Sınıfların Başlangıç Metotları (Constructor)

Bir sınıf nesnesi yaratılırken new operatörü tarafından otomatik olarak çağrılan sınıfın static olmayan metotlarına başlangıç metotları (constructors) denir.

Başlangıç metotları sınıf ismi ile aynı isimli metotlardır. Başlangıç metotlarının geri dönüş değeri kavramı yoktur. Bildirim sırasında geri dönüş değeri kısmına hiçbir şey yazılmaz. Eğer yazılırsa (void anahtar sözcüğü de dahil) error oluşur. Örneğin:

```
package csd;

class App {
	public static void main(String [] args)
	{
		Sample s = new Sample();
	}
}

class Sample {
	public Sample()
	{
		System.out.println("Sample.Sample()");
	}
}
```

Sınıfın başlangıç metotları overload edilebilir. Yani aynı isimde farklı parametrik yapıda birden fazla başlangıç metodu yazılabilir. Sınıfın parametresiz başlangıç metoduna "varsayılan başlangıç metodu (default constructor)" denilmektedir.

new operatörü ile nesne yaratılırken new operatörüne verilen argüman listesi ile hangi metodun çağrılacağı belirlenir. Örneğin:

```
s1 = new Sample(); // Default başlangıç metodu
s2 = new Sample(10); // int parametreli başlangıç metodu
s3 = new Sample(10, "ankara"); // int, string parametreli başlangıç metodu
```

En uygun başlangıç metodu yine "overload resolution" kurallarıyla belirlenmektedir. Tüm başlangıç metotları aday metotlardır. Bunlar arasından uygunlar ve daha sonra en uygun metot belirlenmeye çalışılır. Örneğin:

```
package csd;

class App {
	public static void main(String [] args)
	{
		Sample s1 = new Sample();
		Sample s2 = new Sample(19);
	}
}


class Sample {
	public Sample()
	{
		System.out.println("Sample.Sample()");
	}

	public Sample(int val)
	{
		System.out.println("Sample.Sample(int)");
	}
}
```

Başlangıç metotlarında return kullanılabilir fakat yanına bir ifade yazılamaz.

new operatörü önce heap'te tahsisatı yapar. Sonra o alanı sıfırlar sonra da başlangıç metodunu çağırır. Yani başlangıç metodu içerisinde sınıfın bir veri elemanına değer atanmamışsa o eleman sıfır değerini almış olur. Eğer değer atamışsak o değer gözükecektir. Örneğin:

```
package csd;

class App {
	public static void main(String [] args)
	{
		Sample s = new Sample();

		System.out.printf("s.a=%d%n", s.a);
		System.out.printf("s.b=%b%n", s.b);
	}
}

class Sample {
	public int a;
	public boolean b;

	public Sample()
	{
		System.out.println("Sample.Sample()");
	}

	public Sample(int val)
	{
		System.out.println("Sample.Sample(int)");
	}
}
```

Başlangıç metotlarının içerisinde kullanılan sınıfın static olmayan veri elemanları o anda yaratılmış olan nesnenin static olmayan veri elemanlarıdır. Yani yaratılmış olan nesnenin veri elemanlarına en erken erişim başlangıç metotlarında yapılabilir. Örneğin:

```
package csd;

class App {
	public static void main(String [] args)
	{
		Sample s = new Sample(10, 20);

		//...
	}
}

class Sample {
	public int a, b;

	public Sample(int x, int y)
	{
		a = x;
		b = y;
	}
}
```

s = new Sample(10, 20) ifadesinde new operatörü ile önce heap'te tahsisat yapılacak, sonra sınıfın başlangıç metodu çağrılacak, sonra nesnenin adresi elde edilip s referansına atanacaktır.

Bir sınıf için programcı hiçbir başlangıç metodu yazmamışsa derleyici tarafından varsayılan başlangıç metodu içi boş olarak yazılır. Örneğin, aşağıdaki sınıf için new Sample() biçiminde nesne yaratmak sorun oluşturmaz:

```
package csd;

class App {
	public static void main(String [] args)
	{
		Sample s = new Sample();

		//...
	}
}

class Sample {
	public int a, b;

	//...
}
```

Başlangıç Metotlarına Neden Gereksinim Duyulmaktadır?

Sınıfların başlangıç metotları temel olarak iki amaçla kullanılmaktadır:

1. Nesne yaratıldığında veri elemanlarına birtakım ilk değerleri vermek

2. Nesne yaratıldığında birtakım ilk işleri arka planda yapmak

Şüphesiz başlangıç metotları yerine bu işler başka bir metoda da yaptırılabilir. Ancak başlangıç metotları bu işlemleri arka planda kodda yer kaplamayacak biçimde yapmaktadır. Böylece kodun daha sade gözükmesi sağlanmaktadır. Örneğin:

```
package csd;

class App {
	public static void main(String [] args)
	{
		Point p1 = new Point(50, 50);

		p1.display();

		Point p2 = new Point();

		p2.display();
	}
}

class Point {
	public int X, Y;

	public Point() {}
	public Point(int x, int y)
	{
		X = x;
		Y = y;
	}

	public static double distance(int x1, int y1, int x2, int y2)
	{
		return Math.sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
	}

	public static double distance(Point p1, Point p2)
	{
```

```
		return distance(p1.X, p1.Y, p2.X, p2.Y);
	}
	public double distance(int x, int y)
	{
		return distance(X, Y, x, y);
	}
	public double distance(Point p)
	{
		return distance(X, Y, p.X, p.Y);
	}

	public void display()
	{
		System.out.printf("(%d,%d)%n", X, Y);
	}
}
```