Abstract Factory em TypeScript

Mario, Rafael, Robson e Wagner

TS

Definição

Abstract Factory é um padrão de projeto do tipo criacional que permite a criação objetos relacionados sem a necessidade de especificar suas classes concretas.

```
interface AbstractFactory {

    createProductA(): AbstractProductA;

    createProductB(): AbstractProductB;
}
```

TS

```
class ConcreteFactory1 implements AbstractFactory {
   public createProductA(): AbstractProductA {
       return new ConcreteProductA1();
   }

   public createProductB(): AbstractProductB {
       return new ConcreteProductB1();
   }
}
```

TS

```typescript
class ConcreteFactory2 implements
AbstractFactory {
    public createProductA(): AbstractProductA {
        return new ConcreteProductA2();
    }

    public createProductB(): AbstractProductB {
        return new ConcreteProductB2();
    }
}
```

```
interface AbstractProductA {
    usefulFunctionA(): string;
}

interface AbstractProductB {

    usefulFunctionB(): string;

    anotherUsefulFunctionB(collaborator: AbstractProductA): string;
}
```

TS

```typescript
class ConcreteProductB1 implements AbstractProductB {

  public usefulFunctionB(): string {
      return 'The result of the product B1.';
  }

  public anotherUsefulFunctionB(collaborator: AbstractProductA): string {
      const result = collaborator.usefulFunctionA();
      return `The result of the B1 collaborating with the (${result})`;
  }
}
```

TS

```typescript
class ConcreteProductB1 implements AbstractProductB {

  public usefulFunctionB(): string {
      return 'The result of the product B1.';
  }

  public anotherUsefulFunctionB(collaborator: AbstractProductA): string {
      const result = collaborator.usefulFunctionA();
      return `The result of the B1 collaborating with the (${result})`;
  }
}
```

```typescript
class ConcreteProductB2 implements AbstractProductB {

    public usefulFunctionB(): string {
        return 'The result of the product B2.';
    }

    public anotherUsefulFunctionB(collaborator: AbstractProductA): string {
        const result = collaborator.usefulFunctionA();
        return `The result of the B2 collaborating with the (${result})`;
    }
}
```

TS

```typescript
function clientCode(factory: AbstractFactory) {
    const productA = factory.createProductA();
    const productB = factory.createProductB();

    console.log(productB.usefulFunctionB());
    console.log(productB.anotherUsefulFunctionB(productA));
}
```

```typescript
// Teste client com primeiro factory
clientCode(new ConcreteFactory1());

// Teste com mesmo cliente mas com diferente factory
clientCode(new ConcreteFactory2());
```

TS

```
//Teste client com primeiro factory
~$: The result of the product B1.
~$: The result of the B1 collaborating with the (The result of the
product A1.)

//Teste com mesmo cliente mas com diferente factory
~$: The result of the product B2.
~$: The result of the B2 collaborating with the (The result of the
product A2.)
```

TS

Fim