String Sınıfı ve Karakter İşlemleri

Java içerisinde mimari ile gelen String, Date gibi bazı özel sınıflar vardır. Bu sınıfların ozellikleri ve davranışlar tam anlaşılamayabiliyor. Bu bölümde bu sınıfları inceliyor olacağız.

String sınıfı

Java' da ilkel tipler içerisinde kelime gibi birden fazla karakter tanımlayabilecegimiz bir tip yoktur. char tipi aşağıdaki gibi sadece tek bir karakter tanımlamayı sağlamaktadır.

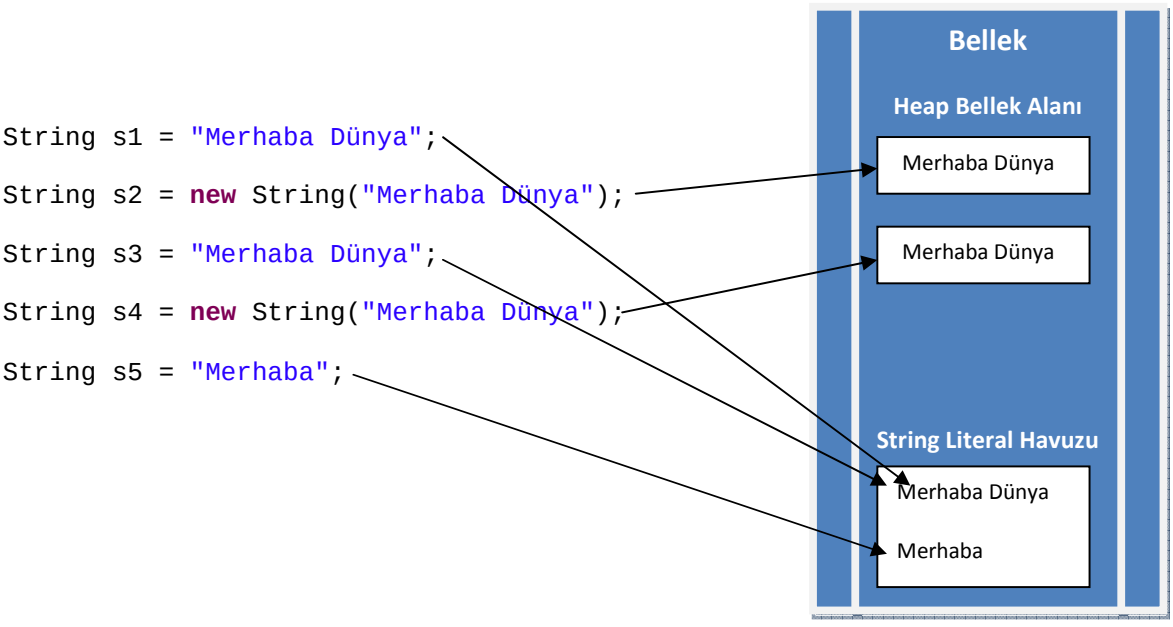
```
char c = 'A';
```

Zaten adının büyük harf ile başlamasından da anlayabileceğimiz gibi String bir sınıftır ve java.lang paketi içerisinde yer almaktadır.

String sınıfı ile aşağıdaki gibi iki şekilde nesne oluşturabiliriz.

```
String s1 = "Merhaba Dünya";
String s2 = new String("Merhaba Dünya");
```

Bu nesne oluşturma şekilleri bellek davranışı olarakta birbirlerinden farklıdır. Direkt atama nesneyi literal havuzuna atar. Bu sayede aynı tipte birden fazla nesne oluşmasına gerek kalmaz. s2 de yer alan ikinci atama tipiyse nesneyi direkt olarak heap alanda yeni bir referansı garanti etmek üzere oluşturur.



Bu grafikte dikkat etmemiz gereken nokta s1 ve s3 literal havuzunda tek bir alanı referans ederken s2 ve s4 ün garantili olarak heap üzerinde yeni referans alanlarını göstermesidir. s5 te yine literal havuzunda farklı bir alanda tutulmaktadır.

Bunun nedeni String tipinde bir atama yapılırken önce literal havuzunun kontrol edilmesi aynı içerikte nesne varsa o alanın referans edilmesidir.

Bu durumda aşağıdaki kodun çıktısı sizce ne olur ?

```
class Ornek {
	public static void main(String[] args) {
		String s1 = "Merhaba Dünya";
		String s2 = "Merhaba Dünya";
		String s3 = new String("Merhaba Dünya");

		System.out.println("s1 == s2 : " + (s1 == s2));
		System.out.println("s1 == s3 : " + (s1 == s3));
	}
}
```

s1 ve s2 literal havuzda aynı alanları gösterdiğinden karşılaştırma true döner. Ancak s3 heap üzerinde yeni bir referans alınını gösterdiği için s1 veya s2 ile karşılaştırması false dönecektir. Yeni çıktımız aşağıdaki gibidir.

```
s1 == s2 : true
s1 == s3 : false
```

Dikkat : String tipindeki nesnelerde bu tarz == ile karşılaştırmalar yerine equals ve compareTo metodlarının kullanılması tavsiye edilmektedir.

String ler bellekte tutulurken byte array olarak tutulurlar. Zaten kaynak koda bakacak olursa char[] value diye bir tanım görüyor olacağız. Yani Merhaba Dünya adında bir String aslında bellek üzerinde aşağıdaki gibi tutuluyor olacaktır.

Karakter	M	e	r	h	a	b	a		D	ü	n	y	a
Dizi İndeksi	0	1	2	3	4	5	6	7	8	9	10	11	12

Not : String içerisinde yer alan char tipindeki karakter indeksi final olarak tanımlanmıştır. Bu yüzden değiştirilemezdir. Buna göre aşağıdaki örnekte literal havuzunda 3 ayrı String bulunmaktadır.

```
String s1 = "Merhaba";
String s2 = "Dünya";
String s3 = s1 + s2;
```

String Metodları

equals ve equalsIgnoreCase

Karşılaştırma için kullanılır ve == gibi referans alanları yerine içeriklerin karşılaştırılmasını sağlar. Aşağıdaki örneği inceleyecek olursak ilk satır false dönerken ikinci satır true dönüyor olacaktır.

```
class Ornek {
	public static void main(String[] args) {
		String s1 = "Merhaba Dünya";
		String s2 = new String("Merhaba Dünya");

		System.out.println(s1 == s2); // false
		System.out.println(s1.equals(s2)); // true
	}
}
```

equalsIgnoreCase ise içerik karşılaştırmasını büyük küçük harf duyarlı olmadan yapar. Aşağıdaki örnekte ilk satır false ikinci satır true dönecektir.

```
class Ornek {
	public static void main(String[] args) {
		String s1 = "Merhaba Dünya";
		String s2 = "MERHABA DÜNYA";

		System.out.println(s1.equals(s2)); // false
		System.out.println(s1.equalsIgnoreCase(s2)); // true
	}
}
```

compareTo ve compareToIgroneCase

Yine karşılaştırma için kullanılır ancak bu sefer true false dönmesi yerine sayısal olarak büyük ve küçüklük döner. Bu yöntem özellikle sıralama algoritmaları için kullanılmaktadır. Aşağıdaki örnekte s1 ve s2 karşılaştırması 12 dönmektedir bunun anlamı s1 nin s2 den daha büyük bir içeriğe sahip olduğudur. Eğer içerikler eşit olsaydı 0 değeri dönüyor olacaktı.

```
class Ornek {
	public static void main(String[] args) {
		String s1 = "Melih";
		String s2 = "Ahmet";
		System.out.println(s1.compareTo(s2)); // Sonuç 12 döner
	}
}
```

compareToIgroneCase ise adından da anlaşılabileceği gibi büyük küçük harfe bakmadan içerik karşılaştırması yapmaktadır.

charAt

İstenen sıradaki karakteri getirmektedir. Aşağıdaki örnekte ekrana e bastırılacaktır. Dizi indeksleri 0 dan başladığı için 1. Eleman "e" dir.

```
class Ornek {
	public static void main(String[] args) {
		String s1 = "Melih";
		System.out.println(s1.charAt(1));
	}
}
```

concat

Birleştirme için kullanılır + ile birleştirmeden bir farkı yoktur. Örnek olarak aşağıdaki kod içerisinde yer alan iki birleştirme satırıda aynı davranışı gösterecektir.

```
class Ornek {
	public static void main(String[] args) {
		String s1 = "Merhaba";
		String s2 = "Dünya";
		System.out.println(s1.concat(s2));
		System.out.println(s1 + s2);
	}
}
```

length

Stringin karakter sayısını verir.

```
class Ornek {
	public static void main(String[] args) {
		String s1 = "Merhaba Dünya";
		System.out.println(s1.length()); // 13
	}
}
```

subString

Strin içerisinden ayrıştırma yapmak için kullanılır. Örneğin aşağıdaki gibi bir String içerisinden ayrı bir String ürettirebiliriz.

```
class Ornek {
	public static void main(String[] args) {
		String s1 = "Merhaba Dünya";
		System.out.println(s1.substring(0, 7)); // 7. karaktere kadar
		System.out.println(s1.substring(8)); //8. karakterden sonrası
	}
}
```

indexOf ve lastIndexOf

String içerisinde geçen karakterin sırasını döndürür. lastIndexOf ise karakterin son geçtiği yeri verir.

```
class Ornek {
	public static void main(String[] args) {
		String s1 = "Merhaba Dünya";
		System.out.println(s1.indexOf("a")); // 4 verir
		System.out.println(s1.lastIndexOf("a")); // 12 verir
	}
}
```

startsWith ve endsWith

startsWith ilgili String in başlangıcını kontrol eder. endsWith ise adından da anlaşılabileceği gibi bitişini.

```
class Ornek {
	public static void main(String[] args) {
		String s1 = "Merhaba Dünya";
		System.out.println(s1.startsWith("Me")); // true döner
		System.out.println(s1.endsWith("ya")); // true döner
	}
}
```

toLowerCase ve toUpperCase

toLowerCase ilgili String I tamamen küçük harfe dönüştürür, toUpperCase ise büyük harfe. Ancak bu dönüşüm de yerelleştirme yapmak için aşağıdaki örnekteki gibi Locale sınıfından yararlanılır.

```
import java.util.Locale;

class Ornek {
	public static void main(String[] args) {
		String s1 = "Melih";
		System.out.println(s1.toLowerCase()); // melih
		System.out.println(s1.toUpperCase()); // MELIH
		System.out.println(s1.toUpperCase(new Locale("tr"))); // MELİH
	}
}
```

replace, replaceAll ve replaceFirst

String içerisinde değişiklik yapmak için kullanılır.

replace : Standart olarak karakter veya String i değiştirir.
replaceAll : Pattern kullarak karakter veya String in değiştirilmesini sağlayabilir.
replaceFirst : Sadece ilk yakaladığı karakter veya String i değiştirir. Pattern kullanılabilir.

```
class Ornek {
	public static void main(String[] args) {
		String s1 = "Merhaba Dünya";
		System.out.println(s1.replace("a", "A")); // MerhAbA DünyA
```

```
            System.out.println(s1.replaceAll("[a|e]", "_")); // M_rh_b_ Düny_
            System.out.println(s1.replaceFirst("a", "A")); // MerhAba Dünya
      }
}
```

split

String i verilen pattern a göre ayrıştırmak için kullanılır ve geriye String dizisi döndürür. Aşağıdaki örnekte verilen cümle kelimelere göre ayrıştırılıyor.

```
class Ornek {
      public static void main(String[] args) {
            String s1 = "Bu örnek bir cümle";

            String[] dizi = s1.split(" ");
            for (String eleman : dizi) {
                  System.out.println(eleman);
            }
      }
}
```

format

Formatlı çıktılar almak için kullanılır. Aşağıdaki örnekte çıktı formatlı olarak yazdırılmaktadır.

```
class Ornek {
      public static void main(String[] args) {
            String format = "|%1$-10s|%2$-20s|";
            System.out.println(String.format(format, "Ad", "Soyad"));
            System.out.println(String.format(format, "Melih", "Sakarya"));
            System.out.println(String.format(format, "Ahmet", "Dursun"));

      }
}
```

Buna göre çıktı aşağıdaki gibi olacaktır.

```
|Ad        |Soyad                |
|Melih     |Sakarya              |
|Ahmet     |Dursun               |
```

valueOf

String değer dönüşüm için kullanılır. İlkel tipler, obje ve karakter dizilerini parametre olarak alabilir.

```
class Ornek {
      public static void main(String[] args) {
            int x = 4;
            String xDegeri = String.valueOf(x);

      }
```

```
}
```

trim

String in sağında ve solunda yer alan boşlukları temizler.

toCharArray

String in karakter dizisi olarak dönüşümünü sağlar. Bu durumda aşağıdaki örnekteki gibi her bir karakter dizinin elemanı olur.

```
class Ornek {
	public static void main(String[] args) {
		String s1 = "Merhaba Dünya";
		char[] dizi = s1.toCharArray();
		for (char eleman : dizi) {
			System.out.println(eleman);
		}
	}
}
```

Character Sınıfı

char tipi için wrapper sınıftır ve karakter operasyonları için yardımcı metodlar içerir. Aşağıdaki örnekte karakterlere ait kontroller bulunmaktadır.

```
class Ornek {
	public static void main(String[] args) {
		char deger = 'A';
		char sayi = '2';

		// sayi mi ?
		System.out.println(Character.isDigit(deger)); // false
		System.out.println(Character.isDigit(sayi)); // true

		// karakter mi ?
		System.out.println(Character.isLetter(deger)); // true
		System.out.println(Character.isLetter(sayi)); // false

		// büyük veya küçük harf mi ?
		System.out.println(Character.isUpperCase(deger)); // true
		System.out.println(Character.isLowerCase(deger)); // false


	}
}
```

StringBuffer ve StringBuilder Sınıfları

StringBuffer ve StringBuilder sınıfları Java içerisinde daha büyük karakter işlemleri için kullanılır. Buna örnek vermek gerekirse bir text dokümanı içerisinde yer alan 1000 satırlık bir yazıyı alıp bir String içerisine atamak istiyorsak burada StringBuffer kullanılabilir. Peki bunun avantajı tam olarak nedir ?

Öncelikle daha önce gördüğümüz üzere String sınıfı içerisinde yer alan karakter değerleri char tipindeki final bir dizide tutulmaktadır. Bunun anlamı bu değerin bir daha değiştirilemeyeceği yeni bir char dizisi yaratmak olduğudur. Yani siz String sınıfı içerisindeki referans değeri aslında değiştiremez sadece yeni bir değer oluşturabilirsiniz. Bunun bu şekilde olmasının nedeni String sınıfında tanımladığınız değişkenin atama sonrası başka bir işaretçi tarafından değiştirilebilmesini engellemektir.

```
public class Ornek {
	public static void main(String[] args) {

		StringBuffer sb = new StringBuffer();
		sb.append("Merhaba");
		sb.append(" ");
		sb.append("Dunya");
		System.out.println(sb);

		System.out.println(sb.length());
		System.out.println(sb.capacity());
	}
}
```

Yukarıdaki örnekte basit bir kullanım bulunuyor. Bu kodun çıktısı aşağıdaki gibi olacaktır.

```
Merhaba Dunya
13
16
```

Buna göre birleştirme işlemi için StringBuffer kullanıldı ve normal bir String ten farkı olmadan çıktı alındı. Ancak sonraki satırlarda uzunluk değeri 13 ken kapasite değeri 16 veriliyor. Yani yoğun bellek hareketi olmaması için öncesinde talep edilenden yüksek kapasitede bir tampon alan ayrılıyor.

Bu örneğe göre bir performans testi yapalım öncelikli olarak aşağıdaki kod örneğini çalıştırıyoruz.

```
public class Ornek {
	public static void main(String[] args) {
		String s = "";

		long baslangic = System.currentTimeMillis();
		for (int i = 0; i < 20000; i++) {
			s += "Merhaba Dunya ";
		}
		long bitis = System.currentTimeMillis();

		System.out.println("Gecen sure : " + (bitis - baslangic));
	}
}
```

Bu örneğe göre çıktı aşağıdaki gibi.

```
Gecen sure : 5219
```

Şimdi bu işlemi aşağıdaki gibi StringBuffer ile gerçekleştiriyoruz.

```
public class Ornek {
	public static void main(String[] args) {
		StringBuffer s = new StringBuffer();

		long baslangic = System.currentTimeMillis();
		for (int i = 0; i < 20000; i++) {
			s.append("Merhaba Dunya ");
		}
		long bitis = System.currentTimeMillis();

		System.out.println("Gecen sure : " + (bitis - baslangic));
	}
}
```

Bu durumda ise aşağıdaki gibi bir çıktı aldık.

```
Gecen sure : 2
```

Aradaki ciddi fark daha az bellek hareketi kullanılmasından dolayı kaynaklanıyor.

Şimdiye kadar örnek içerisinde sadece StringBuffer kullandık. Peki StringBuilder sınıfının görevi nedir ? Aslına bakarsanız bu iki sınıf arasında işlevsel olarak bir fark bulunmuyor yan ikiside aynı amaçla kullanılabilir. StringBuilder sınıfı dize 1.5 ile eklenmiştir ve senkronize değildir. Bunun anlamı çoklu Thread durumlarına göre planlanmamıştır ancak StringBuffer a göre daha avantajlıdır.

StringTokenizer Sınıfı

Daha önce String sınıfı içerisinde split() metodunu görmüştük bu metod bir String içeriğini istediğimiz gibi ayrıştırıp dizi elemanı haline getirmemizi sağlıyordu. StringBuffer sınıfıda aslına bakarsanız bu işlemi yapmaktadır.

Aşağıdaki örnekte s değişkenine atanmış bir içeriği boşluk, virgül ve nokta işaretlerine göre ayrıştırıyoruz. Sonrasında while döngüsü içerisinde hasMoreTokens() metodu işe bu ayrıştırmaya uygun bir içerik var mı diye sorguluyoruz. Eğer cevap true dönerse döngü içerisinde nexTokent() metodu ile ayrıştırılmış olan bu içeriği alıyoruz.

```
import java.util.StringTokenizer;

public class Ornek {
	public static void main(String[] args) {
		String s = "Bu deneme amacli bir yazi. " +
				"Ancak, yazi icerisinde farkli " +
```

```
                        "kelimeler olabilir.";

            StringTokenizer cozumleyici = new StringTokenizer(s," ,.");

            while (cozumleyici.hasMoreTokens()) {
                  System.out.println(cozumleyici.nextToken());
            }

      }
}
```

Peki bu durumda split() gibi daha kolay bir yöntem varken neden StringTokenizer kullanalım ? Bu aslında duruma göre değişebilecek bir seçim. Öncelikle split işlemi metod çağırımı ile tüm ayrıştırmayı yapar. Yani sizi içeriğin tamamını ayrıştırmak istemiyorsanız bile öncelikle işlem yapılacak ve gereksiz yere bir işlem zamanı alacaktır. StringTokenizer ise her nextToken() metodu işe bu işlemi yapar.

Buna örnek verecek olursak eğer bir kitap içerisinde geçen kelimeleri ayrıştırıp 4 ten fazla geçtiğini bulmak istiyorsak StringTokenizer kullanılmalı. Örneğin ilk sayfada zaten 4 tane "Java" kelimesi bulunduysa diğer sayfaları ayrıştırmadan uygulamadan çıkılabilir.