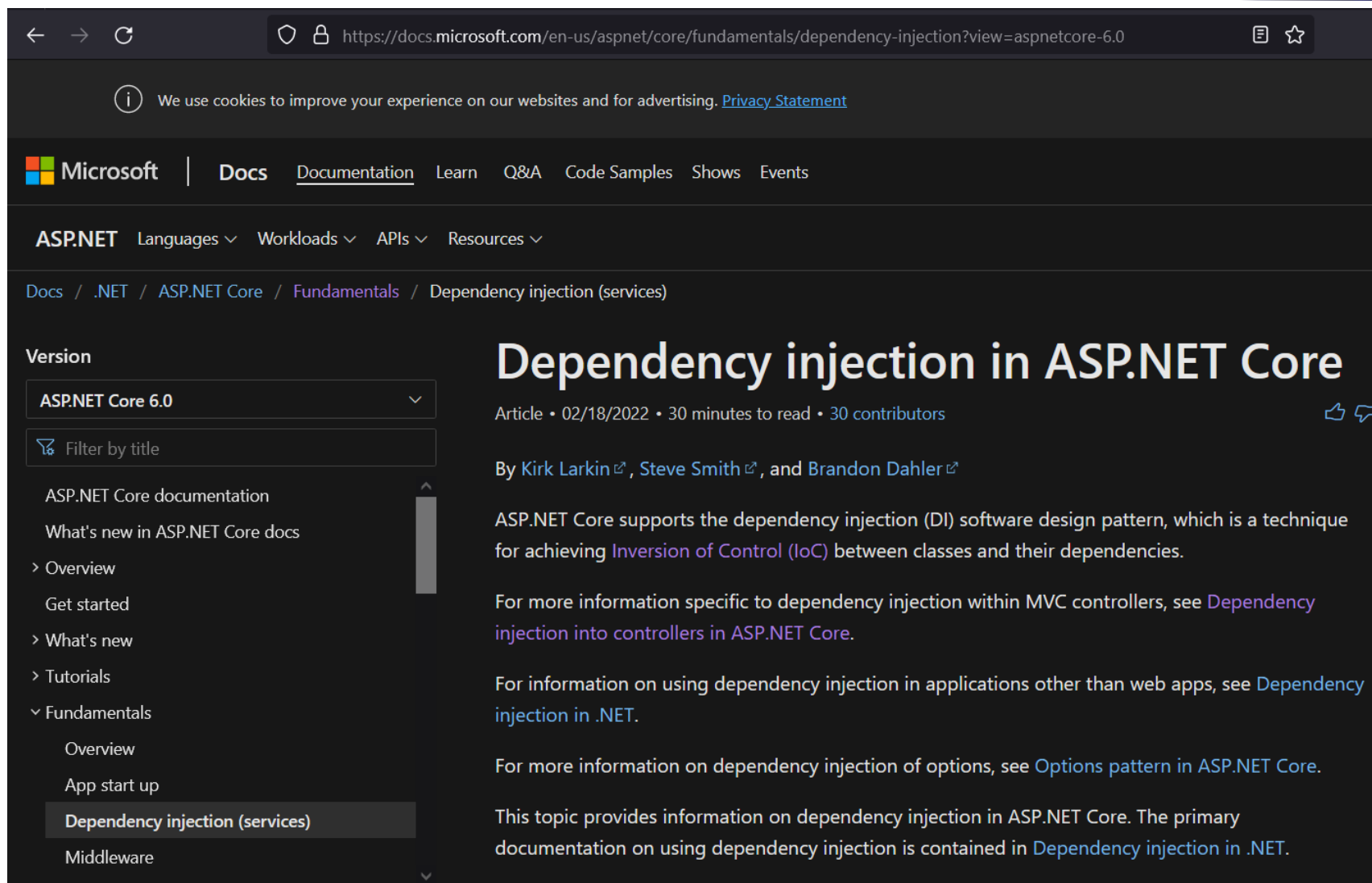
- Singleton – quando queremos partilhar um estado a nível da aplicação
- Scoped – quando queremos partilhar estado a nível de um pedido/acção/actividade
- Transient – quando não temos ou não queremos partilhar nenhum estado



# Que problemas são resolvidos pelo DI?

- Utilização de interfaces para abstrair a implementação da dependência
- Registo da dependência como um serviço utilizando um contentor de serviços (IServiceProvider)
- Injeção do serviço/dependência no construtor da classe onde é utilizado. A framework é responsável por inicializar a instância da dependência e destruir a mesma quando não é necessária

# Referências



The screenshot shows a web browser displaying the Microsoft Docs page for "Dependency injection in ASP.NET Core". The browser's address bar shows the URL: <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-6.0>. A cookie notice is visible at the top. The Microsoft logo and navigation links (Docs, Documentation, Learn, Q&A, Code Samples, Shows, Events) are in the header. Below the header, there are dropdown menus for "ASP.NET", "Languages", "Workloads", "APIs", and "Resources". The breadcrumb trail reads: Docs / .NET / ASP.NET Core / Fundamentals / Dependency injection (services). On the left, a sidebar shows the "Version" dropdown set to "ASP.NET Core 6.0" and a search filter "Filter by title". The sidebar also lists a table of contents with items like "Overview", "Get started", "What's new", "Tutorials", "Fundamentals" (expanded), "Overview", "App start up", "Dependency injection (services)" (selected), and "Middleware". The main content area has the title "Dependency injection in ASP.NET Core" in large font, followed by metadata: "Article • 02/18/2022 • 30 minutes to read • 30 contributors". The authors are listed as "By Kirk Larkin, Steve Smith, and Brandon Dahler". The text explains that ASP.NET Core supports the dependency injection (DI) software design pattern, which is a technique for achieving Inversion of Control (IoC) between classes and their dependencies. It provides links to more information specific to MVC controllers and to applications other than web apps. It also links to information on dependency injection of options. The final paragraph states that the primary documentation on using dependency injection is contained in "Dependency injection in .NET".

← → ↻ <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-6.0> 📄 ☆

📘 We use cookies to improve your experience on our websites and for advertising. [Privacy Statement](#)

**Microsoft** | Docs Documentation Learn Q&A Code Samples Shows Events

**ASP.NET** Languages ▾ Workloads ▾ APIs ▾ Resources ▾

Docs / .NET / ASP.NET Core / Fundamentals / Dependency injection (services)

**Version**

ASP.NET Core 6.0 ▾

🔍 Filter by title

- ASP.NET Core documentation
- What's new in ASP.NET Core docs
- > Overview
- Get started
- > What's new
- > Tutorials
- ▾ Fundamentals
  - Overview
  - App start up
  - Dependency injection (services)**
  - Middleware

## Dependency injection in ASP.NET Core

Article • 02/18/2022 • 30 minutes to read • 30 contributors

By [Kirk Larkin](#), [Steve Smith](#), and [Brandon Dahler](#)

ASP.NET Core supports the dependency injection (DI) software design pattern, which is a technique for achieving [Inversion of Control \(IoC\)](#) between classes and their dependencies.

For more information specific to dependency injection within MVC controllers, see [Dependency injection into controllers in ASP.NET Core](#).

For information on using dependency injection in applications other than web apps, see [Dependency injection in .NET](#).

For more information on dependency injection of options, see [Options pattern in ASP.NET Core](#).

This topic provides information on dependency injection in ASP.NET Core. The primary documentation on using dependency injection is contained in [Dependency injection in .NET](#).

# qualificar

TAL

X

</>

## A <PROGRAMAÇÃO