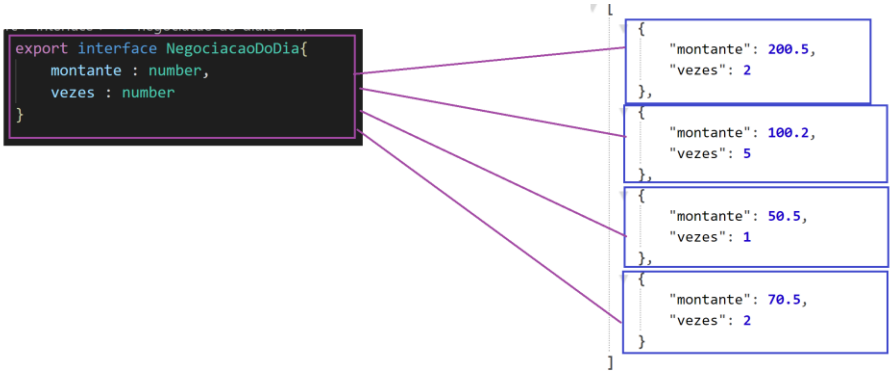
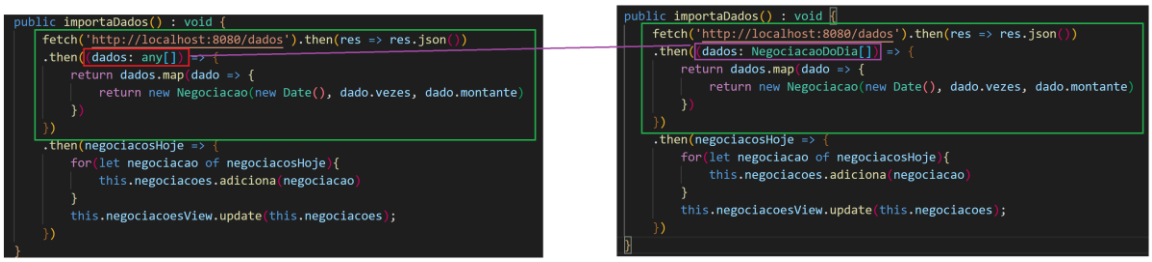
TypeScript – Interface

Se eu quiser **consumir dados de um backend** com o **TypeScript** eu posso usar um recurso excelente que me permite ter **mais segurança nessa operação**, que é a **Interface.**

Uma **Interface** não pode ser instanciada e no TypeScript ela é como se fosse um shape *(um corpo)* para um determinado "objeto", então eu posso usar uma **interface** para representar o **JSON que estou consumindo do backend** por exemplo:



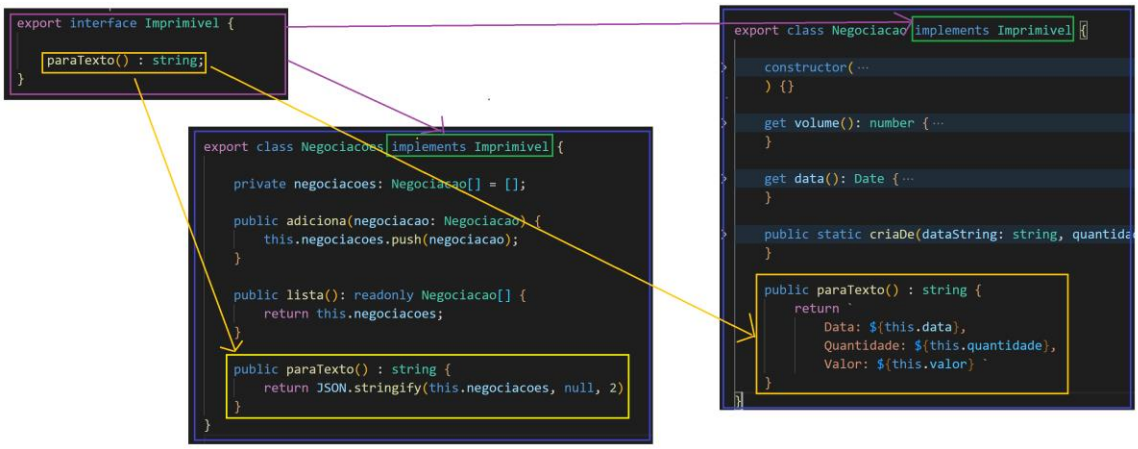
Isso é bom pra aplicação pois eu posso **passar a consumir os dados baseado na Interface**, o que me garante muito **mais segurança e manutenibilidade do que usar um tipo any**:



TypeScript – Métodos em Interface

Interfaces no Java podem ser lidas como *"contratos"* onde eu posso dizer que **a classe que implementa uma interface** *(assina um contrato)* **tem a obrigação de implementar seus métodos.** As Interfaces no **TypeScript** também podem ser utilizadas seguindo esse conceito, se eu quiser que minhas **classes** se tornem **"algo"** através de um contrato, eu posso definir uma **Interface** e fazer com que as **classes** as assinem.

Digamos que eu queria **tornar algumas classes "imprimiveis"**, eu crio uma **Interface** *(contrato)* com a **assinatura de um método "paraTexto"**, **quem** quiser se tornar um **Imprimível** deve **assinar esse contrato e implementar esse método:**



Um grande **benefício que essa abordagem traz é permitir a flexibilidade das minhas classes**, através de **contratos** eu as torno **"algo"**, esse **"algo"** pode **ser utilizado como parâmetro de um método** de maneira que ele **mantenha consistência.**

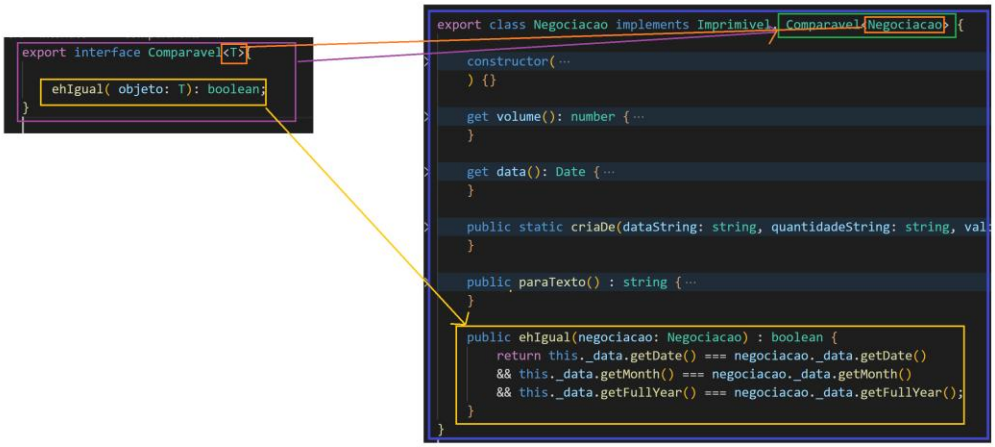
Por exemplo, se eu tiver **uma classe que só faça a impressão de classes**, eu posso garantir que **ela** vai conseguir chamar esse **método de "paraTexto"** deixando explicito que **ela só funciona com "Imprimivel"** (ou seja, a interface como referência para polimorfismo):

```
import { Imprimivel } from "./imprimivel.js";

export function imprimir(... objetos: Imprimivel[]){

    for(let objeto of objetos){
        console.log(objeto.paraTexto())
    }

}
```

TypeScript – Interfaces também podem ter Genericos

Uma **Interface** também pode ser genérica, por exemplo, se **eu quero que todos os meus objetos sejam comparáveis** faz sentido que eu tenha um **contrato(Interface)** que diga como deve ser feita a comparação.

Para que um **objeto seja comparável** ele deve **assinar esse contrato** e já dizer **com o que ele vai ser comparado**, essa flexibilidade é possível graças ao tipo genérico:



Uma classe pode implementar mais de uma interface, e interfaces podem herdar de Interfaces.