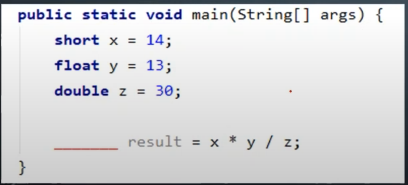
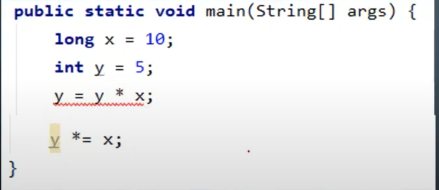
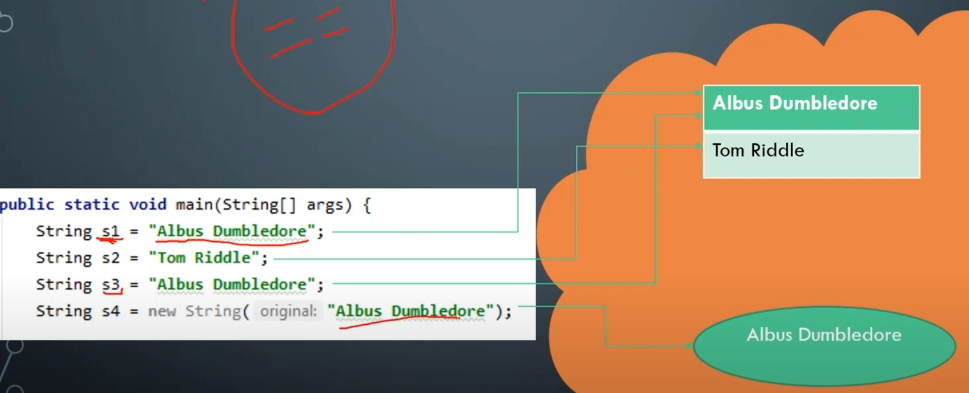
* **Referance tek başına var olamaz Heapte bir şeyle ilişkili olması gerekiyor (null gibi);**
* **Short ve int birlikte işleme girdiğinde short inte promote (terfi) olur**
* **Küçük veri tipleri short double gibi işleme girdiğinde inte promote olur.**

**Burada 3 değişim oluyor ilk önce x short olduğu için işleme girmeden önce inte dönüyor. Ardından float ile işleme giriyor orda float oluyor. En son çıkan sonuç float double ile işleme gireceği için double dönüyor.**

****

**Burada çarpma işlemi yapıp değeri y ye atıyoruz x long döndüğü için ilk işlemde atama yapmamız gerekiyor ikinci işlemde direkt eşitlikte verdiğimizde otomatik olarak inte çevirip sonucu veriyor.**

***Mutable* get set kullanıp başka bi yerde değiştirebiliriz. *Immutable* sadece get kullanılır final ile alınır başka yerde değiştirilemez.**

****

**Aynılık (==) bellekte aynı olan nesneyi ifade eder s1 == s3 → true s1 == s4 → false değer alır.**

**Eşitlik (equals) ise içeriğin aynı olmasına bakar bu durumda s1.equals(s3) → true s1.equals(s4) → true**

** == Operatörü: Referansların aynı nesneyi işaret edip etmediğini kontrol eder.**

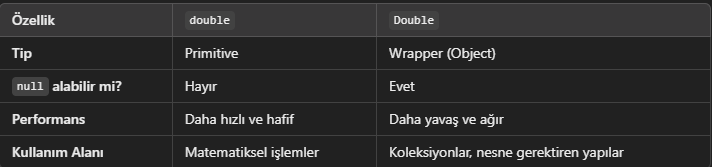
** equals() Metodu: Nesnelerin içeriklerini karşılaştırır.**

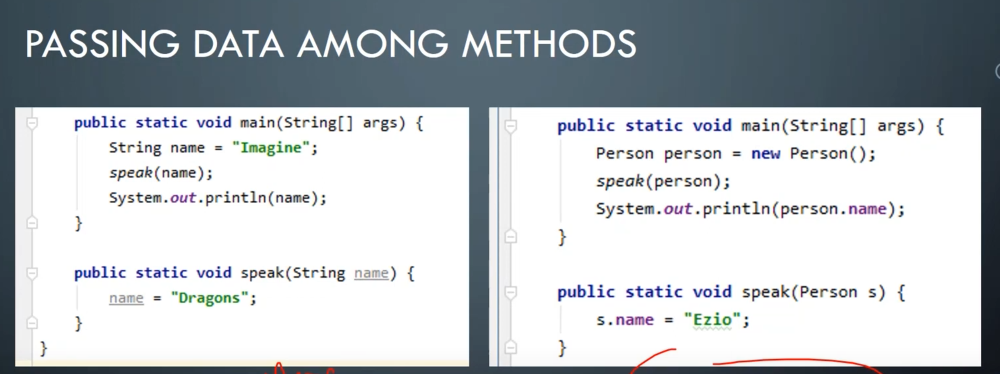
1. **length()**
   * **String'in uzunluğunu döner (karakter sayısı).**
   * **Örnek: "Hello".length() → 5**
2. **charAt(int index)**
   * **Belirtilen konumdaki karakteri döner.**
   * **Örnek: "Hello".charAt(1) → 'e'**
3. **indexOf(String str)**
   * **Verilen alt String'in başlangıç indeksini döner. Bulamazsa -1 döner.**
   * **Örnek: "Hello".indexOf("l") → 2**
4. **substring(int beginIndex, int endIndex)**
   * **Belirtilen başlangıç ve bitiş indeksleri arasındaki alt String'i döner.**
   * **Örnek: "Hello".substring(1, 4) → "ell"**
5. **toLowerCase()**
   * **String'i küçük harflere çevirir.**
   * **Örnek: "HELLO".toLowerCase() → "hello"**
6. **toUpperCase()**
   * **String'i büyük harflere çevirir.**
   * **Örnek: "hello".toUpperCase() → "HELLO"**
7. **equals(Object another)**
   * **İki String'in içeriğinin eşit olup olmadığını kontrol eder.**
   * **Örnek: "Hello".equals("hello") → false**
8. **equalsIgnoreCase(String another)**
   * **Büyük/küçük harf farkını dikkate almadan iki String'in eşit olup olmadığını kontrol eder.**
   * **Örnek: "Hello".equalsIgnoreCase("hello") → true**
9. **startsWith(String prefix)**
   * **String'in belirtilen ön ekle başlayıp başlamadığını kontrol eder.**
   * **Örnek: "Hello".startsWith("He") → true**
10. **endsWith(String suffix)**
    * **String'in belirtilen son ekle bitip bitmediğini kontrol eder.**
    * **Örnek: "Hello".endsWith("lo") → true**
11. **contains(CharSequence seq)**
    * **String'in belirtilen alt diziyi içerip içermediğini kontrol eder.**
    * **Örnek: "Hello".contains("ll") → true**
12. **replace(CharSequence target, CharSequence replacement)** 
    * **String'de belirtilen alt diziyle verilen yeni değeri değiştirir.**
    * **Örnek: "Hello".replace("l", "x") → "Hexxo"**
13. **trim()**
    * **String'in başındaki ve sonundaki boşlukları kaldırır.**
    * **Örnek: " Hello ".trim() → "Hello"**

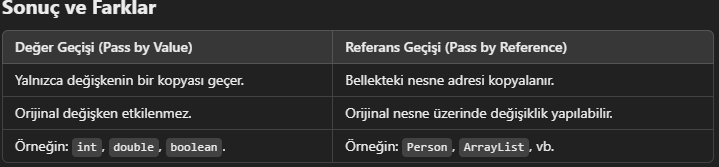
| **Öğe** | **Ne İşe Yarar?** |
| --- | --- |
| **== Operatörü** | **İki referansın aynı nesneyi işaret edip etmediğini kontrol eder. Bellekteki aynı adresi gösterip göstermediklerini belirler.** |
| **equals() Metodu** | **İki nesnenin içeriklerinin aynı olup olmadığını kontrol eder. Özellikle String gibi sınıflarda içerik karşılaştırması için kullanılır.** |
| **hashCode() Metodu** | **Nesnenin hash kodunu döndürür. Hash tabanlı koleksiyonlarda (örneğin HashMap, HashSet) nesnelerin hızlı erişimi için kullanılır.** |
| **compareTo() Metodu** | **İki nesnenin doğal sıralamasını karşılaştırır. Sıralama ve sıralı koleksiyonlarda (örneğin TreeSet, TreeMap) kullanılır.** |
| **StringBuilder Sınıfı** | **Değiştirilebilir (mutable) string oluşturmak için kullanılır. Performans açısından avantajlıdır çünkü yeni string oluşturmak yerine mevcut string üzerinde değişiklik yapar.** |
| **StringBuffer Sınıfı** | **StringBuilder gibi çalışır ancak thread-safe (çoklu iş parçacığı) ortamlar için uygundur. Synchronization sağlar.** |
| **ArrayList Sınıfı** | **Dinamik dizi yapısı sağlar. Eleman ekleme ve erişim işlemlerinde hızlıdır. Sıralı veri depolamak için idealdir.** |
| **LinkedList Sınıfı** | **İkili bağlı liste yapısı sağlar. Eleman ekleme ve çıkarma işlemlerinde daha verimlidir, özellikle listenin başında veya ortasında değişiklik yapıldığında.** |
| **abstract Sınıf** | **Soyut sınıf tanımlamak için kullanılır. Kendinden nesne oluşturulamaz, alt sınıflar tarafından genişletilir ve ortak özellikler/metodlar sağlar.** |
| **interface** | **Sadece metod imzaları ve sabitler içeren bir sözleşme sağlar. Bir sınıfın birden çok interface'i implement etmesine olanak tanır.** |
| **final Anahtar Kelimesi`** | **Değiştirilemez hale getirmek için kullanılır. Değişkenlerde sabit değerler, metodlarda override edilemezlik, sınıflarda genişletilemezlik sağlar.** |
| **static Anahtar Kelimesi`** | **Sınıfın kendisine ait üyeler oluşturmak için kullanılır. Nesne oluşturmadan sınıf seviyesinde erişim sağlar.** |
| **this Anahtar Kelimesi`** | **Mevcut nesnenin referansını belirtir. Özellikle gösterici olarak kullanılır ve metod veya konstrüktörde sınıf üyelerine erişimde kullanılır.** |
| **super Anahtar Kelimesi`** | **Üst sınıfın üyelerine (metod ve değişken) erişmek için kullanılır. Alt sınıflarda üst sınıfın metodlarını çağırmak için idealdir.** |
| **try-catch Bloğu** | **Hata yönetimi için kullanılır. Potansiyel olarak hataya yol açabilecek kodları try bloğunda çalıştırır ve hataları catch bloğunda yakalar.** |
| **finally Bloğu`** | **try-catch bloğundan sonra her zaman çalıştırılan kodları içerir. Kaynakları serbest bırakmak veya temizlik işlemleri için kullanılır.** |
| **lambda İfadeleri** | **Fonksiyonel programlama özelliklerini destekler. Anonim fonksiyonlar oluşturmak için kullanılır, kodu daha kısa ve okunabilir hale getirir.** |
| **Stream API** | **Koleksiyonlar üzerinde fonksiyonel tarzda işlemler yapmayı sağlar. Filtreleme, dönüştürme, toplama gibi işlemleri kolaylaştırır.** |

**Ek Açıklamalar**

* **== vs equals(): Daha önce açıkladığımız gibi, == operatörü referans eşitliğini kontrol ederken, equals() metodu içerik eşitliğini kontrol eder.**
* **hashCode() ve equals() İlişkisi: İki nesne eşitse (equals() metodu true dönerse), hashCode'ları da aynı olmalıdır. Bu, hash tabanlı koleksiyonların doğru çalışması için önemlidir.**
* **StringBuilder vs StringBuffer: StringBuilder daha hızlıdır çünkü synchronized değildir, ancak çoklu iş parçacığı ortamlarında güvenli değildir. StringBuffer ise thread-safe'dir ancak biraz daha yavaştır.**
* **abstract Sınıf vs interface: abstract sınıflar hem somut hem de soyut metodlar içerebilirken, interface genellikle sadece soyut metodlar ve sabitler içerir. Java 8 ile birlikte, interface'ler de varsayılan ve statik metodlar içerebilir hale gelmiştir.**

****

* ** main metodu içinde name = "Imagine" atanır.**
* **speak(name) çağrılır ve burada "Imagine" değerinin bir kopyası speak metoduna geçer.**
* **speak metodu içinde name = "Dragons" atanır, fakat bu değişiklik yalnızca metodun içinde geçerlidir. Orijinal name değişkeni etkilenmez.**
* **System.out.println(name) çıktısı "Imagine" olur. (SOL TARAF)**
* ** main metodu içinde Person nesnesi oluşturulur ve referansı person değişkenine atanır.**
* ** speak(person) çağrıldığında, person nesnesinin referansı metoda geçer. Artık s ve person aynı nesneyi işaret eder.**
* ** speak metodunda s.name = "Ezio"; yazıldığında, person nesnesinin name özelliği değiştirilir. Bu değişiklik orijinal nesne üzerinde etkilidir.**
* ** System.out.println(person.name) çıktısı "Ezio" olur. (SAĞ TARAF)**

****