

# Singleton

## Definição:

O padrão Singleton garante que uma classe tenha apenas **uma instância** e fornece um ponto de acesso global a essa instância.


## Quando Usar?

Quando precisa garantir que só exista **uma instância** de uma classe em toda a aplicação;

Quando você quer controlar o acesso a um recurso compartilhado (ex: banco de dados, logger, configurações globais).

# Aplicação:

Sem Singleton:



```
1 class Logger {
2     log(message: string): void {
3         console.log(`[LOG]: ${message}`);
4     }
5 }
6
7 const logger1 = new Logger();
8 const logger2 = new Logger();
9
10 logger1.log("Primeira instância");
11 logger2.log("Segunda instância");
12
13 console.log(logger1 === logger2);
```

# Aplicação:

Com Singleton:

```
1 class SingletonLogger {
2     private static instance: SingletonLogger;
3
4     private constructor() {}
5
6     static getInstance(): SingletonLogger {
7         if (!SingletonLogger.instance) {
8             SingletonLogger.instance = new SingletonLogger();
9         }
10        return SingletonLogger.instance;
11    }
12
13    log(message: string): void {
14        console.log(`[LOG]: ${message}`);
15    }
16 }
17
18 // Testando
19 const loggerA = SingletonLogger.getInstance();
20 const loggerB = SingletonLogger.getInstance();
21
22 loggerA.log("Usando instância A");
23 loggerB.log("Usando instância B");
24
25 console.log(loggerA === loggerB);
```

# Pontos Fortes/Fracos:

## Pontos Fortes:

- Garante que apenas uma instância exista;
- Controla o acesso a recursos compartilhados;
- Fácil de implementar;

## Pontos Fortes:

- Viola o princípio da responsabilidade única (SRP);
- Dificulta testes unitários (difícil de mockar);
- Pode gerar acoplamento global se mal utilizado;

# Conclusão:

O padrão Singleton é bastante útil em casos onde apenas uma instância de uma classe deve existir, como loggers, configurações globais, gerenciadores de conexão, entre outros.

Apesar de ser simples e eficaz, deve ser usado com cautela para evitar acoplamento excessivo e dificuldade na testabilidade do sistema.

# Factory Method

## Definição:

O **Factory Method** é um padrão de projeto **criacional** que fornece uma interface para criar objetos, mas permite que as subclasses decidam **qual classe concreta será instanciada**. Ele ajuda a **desacoplar o código** que usa o objeto do código que cria o objeto.

## Quando Usar?

Quando você quer evitar acoplamento direto com classes concretas;

Quando seu sistema precisa ser facilmente **extensível**, adicionando novos tipos sem mudar o código existente;

Quando a criação dos objetos envolve lógica complexa ou variações de um mesmo tipo.

# Aplicação:

Sem Factory Method:

```
1 class EmailNotificacao { enviar(msg: string) { console.log([Email] ${msg}); } }
2
3 class SMSNotificacao { enviar(msg: string) { console.log([SMS] ${msg}); } }
4
5 // Código cliente decide qual classe usar const tipo = "sms"; let notificacao;
6
7 if (tipo === "email") { notificacao = new EmailNotificacao(); } else if (tipo === "sms") { notificacao = new SMSNotificacao(); }
8
9 notificacao.enviar("Você tem uma nova mensagem.");
```

# Aplicação:

Com Factory Method:



```
1 interface Notificacao { enviar(msg: string): void; }
2
3 class EmailNotificacao implements Notificacao { enviar(msg: string) { console.log([Email] ${msg}); } }
4
5 class SMSNotificacao implements Notificacao { enviar(msg: string) { console.log([SMS] ${msg}); } }
6
7 abstract class NotificacaoFactory { abstract criar(): Notificacao; }
8
9 class EmailFactory extends NotificacaoFactory { criar(): Notificacao { return new EmailNotificacao(); } }
10
11 class SMSFactory extends NotificacaoFactory { criar(): Notificacao { return new SMSNotificacao(); } }
12
```



# Pontos Fortes/Fracos:

## Pontos Fortes:

- Reduz o acoplamento entre o código que usa e o que cria os objetos;
- Torna o sistema mais extensível e de fácil manutenção;
- Segue os princípios SOLID (especialmente o OCP - Aberto para extensão, fechado para modificação).

## Pontos Fortes:












- Adiciona mais classes e estrutura ao sistema;
- Pode parecer complexo demais para sistemas pequenos ou simples.

# Conclusão:

O Factory Method é uma ótima solução quando precisamos criar objetos de maneira controlada e flexível, especialmente em sistemas que precisam crescer e se adaptar com o tempo.

Embora pareça mais "verbooso" que o uso direto de `new`, ele oferece muito mais poder de organização, manutenção e testabilidade, e evita que o cliente precise conhecer todos os tipos concretos do sistema.

# Comparação:

Aspecto	Singleton	Factory Method
 <b>Objetivo</b>	Garantir que uma classe tenha <b>uma única instância</b>	Delegar a criação de objetos para subclasses
 <b>Categoria</b>	Criacional	Criacional
 <b>Controle de instância</b>	Total: apenas uma instância é criada	Nenhum: múltiplas instâncias podem ser criadas conforme a necessidade
 <b>Flexibilidade</b>	Baixa: restringe a criação de múltiplas instâncias	Alta: permite escolher dinamicamente qual classe instanciar
 <b>Desacoplamento</b>	Baixo: código depende diretamente da classe Singleton	Alto: o cliente depende de uma interface, não da implementação
 <b>Testabilidade</b>	Difícil de testar (instância global é difícil de mockar)	Fácil de testar (objetos podem ser mockados via interface)
 <b>Exemplo típico</b>	Logger, Configuração global, Conexão de BD	Criador de notificações, documentos, interfaces diferentes
 <b>Instância única</b>	Sim	Não
 <b>Complexidade estrutural</b>	Simples e direta	Envolve hierarquia de classes (fábricas e produtos)
 <b>Prós</b>	Controle total da instância, simples de usar	Altamente extensível e desacoplado
 <b>Contras</b>	Acoplamento global, difícil de testar, quebra SRP	Estrutura mais complexa, pode ser "overkill" para casos simples

# Conclusão:

O **Singleton** é ideal quando se quer **garantir uma única instância** de algo no sistema e esse algo precisa estar acessível globalmente. É simples e direto, mas pode trazer problemas de acoplamento e dificultar testes.

Já o **Factory Method** brilha quando queremos **desacoplar a criação de objetos do uso desses objetos**, tornando o sistema mais extensível e aderente aos princípios do SOLID, especialmente o OCP (Aberto para extensão, fechado para modificação).

Use Singleton quando quiser controlar o número de instâncias.

Use Factory Method quando quiser flexibilidade na criação de objetos sem depender das classes concretas.