

20.11.2025 Ders Notları

Meta Android Developer Professional Certificate - OOP & JavaScript Notları

Tarih: 20.11.2025

Kapsanan Konular: Programlama Paradigmaları, OOP Prensipleri, JavaScript'te Class'lar, Inheritance, Prototype

Programlama Paradigmaları

Programlama paradigmaları kod yazma stilini belirleyen sınıflandırmalar. **Functional Programming**'i biliyorsun zaten, bir diğer popüler olan da **Object Oriented Programming (OOP)**.

OOP'de programları *object*'ler kullanarak düzenliyoruz. Functional'da data ile fonksiyonlar ayrı duruyor, OOP'de ise object'in içinde hepsi bir arada.

Functional vs OOP Örneği

Functional Approach:

```
var shoes = 100;
var stateTax = 1.2;

function totalPrice(shoes, tax) {
  return shoes * tax;
}

var toPay = totalPrice(shoes, stateTax);
console.log(toPay); // 120
```

OOP Approach:

```
var purchase1 = {
  shoes: 100,
  stateTax: 1.2,
  totalPrice: function() {
    var calculation = purchase1.shoes * purchase1.stateTax;
    console.log('Total price:', calculation);
  }
};
```

```
purchase1.totalPrice(); // 120
```

OOP'nin güzel yanı: birden fazla object yaratabilirsin:

```
var purchase2 = {  
  shoes: 50,  
  stateTax: 1.2,  
  totalPrice: function() {  
    var calculation = purchase2.shoes * purchase2.stateTax;  
    console.log('Total price:', calculation);  
  }  
};
```

this Keyword'ü

Yukarıdaki kodda `purchase1.shoes` ve `purchase2.shoes` yazmak yerine `this` kullanabiliriz. `this` "bu object" demek:

```
var purchase1 = {  
  shoes: 100,  
  stateTax: 1.2,  
  totalPrice: function() {  
    var calculation = this.shoes * this.stateTax;  
    console.log('Total price:', calculation);  
  }  
};
```

Artık aynı method'u tüm object'lerde kullanabiliriz. Ama yine de her object'e aynı method'u yazmak verimli değil. İşte burada **class'lar** devreye giriyor.

OOP Prensipleri

OOP'nin 4 temel prensibi var:

1. Inheritance (Kalıtım)

Base class'tan sub-class'lar türetmek. JavaScript'te `extends` keyword'ü ile yapılıyor.

```
class Animal { }  
class Mammal extends Animal { }  
class Elephant extends Mammal { }
```

2. Encapsulation (Kapsülleme)

Implementation'ı gizlemek. Kullanıcının nasıl çalıştığını bilmesine gerek yok:

```
"abc".toUpperCase(); // Nasıl çalıştığını bilmeme gerek yok
```

3. Abstraction (Soyutlama)

Kavramı somut implementation'dan ayırmak. Daha genel bir şekilde kod yazmak.

4. Polymorphism (Çok Biçimlilik)

Aynı method'un farklı object'lerde farklı davranması:

```
const bicycle = {
  bell: function() {
    return "Ring, ring! Watch out, please!";
  }
}

const door = {
  bell: function() {
    return "Ring, ring! Come here, please!";
  }
}

function ringTheBell(thing) {
  console.log(thing.bell());
}

ringTheBell(bicycle); // "Ring, ring! Watch out, please!"
ringTheBell(door);    // "Ring, ring! Come here, please!"
```

JavaScript'te Class'lar

Class'lar object'ler için blueprint (şablon) görevi görür.

Basit Class Örneği

```
class Train {
  constructor(color, lightsOn) {
    this.color = color;
    this.lightsOn = lightsOn;
  }

  toggleLights() {
    this.lightsOn = !this.lightsOn;
  }
}
```

```
lightsStatus() {  
    console.log('Lights on?', this.lightsOn);  
}  
}  
  
var myFirstTrain = new Train('red', false);  
console.log(myFirstTrain); // Train {color: 'red', lightsOn: false}  
  
myFirstTrain.toggleLights();  
myFirstTrain.lightsStatus(); // Lights on? true
```

Inheritance ile Class'lar

```
class HighSpeedTrain extends Train {  
    constructor(passengers, highSpeedOn, color, lightsOn) {  
        super(color, lightsOn); // Base class'ın constructor'ını çağır  
        this.passengers = passengers;  
        this.highSpeedOn = highSpeedOn;  
    }  
  
    toggleHighSpeed() {  
        this.highSpeedOn = !this.highSpeedOn;  
        console.log('High speed status:', this.highSpeedOn);  
    }  
  
    // Method overriding  
    toggleLights() {  
        super.toggleLights(); // Base class'ın methodunu çağır  
        super.lightsStatus();  
        console.log('Lights are 100% operational.');    }  
}  
  
var highSpeed1 = new HighSpeedTrain(200, false, 'green', false);  
highSpeed1.toggleLights();  
// Lights on? true  
// Lights are 100% operational.
```

Prototype Inheritance

JavaScript'te inheritance **prototype**'lar üzerinden çalışır. Class syntax'ı aslında prototype'ların daha okunaklı hali.

Object.create() ile Prototype

```
var bird = {
  hasWings: true,
  canFly: true,
  hasFeathers: true
};

var eagle1 = Object.create(bird);
console.log(eagle1); // {} - boş görünür ama...
console.log(eagle1.hasWings); // true - prototype'dan geliyor

var penguin1 = Object.create(bird);
penguin1.canFly = false; // Override ettik
console.log(penguin1.canFly); // false (kendi property'si)
console.log(eagle1.canFly); // true (hala prototype'dan)
```

JavaScript önce object'in kendi property'lerine bakar, bulamazsa prototype'ına bakar.

Class Instance'larını Property Olarak Kullanma

```
class StationaryBike {
  constructor(position, gears) {
    this.position = position;
    this.gears = gears;
  }
}

class Treadmill {
  constructor(position, modes) {
    this.position = position;
    this.modes = modes;
  }
}

class Gym {
  constructor(openHrs, stationaryBikePos, treadmillPos) {
    this.openHrs = openHrs;
    this.stationaryBike = new StationaryBike(stationaryBikePos, 8);
    this.treadmill = new Treadmill(treadmillPos, 5);
  }
}

var boxingGym = new Gym("7-22", "right corner", "left corner");
```

```
console.log(boxingGym.openHrs); // "7-22"  
console.log(boxingGym.stationaryBike); // StationaryBike object
```

Diğer Built-in Object'ler

JavaScript'te bir sürü built-in constructor var:

```
new Date();           // Tarih ve zaman  
new Error();          // Hata object'i  
new Map();            // Key-value collection  
new Set();            // Unique value collection  
new Promise();        // Asenkron işlemler için  
new WeakSet();        // Weak reference'lı Set  
new WeakMap();        // Weak reference'lı Map
```

Bunların hepsi aslında JavaScript'in built-in class'ları ve hepsi prototype inheritance kullanıyor.