9. Tipos Avançados e Utilitários em TypeScript

TypeScript oferece uma variedade de tipos avançados e utilitários que ajudam a lidar com situações mais complexas de tipagem, permitindo uma maior flexibilidade e reusabilidade do código.

Tipos Union e Intersection

1. **Union Types**
    - **Union types permitem que uma variável aceite mais de um tipo de dado.**

```
type NumeroOuString = number | string;
function exibirId(id: NumeroOuString) {
    console.log(`ID: ${id}`);
}
exibirId(101);  // Válido
exibirId("202");  // Válido
```

2. **Intersection Types**
    - **Intersection types combinam múltiplos tipos em um só, juntando suas propriedades.**

```
interface Negocio {
    nome: string;
    fundacao: Date;
}

interface Empregados {
    quantidade: number;
}

type Empresa = Negocio & Empregados;
let minhaEmpresa: Empresa = { nome: "TechCorp", fundacao: new Date(), quantidade: 10
0 };
```

Tipos Condicionais

- **Conditional Types**

- **Tipos condicionais permitem a criação de tipos que dependem de condições.**

```
type PequenoOuGrande<T> = T extends "pequeno" ? number : string;
let tamanhoPequeno: PequenoOuGrande<"pequeno"> = 10;  // number
let tamanhoGrande: PequenoOuGrande<"grande"> = "muito grande";  // string
```

Utilitários Comuns

TypeScript fornece várias funções utilitárias que ajudam a manipular tipos de maneiras úteis.

1. **Partial**
   - **Torna todas as propriedades de um tipo opcional.**

```
interface Carro {
    marca: string;
    modelo: string;
    ano: number;
}

function configurarCarro(config: Partial<Carro>) {
    return config;
}
configurarCarro({ modelo: "Fusca" });  // Outras propriedades são opcionais
```

2. **Readonly**
   - **Torna todas as propriedades de um tipo somente leitura.**

```
type CarroReadonly = Readonly<Carro>;
let meuCarro: CarroReadonly = { marca: "VW", modelo: "Gol", ano: 2020 };
// meuCarro.modelo = "Fusca";  // Erro: não pode modificar uma propriedade readonly
```

3. **Record**
   - **Cria um objeto tipo com um conjunto de propriedades do mesmo tipo.**

```
let cidades: Record<string, string> = {
    Tokyo: "Japão",
    Paris: "França",
    NovaYork: "EUA"
};
```

Exercícios para Praticar

1. **Trabalhando com Union e Intersection:**
   - Crie uma função que aceite um parâmetro que pode ser um `string` ou um `number`, e outro parâmetro que é uma interseção de dois tipos com diferentes propriedades.
2. **Explorando Conditional Types:**
   - Defina um tipo condicional que mude com base em um valor booleano passado, alternando entre dois tipos definidos.
3. **Aplicando Partial e Readonly:**
   - Use `Partial` para criar uma função que aceite configurações parciais de um objeto. Use `Readonly` para garantir que nenhum objeto passado para uma função específica possa ser modificado.
4. **Utilizando Record:**
   - Implemente uma função que use `Record` para mapear nomes de países para suas capitais e permita busca por qualquer país fornecido.
5. **Generics com Utilitários:**
   - Crie uma função generic que aceite um tipo e retorne um array de `Partial` desse tipo, permitindo a criação de listas com configurações parciais.