TÓPICO 12 - INTERFACES FLUENTES

Clean Code - Professor Ramon Venson - SATC 2026.1

Surgimento

Em dezembro de 2005, Martin Fowler publicou um artigo sobre uma conversa com Eric Evans sobre um tipo de interface que eles decidiram chamar de **Fluente**.

Classes Tradicionais

Tomando como exemplo uma biblioteca para manipular horas, temos a seguinte sintaxe:

```
TimePoint cincoHoras, seisHoras;
...
TimeInterval reuniao = new TimeInterval(cincoHoras, seisHoras);
```

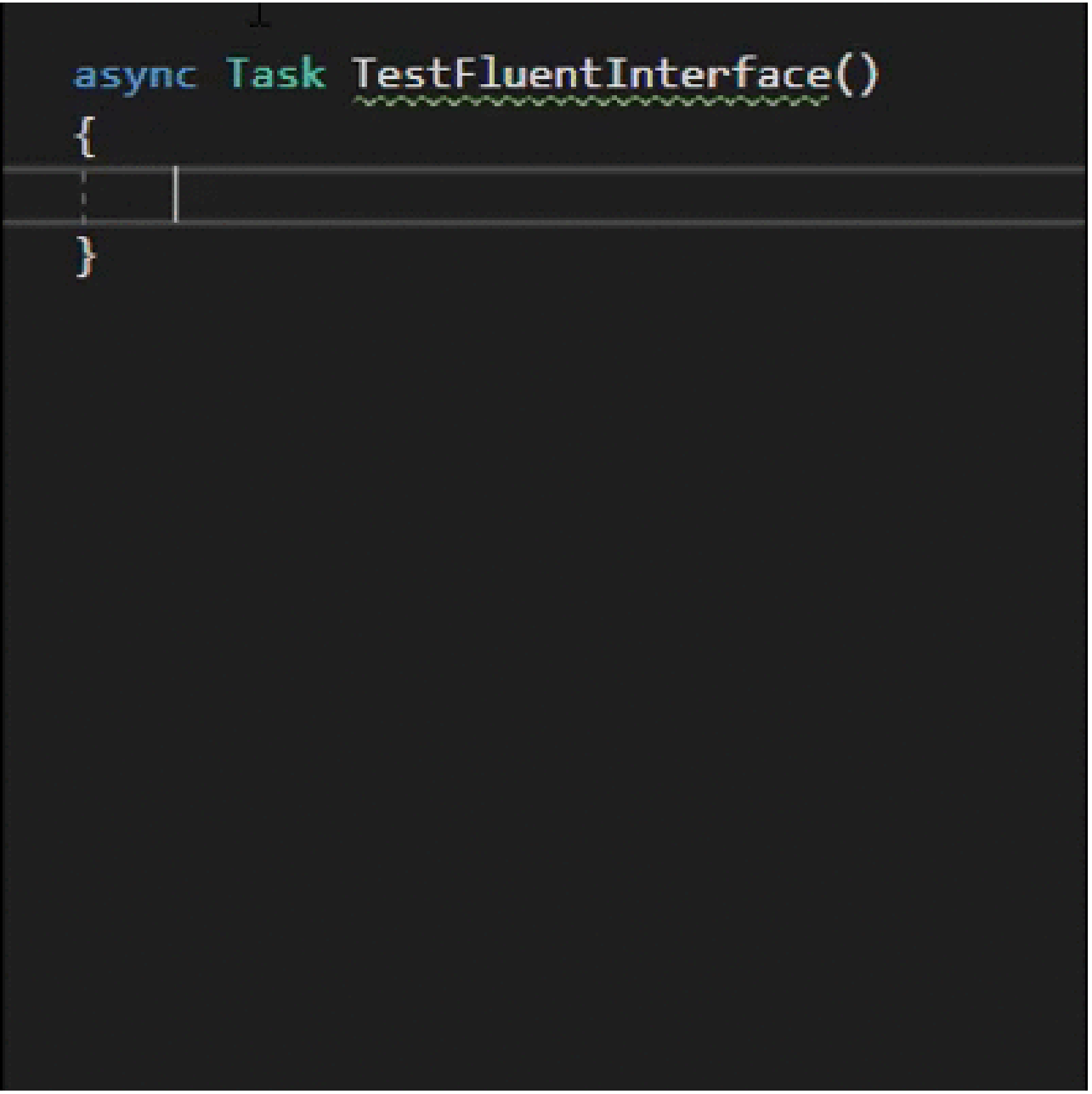
Classes Fluentes

Traduzindo essa sintaxe para uma interface fluente, temos:

```
TimePoint cincoHoras, seisHoras;
...
TimeInterval reuniao = TimeInterval.de(cincoHoras).ate(seisHoras);
```

ou

```
TimePoint cincoHoras, seisHoras;
...
TimeInterval reuniao = cincoHoras.ate(seisHoras);
```



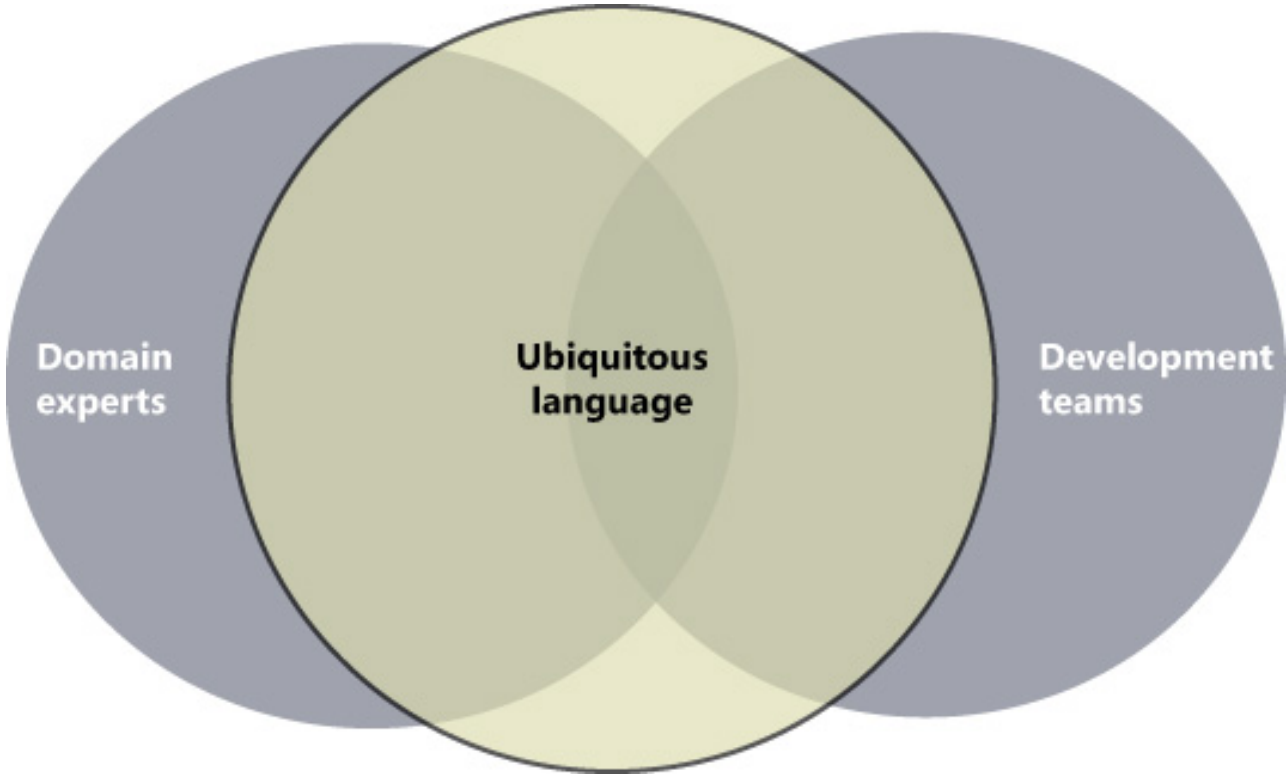
Objetivo da Interface Fluente

O objetivo de uma interface fluente é tornar o código mais legível e fácil de entender.

Isso é possível porque a interface fluente permite que você escreva código que se parece mais com uma linguagem natural, mas seu foco é especialmente facilitar a integração com a linguagem de domínio.

Linguagem de Domínio

Uma linguagem de domínio tem como característica ser uma linguagem escrita para problemas específicos, ao invés de ser uma linguagem genérica que é definida para qualquer tipo de software.



Por exemplo, em um sistema de gerenciamento transações bancárias, criar um objeto `Transferencia` pode ser feito tradicionalmente de duas formas diferentes. A primeira é usando o construtor da classe:

```
Transferencia transfer = new Transferencia(100.0, "798565-8", "255685-9", "USD", "2025-05-10", true);
```

E a segunda é usando os métodos da classe:

```
Transferencia transfer = new Transferencia();
transfer.setValor(100.0);
transfer.setContaOrigem("798565-8");
transfer.setContaDestino("255685-9");
transfer.setMoeda("USD");
transfer.setDataAgendamento("2025-05-10 08:00:00");
transfer.setEstaAgendada(true);
```

No entanto, ambas as formas descrevem de maneira genérica o que é uma transferência bancária, não sendo específicas para o domínio do sistema. Uma forma de transformar isso em uma interface fluente seria:

```
Transferencia transfer = Transferencia.create()
    .comValor(100.0)
    .de("798565-8")
    .para("255685-9")
    .naMoeda("USD")
    .em("2025-05-10")
    .comoAgendada();
```

Como uma Interface Fluente pode ser construída

Uma interface fluente não depende de uma receita única. O ponto central é permitir encadeamento com nomes alinhados ao domínio.

Uma forma comum de construir uma API fluente é:

- Crie um construtor padrão privado.
- Crie um método estático `create()` que retorna uma instância da classe.
- Crie métodos que retornam a instância da classe atual.
- Se houver montagem incremental, crie um método `build()` para validar e finalizar o objeto.

Interface Fluente != Builder

Toda builder API costuma ser fluente, mas nem toda interface fluente é um builder.

- **Interface fluente**: enfatiza legibilidade e linguagem de domínio no uso da API.
- **Builder**: enfatiza construção gradual de um objeto, muitas vezes com validação ao final.

Uma API pode ser fluente em consultas, configurações ou regras de domínio sem necessariamente seguir o padrão Builder completo.

Exemplo

Crie a classe com o construtor privado e os métodos privados:

```
public class Transferencia {
    private double valor;
    private String contaOrigem;
    private String contaDestino;
    private String moeda;
    private String dataAgendamento;
    private boolean estaAgendada;

    private Transferencia() {
        // Construtor privado
    }
    ...
}
```

Adicione um método estático `create()` que retorna uma instância da classe.

```
public class Transferencia {
    ...
    public static Transferencia create() {
        return new Transferencia();
    }
    ...
}
```

Crie métodos que retornam a instância da classe atual. A nomenclatura desses métodos deve seguir a linguagem de domínio, ao invés de seguir a convenção de nomenclatura tradicional (*Getters* e *Setters*).

```
public class Transferencia {
    ...
    public Transferencia comValor(double valor) {
        this.valor = valor;
        return this;
    }

    public Transferencia de(String contaOrigem) {
        this.contaOrigem = contaOrigem;
        return this;
    }
    ...
}
```

Vantagens e Desvantagens

Uma API criada usando essa abordagem é criada para que seja simples, fácil de usar e entender.

Algumas convenções de nomenclatura e de padrões de design são flexibilizadas, como métodos *setters* que retornam a instância atual, ao invés de `void`.

O preço da fluência é que ela pode ser difícil de manter.

Material de Apoio

- Martin Fowler - Fluent Interfaces
- Martin Fowler - Domain Specific Language
- DevMedia - Interfaces Fluentes
- Baeldung - Differences between Fluent Interface and Builder Pattern