Модуль: Typescript. Классы

**Урок:** this и типизация контекста

## КАК НЕ ПОТЕРЯТЬ КОНТЕКСТ

При усложнении функционала всегда можно потерять контекст в классах. В TS существует возможность четко сказать, чем должен быть контекст и не получать такие ситуации:

```
class Player {
    #login: string;

constructor(login: string) {
    this.#login = login;
}

logIn() {
    return `Player ${this.#login} online!`;
}

const player = new Player("Test");

const test = player.logIn;

test(); // Error: Cannot read properties of undefined (reading '#login')
```

Проблему можно решить при помощи **bind** или **стрелочной функции**. Но до запуска кода вы не узнаете об ошибке. Поэтому в TS есть вариант сразу сказать, **чем будет контекст** прямо первым аргументом в функции. И вы увидите ошибку на этапе разработки:

```
class Player {
    #login: string;

constructor(login: string) {
    this.#login = login;
}

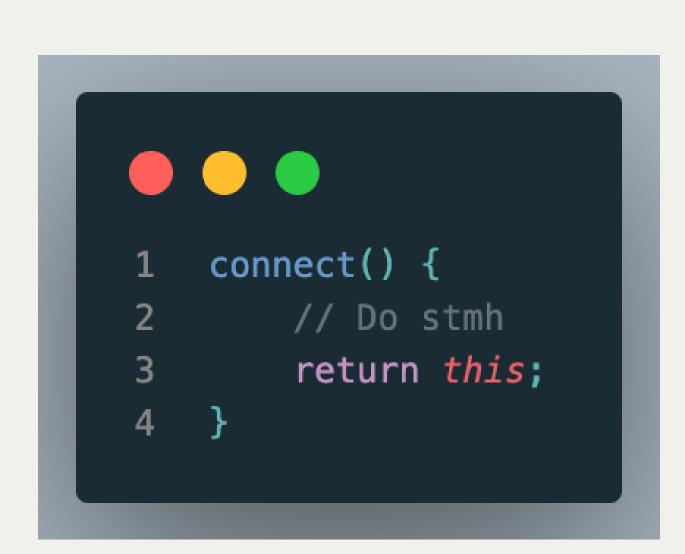
logIn(this: Player) { // this type
    return `Player ${this.#login} online!`;
}

const player = new Player("Test");
const test = player.logIn;
test(); // Error: The 'this' context of type 'void' is not assignable to method's 'this' of type 'Player'
```

Иногда мы из метода возвращаем контекст, то есть ссылку на экземпляр объекта. В таких случаях **не стоит жестко типизировать** возвращаемое значение, так как оно может сломать логику:

```
1 connect(): Player {
2   // Do stmh
3   return this;
4 }
```

**Неправильно**: даже в наследуемых классах будет возвращать Player, что может привести к ошибке



Правильно: всегда будет возвращаться **текущий** экземпляр класса, даже в потомках

При помощи контекста мы можем проверить, к какому классу относится экземпляр и написать свой **защитник типа**:

```
isPro(): this is CompetitivePlayer {
    return this instanceof CompetitivePlayer;
}

const somePlayer: Player | CompetitivePlayer = new CompetitivePlayer("Test");
somePlayer.isPro() ? console.log(somePlayer) : console.log(somePlayer);
```