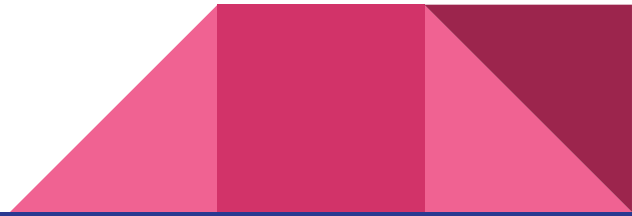
Abstrakt siniflər

Abstrakt siniflər

JavaScript-də özbaşına yaradıla bilməyən, yalnız başqa siniflər tərəfindən genişlənə bilən siniflərdir. Abstrakt sinifın özündəki metodlar konkret əməliyyatlar yerinə qoyula bilməz, bu metodlar onu genişlədən siniflər tərəfindən təyin edilməlidir.

JavaScript-də abstrakt sinifları yaratmaq üçün dilin özündə bir xüsusiyyət yoxdur, ancaq bu funksionalı mimari tətbiq etmək mümkündür.



Misal

```
// Abstract class
class Animal {
  constructor(name) {
      if (new.target === Animal) {
          throw new Error("Bu sinifdan obyekt yaratmaq
olmaz"); // Bu sınıfdan obyekt yaratmaq mümkün deyil
      }
      this.name = name;
  }
```

Misal

```
  // Abstract method
  sound() {
      throw new Error("Bu metodun təyinatını genişlən sinifdə yazmalısınız"); //
Törədilmiş sınıflarda bu metodu mü əyyənləşdirmək lazımdır
  }
}

// "Dog" sinifi "Animal" sinifini genişl ədir
class Dog extends Animal {
  sound() {
      return "Hav hav"; // Köpəyin səsi
  }
}
```

Misal

```
const rex = new Dog("Rex");
console.log(rex.sound());  // "Hav hav" çıxar

// Təkrar "Animal" obyektini yaratmağa çalışdığınızda xəta alacaqsınız
// const creature = new Animal("Creature");  // Xəta: Bu sinifdan obyekt yaratmaq olmaz
```

JavaScript-də get və set metodları

JavaScript-də get və set metodları obyektin xüsusi xassələrinə müraciət etmək üçün və onları dəyişmək üçün istifadə olunur. Bu metodlar obyektin daxilində xassələrlə işləmək üçün daha tənzimlənmiş bir yol təqdim edir. get metodunu xassənin dəyərini almaq üçün, set metodunu isə xassəyə dəyər təyin etmək üçün istifadə edirik.

Aşağıdakı kodda bir Person obyektinin sadə bir nümunəsini göstərəcəyəm ki, bu obyektin içərisində name xassəsi üçün get və set metodları mövcuddur.

Misal

```
class Person {
        constructor(firstName, lastName) {
          // İki daxili dəyişən təyin edirik
          this._firstName = firstName;
          this._lastName = lastName;
        }

        // 'name' xassəsi üçün 'get' metodu
        get name() {
          // Tam adı qaytarır
          return `${this._firstName} ${this._lastName}`;
        }
```

Misal

```
// 'name' xassəsi üçün 'set' metodu
        set name(fullName) {
            // Daxil olunan tam adı iki hissəyə bölmək üçün
            let parts = fullName.split(' ');
            this._firstName = parts[0];
            this._lastName = parts[1];
        }
    }
```

Misal

```
// Nümunə yaratmaq
      let person = new Person("Nurlan", "Aliyev");

      console.log(person.name); // Nurlan Aliyev'i qaytaracaqdır

      // 'name' xassəsinə dəyər təyin etmək
      person.name = "Elnur Hüseynov";

      console.log(person.name); // Elnur Hüseynov'u qaytaracaqdır
```

Bu kodda

1. Person sinfini yaradırıq.
2. Bu sınıfın konstruktoru iki daxili dəyişənə sahibdir: _firstName və _lastName.
3. name xassəsi üçün get və set metodları təyin edirik ki, bu metodlar daxili dəyişənlərlə işləyə bilsin.
4. get metodu tam adı qaytarır.
5. set metodu isə daxil edilən tam adı iki hissəyə bölbilir.
6. Nəticədə, bu kod sayəsində, name xassəsinə müraciət edərək və ona dəyər təyin edərək, daxili dəyişənlərlə asanlıqla işləmək mümkündür.

JavaScript-də prototype

JavaScript-də prototype məfhumu, obyektlər arasında miras almağa imkan verir. Əsas fikir belədir ki, bir obyekt digər obyektin xüsusiyyətlərini və metodlarını miras ala bilər. Bu, obyekt oriyentasiyaşdırılmış proqramlaşdırma (OOP) da belə bir əsas konseptdir.

JavaScript dildə bütün funksiyaların bir prototype xassəsi var. Bu prototype, o funksiyanın konstruktoru olaraq istifadə olunduğunda yeni obyektlər tərəfindən miras alınan xüsusiyyətlər və metodlar toplusudur.

Aşağıdakı nümunədə Animal və Dog funksiyalarını və onların prototype-larını necə istifadə etməli olduğunu göstərirəm:

Misal

```
function Animal(name,type) {
        this.name = name;
        this.type = type;
}

Animal.prototype.makeSound = function(){
        console.log('hav-hav')
}

let dog = new Animal('rex','it')
dog.makeSound()// 'hav-hav'
console.log(dog)// {name:'rex',type:'it'}
```