Java ile Nesne Merkezli Programlamaya Giriş

5. Bölüm
Akış Kontrolü

Akın Kaldıroğlu

www.javaturk.org

Aralık 2016

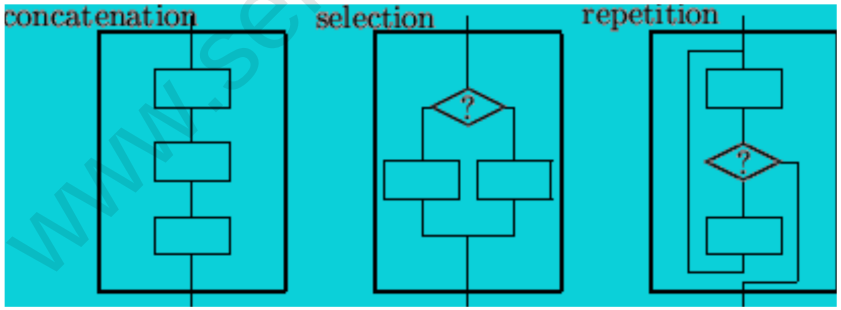
Küçük Ama Önemli Bir Konu

- Bu dosya ve beraberindeki tüm, dosya, kod, vb. eğitim malzemelerinin tüm hakları **Selsoft Yazılım, Danışmanlık, Eğitim ve Tic. Ltd. Şti**.'ne aittir.
- Bu eğitim malzemelerini kişisel bilgilenme ve gelişiminiz amacıyla kullanabilirsiniz ve isteyenleri **http://www.selsoft.academy** adresine yönlendirip, bu malzemelerin en güncel hallerini almalarını sağlayabilirsiniz.
- Yukarıda bahsedilen amaç dışında, bu eğitim malzemelerinin, ticari olsun/olmasın herhangi bir şekilde, toplu bir eğitim faaliyetinde kullanılması, bu amaca yönelik olsun/olmasın basılması, dağıtılması, gerçek ya da sanal/Internet ortamlarında yayınlanması yasaktır. Böyle bir ihtiyaç halinde lütfen benimle, **akin.kaldiroglu@selsoft.academy** adresinden iletişime geçin.
- Bu ve benzeri eğitim malzemelerine katkıda bulunmak ya da düzeltme ve eleştirilerinizi bana iletmek isterseniz çok sevinirim.
- Bol Java'lı günler dilerim.

Kontrol İfadeleri

Akış Kontrolü

- Nesne-merkezli olsun olmasın her programlama dilinde akışı kontrol etmeyi sağlayan ifadeler vardır.
- Edsger Dijkstra'ya göre en temel 3 kontrol yapısı şunlardır:
  - Ardışıllama (concatenation)
  - Seçme/karar verme (selection)
  - Tekrarlama (repetation, looping)



Kontrol İfadeleri

- Diller, ifadelerin ardışıl olarak çalışmalarını tabii olarak sağlarlar.
- Geri kalan kontrol yapıları ise özetle şunlardır:

Tekrarlama İfadeleri

while

- **while**, mantıksal ifade doğru olduğu müddetçe, kendisinden sonra gelen ifade ya da ifade bloğunun çalışmasını sağlar.

```
while (mantıksal ifade)
        ifade
ya da
while (mantıksal ifade) {
        ifade(ler)
}
```

WhileDemo.java

do-while

- **do-while**, **do**'dan sonra gelen ifade ya da ifadeleri, **while**'in mantıksal ifadesi doğru olduğu müddetçe çalıştırır.
  - **while**'dan farkı, ifade bloğunun en az bir kere çalışmasıdır.
  - **while** satırındaki "**;**"ü unutmayın.

```
do
    ifade
while (mantıksal ifade);
```

ya da

```
do{
   ifade(ler)
}
while (mantıksal ifade);
```

DoWhileDemo.java

for

- **for**, ifadeyi ya da ifade bloğunu, başlangıç, bitiş ve değişim değerlerinin merkezi olarak yöneterek, tekrarlı olarak çalıştırmakta kullanılır.
- İlk değer atama, bitiş şartı ve değişim zorunlu değildir ama **for** parantezinde iki tane "**;**" bulunmalıdır.

```
for (ilk değer atama; bitiş şartı; değişim)
        ifade

ya da

for (ilk değer atama; bitiş şartı; değişim){
        ifade(ler)
}
```

ForDemo.java

ListCharacters.java

Gelişmiş for

- **for** ifadesinin genelde “for each” olarak adlandırılan bir gelişmiş hali daha vardır.
- Bu yapıyı ileride göreceğiz.

Karar İfadeleri

if

- **if**, bir mantıksal ifadeye bağlı olarak karar verme ve seçme için kullanılır.
- Mantıksal ifade doğru ise ifade ya da ifade bloğu çalışır yanlışsa çalışmaz.
  - Her iki halde de akış, **if** ifadesi ya da bloğundan sonraki ifadeden devam eder

```
if (mantıksal ifade)
    ifade

ya da

if (mantıksal ifade) {
    ifade(ler)
}
```

ListCharactersWithIf.java

if-else

- **if**'in mantıksal ifadesi doğru olmadığında da çalışacak bir blok varsa, **else** kullanılır.

```
if (mantıksal ifade)
    ifade
else
    ifade

ya da

if (mantıksal ifade){
    ifade(ler)
}else{
    ifade(ler)
}
```

if-else if-else

- Birden fazla şarta bağlı olarak çalışacak alternatif ifadeler ise çoklu **if else if else** ile ifade edilebilir.
- Sondaki **else** zorunlu değildir.

```
if (mantıksal ifade)
    ifade
else if (mantıksal ifade)
    ifade
...
else if (mantıksal ifade)
    ifade
else
    ifade
```

IfElseDemo.java

Uygulama

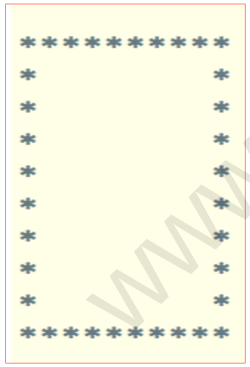
- Aşağıdaki sınıflandırmaya göre girilen bir yaşın hangi aralıkta olduğunu bulan ve yazan bir program yazın.
  - < 0 or > 120 : Çık
  - 0-3 : Bebek
  - 4-12 : Çocuk
  - 13-19 : Ergen
  - 20-30 : Genç
  - 31-49 : Orta Yaş
  - 50-120 : Yaşlı

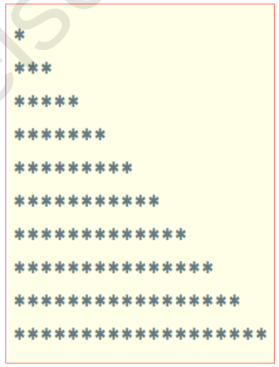
Uygulama

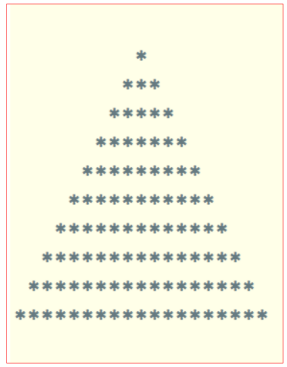
- Yukarıdaki sınıflandırmaya göre bir ailedeki bireylerin yaşlarını alıp, aralığını veren ve en sonunda da ailede hangi aralıktan kaç kişi olduğunu yazan bir program yazın.

Uygulama

- “*” kullanarak verilen bilgilerle aşağıdaki şekilleri çizen programlar yazın:
  - Yükseklik ve genişlik ile dikdörtgen,
  - Yükseklik ile dik üçgen,
  - Yükseklik ile eşkenar üçgen







Üçlü if-else Operatörü « ? : »

- Üçlü koşullu (ternary conditional) *operatör*, "**?:**" 3 tane işlenen alan tek operatördür.
- Mantıksal ifade doğru ise ifade1 değilse ifade2 çalışır:

```
Mantıksal ifade ? ifade1 : ifade2
```

```
double random = Math.random();
String para = random > 0.5 ? "Tura" : "Yazı";
```

```
String para;
double random = Math.random();
if(random > 0.5)
    para = "Tura";
else
    para = "Yazı";
```

TernaryOperatorDemo.java

Cümle Problemi

- Aşağıdaki gibi bir durumla karşılaştınız mı?

```
if(true)
    int u = 7;  // Hata

for (int j = 0; j < 10; j++)
    int k = j;  // Hata
```

- Problem, blok kullanılmadığında, **if** ve bir değişken tanımının tek bir cümlede (statement) ifade edilmesidir,
- Aslında değişken tanımı zaten tek bir cümledir ve bu şekilde bir **if** (ya da **for**, **while** ve **do-while**) cümlesinin içinde kullanılmamalıdır.
  - Çözüm ya blok kullanmak ya da değişkeni önceden tanıtmaktır.

StatementProblem.java

== Operatörü (Tekrar)

- Eşitlik kıyaslamak için "**==**" operatörünü kullanın.
- **if** ya da **while** gibi yapılarda yanlışlıkla eşitlik kıyaslaması için "**==**" yerine atama operatörü "**=**" yazmak, yaygın bir hatadır.

switch

- **switch**, if-else if-else yapısının int ve String olan ifadeler için daha toplu bir şeklidir.
- Bir int sonucu veren int-ifade ya da String ifade **case**'lerdeki hangi ifadeye eşitse ondan sonra gelen blok çalıştırılır.
- Eğer hiç birine eşit değilse **default**'un ifadeleri çalışır.

```
switch(int/String-ifade){
    case: sabite1          ifade(ler)
    case: sabite2          ifade(ler)
    . . .
    default:              ifade(ler)
}
```

SwitchDemo.java

Bazı Noktalar I

- switch'in ifadesi byte, short, char ya da int olmalıdır.
  - Java 7 ile bu durumda artık String de kullanılabilmektedir
- case'den sonraki değerler de yukarıdaki tipten ve ancak sabite olabilir,
  - Değişken kullanılırsa final olmalıdır.
  - Eğer değişken final olsa bile bir metot çağrısıyla üretiliyorsa, derleyici buna izin vermeyecektir.
- Hiç bir iki case sabitesi aynı olamaz.

Bazı Noktalar II

- İfadenin değerine eşit bir sabiteye sahip olan case'in ifade bloğu bir break görünceye kadar çalışır.
- default ifadesi zorunlu değildir ve en fazla bir tane olabilir.
- default ifadesinin ya da diğer case ifadelerinin sırası önemli değildir.

Uygulama

- Kendi projenizde *SwitchDemo* isimli bir sınıf oluşturup, main metotunu, dersin örneği olan *SwitchDemo'*dan aynen kopyalayın.
  - Main metotundaki *switch* bloğuna yapacağınız değişiklikle girilen aya karşılık gelen mevsimi yazmasını sağlayın.
  - Bunu sadece 5 print ve 4 break cümlesiyle yapın.

break ve continue

- break ve continue, birbirlerine benzer iki tekrarlı yapılarda kullanılan kontrol yapısıdır.
- Etiketli ve etiketsiz olmak üzere 2 hali vardır.
  - break, switch, for, while ve do-while içinde, continue ise sadece for, while ve do-while içinde kullanılır, aksi taktirde derleme zamanı hatası oluşur.
- Etiketsiz hallerinde
  - break içinde bulunduğu tekrar yapısını tamamen kırar ve akış, tekrar bloğunu takip eden ifadeden devam eder.
  - continue ise bulunduğu tekrar yapısını sadece o adım için kırar ve akış tekrar bloğunun bir sonraki adımından devam eder.

BreakDemo.java ve Continue.java

Uygulama

- Girilen bir sayının asal olup olmadığını belirleyen bir program yazın.
- Program aynı zamanda girilen sayı asal değilse, ilk bölenini de belirleyip yazacaktır.
- Unutmayın ki bir sayı karekköküne kadar olan tamsayılara bölünemiyorsa sonrakilere zaten bölünemeyecektir.

Etiketli break ve continue

- Eğer iç içe birden fazla blok var ve break ya da continue, içteki bloklardan birindeyse, daha dışarıda olan bloğu kırmanın yolu etiket (label) kullanmaktır:
  - Etiket, dıştaki bloklardan birini işaretler ve break kullanıldığında akış, etiketin işaretlediği bloktan sonra gelen ifadeden devam eder.
  - continue kullanıldığında ise, etiketin işaretlediği tekrar yapısının adımı kırılır ve akış bir sonraki adımdan devam eder.

BreakAndContinueWithLabel.java

- Aşağıdaki durumlarda programın nasıl davranacağını bulmaya çalışın ve sonra da deneyin:
  - break search;
  - break;
  - continue search;
  - continue;

return

- **return**, bir metottan, o metotun çağırıldığı ortama dönüş yapmak için kullanılır.
  - Metotun main olması durumunda bu JVM'in çıkması anlamına gelir.
- İki formu vardır:
  - Metot bir değer döndürüyorsa, değerin tipi ile metotun dönüş tipi, uyumlu olmalıdır,
  - Metot bir değer döndürmüyorsa **return**'e gerek yoktur ama gerekirse dönüş değersiz olarak kullanılabilir.

```
return maas;

ya da

return;
```

ReturnDemo.java

Özyineleme ya da Recursion

Özyineleme ya da Recursion

- Algoritmalar dünyasında bir teknik olan **özyineleme** ya a **recursion**, metotların kendilerini tekrarlı olarak çağırmalarına denir.
- Bazı algoritmalar özyinelemeli (recursive) olarak çalışabilir.
- Özyinelemeli olarak çalışan algoritmaların, özyinelemesiz yani özyineleme kullanmadan yazılmaları da söz konusudur.

Factorial Algoritmaları

- algorithms paketinde
  - FactorialExampleByFor.java
  - FactorialExampleByRecursion.java

Fiboniacci Algoritmaları

- Fiboniacci sayılarını bulan algoritmayı, yinelemeli ve yinelemesiz olarak yazın.

Diğer Kontrol Yapıları

Erişilemeyen Kod

- Java, erişilemeyen kod'a (unreachable code) izin vermez.
  - Dolayısıyla return'den sonra kod olmamalıdır.
  - Sonsuz döngüler de zaman zaman erişilemeyen kod parçalarına sebep olabilir.

```
while(true)
    System.out.println("Selam");

int i = 5; // Unreachable code!

ya da

while(false)
    System.out.println("Selam");
    // Unreachable code!
```

UnreachableCode.java

Sıradışı Durum Yönetimi

- Java'da sıradışı durum yönetimi de akış kontrolu içerir.
  - İleride ayrı bir bölümde ele alınacaktır.

```
try {
    ifadeler
} catch (SıradışıDurumTipi nesne) {
    ifade(ler)
} finally {
    ifade(ler)
}
```

Java'a **goto** Var Mı?

- **goto**, Java'da henüz bir anlamı ve kullanımı olmayan ama ileride olabilir diye saklanan bir anahtar kelimedir.
- Klasik kullanışıyla **goto**, programları yapısal olmaktan çıkarmakla eleştirilmişti,
  - Edsger Dijkstra, 1968'de ünlü "**Goto Considered Harmful**" makalesini yayınladı.

Özet

- Bu bölümde akış kontrolünde kullanılan yapıları gördük.
- Ayrıca erişilemeyen cümleler ve **goto**'dan bahsettik.
- Bu bölüm ile birlikte artık nesne-merkezli olmasa da prosedürel Java programlarını büyük ölçüde yazabilirsiniz.
  - Algoritmik yapılar bu bölüm için güzel örnek oluştururlar.

Ödevler

Ödevler

- 2. dereceden ax^2+bx+c = 0 şeklindeki bir denklemin köklerini hesaplayan bir program yazın.
- Girilen bir sayıya kadar kaç tane asal sayı olduğunu hesaplayan bir program yazın.
- Monte Carlo yöntemini kullanarak Pi sayısını hesaplayan bir program yazın.
- FizzBuzz oyununu yazın. Bilgi için https://en.wikipedia.org/wiki/Fizz_buzz

Ödevler

- Girilen bir sayının rakamlarının yerlerini değiştirerek tersten yazan bir program yazın.
- 1, 12 ve 25 sayfa fotokopinin fiyatları aşağıda verilmiştir. Buna göre çekilecek n sayfalık fotokopinin minimum fiyatını "**int fiyatHesapla(int n)**" şeklinde arayüze sahip bir metod ile hesaplayan bir program yazın:
  - 1 copy: 5 Kuruş
  - 12 copies: 50 Kuruş
  - 25 copies: 1 Lira

Aynı programı "**int özyinelemeliFiyatHesapla(int n)**" şeklinde bir arayüze sahip özyinelemeli (recursive) bir metotla tekrar yazın.

Ödevler

- Bir tam sayının bölenlerini bulan bir program yazınız.
- Bir tam sayının asal bölenlerini bulan bir program yazınız.
  - Fundamental theorem of arithmetic (the unique factorization theorem or the unique-prime-factorization theorem): Every integer greater than 1 either is prime itself or is the product of prime numbers, and that this product is unique, up to the order of the factors.
  - https://en.wikipedia.org/wiki/Fundamental_theorem_of_arithmetic
  - 100 = 2^2 * 5^2, 1000 = 2^3 * 5^3, 17248 = 2^5 * 7^2 * 11