**VARARGS**

Java, değişken sayıda argüman alması gereken metotların oluşturulmasını basitleştiren bir özelliğe sahiptir. Bu özelliğe değişken uzunluklu argümanlar(variable-length arguments) anlamındaki varargs denir.

Varargs tek method’a istediğimiz kadar parametre yollayarak sonuç alamamızı sağlar. Yani parametre sayimiz değişken, ancak method’un yapacagi iş sabitse VARARGS kullanarak tek method’la kodumuzu yazabiliriz.

add(5,7);

add(5,7,-15);

add(5,7,-15,20);

add(5,7,-15,20,30);🡺 bu sekilde bir kod bloğu hazırlamak istediğimizde her bir işlem için ayrı ayrı method’lar oluşturmamız gerecekti.

Bu durumu kolaylaştırmak için java yeni bir date type oluşturmuş.

Varargs’ta parametre sayısının önemi yoktur.Kac tane olursa da alır,işte bu kısımda VARARGS’ ın içinde kac tane sayı oldugunu bilmediğimiz için FOR..EACH LOOP’a ihtiyacımız vardır.

For Each index ile lenght,size ile uğraşmaz,her elemanı sırasıyla getirir.

*topla*(5,10,13,0);

private static void topla(int...sayilar) {

int toplam=0;

for (int each : sayilar) {🡪sayilar varargs’ına git oradaki her bir int bana getir.

toplam+=each;

}

System.*out*.println("girilen sayilarin toplami : " + toplam); }

Topla method’unda birden fazla sayıyı toplamak istediğimde argument sayısı kadar parametre oluşturmalıyım fakat VARARGS int...sayilar satırı ile bu işlemi kolayca yapıyor.

Varargs List gibi davranır ama Array Method’ları ile çalışır.

Verilen Sayilardan İlkini Ayırarak İşlem Yapmak İstiyorsak;

*KafanaGoretopla*(5,10,13,0);

}

private static void KafanaGoretopla(int sayi1, int...sayilar) {

int toplam=0;

for (int each : sayilar) {

toplam+=each;

}

System.*out*.println("ilk sayi ile digerlerinin toplaminin carpimi : " + sayi1\*toplam);

VARARGS’ta hiç eleman olmasa bile Java itiraz etmez. Önce int olarak tanımlanan sayıları eşleştirir, kalan tüm sayıları Varargs’a doldurur.

Varargs Kullanırken Dikkat Edilmesi Gereken En Önemli Nokta Şudur;

*topla*(5,10,13,0);

private static void topla(int...sayilar, int sayi1);

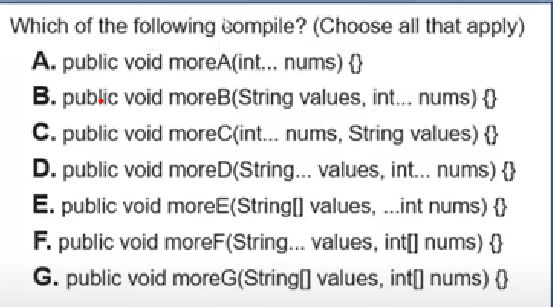
🡪Bu şekilde bir yazımda VARARGS tüm sayıları kendine toplar dolayısıyla Varargs kullanırken Varargs’tan sonra bir parametre yazılmaz. Bu şekilde bir kullanımda Java hat verir.

*topla*(5,10,13,0,"Ali Can");

private static void topla(int...sayilar, String sonParametre);

🡪Böyle bir yazım türü de hata verir,varargs’tan sonra parametre olamaz,Varargs sonuncu olmalı. Bu şekilde bir kod yazımında Java Varargs son parametre olmalı diye uyarır.Şayet buradaki String parametreyi de Varargs yapmak istersek yine hata verir.

🡪 Bir method’da sadece bir Varargs olabilir.



**STRING BUILDER**

🡪String Builder “mutable” yani değiştirilebilir String elde etmemize olanak tanır.

StringBuilder sb =**new** StringBuilder("Yasasin Java");

sb.append("!!!");

System.***out***.println(sb);

🡪 Böylece hafızada her seferinde yeni bir alan açılmadan var olan alan üzerinde değişiklik yapılabilir. Bu da StringBuilder sınıfını hafıza kullanımı olarak String sınıfının önüne geçirir.

🡪String Builder thread-safe değildir. Yani synchronized değildir. Thread’lı bir işlem kullanılacaksa StringBuilder kullanılması güvenli değildir.

🡪Not:StringBuffer,StringBuilder’a benzer. StringBuilder, StringBuffer’dan hızlıdır. Multi-thread için StringBuffer kullanılır.

1-StringBuilder sb1 =**new** StringBuilder()🡪boş bir sb oluşturur.

2- StringBuilder sb2 =**new** StringBuilder(“animal”);🡪Belli bir değeri olan StringBuilder oluşturur.

3- StringBuilder sb =**new** StringBuilder(11); 🡪Tüm non-primitive data türleri için, değişken olmakla beraber ilk başta oluşturulurken bir hafıza tanımlanır. Bu StringBuilder için 16 bit, yani bu tanımlanan kadar yer açar. Doldurmak ya da doldurmamak kullanıcıya kalmıştır. Standartları olan veriler için örneğin tel. numbers 11 karakterden oluşur.11 karakter yeterli iken 16 karakterlik yer işgal etmek büyük programlarda boşuna gereksiz alan kullanımına sebep olur.

Bu durumda oluşturulan StringBuilder’ın kaç karakterli olabileceği belilenebilir. Adım attıkça basacağımız yeri oluşturur

