ОБОБЩЕННЫЕ КЛАССЫ

Обобщенные классы действуют по тому же принципу: абстрактные идентификаторы при создании экземпляра будут заменены чем угодно:

```
class User<T, S> {
    name: T;
    age: S;

    constructor(name: T, age: S) {
        this.name = name;
        this.age = age;
    }
}

const ivan = new User("Ivan", 30);
console.log(ivan);
```

Альтернативный синтаксис позволяет прописать типы вручную при создании экземпляра. Это полезно, когда данные приходят из **других источников**, а не прописаны вручную:

```
const ivan = new User<string, number>("Ivan", 30);  // Не очень полезно
const nameData = 'Alex';
const ageData = 50;
const alex = new User<string, number>(nameData, ageData); // Теперь неправильные данные не пройдут
```

Методы классов могут быть дженериками. Причем одинаковые названия идентификаторо **не пересекаются:**

```
class User<T, S> {
    name: T;
    age: S;

    constructor(name: T, age: S) {
        this.name = name;
        this.age = age;
    }

    sayMyFullName<T>(surname: T): string {
        if (typeof surname !== "string") {
            return `I have only name: ${this.name}`;
        } else {
            return `${this.name} ${surname}`;
        }
    }
}
```

При наследовании классов мы должны указывать то, **какими типами данных были идентификаторы у родительского класса.** Только так мы можем передавать их наследнику. Сам же наследник тоже может быть дженериком:

```
class AdminUser<T> extends User<string, number> {
    rules: T;
}
```