Nesne Yaşam Döngüsü

Constructor (Yapılandırıcı Metod)

Constructor' lar bir nesnenin oluşturulması ile çalışan metodlardır. Nesnenin oluşturulması new ile o nesneden bir tane yaratılması anlamına gelir. Aşağıda ornek bir constructor tanımı bulunmaktadır.

```
class Sınıf_Adı {

	Sınıf_Adı() {
		//Constructor içeriği
	}

}
```

Bunu bir örneğe dökecek olursak aşağıdaki sınıfa ait bir constructor bulunmaktadır.

```
class Ornek {

	Ornek() {
		System.out.println("Constructor çalıştı");
	}

}
```

Bu örnekte dikkatimizi çeken bir şey olmalı. Birincisi constructor metodunun geriye ne tipte bir değer döndürüceği bilinmiyor ? Bunun haricinde isim olarakta sınıf ismi ile aynı. Evet bu bir zorunluluktur bir metodun constructor olabilmesi için sınıf ismi ile aynı olması gerekmektedir ayrıca geriye döndürülecek bir tip yoktur ve void' te değildirler.

Constructor özelliklerini maddeler şeklinde sayacak olursak.

- Sınıf ismi ile aynı olmak zorundadırlar.
- Geriye döndürülecek dönüş tipleri yoktur ve void değildirler.
- İstenen sayıda parametre alabilirler.
- Birden fazla olabilirler yani metod overloading constructor' lar içinde geçerlidir.
- Eğer constructor yazılamışsa sanal makina bir tane yaratacaktır.

Peki constructor' devreye girmesi ne zaman olur ? Daha öncede bahsettiğimiz gibi new ile nesnenin yaratılması gerekmektedir. Aşağıdaki iki adet nesne yaratıyoruz.

```
class Ornek {

	Ornek() {
		System.out.println("Constructor içeriği");
	}
```

```
	public static void main(String[] args) {
		Ornek o1 = new Ornek();
		Ornek o2 = new Ornek();
	}
}
```

Bu durumda kodun çıktısı aşağıdaki gibi olacaktır.

```
Constructor içeriği
Constructor içeriği
```

Bunun anlamı constructor iki kere çalışmış demektir. Peki burada constructor' ı çağıran yer neresidir ? Yukarıdaki kod içerisinde yer alan `new Ornek();` satırları nesnenin yaratılması ile birlikte () işaretleri bir metod çağırımını ifade eder. İşte bu constructor' dır.

Peki bunun yanında biz constructor ile parametre görmek istersek ne yapmalıyız ? Aşağıdaki örnekte bir öğrenci sınıfı yaratıyorum ve constructor ile öğrenci bilgilerini alıp yazdırıyorum.

```
class Ogrenci {

	Ogrenci(String ad, String soyad) {
		System.out.println(ad + " " + soyad);
	}

	public static void main(String[] args) {
		Ogrenci o1 = new Ogrenci("Melih", "Sakarya");
		Ogrenci o2 = new Ogrenci("Ahmet", "Dursun");

	}
}
```

Dikkat : Yukarıdaki örnekte bir adet constructor bulunmaktadır ve iki adet String değeri istemektedir. Bu durumda nesne yaratırken bu değerleri göndermek zorundayım. Örnek olarak Ogrenci nesnesi yaratırken `Ogrenci()` diyemezdik.

Birden fazla constructor kullanımı

Eğer bir sınıf içerisinde birden fazla constructor kullanılacaksa yani overloading varsa ne olur ? Bu durumda aynı standart metodlardaki gibi hangi tipte parametre gönderilirse o constructor çağırılır.

Örnek vermek gerekirse aşağıdaki kod içerisinde üç adet constructor bulunmakta ve main içerisinde nesneler yaratılırken ilgili constructor' lara erişilmektedir.

```
public class Ogrenci {
	Ogrenci() {
		System.out.println("1. constructor");
	}
```

```
	Ogrenci(String ad) {
		System.out.println("2. constructor");
	}

	Ogrenci(String ad, String soyad) {
		System.out.println("3. constructor");
	}

	public static void main(String[] args) {
		Ogrenci o1 = new Ogrenci();
		Ogrenci o2 = new Ogrenci("Melih");
		Ogrenci o3 = new Ogrenci("Melih", "Sakarya");
	}
}
```

Buna göre yukarıdaki kodu çalıştırırsak çıktı aşağıdaki gibi olacaktır.

```
1. constructor
2. constructor
3. constructor
```

this kelimesi

this kelimesinin birinci işi sınıf içerisinde nesnenin referansına ulaşmaya yarar. Bunun anlamı siz sınıf içerisinde herhangi bir metodtayken o metodun bağlı olduğu nesneye erişmeyi sağlar.

Dikkat : this kelimesi static metodlarda kullanılamaz.

Peki aynı nesneyi ifade ettiklerini nasıl anlayabiliriz. Örnek verecek olursak aşağıdaki sınıfta nesneden bir tane oluşturup hash değerini yazdırıyoruz. Yine bir alt satırda o nesne üzerinden çağırılan metod içerisinde this' in hash değerini alıyoruz.

```
class Ornek {

	void bilgiYazdır() {
		System.out.println(this.hashCode());
	}

	public static void main(String[] args) {
		Ornek orn = new Ornek();
		System.out.println(orn.hashCode());
		orn.bilgiYazdır();
	}
}
```

Bu durumda yukarıdaki kodun çıktısı aşağıdaki gibi iki satır içinde aynı olacaktır. Yani this kelimesi ile orn aynı şeyi ifade etmiş oldu.

```
4072869
```

```
4072869
```

Not : hash değeri tekildir ve her nesne için tekil oluşturulur.

this ile nesneye ait attribute ve metodlara da erişebiliriz. Aşağıdaki örneği inceleyecek olursak

```
public class Ogrenci {

	String ad = "Ahmet";

	Ogrenci(String ad) {
		System.out.println(ad);
		System.out.println(this.ad);
	}

	public static void main(String[] args) {
		Ogrenci ogr = new Ogrenci("Melih");
	}
}
```

Buna göre sadece ad değerini yazdırırsak "Melih" yazacaktır ancak this.ad diyecek olursak "Ahmet" yazdıracaktır. Bunun sebebi metodlar içerisinde yer alan yerel değişkenler ile sınıflar içerisinde yer alan nesne değişkenlerinin ayrı alanlarda bulunmasıdır.

Not : Bir metod içerisinde nesne değişkeni ile aynı isimde yerel değişken var ise öncelik yerel değişkene verilir. Bu durumda nesne değişkenine yukarıdaki gibi this kelimesi ile ulaşılır.

this kelimesi için sık kullanımlardan birisi aşağıdaki örnekteki gibi nesneye ait ilk değerlerin constructor ile birlikte gönderilmesidir.

```
public class Ogrenci {

	String ad;
	String soyad;
	int yas;

	public Ogrenci(String ad, String soyad, int yas) {
		this.ad = ad;
		this.soyad = soyad;
		this.yas = yas;
	}

	public static void main(String[] args) {
		Ogrenci ogr = new Ogrenci("Melih", "Sakarya", 30);
	}
}
```

this kelimesinin bir başka görevi ise aynı sınıf içerisindeki constructor' lara ulaşmaktır. Aşağıdaki örnekte `new Ogrenci()` ile parametresiz constructor çağırılmaktadır. Ardından bu constructor' ın ilk satırında bulunan this kelimesi ikinci constructor' ı çağırmaktadır.

```
public class Ogrenci {

	Ogrenci() {
		this("Melih");
		System.out.println("1. constructor bitti...");
	}

	Ogrenci(String ad) {
		System.out.println(ad);
		System.out.println("2. constructor bitti...");
	}

	public static void main(String[] args) {
		Ogrenci ogr = new Ogrenci();
	}
}
```

Buna göre bu kodun çıktısı aşağıdaki gibi olacaktır.

```
Melih
2. constructor bitti...
1. constructor bitti...
```

Dikkat : this ile başka bir constructor ancak bir constructor metodu içerisinden çağırılabilir ve bu ilk satırda olmalıdır. Başka bir metod üzerinden bu şekilde ulaşım sağlanamaz.

Garbage Collector (Çöp toplayıcısı) mekanizması

Java ile bellek üzerinde yarattığınız nesneler herhangi bir referansa sahip değillerse bir süre sonra garbage collector tarafından bellekten silineceklerdir. Eğer C++ gibi bir dil kullansaydık bu işi manuel olarak bizim yapmamız gerekecekti.

Garbage Collector mekanizmasının amacı JVM üzerinde gereksiz bellek kullanımı engellemektir. Örnek verecek olursak aşağıda bir döngü içerisinde 10.000 adet nesne oluşturuyoruz. Ancak bu nesneler for alanı sonunda referanslarını kaybediyorlar. Bu durumda bir süre sonra Garbage Collector devreye girip bu nesneleri çöp olarak görecek ve bellekten temizleyeceklerdir.

```
class Ornek {

	int deger = 1234;

	public static void main(String[] args) {
		for (int i = 0; i < 10000; i++) {
			Ornek ogr = new Ornek();
		}
	}
}
```

Dikkat : Garbage Collecter mekanizması belirli aralıklarda ve bellek ihtiyacı olduğu sürece çalışacaktır. Bir başka yöntem olarak `System.gc()` metodu ile bizde bu mekanizmayı devreye soksakta bu çok tavsiye edilen bir yöntem değildir.

finalize metodu

Nesnenin yaşamını constructor ile görebiliyorduk. Peki nesne yaşamının sona ermesini yani Garbage Collector mekanizması ile bellekten temizlenmesini nasıl görebiliriz ? Bu durumda Object sınıfından gelen finalize metodunu kullanabiliriz.

Not : finalize metodu sınıf içerisinde override edilerek kullanılır şu anda bu mekanizmayı bilmesekte ilerleyen bölümlerde görüyor olacağız.

Aşağıdaki örnekte yine 10.000 adet nesne yaratıp yaşamını constructor ile görüyoruz. Daha sonrasında nesne yaşamı kaybolduğunda ki biz bunu System.gc metodu ile manuel olarak yapıyoruz finalize metodu çalışarak nesnenin sonlandığını gösterecektir.

```
class Ornek {

	public Ornek() {
		System.out.println("Nesne yaşamı başladı");
	}

	@Override
	protected void finalize() throws Throwable {
		System.out.println("Nesne yaşamı sonlandı");
	}

	public static void main(String[] args) {
		for (int i = 0; i < 10000; i++) {
			Ornek ogr = new Ornek();
		}
		System.gc();
	}
}
```

Dikkat : finalize metodunun çalışması garanti değildir. Eğer Garbage Collector mekanizması devreye girmeden JVM işlemi sonlandırılırsa finalize metoduna uğranmadan nesne bellekten yok olacaktır. Bu yüzden finalize içerisine iş süreçlerinin eklenmesi tavsiye edilmez.