

**T.C.
MİLLÎ EĞİTİM BAKANLIĞI**

BİLİŞİM TEKNOLOJİLERİ

**NESNE TABANLI PROGRAMLAMADA
KARAR VE DÖNGÜ YAPILARI
482BK0161**

Ankara, 2011

- Bu modül, mesleki ve teknik eğitim okul/kurumlarında uygulanan Çerçeve Öğretim Programlarında yer alan yeterlikleri kazandırmaya yönelik olarak öğrencilere rehberlik etmek amacıyla hazırlanmış bireysel öğrenme materyalidir.
- Millî Eğitim Bakanlığınca ücretsiz olarak verilmiştir.
- **PARA İLE SATILMAZ.**

İÇİNDEKİLER

AÇIKLAMALAR	ii
GİRİŞ	1
ÖĞRENME FAALİYETİ-1	3
1. MANTIKSAL OPERATÖRLER.....	3
1.1. Boolean Operatörler	4
1.1.1. Eşitlik ve İlişkisel Operatörler	4
1.1.2. Koşullu Mantıksal Operatörler	5
UYGULAMA FAALİYETİ	9
ÖLÇME VE DEĞERLENDİRME	11
ÖĞRENME FAALİYETİ-2	13
2. ŞART İFADELERİ	13
2.1. IF İfadesi	13
2.2. İç İçe If İfadesi	15
2.3. If-Else if İfadesi	16
2.4. Switch İfadesi.....	20
UYGULAMA FAALİYETİ	24
ÖLÇME VE DEĞERLENDİRME	26
ÖĞRENME FAALİYETİ-3	28
3. DÖNGÜ YAPILARI	28
3.1. Bileşik Atama Operatörleri	28
3.2. While İfadeleri	30
3.3. For İfadeleri	32
3.4. Do İfadeleri	34
3.4.1. Break ve Continue İfadeleri	35
UYGULAMA FAALİYETİ	38
ÖLÇME DEĞERLENDİRME.....	40
ÖĞRENME FAALİYETİ-4	42
4. HATA AYIKLAMA	42
4.1. Try - Catch Bloku	42
4.2. Birden Çok Catch Bloku	45
4.3. Denetlenmiş İfadeler.....	47
4.4. Denetlenmiş Deyimler	50
4.5. Özel Durumlar	50
4.6. Finally Bloku	53
UYGULAMA FAALİYETİ	55
ÖLÇME VE DEĞERLENDİRME	57
MODÜL DEĞERLENDİRME	58
CEVAP ANAHTARLARI.....	60
KAYNAKÇA	62

AÇIKLAMALAR

KOD	482BK0161
ALAN	Bilişim Teknolojileri
DAL/MESLEK	Veritabanı Programcılığı
MODÜLÜN ADI	Nesne Tabanlı Programlamada Karar ve Döngü Yapıları
MODÜLÜN TANIMI	Mantıksal operatörler vasıtasıyla şartlı ifadeler ve tekrarlı işlemler gerçekleştiren kodlamalar ile bu kodlamalardaki hataların yönetiminin kazandırıldığı bir öğrenme materyalidir.
SÜRE	40/32
ÖN KOŞUL	Nesne Tabanlı Programlamada Metotlar modülünü almış olmak
YETERLİK	Karar ve döngü ifadelerini kullanmak
MODÜLÜN AMACI	Genel Amaç Bu modül ile gerekli ortam sağlandığında karar ve döngü ifadelerini program içerisinde kullanabileceksiniz. Amaçlar 1. Mantıksal operatörleri kullanabileceksiniz. 2. Şart ifadelerini kullanarak programın işleyişini yönlendirebileceksiniz. 3. Döngü ifadeleri ile tekrarlı işlemler yapabileceksiniz. 4. Kod içeriğinde hataları ve özel durumları yönetebileceksiniz.
EĞİTİM ÖĞRETİM ORTAMLARI VE DONANIMLARI	Ortam: Atölye, laboratuvar, bilgi teknolojileri ortamı (internet) vb., kendi kendinize veya grupta çalışabileceğiniz tüm ortamlar Donanım: Nesne tabanlı programlama yazılımını çalıştırabilecek yeterlikte bilgisayar, yedekleme için gerekli donanım (CD yazıcı, flash bellek), raporlama için yazıcı, kâğıt ve kalem
ÖLÇME VE DEĞERLENDİRME	Modül içinde yer alan her öğrenme faaliyetinden sonra verilen ölçme araçları ile kendinizi değerlendireceksiniz. Öğretmen modül sonunda ölçme aracı (çoktan seçmeli test, doğru-yanlış testi, boşluk doldurma, eşleştirme vb.) kullanarak modül uygulamaları ile kazandığınız bilgi ve becerileri ölçerek sizi değerlendirecektir.

GİRİŞ

Sevgili Öğrenci,

Şu ana kadar adını çok sık duyduğunuz ancak kendisiyle yeni tanıştığınız nesne tabanlı programlama mantığı ile onuncu sınıftan bu tarafa gördüğünüz programlama mantığını bu dersle birleştiriyorsunuz. Sene sonunda bu dersi başardığınızda nesne tabanlı programlamanın gerçekten programcılık sektörüne ne büyük avantajlar getirdiğini daha iyi anlayacaksınız.

Bu modülle programcılık sektöründe, bilgisayarı bir insan gibi düşündürüp önüne getirilen birkaç seçenektan en uygununu seçmesini sağlayabileceksiniz. Aynı zamanda, aynı kodları tekrar tekrar yazmak yerine döngüler vasıtasıyla bir defa yazıp tekrarını sağlayabileceksiniz.

Onuncu sınıftan beri şu şekilde bir tecrübe edinmiş olmalısınız: “Hataların yönetimi programın profesyonelleşmesindeki ilk adımdır.” Çünkü hatalar programın kullanılabilirliğini düşürür veya artırır. Hataların kontrol altına alınması programınızın doğru bir şekilde kullanılmasını sağlayacak, tersi durumda ise programınızın kullanılabilirliğini azaltacaktır. İşte bu modül de aynı zamanda hataların program içerisinde nasıl yönetileceğini öğrenmiş olacaksınız.

ÖĞRENME FAALİYETİ-1

AMAÇ

Mantıksal operatörleri program içerisindeki karar ve döngü yapılarında kullanacak şekilde öğrenebileceksiniz.

ARAŞTIRMA

- Gündelik hayatımızda sonucu doğru ya da yanlış olan pek çok önermeyle karşılaşırız. Örneğin, meteorolojiden havanın yağışlı olup olmayacağını sorduğunuzda cevap evet ya da hayır olacaktır. Siz de sonucu true (doğru) veya false (yanlış) olan önermelerden örnekler bulunuz. Bu örnekleri öğretmenimize teslim edecek veya sınıfta tartışacak şekilde hazırlayınız.

1. MANTIKSAL OPERATÖRLER

Mantıksal operatörler programlama dünyasında daha çok karşılaştırma amacıyla kullanılan operatörlerdir. Bilgisayarın karar verme mekanizmasını harekete geçiren operatörler olarak da bilinmektedir.

Örneğin, “Öğretmenimiz kendi dersinin sınavından 90 üzeri alan öğrencileri ödüllendirdi.” dediği zaman buradaki mantıksal operatörü 90 üzeri yani 90’dan büyük demektir, dolayısıyla da BÜYÜKTÜR olarak ifade edilebilir.

Başka bir örnek vermek gerekirse “Beden eğitimi dersi öğretmenimiz voleybol maçı kadrosuna sınıfımızdaki öğrencilerden isminin baş harfi A olmayanlardan seçti.” dediğinde, buradaki mantıksal operatör isminin baş harfi A olmayanlar yani A’ya eşit olmayanlar demektir ki buda EŞİT DEĞİL olarak ifade edilir.

Yukarıdaki örnekler gibi gerçek hayattan pek çok örnekler verilebilir. Bütün bu örneklerin programlama dilinde nasıl ifade edildiği aşağıda görülecektir.

Değişken, bir programda kullanılacak verileri geçici bir süre fiziksel bellekte (RAM) barındıran yapılardır. Boolean değişkenler de program içerisindeki mantıksal bir sonucu barındıran yapılardır fakat bu değişkenler sadece iki değer alabilir: “True ya da False”dir. True; doğru, evet, açık gibi anlamlarda; False ise yanlış, hayır, kapalı gibi anlamlarda kullanılır. Yani boolean değişkenlere iki durumlu değişkenlerdir de denilebilir.

Örnek: Nesne tabanlı programlama dersinden geçebilmek için gerekli ortalama 45'tir. Öyleyse şöyle bir düşünüldüğünde ortalama 45'ten büyükse öğrenci dersten geçmiş olacaktır.

Bir öğrencinin ortalaması 58 olsun. Bu durumda şöyle bir sorgu geliştirilir: Öğrencinin ortalaması 45' ten büyük mü? Cevap: Evet (Doğru)

Bir öğrencinin de ortalaması 36 olsun. Bu durumda ise soru ve cevap da şu şekilde olacaktır: Öğrencinin ortalaması 45' ten büyük mü? Cevap: Hayır (Yanlış)

Yukarıdaki örnekte görüldüğü gibi yalnız iki değerden bahsedildi. Bu da doğru ve yanlıştır. İşte boolean değişkenler sadece bu değerleri alabilen değişkenlerdir.

Programlama dilinde bir boolean değişken tanımlamak şu şekildedir:

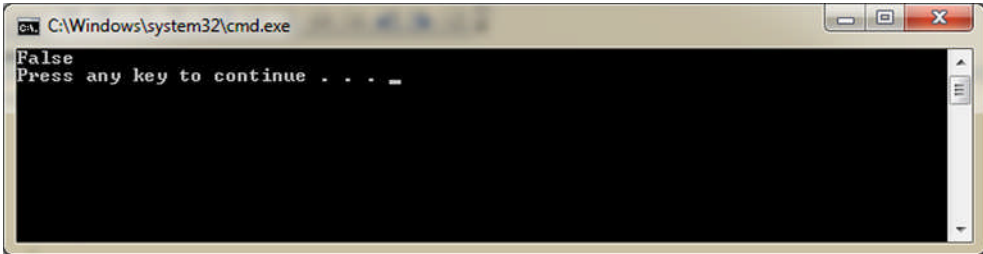
bool [değişken ismi];

Aşağıda Resim 1.1'deki örnek incelenmelidir:

```
static void Main(string[] args)
{
    bool sonuc; //boolean değişken tanımlaması yapılmıştır
    sonuc = false; //boolean değişkene değer ataması yapılmaktadır
    Console.WriteLine(sonuc); //değişken değeri ekrana yazılmaktadır
}
```

Resim 1.1: Boolean değişkenlerin program kodu

Yukarıdaki örnek çalıştırıldığında ekrana aşağıdaki pencere gelecektir.



Resim 1.2: Boolean değişkenlerin ekran çıktısı

1.1. Boolean Operatörler

Boolean operatörler sonucu doğru ya da yanlış olan hesaplamayı gerçekleştiren operatörlerdir. Programlama dillerinde pek çok boolean operatör vardır.

1.1.1. Eşitlik ve İlişkisel Operatörler

Bu operatörler için belki de en sık kullanılan operatörlerdir denebilir. Eşitlik operatörü, aynı türdeki iki değer birbirine eşit olup olmadığını sorgulayan operatördür. İki

değerin birbirine eşit (= =) olup olmadığı sorgulanabileceği gibi birbirinden farklı (!=) olup olmadığı da sorgulanabilir.

NOT: Dikkat edildiyse eşittir operatörü, yan yana iki eşittir (= =) işaretinden oluşmaktadır. Bir de tek eşittir (=) işareti vardır ki bu da atama işlemleri için kullanılmaktadır. Örneğin, a==b yi karşılaştırır ve değerler aynıysa true değerini üretir. Ancak a = b ifadesi, b değerini a ' ya aktarır. Bu iki ayrımı iyi yapmak gerekir.

Aşağıdaki tabloda bu operatörler örneklerle kısaca özetlenmiştir.

Operatör	Anlamı	Örnek	Ödül Puanı 90 ise Sonuç
==	Eşittir	Puan == 77	False (77 eşit 90 mı?)
!=	Eşit değil (Farklı)	Puan != 77	True (77 farklı 90 mı?)

Tablo 1.1: Eşitlik operatörler

Yukarıdaki tabloda gördüğünüz gibi hem eşit işaretiyle hem de eşit değil yani farklı işaretiyle sorgulama yapılmıştır.

Diğer sık kullanılan operatör çeşidi olarak da ilişkisel operatörler söylenebilir. İlişkisel operatörler, bir verinin aynı türden diğer veriden büyük veya küçük olup olmadığını sorgulamak için kullanılan operatörlerdir.

Operatör	Anlamı	Örnek	Ödül Puanı 90 ise Sonuç
<	Küçük	Puan < 77	True (77 küçük 90 mı?)
<=	Küçük veya eşit	Puan <= 77	True (77 küçük veya eşit 90 mı?)
>	Büyük	Puan > 77	False (77 büyük 90 mı?)
>=	Büyük veya eşit	Puan >= 77	False (77 büyük veya eşit 90 mı?)

Tablo 1.2: İlişkisel operatörler

1.1.2. Koşullu Mantıksal Operatörler

Bütün programlama dillerinde birden fazla mantıksal operatör kullanılmak durumunda kalınabilir. Bu durumda bu mantıksal operatörlerin birleştirilerek sonuçta yine tek bir boolean (true veya false) değerinin üretilmesi gerekmektedir. İşte bu birleşimi sağlayan operatörlere koşullu mantıksal operatörler denir. Bu operatörler **&&** simgesi ile gösterilen "Mantıksal AND" ile **||** simgesi ile gösterilen "Mantıksal OR" operatörleridir.

&& (Mantıksal AND Operatörü): bağladığı iki boolean değerinin her ikisinin de "True" olması durumunda "True" değerini üretir. Bunun haricindeki bütün durumlarda "False" değerini üretecektir yani bu operatörün bağladığı boolean değerlerden sadece birinin "False" olması sonucu da "False" yapacaktır.

Örnek: Bir öğrencinin bir dersin sınavından alabileceği puan 0 ile 100 arasında olmalıdır. Dolayısıyla puan değeri mantıksal olarak şu şekilde tanımlanabilir.

puan >= 0 && puan <= 100 (puan 0'dan büyük veya 0'a eşit olmalıdır, aynı zamanda 100'den küçük veya 100'e eşit olmalıdır.)

Yukarıdaki ifade kodlamaya şu şekilde dökülebilir (*Bir öğrenci için belirlenen ders puanının 0 ile 100 arasında olup olmadığını gösteren kodlamadır.*).

```
static void Main(string[] args)
{
    bool PuanGecerliMi;
    int puan = 70;
    PuanGecerliMi = (puan >= 0) && (puan <= 100);
    Console.WriteLine(PuanGecerliMi);
}
```

Resim 1.3: Mantıksal AND operatörü

Bu programda *PuanGecerliMi* değişkeni puanın 0 ile 100 arasında olup olmadığını gösteren boolean değişkendir. Programa göre bu değişken “*True*” değerini alırsa (ekranda true yazarsa) puanın geçerli olduğunu, “*False*” değerini alırsa (ekranda false yazarsa) puanın geçerli olmadığını belirtiyor demektir.

Yukarıdaki programda puan değişkeninin değeri 70 olarak görülmektedir. *PuanGecerliMi* değişkeni, 70 puanının 0 ile 100 arasında olup olmadığı ekrana aşağıdaki resimde görüldüğü gibi yazılacaktır.



Resim 1.4: Mantıksal AND operatörü

Şimdi puan değişkeninin değeri 140 olarak değiştirilip çıktı ekranının nasıl değiştiği görülsün.

|| (Mantıksal OR Operatörü) ise birbirine bağladığı boolean değerlerinden birinin “*True*” olması sonucun “*True*” olması için yeterli olacaktır. Sonucun “*False*” olması için her iki boolean değerinin de “*False*” olması gerekmektedir.

Örneğin, “*Öğretmenimiz, Selim ya da Fatih’i yanına çağırdı.*” cümlesinde iki öğrenciden birinin öğretmenin yanına gitmesinin yeterli olacağı belirtilmektedir.

Örnek: Bir öğrencinin bir dersin sınavından alabileceği puan 0 ile 100 arasında olmalıdır. Dolayısıyla puan değeri mantıksal olarak şu şekilde tanımlanabilir (*Yukarıda AND operatörü ile yaptığımız örneği OR operatörü ile yapınız.*):

Puan < 0 || puan >100 (puan deęişkeninin deęerinin 0'dan küçük olması, puan deęerinin geçersiz olması için yeterlidir veya puan deęişkeninin deęerinin 100'den büyük olması puan deęerinin geçersiz olması için yeterlidir.)

Yukarıdaki ifade kodlamaya şu şekilde dökülebilir (Bir öğrenci için belirlenen ders puanının 0 ile 100 arasında olup olmadığını bulunuz).

```
static void Main(string[] args)
{
    bool PuanGecersizMi;
    int puan = 703;
    PuanGecersizMi = (puan < 0) || (puan > 100);
    Console.WriteLine(PuanGecersizMi);
}
```

Resim 1.5: OR operatörü

Programda “PuanGecersizMi” adında tanımlanan deęişken puanın geçerli bir puan olup olmadığı bildirecektir. Yukarıdaki kodlamada puan deęişkenine deęer olarak 703 atanmıştır. Program çalıştığında şöyle bir sorgulama geliştirecektir.

- 703 deęeri 0 ‘dan küçük mü? Hayır (False)
- 703 deęeri 100 den büyük mü? Evet (True)

Öyleyse sonuçta “True” olacaktır. Çünkü iki deęerden bir tanesinin “True” olması sonucu da “True” yapacaktır. Aşağıda çıktı ekranı görülebilir.



Resim 1.6: OR operatörü çıktı ekranı

1.1.3. Operatör Öncelięi ve Birleşim Özellięi

Gündelik hayatta olduęu gibi programlamada da işlem öncelięi diye bir kavram yer alır. Öyle ya işlem yaparken bilgisayarın rastgele bir yerden başlaması aynı verilerden oluşan aritmetiksel veya mantıksal işlemlerin dahi farklı olmasına sebep olacaktır. Bu nedenle matematikten bilinen “işlem öncelięi” kavramı burada da geçerlidir.

Aşağıdaki örnek dikkatle incelenmelidir.

$$islem = 3 + 5 * 9$$

Yukarıdaki işlemin iki farklı sonucu çıkarılabilir:

- Birinci Sonuç → $islem = 48$
- İkinci Sonuç → $islem = 72$

olarak bulunacaktır. Aslında işlem önceliği denilen olay uygulandığında tek bir sonuç bulunmuş olur. Yukarıdaki işlem, işlem önceliğine göre gerçekleştirecek olunursa (*aşağıdaki gibi bir yol izleyerek*) doğru sonuca ulaşılabilir.

- Öncelikle çarpma işlemi gerçekleştirilir. ($5 * 9 = 45$).
- Sonrada toplama işlemi yapılır. Çıkan sonuçla (45 ile), 3 toplanır,
- Doğru sonuç bulunmuş olur ($islem = 45 + 3 = 48$).

Aşağıdaki Tablo 1.3, operatörlerin tamamının işlem önceliğini göstermektedir. Üst kategoride yer alan operatör, alt kategorilerde bulunanlardan önceliklidir. Aynı kategorideki operatörlerin önceliği ise bu tablodaki birleşim alanında belirtilen yöndeki operatör şeklindedir.

Kategori	Operatör	Açıklama	Birleşim
Birincil	()	Parantez	Sol
	++	Artırma	
	--	Azaltma	
Çarpımsal	*	Çarpma	Sol
	/	Bölme	
	%	Kalanı Bulma	
Toplamsal	+	Toplama	Sol
	-	Çıkarma	
Karşılaştırma	<	Küçük	Sol
	<=	Küçük veya eşit	
	>	Büyük	
	>=	Büyük veya eşit	
Eşitlik	=	Eşit	Sol
	!=	Eşit değil	
Koşullu mantıksal	&&	Mantıksal AND	Sol
		Mantıksal OR	

Tablo 1.3: İşlem önceliği

UYGULAMA FAALİYETİ

Mantıksal operatörleri kullanınız.

İşlem Basamakları	Öneriler
<p>➤ Aşağıdaki işlem ile belirtilen formülle belirlenen sonucu bulunuz. ($a=16, b=12, c=10$) işlem = $a + c / (b - c) * 10$</p>	<p>➤ Nesne tabanlı programlama yazılımı programı size yardımcı olabilir.</p>
<p>➤ Aynı işlemi parantezleri kaldırarak gerçekleştiriniz.</p>	<p>➤ Önceki bulduğunuz sonuçla şimdi bulduğunuz sonucu karşılaştırınız(parantezlerin işlem önceliğine olan etkisi).</p>
<p>➤ Aynı işlemi çarpma (*) operatörü ile bölme (/) operatörünü yer değiştirerek gerçekleştiriniz.</p>	<p>➤ Çarpma ve bölme operatörlerinden önceliğin hangisi olduğunu anlamaya çalışınız.</p>
<p>➤ İşlem değerinin pozitif mi yoksa negatif mi olduğunu bulunuz.</p>	<p>➤ Koşullu mantıksal operatörleri kullanarak sonucu, doğruysa true, yanlışsa false olarak bulunuz.</p>

KONTROL LİSTESİ

Bu faaliyet kapsamında aşağıda listelenen davranışlardan kazandığınız beceriler için **Evet**, kazanamadıklarınız için **Hayır** kutucuklarına (X) işareti koyarak öğrendiklerinizi kontrol ediniz.

Değerlendirme Ölçütleri	Evet	Hayır
1. Mantıksal değişkenleri tanımladınız mı?		
2. Boolean operatörleri kullandınız mı?		
3. Eşitlik operatörlerini kullandınız mı?		
4. İlişkisel operatörleri kullandınız mı?		
5. Koşullu mantıksal operatörleri kullandınız mı?		
6. Operatör önceliği ve birleşim özelliğini özetlediniz mi?		

DEĞERLENDİRME

Değerlendirme sonunda “Hayır” şeklindeki cevaplarınızı bir daha gözden geçiriniz. Kendinizi yeterli görmüyorsanız öğrenme faaliyetini tekrar ediniz. Bütün cevaplarınız “Evet” ise “Ölçme ve Değerlendirme” ye geçiniz.

ÖLÇME VE DEĞERLENDİRME

Aşağıdaki soruları dikkatlice okuyunuz ve doğru seçeneği işaretleyiniz.

1. $islem = y + x * (y - z) / 2$

Yukarıdaki formülde ilk önce hangi işlem uygulanır?

- A) $y + x$ B) $x * y$ C) $y - z$ D) $z / 2$

2. $islem = y + x * (y - z) / 2$

Yukarıdaki formülde $x = 2$, $y = 10$ ve $z = 6$ ise “*islem*” değeri kaçtır?

- A) 4 B) 14 C) 24 D) 34

3. $islem = y + x * y - z / 2$

Yukarıdaki formülde $x = 2$, $y = 10$ ve $z = 6$ ise “*islem*” değeri kaç olarak değişecektir?
(NOT: Dikkat edilirse 2. sorudaki formülün parantezi kaldırılmış hâlidir.)

- A) 27 B) 17 C) 14 D) 7

4. Bir firma işe alacağı personelin tanımını şu şekilde yapmaktadır:

- Yaşı 26 – 33 aralığında (26 ve 33 hariç) olan evli erkek veya
- Yaşı 25 – 30 aralığında (25 yaş dahil, 30 yaş hariç) evli bayan

Buna göre bu koşulların birleşiminden “True” sonucunu elde etmek için aşağıdaki ifadede boşluklara gelmesi gerekenler hangi seçenekte doğru olarak verilmiştir?

Sonuc = (yas > 26 yas < 33 cinsiyet == “e”) (yas >= 25 yas < 30 cinsiyet == “k”) MedeniDurum = “evli”

(NOT: cinsiyet için “e” ya da “k”, medeni hâl için “evli” ya da “bekâr” ifadelerini kullanınız.)

- A) &&, &&, ||, &&, ||, ||
B) ||, ||, &&, ||, ||, ||
C) &&, &&, ||, &&, &&, ||
D) &&, &&, ||, &&, &&, &&

5. $(1=2 \ \&\& \ 5!=7) \ \&\& \ (“7” = 7 \ || \ “ali” != “salih”) \ || \ (7 < 12 \ \&\& \ 6 <= 8)$ ifadesinin açıklaması aşağıdakilerden hangisinde doğru verilmiştir.

- A) False && False || True → Sonuç False
B) True && True || True → Sonuç True
C) True && False || True → Sonuç True
D) True && False || False → Sonuç True

DEĞERLENDİRME

Cevaplarınızı cevap anahtarıyla karşılaştırınız. Yanlış cevap verdiğiniz ya da cevap verirken tereddüt ettiğiniz sorularla ilgili konuları faaliyete geri dönerek tekrarlayınız. Cevaplarınızın tümü doğru ise bir sonraki öğrenme faaliyetine geçiniz.

ÖĞRENME FAALİYETİ-2

AMAÇ

Şart ifadelerini kullanarak iki farklı kod bloku arasında seçim yapabilecek, bir koşulun farklı değerlerini değişik ifadelerle ilişkilendirebileceksiniz.

ARAŞTIRMA

- Gündelik hayatınızda bir şarta bağlı olarak yaptığınız eylemleri düşününüz ve üç adet örnek veriniz (hafta sonları yağışlı havalarda sinemaya gitmeyi, yağış olmazsa da halı sahada futbol oynamayı tercih ederim.).

2. ŞART İFADELERİ

Belki de gündelik hayatta en çok kullanılan yapılardan bir tanesidir. Bazen okulda öğretmenlerden “Yeterli şekilde ders çalışmazsanız sınıfı geçemezsiniz.” şeklindeki uyarılar, bazen evde anne ya da babadan “Kardeşine iyi davranırsan sana güzel bir bisiklet alacağım.” şeklindeki hatırlatmalarla karşılaşılır. Bu gibi durumların tamamında şart ifadeleri ile karşı karşıya kalınır.

Şart ifadeleri programın akışını yönlendiren güçlü bir yapıdır. Gündelik hayatta çıkan pek çok seçenekten bir tanesi seçilmek zorunda olduğunda iyi düşünüp ondan sonra karar verilir. Farklı yazım şekillerine rağmen programlama dili içinde farklı seçeneklerden en uygununu seçmek için kullanılan yapılardır.

2.1. IF İfadesi

Türkçe karşılığını “Eğer” olarak nitelendirilebilecek if ifadesi, en fazla iki seçenekli durumlar için kullanılan bir yapıdır. Koşulun sonucuna bağlı olarak iki farklı kod bloku arasında seçim yapmak istenildiğinde kullanılacak yapıdır.

- If ifadesinin söz dizimi şu şekildedir:

```
if (koşul)
    İfade - 1
Else
    İfade - 2
```

Yukarıdaki yazım kuralına göre koşul doğru (*true*) ise ifade - 1 işletilecek, koşul doğru değilse (*false*) ifade - 2 işletilecektir. Aşağıdaki örnek dikkatlice incelenmelidir.

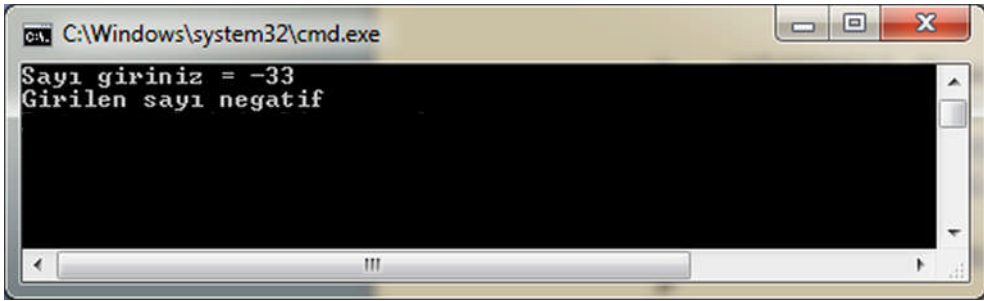
Örnek: Klavyeden girilen bir sayının pozitif bir sayı olup olmadığını bulan programı yazınız.

```
static void Main(string[] args)
{
    int sayi;
    Console.Write("Sayı giriniz = ");
    sayi = Convert.ToInt32(Console.ReadLine());
    if (sayi > 0)
        Console.WriteLine("Girilen sayı pozitif");
    else
        Console.WriteLine("Girilen sayı negatif");
}
```

Resim 2.1: if ifadesi

Yukarıdaki resimde (Resim 2.1) yer alan kodlama bu örnek için kullanılacak bir kodlamadır. Bu kodlamada *int* tipinde tanımlanan bir *sayi* değişkeni değerini klavyeden almaktadır. Böylece *sayi* değişkenine göre program yönlendirilmektedir.

Bu programdaki koşul, *sayi* değişkeninin değerinin 0 (sıfır)'dan büyük olmasıdır. Eğer *sayi* değişkeni 0 (sıfır)'dan büyükse koşulun sonucu *true* olacak ve ekrana, "Girilen sayı pozitif" yazacaktır. Aksi hâlde yani *sayi* değişkeninin değeri 0 (sıfır)'dan küçükse koşulun sonucu *false* olacak ve ekrana "Girilen sayı negatif" yazacaktır. Aşağıda her iki durum içinde ekran çıktıları incelenebilir.



Resim 2.2: if ifadesi çıktı ekranı (Koşul sonucu: true)



Resim 2.3: if ifadesi çıktı ekranı (koşul sonucu: false)

2.2. İç İçe If İfadesi

Bir programda tek bir if bloku kullanılabilceği gibi bir if ifadesinin içinde başka bir if ifadesi de kullanılabilir. Bu kullanımda sınır yoktur. Gereklikçe iç içe if ifadelerinin sayısı artabilir.

Örnek: Bir öğrencinin bir dersten almış olduğu ortalamaya göre öğrencinin dersten geçip geçmediği bulan programı şu kurallara göre kodlanır.

- Ortalama 45 ve üstüyse öğrenci dersten geçecektir.
- Ortalama 45'in altında ise öğrencinin sorumluluk sınav notu klavyeden girilecek ve bu not 45 ve üstüyse öğrenci dersten geçecek, bu not 45'in altında ise öğrenci dersten kalacaktır.

Bu açıklamalara göre kodlama şu şekilde gerçekleştirilebilir:

```
static void Main(string[] args)
{
    int ortalama, sorumluluk;
    Console.Write("Ders ortalamanızı giriniz = ");
    ortalama = Convert.ToInt32(Console.ReadLine());
    if (ortalama >= 45)
    {
        Console.WriteLine("Dersten geçtiniz...");
    }
    else
    {
        Console.WriteLine();
        Console.Write("Sorumluluk sınavı notunuz = ");
        sorumluluk = Convert.ToInt32(Console.ReadLine());
        if (sorumluluk >= 45)
            Console.WriteLine("Dersten sorumluluk sınav notu ile geçtiniz...");
        else
            Console.WriteLine("Dersten kaldınız...");
    }
}
```

Resim 2.4: İç içe if ifadesi kullanımı

Yukarıdaki Resim 2.4'te görülen gibi ilk olarak ortalama değişkeninin değeri klavyeden alınmaktadır. Daha sonra if yapısı ortalama değişkeninin değerine bakmaktadır. Bu değer eğer 45'ten büyük veya 45'e eşit ise ekrana "Dersten geçtiniz." mesajını yazacaktır. Else bölümünde ise koşulun *false* (ortalama değişkeninin değerinin 45'ten büyük olmaması) değerini alması ile bu kez de klavyeden sorumluluk değişkeni değeri girilmektedir.

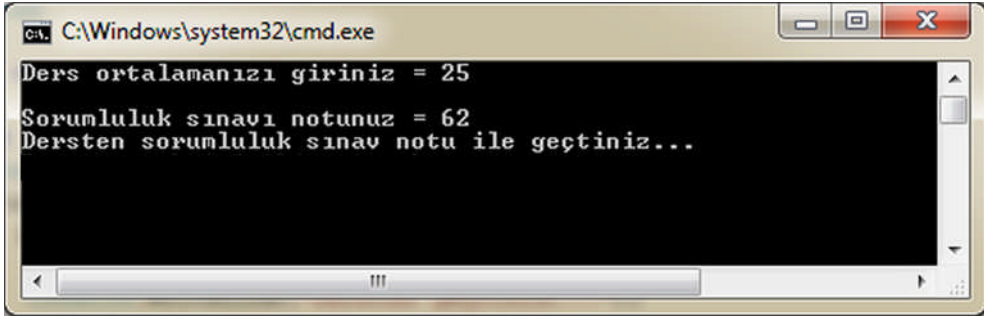
Sorumluluk değişkeninin değeri hemen sonra gelen if blokunda sorgulanmaktadır. Bu değer 45'ten büyük ya da 45'e eşit ise ekrana "Dersten sorumluluk sınav notu ile geçtiniz..."

yazmaktadır. Sorumluluk deęişkeninin deęeri 45'in altında ise ekrana "Dersten kaldınız..." mesajı yazılmaktadır.

Bu kodlamada aynı zamanda kodlar { } karakterleri kullanılarak gruplandırılmıştır. Yani ilk if ifadesinde yazılan kodlar { } karakterleri arasına yazılarak gruplandırılmış, else bölümünde yer alan if blokundaki kodlamalarda ise böyle bir şey söz konusu olmamıştır. Bunun sebebi şudur: Eğer if blokunda yazılan kodlama birden fazla satırdan oluşuyorsa bu kodlama { } karakterleri arasına yazılır. Aksi hâlde tek satırdan oluşan kodlamada { } karakterlerini kullanmaya gerek yoktur.

Program çalıştırıldığında aşağıdaki ekran çıktıları elde edilebilir:

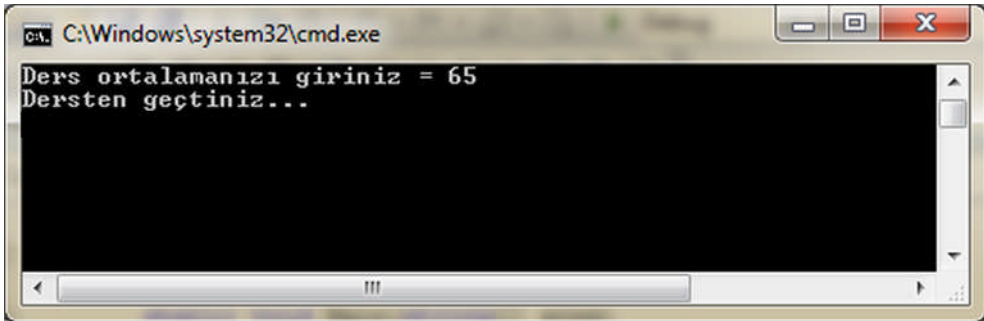
- Ders ortalaması 45'in altında ise Resim 2.5 ekran çıktısı olacaktır.



```
C:\Windows\system32\cmd.exe
Ders ortalamanızı giriniz = 25
Sorumluluk sınavı notunuz = 62
Dersten sorumluluk sınav notu ile geçtiniz...
```

Resim 2.5: İç içe if ifadesi ekran çıktısı

- Ders ortalaması 45 ve üstüye Resim 2.6 ekran çıktısı olacaktır.



```
C:\Windows\system32\cmd.exe
Ders ortalamanızı giriniz = 65
Dersten geçtiniz...
```

Resim 2.6: İç içe if ifadesi ekran çıktısı

2.3. If-Else if İfadesi

İkiden fazla koşulun olduğu durumlarda kullanılan yapıdır. Böylece *boolean* deyimleri içlerinden bir tanesi *true* deęerini üretene kadar basamaklandırılabilir.

Yazım şekli aşağıdaki gibidir:

if (koşul-1)

Kodlamalar

else if (koşul-2)

Kodlamalar

else if (koşul-3)

Kodlamalar

Else

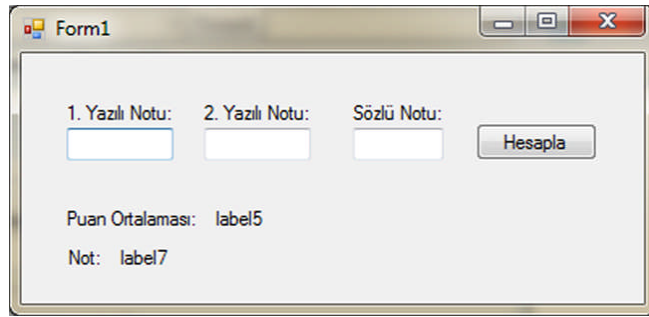
Kodlamalar

Yazım şeklinde de görüldüğü gibi herhangi bir koşul true değerini üretene kadar *if* yapıları birbirine bağlanabilir. Ayrıca en sonda *else* ifadesi ekrana çıkmaktadır. Hiçbir koşul *true* değerini üretmez ise buradaki kodlamalar işletilecektir.

Aşağıdaki örnek bu anlamda incelenebilir. Bu örnekte, iki sınav ve bir sözlü notu girilen öğrencinin ortalama puanı hesaplanarak bu puanın nota dönüştürülmesi anlatılmaktadır. Ortalamaya göre notlar şu şekilde olacaktır:

- Ortalama 0 – 24 arasında ise (24 dâhil) not = 0
- Ortalama 25 - 44 arasında ise (44 dâhil) not = 1
- Ortalama 45 – 54 arasında ise (54 dâhil) not = 2
- Ortalama 55 – 69 arasında ise (69 dâhil) not = 3
- Ortalama 70 – 84 arasında ise (84 dâhil) not = 4
- Ortalama 85 – 100 arasında ise not = 5

Bu örnekte kullanılacak form görüntüsü, Resim 2.7’de görüldüğü gibi oluşturulur.



The image shows a Windows application window titled "Form1". Inside the window, there are three text boxes for input, each preceded by a label: "1. Yazılı Notu:", "2. Yazılı Notu:", and "Sözlü Notu:". To the right of these input fields is a button labeled "Hesapla". Below the input fields, there are two more labels: "Puan Ortalaması: label5" and "Not: label7". The window has standard Windows window controls (minimize, maximize, close) in the top right corner.

Resim 2-7: Öğrenci not dönüşümü form görüntüsü

Resim 2.7’de ki formdaki kontroller şunlardır: 1.Yazılı Notu, 2.Yazılı Notu, Sözlü Notu, Puan Ortalaması ve Not Label kontrolleri; 1 ve 2. Yazılı notları ile sözlü notuna değer girmek için textbox kontrolü, Hesapla ise bir buton kontrolü kullanılmıştır.

“Hesapla” butonuna tıkladığında gerekli işlemler yapılarak *label5* kontrolünde puan ortalaması, *label7* kontrolünde ise not değeri görüntülenecektir. Her iki durumda “Hesapla” butonuna tıkladığında gerçekleşecektir. Bu anlamda bu iki durumlu program kodlaması iki aşamada gerçekleştirilebilir.

- İlk olarak butona tıklama olayında “1.Yazılı”, “2.Yazılı” ve “Sözlü” notları toplanıp üçe bölünerek ortalama bulunup yazılacaktır. Bu ilk durum için kodlama şu şekilde olacaktır:

```
//1. Durum: Ortalama hesaplanıyor
int y1, y2, s, ortalama, not;
y1 = Convert.ToInt16(textBox1.Text);
y2 = Convert.ToInt16(textBox2.Text);
s = Convert.ToInt16(textBox3.Text);
ortalama = (y1 + y2 + s) / 3;
label5.Text = Convert.ToString(ortalama);
```

Resim 2-8: Not dönüşüm program kodlaması

Resim 2.8’deki kodlamada gördüğümüz gibi *textbox1*, *textbox2* ve *textbox3* kontrollerine girilen veriler *Int16* veri türüne dönüştürülerek ilgili değişkenlere aktarılmakta ve daha sonrada işlemlerle *ortalama* değeri bulunmaktadır. Bulunan *ortalama* değeri *label5* kontrolünde görüntülenirken *string* türüne dönüştürülmektedir.

- İkinci olarak da bu ortalama göre dersi geçme notu, *else if* yapısıyla bulunacaktır. Buna göre program kodlaması şu şekilde olacaktır:

```

//2. Durum: Not dönüşüm işlemi yapılıyor
if (ortalama >= 0 && ortalama <= 24)
    not = 0;
else if (ortalama >= 25 && ortalama <= 44)
    not = 1;
else if (ortalama >= 45 && ortalama <= 54)
    not = 2;
else if (ortalama >= 55 && ortalama <= 69)
    not = 3;
else if (ortalama >= 70 && ortalama <= 84)
    not = 4;
else
    not = 5;
label7.Text = not.ToString();

```

Resim 2.9: Not dönüşüm program kodlaması

Resim 2.9’da görünen kodlamada altı adet koşul yer almaktadır. Bu koşullardan sadece bir tanesi *true* değerini üretecektir. Kodlamada aynı zamanda puan aralık sorgulaması yapıldığı için && (AND Operatörü) kullanılmıştır. Programın çalışması şu şekilde olacaktır: Hesaplanan ortalama değeri hangi aralıkta ise o aralık için belirtilen boolean ifade *true* değerini alacak ve o aralık için belirlenen değer *not* değişkenine aktarılacaktır. Programı çalıştırarak çıktı ekranı aşağıdaki Resim 2.10’da görülebilir.

The screenshot shows a Windows application window titled "Form1". It contains three text input fields for "1. Yazılı Notu" (containing 48), "2. Yazılı Notu" (containing 81), and "Sözlü Notu" (containing 70). To the right of these fields is a button labeled "Hesapla". Below the input fields, the application displays the calculated results: "Puan Ortalaması: 66" and "Not: 3".

Resim 2-10: Not dönüşüm programı çıktı ekranı

Resim 2.10’da görüldüğü gibi “1. Yazılı Notu” 48, “2. Yazılı Notu” 81 ve “Sözlü Notu” da 70 olarak girilmiştir. Buna karşılık gelen puan ortalaması 66 olarak hesaplanmıştır. İkinci duruma göre bakıldığında 66 puanının hangi aralıkta olduğu bulunacaktır. 55 – 69 aralığında olduğu görülmektedir. Buradaki not değeri de 3 olarak atanmıştır.

2.4. Switch İfadesi

Else – if ifadesinin program içerisinde ikiden fazla koşul var ise kullanılan bir yapı olduğunu daha önce söylenmişti. Bununla beraber *else – if* yapısında koşul, sürekli olarak elde edilen değer yeni değerlerle kıyaslanmaktadır. *Switch* ifadesinde ise elde edilen değer yeni değerlerle kıyaslanmak yerine eşleştiği değere göre kodları işletmektedir. Bu durum aynı zamanda programın okunabilirliğine de katkı yapmaktadır.

Switch ifadesinin yazım kuralı aşağıdaki şekildedir:

```
switch (kontrol-ifadesi)
{
    case durum-ifadesi-1:
        kodlamalar;
        break;
    case durum-ifadesi-2:
        kodlamalar;
        break;
    ...
    Default;
        kodlamalar
        break();
}
```

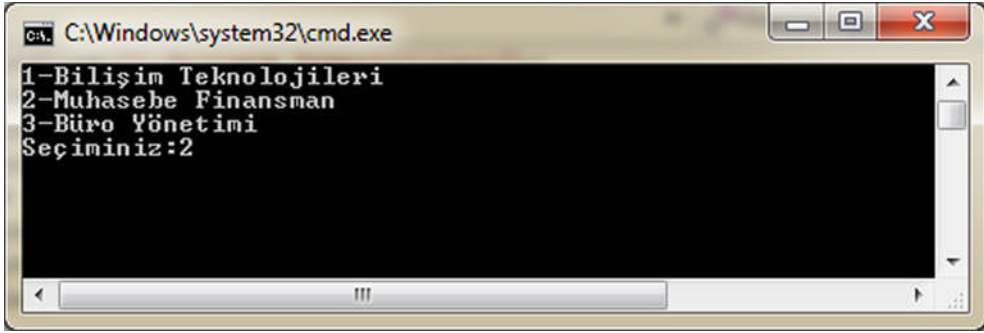
Burada *kontrol-ifadesi* bir kez değerlendirilir ve eşit olduğu “*durum-ifadesi*”nin yer aldığı *case* bloku çalıştırılır. Break ifadesine kadar olan kodlar çalıştırılır. Bu noktada *switch bloku* sonlanarak program *switch* ifadesinden sonraki satırdan itibaren devam eder. Kontrol ifadesi hiçbir durum ifadesine eşit değilse *Default* bloku çalıştırılır. Ancak *default bloku* isteğe bağlıdır. Bu blok olmadığında program kod blokunu çalıştırmadan *switch blokundan* çıkar.

Örnek: Mesleki okulda bulunan alanlara ait sınıfları görüntüleyen program aşağıdaki yönergeye göre yapılsın.

- Program ilk çalıştırıldığında ekrana aşağıdaki gibi numaralandırılmış bölüm listesi gelecektir (Resim 2.11).

Bölümler:

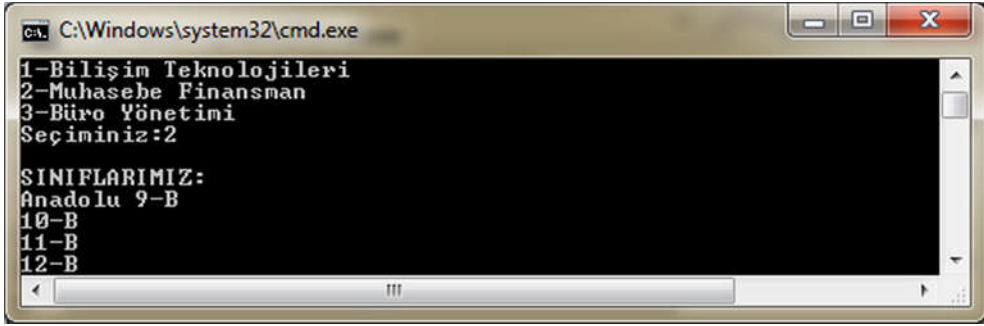
1. Bilişim Teknolojileri
Sınıflar: Anadolu 9-A, Anadolu 10-A, Anadolu 11-A, 10-A, 11-A, 12-A
2. Muhasebe Finansman
Sınıflar: Anadolu 9-B, Anadolu 10-B, 10-B, 11-B, 12-B
3. Büro Yönetimi
Sınıflar: 10-C, 11-C, 12-C



```
C:\Windows\system32\cmd.exe
1-Bilişim Teknolojileri
2-Muhasebe Finansman
3-Büro Yönetimi
Seçiminiz:2
```

Resim 2-11: Program çalışma ekranı

- İlgili bölüm numarası seçildiğinde (1, 2, 3) o bölüme ait sınıflar alt alta listelenecektir (Resim 2.12).



```
C:\Windows\system32\cmd.exe
1-Bilişim Teknolojileri
2-Muhasebe Finansman
3-Büro Yönetimi
Seçiminiz:2
SINIFLARIMIZ:
Anadolu 9-B
10-B
11-B
12-B
```

Resim 2.12: Program çalışma ekranı

Kodlama bu yönergelere göre yapılacak olursa aşağıdaki gibi bir kodlama ortaya çıkacaktır.

```

static void Main(string[] args)
{
    Console.WriteLine("1-Bilişim Teknolojileri");
    Console.WriteLine("2-Muhasebe Finansman");
    Console.WriteLine("3-Büro Yönetimi");
    Console.Write("Seçiminiz:");
    int bolum = Convert.ToInt16(Console.ReadLine());
    Console.WriteLine();
    Console.WriteLine("SINIFLARIMIZ:");
    switch (bolum)
    {
        case 1: //bilişim teknolojileri
            Console.WriteLine("Anadolu 9-A"); Console.WriteLine("Anadolu 10-A");
            Console.WriteLine("Anadolu 11-A");
            Console.WriteLine("10-A"); Console.WriteLine("11-A");
            Console.WriteLine("12-A");
            break;
        case 2: //muhasabe finansman
            Console.WriteLine("Anadolu 9-B"); Console.WriteLine("10-B");
            Console.WriteLine("11-B"); Console.WriteLine("12-B");
            break;
        case 3: //büro yönetimi
            Console.WriteLine("10-C"); Console.WriteLine("11-C");
            Console.WriteLine("12-C");
            break;
    }
}

```

Resim 2.13: Switch program kodlaması

Resim 2.13'te ki kodlamada ilk olarak bölümlerin listesi ekrana getirilmiştir.

int bolum = Convert.ToInt16(Console.ReadLine())

satırıyla da klavyeden bölüm bilgisi alınıyor. *bolum* değişkeni aynı zamanda *switch* ifadesindeki kontrol deyimidir ve bu yapıdaki durumlardan *bolum* değişkenine eşit olan durum ifadesinin bulunduğu kod bloku çalıştırılıyor. Bu kod blokunda da bölüme ait sınıf listesi alt alta ekrana yazdırılıyor.

Switch ifadesini programlarda kullanmak birçok avantaj sağlamaktadır. Ancak bu ifade her durumda kullanılamamaktadır. Bu ifadeyi kullanabilmek için bir takım şartların yerine gelmesi gerekmektedir. Aşağıda bulunabilecek bu şartlar hiçbir zaman dikkatten çıkarılmaz.

Switch ifadesi kullanırken dikkat edilecek noktalar:

- *Switch* ifadesi yalnızca *string* veya *int* veri türleriyle kullanılabilir.
- *case* ifadesinde kullanılan *durum değerleri* sabit değerler olmalıdır. Örneğin, case 1, case "sınıf", ...vb

- Farklı durumlar için aynı kodlamalar çalıştırılabilir. Örneğin, aşağıdaki Resim 2.14'te yer alan kodlamalar incelenmelidir.

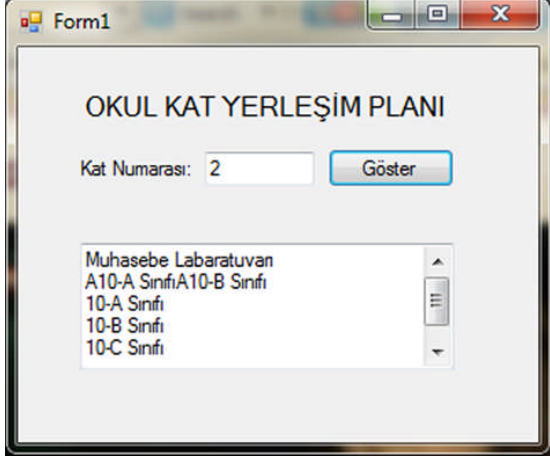
```
private void button1_Click(object sender, EventArgs e)
{
    string bolum = textBox1.Text;
    int burs = 100;
    switch (bolum)
    {
        case "muhasabe":
        case "büro":
            burs += 150;
            break;
        case "bilisim":
            burs += 200;
            break;
        default:
            burs += 50;
            break;
    }
    label3.Text = "Burs Miktarı: " + burs + " TL";
}
```

Resim 2.14: Farklı durumlar için aynı kodlama

Burada mesleki eğitim kurumunda okuyan bir öğrencinin bölümüne göre alacağı burs miktarı hesaplanmaktadır. Burada dikkat edilecek nokta hem muhasebe hem de büro alanlarında okuyan öğrencilerin aynı burs miktarını alacağı anlaşılmaktadır. Çünkü muhasebe durumu için herhangi bir kodlama yapılmamıştır. Dolayısıyla hemen devamında bulunan durumun kodlaması bu durum için geçerli olmaktadır.

UYGULAMA FAALİYETİ

Şart ifadelerini kullanarak programın işleyişini yönlendiriniz.

İşlem Basamakları	Öneriler
<ul style="list-style-type: none">➤ Bir okulun kat yerleşim planını gösteren kodlamayı else if yapısını kullanarak yapınız.	<ul style="list-style-type: none">➤ Nesne tabanlı programlama yazılımı size yardımcı olabilir.
<ul style="list-style-type: none">➤ Kullanıcı görmek istediği kat numarasını klavyeden girecek ve o katta nelerin bulunduğu dair listeyi alabilecektir.	<ul style="list-style-type: none">➤ Ekran görüntüsü şu şekilde olabilir. ➤ Resim 2.15: Program ekran görüntüsü
<ul style="list-style-type: none">➤ Okul zemin kat hariç üç kattan oluşmaktadır.	<ul style="list-style-type: none">➤ Zemin kat için 0, 1, 2 ve 3 verilerini kullanabilirsiniz.
<ul style="list-style-type: none">➤ Kullanıcı var olmayan bir kat girdiğinde uyarılmalıdır.	<ul style="list-style-type: none">➤ Uyarı “Lütfen 0-3 arasında bir kat numarası giriniz...” şeklinde olabilir.
<ul style="list-style-type: none">➤ Aynı uygulamayı switch yapısını kullanarak da gerçekleştiriniz.	<ul style="list-style-type: none">➤ Her iki kod ekranını veya programın çalışmasını karşılaştırınız.

KONTROL LİSTESİ

Bu faaliyet kapsamında aşağıda listelenen davranışlardan kazandığınız beceriler için **Evet**, kazanamadığınız beceriler için **Hayır** kutucuğuna (X) işareti koyarak kendinizi değerlendiriniz.

Değerlendirme Ölçütleri	Evet	Hayır
1. if ifadesinde kullanacağınız koşul değişkenini uygun şekilde tanımladınız mı?		
2. if yapısının şart ifadesini doğru bir şekilde yazdınız mı?		
3. if ifadesini kullandınız mı?		
4. İfadeleri gruplamak için bloklar kullandınız mı?		
5. if ifadelerini basamakladınız mı?		
6. Switch ifadesini kullandınız mı?		
7. Switch ifadesinde kullanacağınız kontrol değişkenini uygun bir şekilde tanımladınız mı?		
8. Switch ifadesindeki her bir durum ifadesini belirttiniz mi?		

DEĞERLENDİRME

Değerlendirme sonunda “**Hayır**” şeklindeki cevaplarınızı bir daha gözden geçiriniz. Kendinizi yeterli görmüyorsanız öğrenme faaliyetini tekrar ediniz. Bütün cevaplarınız “**Evet**” ise “Ölçme ve Değerlendirme”ye geçiniz.

ÖLÇME VE DEĞERLENDİRME

Aşağıdaki soruları dikkatlice okuyunuz ve doğru seçeneği işaretleyiniz.

```
private void button1_Click(object sender, EventArgs e)
{
    if (textBox1.Text == "") textBox1.Text = "0";
    if (textBox2.Text == "") textBox2.Text = "0";
    int s1 = Convert.ToInt32(textBox1.Text);
    int s2 = Convert.ToInt32(textBox2.Text);
    if (s1 < 0) s1 = s1 * -1;
    if (s2 < 0) s2 = s2 * -1;
    int sonuc = s1 + s2;
    label3.Text = "Sonuç= " + sonuc.ToString();
}
```

Resim 2.16: Program kodları

1'den 4.soruya kadar olan soruları yukarıdaki Resim 2.16'da yer alan kodlamaya göre cevaplayınız.

1. *textBox2* kontrolüne girilen değer -25 ve *textBox1* kontrolü de boş bırakılıp *button1* kontrolüne tıklanırsa *sonuc* değişkeninin son değeri aşağıdakilerden hangi olur?
A) 25 B) -25 C) 26 D) 24
2. Resim 2.16'da yer alan kodlamaya göre *if (s1 < 0) s1 *= -1;* satırı ile yapılmak istenen aşağıdaki seçeneklerin hangisinde doğru olarak ifade edilmiştir?
A) *textBox1* kontrolüne girilen değeri, sayısal değere dönüştürür.
B) *textBox1* kontrolüne veri girilmemişse -1 girilmiş gibi değerlendirir.
C) *textBox1* kontrolüne girilen veriden 1 eksiltir.
D) *textBox1* kontrolüne girilen değeri pozitif değere dönüştürür.
3. Yukarıdaki Resim 2.16'da yer alan kodlamayla ilgili aşağıdakilerden hangi veya hangileri yanlıştır?
I. *textBox* kontrollerine veri girilmediğinde program hata vermez.
II. Kodlamada mantıksal operatör kullanılmamıştır.
III. *sonuc* değişkeninin değeri *textBox1* ile *textBox2* kontrollerinin toplamıdır.
A) Yalnız I B) Yalnız II C) Yalnız III D) II ve III
4. Switch yapısında her bir durum hangi komutla ifade edilir?
A) case B) break C) default D) else

5. Aşağıdaki veri türlerinden hangi veya hangileri ile tanımlanan bir değişken *switch* ifadesinde kontrol deyimi olarak kullanılabilir?

- I. int
- II. float
- III. string

A) Yalnız I B) I ve II C) I ve III D) II ve III

Aşağıdaki cümlelerin başında boş bırakılan parantezlere, cümlelerde verilen bilgiler doğru ise D, yanlış ise Y yazınız.

6. () Switch ifadesinde case ile belirtilen durumlardaki kodlamaları sonlandıran komut “*break*”tir.
7. () Switch ifadesindeki kontrol deyimi (*switch(kontrol-deyimi)*), durum ifadelerinden (case)hiçbiri ile uyuşmuyorsa *else* bölümündeki kodlar çalışır.
8. () Resim 2-16’daki kodlamaya *textBox2* kontrolü boş bırakılıp *button1* kontrolüne tıklanırsa *s2* değişkeninin değeri 0 (sıfır) olur.

DEĞERLENDİRME

Cevaplarınızı cevap anahtarıyla karşılaştırınız. Yanlış cevap verdiğiniz ya da cevap verirken tereddüt ettiğiniz sorularla ilgili konuları faaliyete geri dönerek tekrarlayınız. Cevaplarınızın tümü doğru ise bir sonraki öğrenme faaliyetine geçiniz.

ÖĞRENME FAALİYETİ-3

AMAÇ

Döngü ifadeleri ile tekrarlı işlemler yapabileceksiniz.

ARAŞTIRMA

- Gündelik yaşantımızda bir şart gerçekleşene kadar tekrarlanan işlemler vardır. Örneğin, sabahleyin kapat düğmesine basılıncaya kadar telefonun alarmı çalar. Gece belirli saate kadar düzenli aralıklarla otobüs seferleri vardır. Cep telefonunda “*Cevap Yok*” mesajı gelene kadar belirli sayıda telefon çalar. Siz de bir şart gerçekleşene kadar belirli sayıda tekrar eden durumlara örnekler bulunuz. Araştırma sonuçlarınızı öğretmene teslim edecek veya sınıfta sunacak şekilde hazırlayınız.

3. DÖNGÜ YAPILARI

Okulda öğretmenlerin “Bu soruyu yapana kadar sınıftan ayrılmak yok.” şeklinde pek çok kez uyarılarıyla karşılaşmıştır. Sürekli soruya odaklanmaya çalışır, doğru cevap bulunamadığında tekrar başa döner yeniden yapılmaya çalışılır. İşte burada döngüsel bir durum oluşmuş olur. Sürekli başa dönmek doğru cevabı bulana kadar devam eder.

Öğrenme Faaliyeti – 2’de bahsi geçen, “*Şartlı İfadeler*” başlıklı bölümde bir şarta bağlı olarak işletilen kodlama yapısından bahsedilmişti. Döngüsel ifadelerde de bir şarta bağlılık vardır. Ama burada kodların işletilmesi süreklilik arz etmektedir.

Ayrıca bundan böyle kullanılacak kodlamalarda yapılacak işlemlerin daha kısa bir şekilde yapılması için bu bölümde yeni operatörler öğrenilecektir. Bu operatörler, “Bileşik Atama Operatörleri” olarak adlandırılmaktadır

3.1. Bileşik Atama Operatörleri

Aritmetiksel operatörlerle (+, -, /, *) atama operatörünün (=) birleştirilmesi ile oluşan operatörlerdir. Aşağıdaki toplama işlemine bir göz atılmalıdır.

$$\text{toplam} = \text{toplam} + 36$$

Burada toplam değerine 36 ilave edilmiştir. Görüldüğü gibi toplama (+) operatörü ile atama operatörü (=) bu sonucu doğurmuştur. Yukarıdaki ifade çalışıyor olmakla birlikte deneyimli bir programcı asla böyle ifadeler kullanmamaktadır. Bunun yerine, bu ifadeyi

daha da kısaltan += şeklinde *bileşik atama operatörünü* kullanmaktadır. Bu ifade gibi kodlamada bir değere başka değerler ekleme, çıkarma, ...vb. durumlarla o kadar çok karşılaşılacak ki *bileşik atama operatörleri* yardımıyla bu işlemler çok daha kısa yoldan yapılabilecektir.

Aşağıdaki tabloda bileşik atama operatörleri bulunabilir.

İşlem	Basit Atama	Bileşik Atama
Toplama	değer = değer + sayı	değer += sayı
Çıkarma	değer = değer - sayı	değer -= sayı
Çarpma	değer = değer * sayı	değer *= sayı
Bölme	değer = değer / sayı	değer /= sayı
Kalanı bulma	değer = değer % sayı	değer %= sayı
1 artırma	değer = değer + 1	değer ++
1 azaltma	değer = değer - 1	değer --

Tablo 3-1: Bileşik atama operatörleri

Örnek: Aşağıdaki Resim 3.1’de görünen form hazırlanır. Formda görüldüğü gibi iki adet *textbox* kontrolü mevcuttur: “Sayı” ve “Miktar”. Burada yapılacak şudur: “Miktar” olarak belirlenen *textbox* kontrolündeki sayı kadar “Sayı” olarak belirlenen *textbox* kontrolündeki değeri artırmak veya azaltmaktır. “Artır” butonuna tıklandığında “Sayı”, “Miktar” değerine göre artacak, “Azalt” butonuna tıklandığında “Sayı”, “Miktar” değerine göre azalacaktır.

Resim 3.1: Form Görüntüsü

Kodlama şu şekilde hazırlanmalıdır:

```

private void button1_Click(object sender, EventArgs e)
{
    //Artır butonu
    int sayi = Convert.ToInt32(textBox1.Text);
    int miktar = Convert.ToInt16(textBox2.Text);
    sayi += miktar;
    textBox1.Text = sayi.ToString();
}

private void button2_Click(object sender, EventArgs e)
{
    //Azalt butonu
    int sayi = Convert.ToInt32(textBox1.Text);
    int miktar = Convert.ToInt16(textBox2.Text);
    sayi -= miktar;
    textBox1.Text = sayi.ToString();
}

```

Resim 3.2: Kodlama ekranı

Kodlamada Artır butonu için button1_Click, azalt butonu için button2_Click olay yordamları kullanılmıştır. Bu yordamlarda da görüldüğü gibi

artırma işlemi için sayi += miktar
azaltma işlemi için sayi -= miktar

ifadeleri kullanılmıştır.

Aynı örnek yapılarak bileşik atama operatörlerinin kullanımına ilişkin uygulama gerçekleştirilebilir.

3.2. While İfadeleri

Bir koşula bağlı olarak (koşul sonucu true olduğu sürece) bir takım kodlamaları tekrar tekrar çalıştırmak için kullanılan döngüsel yapıdır. While yapısının söz dizimi şu şekildedir:

```

while (koşul)
{
    Kodlamalar;
    Denetim değişkeni güncellemesi (sayaç);
}

```

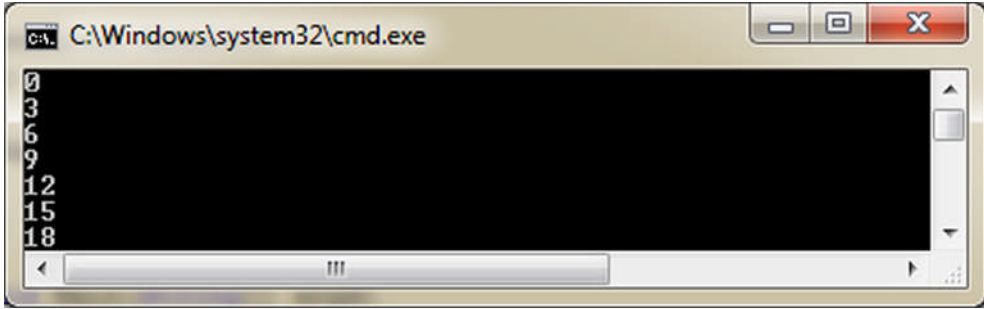
Koşul değerlendirilir ve sonucu *true* ise döngü içerisindeki kodlar çalıştırılır. Döngü içindeki son kodlamadan sonra tekrar koşul sorgulanır. Koşul sonucu yine *true* ise tekrar döngü içerisindeki kodlar çalıştırılır. Koşul sonucu *false* olana kadar döngü içerisindeki kodlar çalıştırılır. Sonuç *false* olduğunda ise *while* döngüsünden hemen sonraki komuttan itibaren program devam eder.

Söz dizimindeki *Denetim değişkeni güncellemesi* (bu ifade aynı zamanda *sayaç olarak da adlandırılmaktadır*. Çünkü döngünün ilerlemesi için sürekli bir sayım yapılmaktadır) satırı, döngünün sonsuza kadar tekrar etmesini engelleyen bir ifadeden oluşmaktadır. Düşünüldüğünde programlar içerisindeki hiçbir döngü sonsuza kadar çalışmamalıdır. Bir şekilde söz diziminde ifade edilen koşul sonucu *false* olacak şekilde ayarlanmalı ve döngü sonlanmalıdır. Aksi hâlde programda mantıksal bir hata oluşacaktır. İşte bu işlemi yapmak için bu satır vazgeçilmez olacaktır. *While* yapısında, döngüyü sonlandırmak için kullanılan denetim değişkeni, döngü içerisinde güncellenir.

Aslında bakıldığında bu döngüdeki koşul ile *if* yapısındaki koşulun kullanımı benzerlik göstermektedir. Ancak yine de bir takım yazım farklılıkları vardır. Aşağıda *while* döngüsündeki koşul yazılırken dikkat edilmesi gerekenler bulunabilir.

- *Koşul*, bir *boolean* ifadeden oluşmalıdır.
- *Koşul*, her zaman parantez içerisinde yazılmalıdır.
- *Koşuldaki boolean* deyiminin sonucu ilk esnada *false* olursa döngüdeki kodlar işlemez.

Örnek: Ekranı aşağıdaki Resim 3.3'te görüldüğü gibi alt alta 1 ile 20 arasındaki 3'ün katı olan sayıları yazdıran kodlama yapılmalıdır.



Resim 3.3: Çıktı ekranı

Döngü kullanılmazsa yapılacak kodlama şu şekilde olacaktır:

```
static void Main(string[] args)
{
    Console.WriteLine(0);
    Console.WriteLine(3);
    Console.WriteLine(6);
    Console.WriteLine(9);
    Console.WriteLine(12);
    Console.WriteLine(15);
    Console.WriteLine(18);
}
```

Resim 3.4: Programlama mantığına aykırı kodlama

Aynı soru *while* döngüsü ile yapıldığında aşağıdaki kodlama (Resim 3.5) için yeterli olacaktır.

```
static void Main(string[] args)
{
    int sayi = 0;
    while (sayi<20)
    {
        Console.WriteLine(sayi);
        sayi += 3;
    }
}
```

Resim 3.5: Döngülerle yapılan kodlama

Kodlama Resim 3.4'te görüldüğü gibi yapılırsa aynı komut tekrar tekrar yazılmak durumunda kalınır. Yine de burada toplamda yedi satır yazıldı. Çünkü 20 ile sınırlandırılmış bir soruydu bu. Aynı soru 20 değil de 100 deseydi kaç satır yazılmak zorunda kalınacaktı? Elbette ki daha fazla. İşte bu gibi durumların önüne geçmek için döngüler kullanılmalıdır.

Aynı soru Resim 3.5'te görüldüğü gibi döngüyle yapılsın. Aslında döngülerle işin ne kadar kolay olduğu görülebilir. *while* döngüsünde kullanılan koşul, *sayi* değişkeninin değerinin 20'den küçük olmasıdır. *sayi* değişkeninin değeri, 20'den küçük olduğu sürece döngü içerisindeki komutlar işletilecektir.

Programın çalışması şu şekildedir: *sayi* değişkeninin ilk değeri 0 (sıfır)'dır. Program akışı *while* kısmına geldiğinde koşula bakacaktır. Koşula göre *sayi* değişkeni, 20'den küçük olduğu için döngüye girilecektir. İlk olarak *sayi* değişkeni ekrana yazdırılacak ve daha sonra değeri 3 artırılabilecektir. Böylece *sayi* değişkeninin son değeri 3 olacaktır. Döngü sonunda tekrar koşul sorgulanacak ve koşul *true* değeri ürettiği için (*sayi* < 20 olduğu için) tekrar döngüdeki komutlar sırasıyla işletilecektir. Bu işlemler son değere kadar devam edip gidecektir. *sayi* değişkeninin son değeri (Resim 3.3'te görüldüğü gibi) 18'dir. *sayi* değişkeni 18 değeri ile tekrar ekrana yazdırılacak ve hemen sonrasında değeri yine 3 arttırılacaktır. Böylece *sayi* değişkeninin son değeri 21 olacaktır. Döngü sonuna gelindiğinden tekrar koşul sorgulanacak ve koşul *false* değerini ürettiği için (*sayi* < 20 olmadığı için) döngü sonlanacaktır.

3.3. For İfadeleri

Tıpkı *while* gibi bir döngü yapısıdır. *while* döngüsünden en belirgin farkı, döngünün başlama, bitiş ve denetim değişkeni güncellemesinin döngü başında yapılmasıdır. Bu nedenle de döngünün başlangıç ve bitiş değeri belli ise genelde bu döngü kullanılmaktadır. *for* ifadesinin söz dizimi şu şekildedir:

```
for (başlangıç değeri; bitiş değeri (koşul); denetim değışkeni güncellemesi)
{
    kodlamalar;
}
```

Öyleyse Resim 3.5 'te *while* döngüsü ile yapılan örnek bir de aşağıdaki Resim 3.6 'da görüldüğü şekilde *for* döngüsü ile kodlanmalıdır.

```
static void Main(string[] args)
{
    for (int sayi = 0; sayi < 20; sayi += 3)
    {
        Console.WriteLine(sayi);
    }
}
```

Resim 3.6: For döngüsü örnek kodlama

Kodlamada (Resim 3.6) görüldüğü gibi *while* döngüsüyle (Resim 3.5) çok fazla fark olmadığı gibi başlangıç ve bitiş değerleri belli olan verilerde *for* döngüsünün kullanılması daha mantıklıdır.

Yukarıda da detaylıca değinildiği gibi(gerek *while* gerekse de *for* döngüsünde) döngünün devam edip etmeyeceği kullanılan koşul (boolean ifade) belirlemektedir. Eğer döngüde koşul kullanılmazsa ya da yanlış kullanılırsa bu durumda döngü sonsuza kadar devam edecek ve program mantıksal bir hataya sebebiyet verecektir. Bu şekilde olan mantıksal hataya *sonsuz döngü* adı verilmektedir.

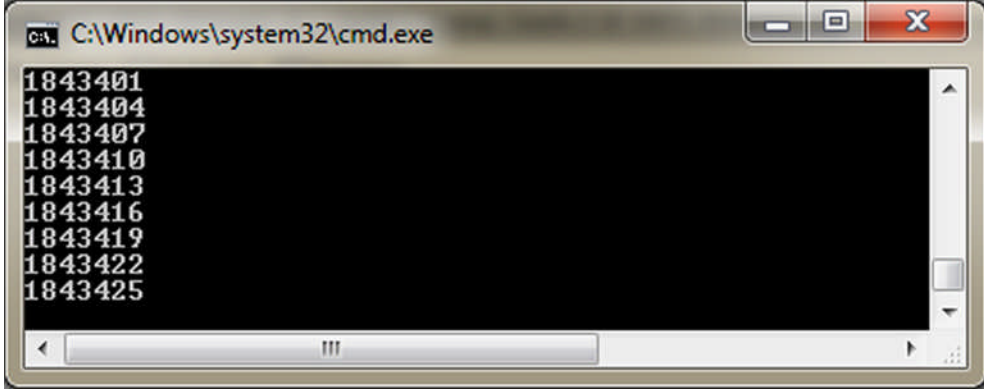
Örneğin kodunuzu Resim 3.6'da görüldüğü gibi değil, aşağıda Resim 3.7'de görüldüğü gibi yazılırsa döngü, sonsuz döngü olmuş olacaktır.

```
static void Main(string[] args)
{
    for (int sayi = 0; sayi != 20 ;sayi += 3)
    {
        Console.WriteLine(sayi);
    }
}
```

Resim 3.7: Sonsuz döngü kodlaması

Resim 3.7'de görünen kodlamada bitiş koşulu olan *sayi!=20* ifadesi, döngünün devam edip etmeyeceğini belirleyen ve *boolean* ifadeden oluşan koşul bölümüdür. Burada belirtilen koşul, *sayi* değışkeninin 20'ye eşit olmama durumudur. Yani *sayi* değışkeni 20'ye eşit olana dek döngü tekrar edecek kodlamada *sayi* değışkeninin yeni değerini ekrana yazacaktır.

Aslında koşul dikkatli incelendiğinde *sayi* değişkeni 0'dan itibaren üçer artarak devam etmektedir. Bu durumda *sayi* değişkeni hiçbir zaman 20'ye eşit olamayacağı için döngünün sonlanması gibi bir durum olmayacaktır. İşte bu durum sonsuz döngü olarak ekrana çıkmaktadır. Bu kodlama çalıştırıldığında sayıların sürekli yeniden güncellendiği aşağıdaki görüntüye (Resim 3.8) benzer bir ekranla karşılaşılacaktır.



Resim 3.8: Sonsuz döngü çıktı ekranı

3.4. Do İfadeleri

While ve *for* ifadelerinin her ikisinde de *koşul* (*boolean ifade*) döngünün hemen başında sorgulanmaktadır. Dolayısıyla belirtilen koşul, *true* değerini üretmezse döngüdeki kodlar işletilmeyecektir. Oysa *do* ifadesinde durum farklıdır. *do* ifadesinde *koşul* (*boolean ifade*) döngünün sonunda verilmektedir. Bu durum döngünün en az bir defa mutlaka işletileceği anlamına gelmektedir.

do ifadesinin söz dizimi şu şekildedir:

```
do
{
    Kodlamalar;
}
While (koşul – boolean ifade)
```

Daha önce *while* (Resim 3.5 'de ki kodlama) ve *for* (Resim 3.6 'da ki kodlama) döngülerinde kullanılan "0 ile 20 arasındaki 3'ün katı olan sayıları ekrana yazan program" örneği bu kez de *do* ifadesi ile yapılmalıdır.

Kodlama aşağıdaki Resim 3.9' da görüldüğü gibi olacaktır.

```

static void Main(string[] args)
{
    int sayi = 0;
    do
    {
        Console.WriteLine(sayi);
        sayi += 3;
    }
    while (sayi < 20);
}

```

Resim 3.9: do ifade ile kodlama

Resim 3.9'daki kodlamada görüldüğü gibi *while* döngüsünden tek farkı *koşulun* (*boolean ifade*) döngü sonunda verilmesidir. Kodlamalarda hangi yapı gerekli ise onu tercih edilmelidir. Eğer döngünün mutlaka en az bir defa çalışması gerekiyorsa *do* ifadesini döngünün başlangıç ve bitiş değerleri belli ise *for* ifadesi tercih edilmelidir.

3.4.1. Break ve Continue İfadeleri

Döngülerde çok fazla tavsiye edilmemesine rağmen özel durumlarda ihtiyaç hâlinde kullanılabilir ifadelerdir. *Break* komutu, *switch* ifadesinde kodlamayı bulunduğu yerde bitirmek için yani kodlamayı kesmek için kullanılmıştır. Döngülerdeki kullanım alanı da bu şekildedir. *Break* komutu, döngüden çıkmak için kullanılan bir komuttur. İşleyiş, döngüden sonraki komuttan devam edecektir.

Aşağıdaki Resim 3.10'da yer alan kodlama incelenmelidir.

```

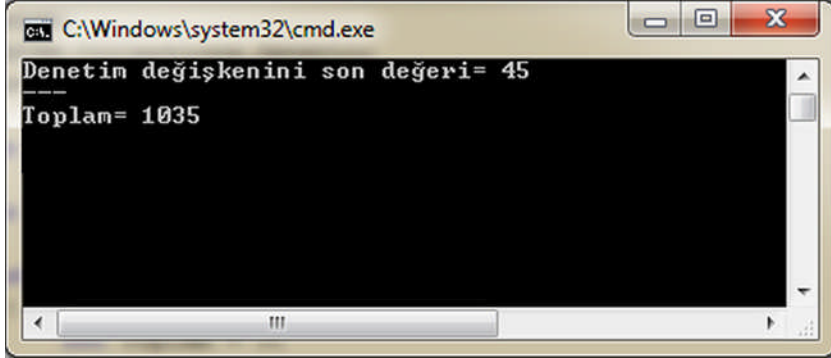
static void Main(string[] args)
{
    int toplam = 0;
    int sayi;
    for (sayi = 0; sayi <= 100; sayi++)
    {
        toplam += sayi;
        if (toplam > 1000) break;
    }
    Console.WriteLine("Denetim değişkenini son değeri= " + sayi);
    Console.WriteLine("----");
    Console.WriteLine("Toplam= " + toplam);
}

```

Resim 3.10: Break komutunun kullanılması

Kodlama 0 ile 100 arasındaki sayıların toplamını bulmak şeklinde tasarlanmıştır. Programın işleyişi şu şekildedir: *sayi* ve *toplam* değişkenlerinin ilk değeri 0 (sıfır) olarak alınmıştır. Koşul, *sayi* değişkeninin 100'den küçük veya eşit olmasıdır. Yani *sayi* değişkeni

101'den küçük olduğu sürece döngüdeki komutlar işletilecektir. Döngüdeki ilk komut *toplam* değişkenine *sayi* değişkeninin ilave edilmesidir(*toplam += sayi*). Daha sonraki satır olan if blokunda söylenmek istenen şudur: “Eğer *toplam* değişkeni 1000'den büyükse döngüyü sonlandır.” Aslında döngü 100'e kadar çalışacak şekilde kodlanmış olmasına rağmen bu satırı (*if bloku*) gördüğünde toplam değişkeninin değerine bakılacak ve bu değer 1000'den büyükse *denetim değişkeni* (*sayaç*) kaç olursa olsun döngüden çıkılacaktır. Program çalıştırıldığında aşağıdaki Resim 3.11'de görüldüğü gibi bir ekranla karşılaşılır.



Resim 3.11: Break komutunun kullanılması

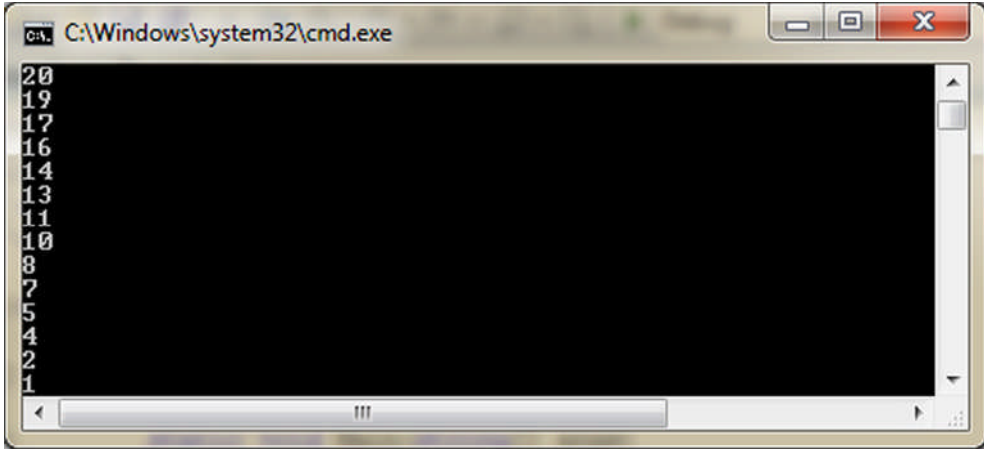
Resim 3.11'deki ekran çıktısında görüldüğü gibi denetim değişkeni, 100 değerini almadan (45 değerinde iken)döngü sonlanmıştır.

continue ifadesi ise döngüdeki denetim ifadesinin (*sayaç*) değerini bir sonraki konuma getirmek için kullanılır. Aşağıdaki örnek incelenmelidir.

```
static void Main(string[] args)
{
    for (int sayac = 20; sayac >= 0; sayac--)
    {
        if (sayac % 3 == 0) continue;
        Console.WriteLine(sayac);
    }
}
```

Resim 3.12: continue ifadesi kullanımı

Yukarıdaki Resim 3.12' de görülen kodlama; 3'ün katı olan sayılar hariç 0 ile 20 arasındaki sayıları ekrana yazdırmak için kullanılmaktadır. 3'ün katı olan sayıları hariç tutmak için *continue* ifadesi kullanılmıştır. Aşağıdaki Resim 3.13'te bu söylenen gibi olup olmadığı görülebilir.

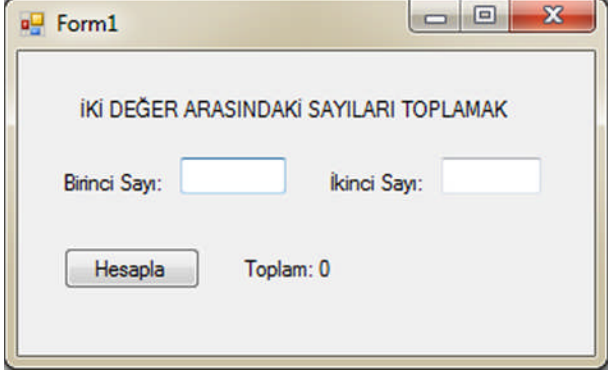


Resim 3.13: Continue ifadesi sonuç ekranı

Resim 3.13'te görüldüğü gibi 0 ile 20 arasında olup da (18, 15, 12, 9, 6, 3, 0) 3'ün katı olan hiçbir sayı çıktı ekranında görünmemektedir. Döngü içerisinde bir takım değerlerin işletilmemesi isteniyorsa *continue* ifadesi kullanılabilir.

UYGULAMA FAALİYETİ

Döngü ifadeleri ile tekrarlı işlemler yapınız.

İşlem Basamakları	Öneriler
<ul style="list-style-type: none">➤ Klavyeden girilen iki sayı arasındaki bütün sayıları toplayan kodlamayı <i>for</i> döngüsünü kullanarak yapınız.	<ul style="list-style-type: none">➤ Nesne tabanlı programlama yazılımı size yardımcı olabilir.
<ul style="list-style-type: none">➤ Programı şu yönergeye göre kodlayınız.➤ Kullanıcı “Birinci Sayı” bölümüne ilk sayıyı, “İkinci Sayı” bölümüne de ikinci bir sayı girecektir. “Hesapla” butonuna tıkladığında iki sayı arasındaki sayılar toplanarak sonucu “Toplam” bölümünde görüntülenecektir.	<ul style="list-style-type: none">➤ Ekran görüntüsü şu şekilde olabilir: 
<ul style="list-style-type: none">➤ Aynı uygulamayı <i>while</i> döngüsü kullanarak yapınız.	<ul style="list-style-type: none">➤ <i>for</i> döngüsünde kullandığımız denetim değişkenini (sayaç) kullanınız ve iki kodlama arasındaki farkları inceleyiniz.
<ul style="list-style-type: none">➤ Aynı uygulamayı <i>do</i> döngüsü kullanarak yapınız.	<ul style="list-style-type: none">➤ <i>for</i> ve <i>while</i> döngüsünde kullandığımız denetim değişkenini (sayaç) kullanınız ve iki kodlama arasındaki farkları inceleyiniz.

KONTROL LİSTESİ

Bu faaliyet kapsamında aşağıda listelenen davranışlardan kazandığınız beceriler için **Evet**, kazanamadıklarınız için **Hayır** kutucuklarına (X) işareti koyarak öğrendiklerinizi kontrol ediniz.

Değerlendirme ölçütleri	Evet	Hayır
1. Bileşik atama operatörlerini kullandınız mı?		
2. While döngüsünü yazdınız mı?		
3. While döngüsünün çalışması için gereken koşulu yazdınız mı?		
4. While döngüsündeki denetim değişkenini güncellediniz mi?		
5. For döngüsünü yazdınız mı?		
6. For döngüsü içindeki koşul değerini yazdınız mı?		
7. For döngüsü içindeki artım miktarını yazdınız mı?		
8. Do ifadesini yazabildiniz mi?		
9. Do ifadesindeki koşulu yazdınız mı?		

DEĞERLENDİRME

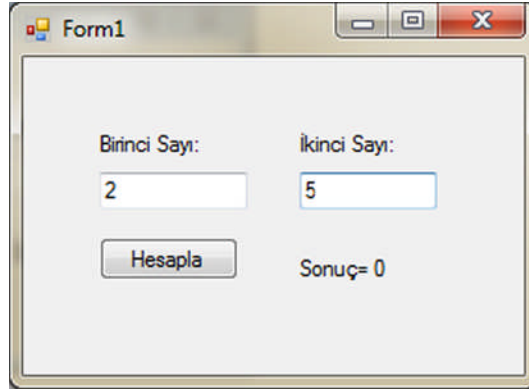
Değerlendirme sonunda “Hayır” şeklindeki cevaplarınızı bir daha gözden geçiriniz. Kendinizi yeterli görmüyorsanız öğrenme faaliyetini tekrar ediniz. Bütün cevaplarınız “Evet” ise “Ölçme ve Değerlendirme” ye geçiniz.

ÖLÇME VE DEĞERLENDİRME

Aşağıdaki soruları dikkatlice okuyunuz ve doğru seçeneği işaretleyiniz.

```
private void button1_Click(object sender, EventArgs e)
{
    int s1 = Convert.ToInt16(textBox1.Text);
    int s2 = Convert.ToInt16(textBox2.Text);
    int s = 1;
    int x = 1;
    while (x <= s2)
    {
        s *= s1;
        x++;
    }
    label3.Text = "Sonuç= " + s.ToString();
}
```

Resim 3.15: Kodlama görüntüsü



Resim 3.16: Ekran görüntüsü

1. Yukarıdaki Resim 3.15'te görülen kodlama *denetim değişkeni* (*sayaç*) değişkeni aşağıdakilerden hangisidir?
A) x B) s C) s1 D) s2
2. Yukarıdaki Resim 3.15 'teki kodlamanın form görüntüsü Resim 3.16 'da görüldüğü gibi ise "Hesapla" butonuna tıkladığında sonuç aşağıdakilerden hangisidir?
A) 25 B) 7 C) 10 D) 32
3. Yukarıdaki Resim 3.15 'teki kodlamanın form görüntüsü Resim 3.16'da görüldüğü gibi ise "Hesapla" butonuna tıkladığında *x* değişkeninin son değeri aşağıdakilerden hangisidir?
A) 4 B) 5 C) 6 D) 7

4. Döngülerle ilgili olarak aşağıdakilerden hangisi veya hangileri doğrudur?
I. Kodlamalardaki şartlı işlemleri yerine getiren yapılardır.
II. Döngünün ilerlemesi ve sonlanması denetim değişkeniyle gerçekleşir.
III. For döngüsünde denetim değişkeni kullanılmamaktadır.
- A) Yalnız I B) Yalnız II C) Yalnız III D) I ve II
5. Başlangıç ve bitiş değeri bilinen tekrarlı işlemlerde kullanılması en uygun olan döngü aşağıdakilerden hangisidir?
A) while B) for C) do D) switch
6. Aşağıdaki döngülerden hangisi içine yazılan kodlar, en az bir kere mutlaka çalışır?
A) while B) for C) do D) switch

Aşağıdaki cümlelerin başında boş bırakılan parantezlere, cümlelerde verilen bilgiler doğru ise D, yanlış ise Y yazınız

- 7 () “*sonuc *=i*” işlemi ile “*sonuc = sonuc * i*” işlemi denk işlemlerdir.
- 8 () *for* döngüsünde, hem koşul hem de denetim değişkeni (koşul) döngünün başında yer almaktadır.
- 9 () *do* döngüsünde denetim değişkeni döngünün içinde koşul ise döngünün başında kullanılmaktadır.
- 10 () Yukarıdaki Resim 3.15’te yer alan kodlamadaki *while* satırı şu şekilde değiştirilmiştir:

while (x >= s2)

Bu değişiklikten sonra Resim 3.16’da yer alan ekranı çalıştırıldığında sonuç 1 olur.

DEĞERLENDİRME

Cevaplarınızı cevap anahtarıyla karşılaştırınız. Yanlış cevap verdiğiniz ya da cevap verirken tereddüt ettiğiniz sorularla ilgili konuları faaliyete geri dönerek tekrarlayınız. Cevaplarınızın tümü doğru ise bir sonraki öğrenme faaliyetine geçiniz.

ÖĞRENME FAALİYETİ-4

AMAÇ

Program kodlaması esnasında oluşabilecek özel durumlara dikkat ederek hatasız programlar yazabileceksiniz.

ARAŞTIRMA

- Gündelik hayatımızda bir takım olumsuzluklarla karşılaşmamak için değişik önlemler alırız. Örneğin, okula giderken evden çıkmadan önce gerek kılık kıyafetimizi, gerekse defter ve kitaplarımızı kontrol ederiz. Çünkü kılık kıyafetimiz düzgün olmaz ise okul idaresinden tepki alırız. Aynı şekilde o gün hava yağacak gibiyse ona göre giyinir ve şemsiyemizi alırız. Aksi hâlde yağmur altında kalabilir ve ıslanabiliriz. Sizler de bu şekilde hayatımızda karşılaşabileceğimiz özel durumlara ait üç adet yazınız.
- Araştırma sonuçlarınızı öğretmene teslim edecek veya sınıfta sunacak şekilde hazırlayınız.

4. HATA AYIKLAMA

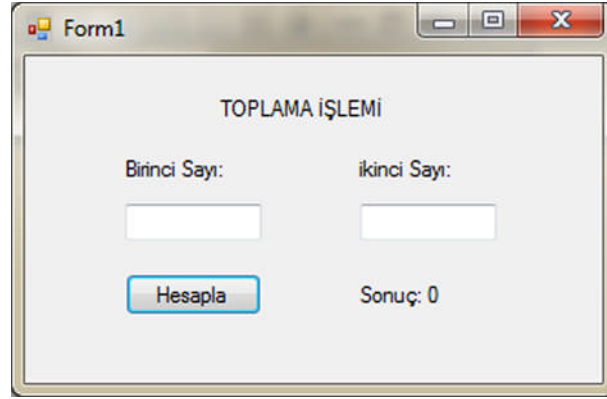
Program çalışırken hemen hemen her aşamasında hatalar meydana gelebilir. Kod parçalarının her zaman beklenildiği gibi çalışacağından emin olmak oldukça zordur. Bu hatalar kişiden kaynaklanabileceği gibi kişinin dışından da kaynaklanabilir. Bilgisayara bir program kurulduğu düşünülün. Çok gerekli bir program olsa da sürekli programda hatalarla karşılaşılırsa ne kadar gerekli olursa olsun, bir daha bu program kullanılmak istenilmez. Oysa programın hataları giderilmiş, hata verdiğinde hatanın nedeninin ne olduğu söyleniyorsa elbette ki bu programı kullanmak kişiye daha yakın gelecektir. O nedenle de hataların programdan ayıklanması son derece önemlidir.

4.1. Try - Catch Bloku

Hataları yakalamadan bir program yazılırsa yukarıda da açıkça belirttiğimiz gibi insanlar bu programı kullanmak istemez. Aşağıdaki örnek, hataların bir programı ne hâle getirdiğine çok güzel bir örnektir.

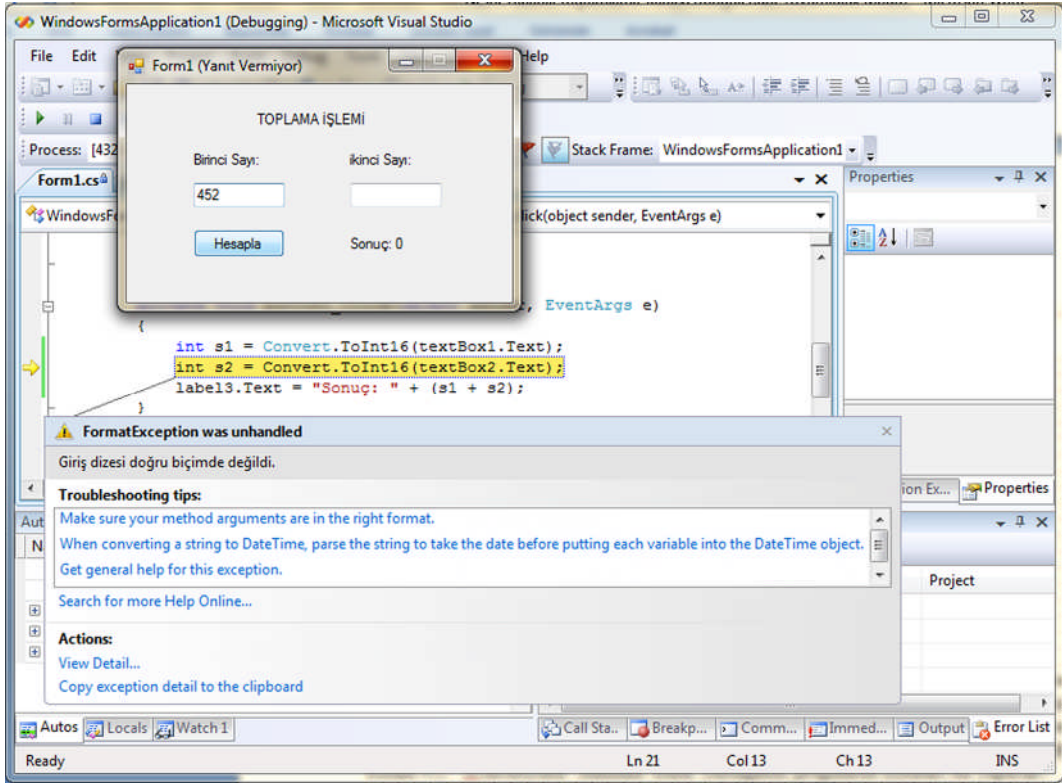
Örnek: Klavyeden girilen iki sayıyı toplayıp sonucu ekrana yazan kodlama yapılmalıdır.

Programın form görüntüsü aşağıdaki gibi olsun.



Resim 4.1: Toplama işlemi ekran görüntüsü

Programın çalışması şu şekilde olacaktır: İki sayı girilecek ve “*Hesapla*” butonuna tıklandığında “*Sonuç*” bölümünde iki sayının toplamı çıkacaktır. Örneğin kullanıcı ilk sayıyı girdi fakat ikinci sayıyı girmede. Normal olarak düşünüldüğünde ikinci sayı otomatik sıfır olmalıdır. Dolayısıyla da “*Sonuç*” bölümünde ilk sayı görünmelidir. Şimdi aşağıdaki Resim 4.2 incelenmelidir. Böyle bir durumda yazılan program bu resimde görüldüğü gibi bir hata ile karşılaşacaktır.



Resim 4.2: Hata ile karşılaşılmış ekran görüntüsü

Oysa bu gibi durumlarda ya kullanıcı ikinci sayıyı girmesi içinde uyarılmalı ya da ikinci sayı otomatik olarak sıfır alınmalıdır. Bu gibi işlemler yapılırsa hem programın olur olmadık yerde kesilmesinin önüne geçmiş hem de kodun daha da profesyonel olması sağlanmış olunur. *try-catch* bloku ile kodu daha sağlam bir zemine oturtulabilir ve programın kullanılabilirliği artırılmış olunur.

Try-catch ifadesinin söz dizimi aşağıdaki gibidir:

```
Try
{
    Hata olmadığı sürece çalışacak kodlar;
}
Catch (özel durum)
{
    Hata oluştuğunda çalışacak kodlar;
}
```

Yukarıdaki söz diziminde görüldüğü gibi normal kodları *try* bölümüne, herhangi bir hata oluştuğunda ise bu gibi durumları yakalamak için de *catch* bölümüne yazılmalıdır. Buradaki özel durum ise hatanın nedenini yakalamak için kullanılan bir ifadedir.

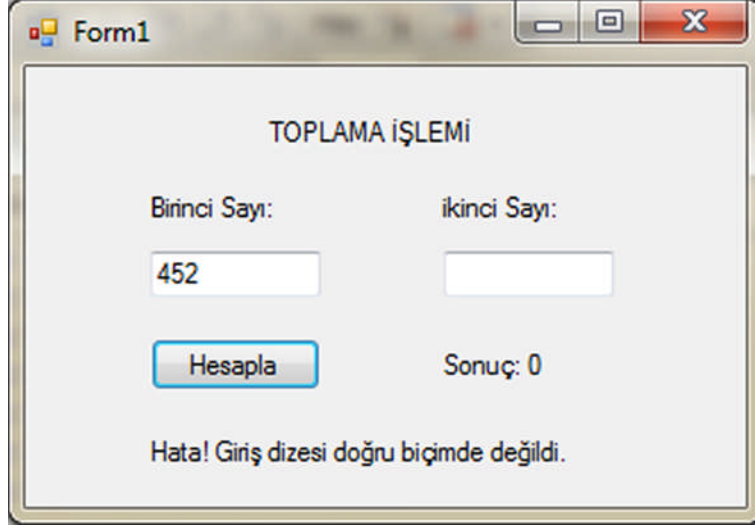
Öyleyse hataların önüne geçmek için try-catch yapısının yukarıdaki örnekte nasıl kullanıldığı aşağıdaki Resim 4.3 'te incelenmelidir.

```
private void button1_Click(object sender, EventArgs e)
{
    try
    {
        int s1 = Convert.ToInt16(textBox1.Text);
        int s2 = Convert.ToInt16(textBox2.Text);
        label3.Text = "Sonuç: " + (s1 + s2);
    }
    catch (Exception hata)
    {
        label5.Text = "Hata! " + hata.Message;
    }
}
```

Resim 4.3: Try-catch yapısının kullanılması

Böylece program kodu hatalara karşı bir öncekine göre daha sağlam bir yapıya kavuşmuş oldu. Resim 4.3'teki kodlamada görüldüğü gibi programın normal kodlaması *try* bölümüne, hata oluştuğunda çalışması gereken kısım ise *catch* bölümüne yazılmıştır. *Catch* bölümünde oluşan özel durumu yakalamak içinse *Exception* (özel durum) türünde hata ifadesi tanımlanmıştır. Oluşan özel durum ile ilgili her türlü bilgi bu ifadeye atanmış

olacaktır. Bu ifadeye Resim 4.3'te görüldüğü gibi label5 kontrolüne aktarılmaktadır. Programın çalışması aşağıda görüldüğü gibi olacaktır.



Resim 4.4: Özel durumun yakalandığı ekran görüntüsü

Yukarıdaki Resim 4.4'te görüldüğü gibi program ikinci sayı girilmemesine rağmen rastgele kesilmemiş ve kullanıcıya oluşan hatanın nedeni bildirilmiştir.

4.2. Birden Çok Catch Bloku

.NET Platformu tarafından sağlanan özel durum kütüphanesi oldukça kapsamlıdır. .NET Platformunda birçok özel durum tanımlanmıştır ve yazılan herhangi bir programda bu özel durumlarla karşılaşma oranı son derece yüksektir. Örneğin, yazılan bir programda Resim 4.2'deki gibi metinsel veriyi sayısal bir veriye dönüştürme özel durumu ile karşılaşılabilir. Bunun yanında *int16* veri türüne sahip bir değişkene *int32* kapsamındaki bir veriyi (overflow, taşma hatası) atama gibi bir özel durumla da karşılaşılabilir. Bunun gibi daha birçok özel durumla karşılaşma durumu söz konusudur.

Karşılaşılan her bir özel durum için ayrı ayrı try-catch bloku yazmak elbette son derece mantıksızdır. Bunun yerine aynı try-catch blokunda farklı catch bölümler kullanılabilir.

Birden çok catch bloku için söz dizimi aşağıdaki gibidir:

```
Try
{
    Hata olmadığı sürece çalışacak kodlar;
}
Catch (özel durum – 1)
{
    İlgili hata oluştuğunda çalışacak kodlar;
}
Catch (özel durum – 2)
{
    İlgili hata oluştuğunda çalışacak kodlar;
}
```

En sık kullanılan üç özel durum aşağıda yer almaktadır.

- **FormatException:** Sayısal veri türüne sahip bir değişkene bir harf girilmesi veya atanması
- **OverflowException:** Bir değişkene atanan veya girilen sayı, değişkenin tanımlandığı veri türünün alabileceği aralığın dışında kalması
- **ArgumentNullException:** Sayısal veri türüne sahip bir değişkene *null* değerini atamak

Örneğin Resim 4-3'teki kod, aşağıdaki Resim 4.5'teki gibi birden çok catch bloku hâline getirilmelidir.

```
private void button1_Click(object sender, EventArgs e)
{
    try
    {
        int s1 = Convert.ToInt16(textBox1.Text);
        int s2 = Convert.ToInt16(textBox2.Text);
        label3.Text = "Sonuç: " + (s1 + s2);
        label5.Text = "";
    }
    catch (FormatException fhata)
    {
        label5.Text = "Hata! Lütfen alanlara sayısal veriler giriniz...";
    }
    catch(OverflowException ohata)
    {
        label5.Text = "Hata! Alanlara çok büyük sayılar giriyorsunuz...";
    }
}
```

Resim 4.5: Birden çok catch bloku kullanımı

Resim 4.5'te görüldüğü gibi her bir özel durum için ayrı *catch* blokları kullanılmıştır. Her özel durum içinde kullanıcı ayrı uyarılmıştır. Bu kod çalıştırıldığında aşağıdaki Resim 4.6'daki gibi bir ekran çıktısı ile karşılaşılır.

Resim 4.6: Birden çok catch bloğunun çalışması

Resim 4.6 'da görüldüğü gibi “*Birinci Sayı*” alanına *int16* veri türünden daha büyük bir sayı girildiği için *overflow* hatası oluşmuştur. Kodlamada da bu hata yakalandığı için ekrana “*HATA! Alanlara çok büyük sayılar giriyorsunuz...*” şeklinde uyarı gelmektedir.

Akla şu şekilde bir soru gelebilir: “*Her iki özel durumda aynı anda meydana gelirse hangisi çalışacaktır?*” Kodlamada ilk olarak bulunan *catch* bloku devreye girecektir.

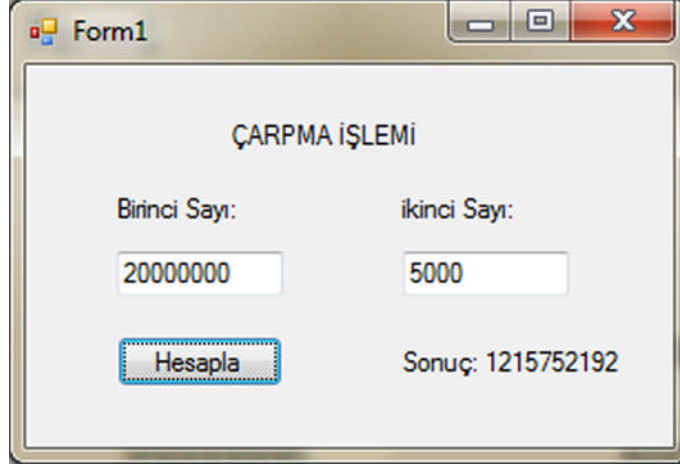
4.3. Denetlenmiş İfadeler

Belirtilen sınırların dışında bir sayıyı *int* türünde bir değişkende tutmak istersek taşma meydana gelir. Eğer -2.147.483.648 sayısından daha küçük bir sayı tutulmak istenirse *underflow*, 2.147.483.647 sayısından büyük bir sayı tutulmak istenirse *overflow* durumu oluşur. Genel olarak her iki durum da *overflow* olarak adlandırılmaktadır. Varsayılan olarak programlama dili derleyicisi taşma problemlerinde yanlış değer üreten bir yapıya sahiptir. Resim 4.8'deki gibi bir görünüme sahip olan form için aşağıdaki Resim 4.7'de görüldüğü gibi bir kodlama yapılır.

```
private void button1_Click(object sender, EventArgs e)
{
    int s1 = Convert.ToInt32(textBox1.Text);
    int s2 = Convert.ToInt32(textBox2.Text);
    int s = s1 * s2;
    label3.Text = "Sonuç: " + s.ToString();
}
```

Resim 4.7: İki sayının toplanması için kullanılan kodlama

Böyle bir kodlama çalıştırıldığında aşağıdaki ekranda yer alan veriler girilerek “Hesapla” butonuna tıklanır.



Resim 4.8: Taşma oluşan bir ekran görüntüsü

Normalde Resim 4.8’deki sayıların çarpılması sonucu 100000000000 sonucu oluşması gerekirken 1215752192 sonucu oluşmuştur. İşte burada bir taşma meydana gelmiştir. Bu taşmanın önüne geçmek için iki seçenek mevcuttur. Bunlardan ilki programlama dilinin derleyicisini projelerdeki taşma problemlerine karşı etkin hâle getirerek yanlış sonuç üretmesi yerine hata ile kesilmesi sağlanabilir. İkinci seçenek ise *checked* ifadesiyle kodlama esnasında bu işlemi yapmaktır.

İlk seçenek için şu adımlar uygulanır: Programda *Project* menüsünden *Projeniz Properties* seçeneği seçilir (Projeniz, projenin adıdır.). Proje özellikleri iletişim kutusunda, *Build* sekmesi tıklanır. Sayfanın sağ alt köşesindeki *Advanced* düğmesi tıklanır. *Advanced Build Settings* iletişim kutusunda, *Check for arithmetic overflow/underflow* onay kutusu işaretlenir. Bundan böyle projeniz taşma problemine karşı etkin hâle gelmiş olacaktır.

İkinci seçenek ise bu işi kodlamayla gerçekleştirmektir. Eğer proje özelliklerindeki *Check for arithmetic overflow/underflow* seçeneği işaretli değilse ve programın yanlış değer üretmesi istenmiyorsa bu durumda *checked* ifadesi kullanılmalıdır. Şimdi Resim 4.7’de yer alan kodlamayı aşağıdaki Resim 4.9’da görüldüğü gibi yeniden yapılır.

```

private void button1_Click(object sender, EventArgs e)
{
    int s1 = Convert.ToInt32(textBox1.Text);
    int s2 = Convert.ToInt32(textBox2.Text);
    int s;
    checked
    {
        s = s1 * s2;
    }
    label3.Text = "Sonuç: " + s.ToString();
}

```

Resim 4.9: checked ifadesi ile taşma problemini etkin hale getiren kodlama

Resim 4.9'da yer alan kodlamaya göre programın çalışması esnasında *checked* blokunda yer alan aritmetiksel işlemde taşma olduğunda, program hata verecektir. Burada yapılan şudur: Programdaki bazı kodlar, oluşabilecek taşmalardan dolayı hatalı sonuçlar üretilmesinin önüne geçilmiştir. Şimdi tekrar program çalıştırılır ve Resim 4.8'de girdiğiniz değerler tekrar girilir. Programın kesilerek hata verdiği görülecektir. Eğer programın hata sırasında kesilmesi istenmiyorsa try-catch bloku kullanılarak hata yönetilebilir.

Bazı durumlarda ise programın taşma hatası vermesi yerine, ne olursa olsun bir sonuç üretmesi sizin için önemli olabilir. Böyle bir sonucu almak için proje özelliklerindeki *Check for arithmetic overflow/underflow* seçeneğinin seçili olmaması gerekmektedir. Eğer bu seçenek seçili ise bu durumda *unchecked* ifadesi ile yanlıştta olsa bir sonuç alınabilir. Resim 4.7'de yer alan kodlardaki aritmetiksel işlemde, oluşan taşmadan dolayı hata aldığımızı düşünerek bu hatanın alınmamasını sağlayabilirsiniz. Aritmetiksel işlemi *unchecked* ifadesi ile denetim dışına çıkararak bu işlemi gerçekleştirebilirsiniz. Şimdi Resim 4.7'deki kodu aşağıdaki şekilde yeniden yazınız.

```

private void button1_Click(object sender, EventArgs e)
{
    int s1 = Convert.ToInt32(textBox1.Text);
    int s2 = Convert.ToInt32(textBox2.Text);
    int s;
    unchecked
    {
        s = s1 * s2;
    }
    label3.Text = "Sonuç: " + s.ToString();
}

```

Resim 4.10: unchecked ile denetim ifadesi

Resim 4.10'da görüldüğü gibi kodlamadaki aritmetiksel işlem denetim dışı bırakılmıştır. Böylece aritmetiksel işlemde meydana gelebilecek bir taşmada program hata vermeyecek, yanlıştta olsa bir sonuç üretecektir.

4.4. Denetlenmiş Deyimler

Checked ve *Unchecked* ifadelerini, parantez içinde yazılmış bir tam sayı deyiminin başında kullanarak bu tür deyimlerdeki taşma denetlemek için de kullanılabilir. Örneğin, Resim 4.9’da görünen kodlama aşağıdaki gibi yeniden yazılmalıdır.

```
private void button1_Click(object sender, EventArgs e)
{
    int s1 = Convert.ToInt32(textBox1.Text);
    int s2 = Convert.ToInt32(textBox2.Text);
    label3.Text = "Sonuç: " + checked(s1 * s2).ToString();
}
```

Resim 4.11: checked ifadesinin denetleyen deyim olarak kullanılması

Resim 4.11’deki kodlama yazıldıktan sonra çalışmasının Resim 4.9’da yer alan kodlamadan herhangi bir farkının olmadığı görülecektir. Burada da aynı şekilde parantez içinde olan aritmetiksel işlem sonucu oluşacak taşmalar, denetimli hâle gelmiş olmaktadır. Aynı şekilde Resim 4.10’da yer alan kodlama da aşağıdaki kodlamayla aynı görevi görmektedir. Bu kodlamayı yaparken proje özelliklerinde yer alan *Check for arithmetic overflow/underflow* seçeneğinin seçili olması gerekmektedir.

```
private void button1_Click(object sender, EventArgs e)
{
    int s1 = Convert.ToInt32(textBox1.Text);
    int s2 = Convert.ToInt32(textBox2.Text);
    label3.Text = "Sonuç: " + unchecked(s1 * s2).ToString();
}
```

Resim 4.12: unchecked ifadesinin denetleyen deyim olarak kullanılması

Resim 4.12’deki kodlamada yer alan çarpma işlemi artık denetimsiz hâle gelmiştir. Dolayısıyla çalışma anında işlem sonucunda herhangi bir taşma meydana geldiğinde program hata vermeyecek, sonuç yanlış da olsa işlem devam edecektir.

Checked ve *unchecked* ifadeleri, tam sayı olmayan aritmetiksel işlemleri denetlemek için kullanamazsınız. Bu ifadeler sadece *int* ve *long* veri türleri için kullanılabilir.

4.5. Özel Durumlar

Programlama dili, programınızın çalışması esnasında oluşabilecek hatalara karşı pek çok özel durum içermektedir. Birçok durumda bu özel durumlardan birinin özel duruma uygun olduğu görülecektir. Örneğin, “*Denetlenmiş Deyimler*” bölümünde sayısal işlemlerde meydana gelebilecek taşmalardan konusu geçmişti veya “*Birden Çok Catch Bloku*” başlıklı bölümde *FormatException* hatasından bahsedildi. Bu gibi hatalar sistem tarafından tanınan hatalardır. Ancak bazı programlarda belirlenebilecek özel durumlar oluşabilecektir. Bu gibi özel durumlar içinde programlama dili kendi içerisinde pek çok özel durum kullanıma sunulmaktadır.

Örneğin, yapılan bir programda kullanıcıya sunulan üç seçenektan birinin seçilmesi istendi (1, 2 ve 3 gibi). Ancak kullanıcı bunlar dışında bir seçeneği seçti (örnek 4 gibi). Bu durumda kullanıcı programda mantıksal bir hata oluşmasına sebep olmuştur. Dolayısıyla bu durumun kullanıcıya bildirilmesi gerekmektedir. Bunu yapabilmek için aşağıda söz dizimi görünen kodlama incelenmelidir.

```
Switch(secenek)
{
    Case 1:
        İşlemler;
    Case 2:
        İşlemler;
    Case 3:
        İşlemler;
    Default:
        throw ArgumantOutOfRangeException("Yanlış seçenek");
}
```

Bu kodlamada görülen gibi kullanıcı 1, 2 ya da 3 'ün dışında bir seçenek seçildiğinde "Yanlış seçenek" mesajıyla uyarılacak bir özel durum oluşturulmuştur. Programcı olarak bundan sonra yapılacak şey ise bu hatayı *catch* ile yakalayıp kullanıcı uyarılmaktır. Özel durum oluşturmak için kullanılan söz dizim aşağıdaki gibidir:

```
throw new exception();
```

Yukarıda söz diziminde de görülen özel durum oluşturmak için *throw* ifadesi kullanılmalıdır. Bu ifade ile özel durum olarak belirlenen işlemler, kodlamada meydana gelen normal hatalar gibi *catch* ifadesi ile yakalanabilir.

Aşağıda programcıya has başka bir özel durum içeren bir örnek kodlama bulunacaktır. Bu örnek, klavyeden girilen iki sayı arasındaki işlem, yine klavyeden girilen işlem türüne (+, -, *, /) göre gerçekleştiren basit bir hesap makinesi örneğidir.

Örneğin, form görüntüsü aşağıdaki Resim 4.13'te görüldüğü gibi olacaktır. Bu formda görüldüğü gibi "Birinci Sayı" ve "İkinci Sayı" kısımlarına girilecek iki sayı arasında, "Operatör" kısmına girilen karaktere göre (+, -, *, /) işlem yapan bir kodlama olacaktır. Ancak burada "Operatör" alanına bizim belirlenen karakter dışında da karakterler girilebilir. İşte bu durumun önüne geçmek için özel durum tanımlaması yapılacaktır.

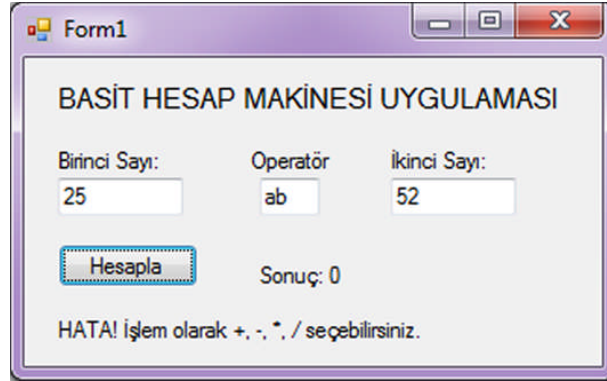
Resim 4.13: Hesap makinesi uygulaması form görüntüsü

Resim 4.13'teki form görüntüsüne göre aşağıdaki gibi bir kodlama yapılabilir. Bu kodlamada da görülebileceği gibi şartlı dallanma yapısı için *switch* yapısı kullanılmıştır.

```
private void button1_Click(object sender, EventArgs e)
{
    string islem = textBox2.Text;
    int s1 = Convert.ToInt32(textBox1.Text);
    int s2 = Convert.ToInt32(textBox3.Text);
    int sonuc = 0;
    try
    {
        switch (islem)
        {
            case "+":
                sonuc = s1 + s2; break;
            case "-":
                sonuc = s1 - s2; break;
            case "*":
                sonuc = s1 * s2; break;
            case "/":
                sonuc = s1 / s2; break;
            default:
                throw new InvalidOperationException
                    ("HATA! İşlem olarak +, -, *, / seçebilirsiniz.");
        }
        label5.Text = "Sonuç: " + sonuc.ToString();
    }
    catch (Exception hata)
    {
        label6.Text = hata.Message;
    }
}
}
```

Resim 4.14: Programa göre özel durum oluşturma

Resim 4.14'te görülen kodlamada özel durum olarak *InvalidOperationException* (Geçersiz İşlem İstisnası) özel durumu kullanılarak hata tanımlaması yapılmıştır. Kullanıcı belirlediği karakterler (+, -, *, /) dışında bir karakteri “Operatör” alanına girdiğinde; “HATA! İşlem olarak +, -, *, / seçebilirsiniz.” hatası ile karşılaşacaktır. Çünkü oluşturulan özel durum aynı zamanda *catch* ifadesi ile yakalanmaktadır. Aşağıda böyle bir ekranla karşılaşma durum görülebilir.



Resim 4.15: Özel durumu catch ile yakalama ekranı

4.6. Finally Bloku

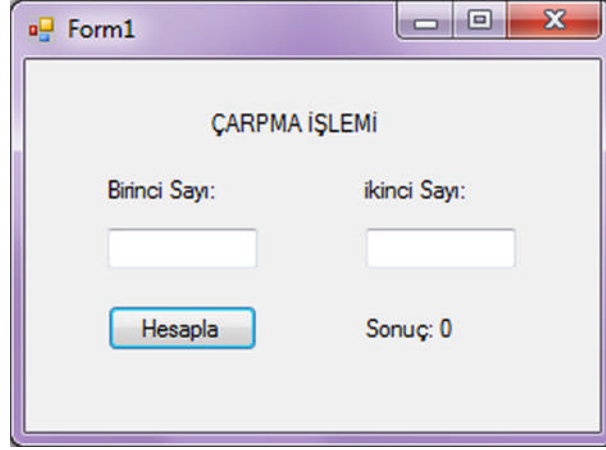
Bir kodlamada bir özel durum oluşsa da oluşmasa da ifadenin her zaman çalıştığından emin olmanın yolu onu bir *finally* bloğunun içine yazmaktır. *Finally* bloku, bir *try* blokundan hemen sonra ya da *try* blokundan sonraki son *catch* blokundan hemen sonra ortaya çıkar. Program *finally* blokuyla ilişkili *try* blokuna girdiği sürece *finally* bloku her zaman çalışacaktır.

finally bloğunun söz dizimi aşağıdaki gibidir:

```
try
{
    Hata oluşmadığında çalışacak kodlamalar;
}
catch(hata yakalama)
{
    Hata oluştuğunda çalışacak kodlamalar;
}
finally
{
    En son çalışacak kodlamalar;
}
```

Yukarıdaki söz diziminde de görüldüğü gibi program, *try* blokuna girdiğinde öncelikle burası çalışacaktır. Eğer buradaki kodların çalışması anında bir özel durum oluşursa *catch* blokunda bu özel durum yakalanacak ve *catch* blokundaki kodlar çalışacaktır. En sonunda da *finally* blokunda yer alan kodlar çalışacaktır. *try* blokundaki kodların çalışması anında herhangi bir hata oluşmazsa *catch* bloku atlanacak ve *finally* bloku çalışacaktır. Kısacası *finally* bloku her durumda çalışacaktır. *finally* bloku bu anlamda daha çok, oluşturulan nesnelerin bellekten temizlenmesi, açılan dosyaların kapatılması, veri tabanı bağlantısının ve nesnelerinin kapatılması, formda yer alan kontrollerdeki verilerin temizlenmesi gibi işlemler için kullanılan bir bölümdür.

Örnek: Aşağıdaki Resim 4.16’da yer alan form görüntüsüne göre klavyeden girilen iki sayıyı çarpıp sonucu “Sonuç” alanında görüntüleyen ve hesaplama işleminin sonunda her iki *textbox* kontrolündeki sayıyı temizleyen kodlama yapılmalıdır.

A screenshot of a Windows application window titled "Form1". The window has a title bar with standard minimize, maximize, and close buttons. The main content area has a title "ÇARPMA İŞLEMİ" centered at the top. Below the title, there are two text boxes: "Birinci Sayı:" on the left and "ikinci Sayı:" on the right. Below the "Birinci Sayı:" text box is a "Hesapla" button. Below the "ikinci Sayı:" text box is a label "Sonuç: 0".

Resim 4.16: Çarpma işlemi form görüntüsü

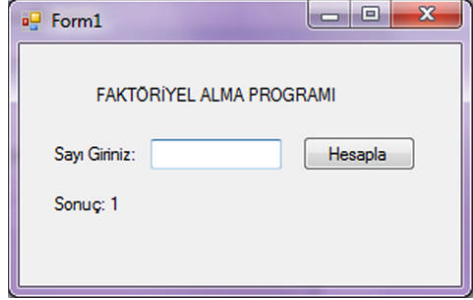
Örnekte kontrol temizleme işi olduğu için bu işi *finally* blokunda yapılmalıdır. Buna göre kodlama aşağıdaki şekilde yapılabilir:

```
private void button1_Click(object sender, EventArgs e)
{
    int s1 = Convert.ToInt32(textBox1.Text);
    int s2 = Convert.ToInt32(textBox2.Text);
    try
    {
        int sonuc = s1 * s2;
        label3.Text = s1 + " x " + s2 + " = " + sonuc;
    }
    catch (Exception hata)
    {
        label5.Text = hata.Message;
    }
    finally
    {
        textBox1.Text = "";
        textBox2.Text = "";
    }
}
```

Resim 4.17: Çarpma işlemi için yapılan kodlama

UYGULAMA FAALİYETİ

Kod içeriğinde hataları ve özel durumları yönetiniz.

İşlem Basamakları	Öneriler
<ul style="list-style-type: none">➤ Klavyeden girilen bir tam sayının faktöriyelini alan programı kodlayınız.	<ul style="list-style-type: none">➤ Nesne tabanlı programlama yazılımı size yardımcı olabilir.
<ul style="list-style-type: none">➤ Kullanıcı "Sayı Giriniz" bölümüne sayıyı girip "Hesapla" butonuna tıkladığında "Sonuç" bölümünde sonucu görüntülenmelidir.	<ul style="list-style-type: none">➤ Ekran görüntüsü şu şekilde olabilir: ➤ Resim 4.18: Program ekran görüntüsü
<ul style="list-style-type: none">➤ Kodlamada meydana gelebilecek hataları yakalayıp gerekli işlemleri yapınız.	<ul style="list-style-type: none">➤ Hataları yakalamak ve işlemek için try – catch yapısını kullanınız.
<ul style="list-style-type: none">➤ Resim 4.18'deki formda yer alan textbox kontrolüne girilecek veriyi, boş bırakılmasına ve karakter girilmesine karşı gerekli hata yakalama kodlamasını yapınız.	<ul style="list-style-type: none">➤ İki adet catch ifadesinde ArgumentException ve Format Exception özel durum yakalama ifadelerini kullanınız.
<ul style="list-style-type: none">➤ Resim 4.18'deki formda yer alan textbox kontrolüne girilecek sayı pozitif olmalıdır. Eğer kullanıcı negatif bir sayı girdiğinde uyarılmalıdır.	<ul style="list-style-type: none">➤ throw ifadesini kullanarak yeni bir özel durum oluşturunuz.
<ul style="list-style-type: none">➤ Aritmetiksel işlemleri, sayı taşmalarına karşı denetimli hâle getirin.	<ul style="list-style-type: none">➤ Proje özellikleri > Buil Sekmesi > Check for arithmetic overflow/underflow seçeneği ile ya da checked ifadesiyle bunu yapabilirsiniz.
<ul style="list-style-type: none">➤ Sonuç görüntüledikten sonra textbox kontrolünü temizleyerek imlecin bu kontrole konumlanmasını sağlayınız.	<ul style="list-style-type: none">➤ Bu işlemleri finally blokunda yapınız.

KONTROL LİSTESİ

Bu faaliyet kapsamında aşağıda listelenen davranışlardan kazandığınız beceriler için **Evet**, kazanamadıklarınız için **Hayır** kutucuklarına (X) işareti koyarak öğrendiklerinizi kontrol ediniz.

Değerlendirme Ölçütleri	Evet	Hayır
1. Kodlamadaki özel durumu yakalayıp işlediniz mi?		
2. Birden çok catch bloku kullandınız mı?		
3. Birden çok özel durumu yakaladınız mı?		
4. Denetlenmiş ifadeler yazdınız mı?		
5. Denetlenmiş deyimler yazdınız mı?		
6. Özel durumlar oluşturduunuz mu?		
7. Bir finally bloku kullandınız mı?		

DEĞERLENDİRME

Değerlendirme sonunda “**Hayır**” şeklindeki cevaplarınızı bir daha gözden geçiriniz. Kendinizi yeterli görmüyorsanız öğrenme faaliyetini tekrar ediniz. Bütün cevaplarınızı “**Evet**” ise “Ölçme ve Değerlendirme”ye geçiniz.

ÖLÇME VE DEĞERLENDİRME

Aşağıdaki soruları dikkatlice okuyunuz ve doğru seçeneği işaretleyiniz.

1. Aşağıdakilerden hangisi kodlamalarda denetimli ifade oluşturmak için kullanılır?
A) try B) catch C) throw D) checked
2. Programcıya has özel durumlar oluşturmak için kullanılan ifade aşağıdakilerden hangisidir?
A) throw B) checked C) try D) catch
3. Program içerisinde oluşan hataları yakalayarak bu hata ile ilgili bilgileri kullanıcıya sunmak için kullanılan ifade aşağıdakilerden hangisidir?
A) try B) checked C) catch D) throw
4. (*FormatException hata*) şeklindeki tanımlamanın yapıldığı bölüm aşağıdakilerden hangisidir?
A) try B) catch C) checked D) throw
5. Programlardaki aritmetiksel işlemlerde meydana gelebilecek taşma hatalarına karşı kodumuzu korumak için kullanılan ifade aşağıdakilerden hangisidir?
A) checked B) tyr C) throw D) catch

Aşağıdaki cümlelerin başında boş bırakılan parantezlere, cümlelerde verilen bilgiler doğru ise D, yanlış ise Y yazınız.

6. () Kodlamalardaki özel durumları yakalamak için birden fazla *catch* bloku kullanılabilir.
7. () *finally* bloku, her *try – catch* ifadesinden sonra mutlaka kullanılmalıdır.
8. () *Try – catch* bloku kullanıldığı bir programın çalışmasında hata meydana geldiğinde bu hatanın görüntülenmesi için *message* özelliği kullanılır.

DEĞERLENDİRME

Cevaplarınızı cevap anahtarıyla karşılaştırınız. Yanlış cevap verdiğiniz ya da cevap verirken tereddüt ettiğiniz sorularla ilgili konuları faaliyete geri dönerek tekrarlayınız. Cevaplarınızın tümü doğru ise “Modül Değerlendirme”ye geçiniz.

MODÜL DEĞERLENDİRME

Aşağıdaki soruları dikkatlice okuyunuz ve doğru seçeneği işaretleyiniz.

1. Boolean değişkenler tanımlamak için aşağıdaki ifadelerden hangisi kullanılır?
A) bool B) boolean C) switch D) while
2. Aşağıdaki yapılardan hangisi veya hangileri kodlamalardaki tekrarlı işlemleri yapmak için kullanılır?
I. if
II. for
III. switch
A) Yalnız I B) Yalnız II C) I ve II D) I ve III
3. *break* komutuyla işlemlerin sonlandırılıp yapı dışına çıkılan şart ifadesi aşağıdakilerden hangisidir?
A) for B) if C) do D) switch
4. İki şartın bulunduğu bir programda aşağıdaki yapılardan hangisi kesinlikle kullanılmalıdır?
A) while B) if C) if – else D) switch
5. Switch yapısında, programda meydana gelebilecek her bir durum hangi komutla ayrılır?
A) break B) case C) continue D) for
6. Bir programda kullanılan switch yapısındaki default ifadesi için hangisi doğrudur?
A) Her switch yapısında olması gereken bir ifadedir.
B) Kendinden önceki şartlar gerçekleşmediğinde çalışır.
C) switch yapısında, bütün durumlardan önce değerlendirilir.
D) Switch yapısından çıkmak için kullanılır.
7. Bir döngüdeki denetim değişkeni (sayaç) için aşağıdakilerden hangisi veya hangileri kesinlikle doğrudur?
I. Her döngüde olmak zorunda değildir.
II. Bir döngünün ilerlemesini sağlar.
III. Döngünün bitiş değerinden küçük olmalıdır.
A) Yalnız I B) Yalnız II C) Yalnız III D) II ve III
8. Nesne tabanlı programlama yazılımında oluşturulan projenin varsayılan özelliği aritmetiksel işlemlerdeki taşmaları denetleyecek şekilde ayarlanmıştır. Bu projedeki bir aritmetiksel işlemde taşmadan dolayı oluşacak hatayı denetim dışına çıkarmak için hangi ifade kullanılır?
A) catch B) throw C) unchecked D) finally

9. Aşağıdaki eşleştirmelerden hangisi doğrudur?
A) throw - şartlı ifadeler oluşturma
B) switch - tekrarlı işlemler gerçekleştirme
C) while - özel durumlar oluşturma
D) catch – özel durum yakalama
10. İf yapısı için aşağıda yazılanlardan hangisi doğrudur?
I. Her if yapısında mutlaka else ifadesi kullanılmalıdır.
II. İf yapısı içinde başka bir if yapısı kullanılabilir.
III. En az iki seçeneğin olduğu durumlarda kullanılan şart yapısıdır.
A) Yalnız I B) Yalnız II C) II, III D) I, III

DEĞERLENDİRME

Cevaplarınızı cevap anahtarıyla karşılaştırınız. Yanlış cevap verdiğiniz ya da cevap verirken tereddüt ettiğiniz sorularla ilgili konuları faaliyete geri dönerek tekrarlayınız. Cevaplarınızın tümü doğru ise bir sonraki modüle geçmek için öğretmeninize başvurunuz.

CEVAP ANAHTARLARI

ÖĞRENME FAALİYETİ-1'İN CEVAP ANAHTARI

1	C
2	B
3	A
4	D
5	C

ÖĞRENME FAALİYETİ-2'NİN CEVAP ANAHTARI

1	A
2	D
3	B
4	A
5	C
6	Doğru
7	Yanlış
8	Doğru

ÖĞRENME FAALİYETİ-3'ÜN CEVAP ANAHTARI

1	A
2	D
3	C
4	B
5	B
6	C
7	Doğru
8	Doğru
9	Yanlış
10	Doğru

ÖĞRENME FAALİYETİ-4'ÜN CEVAP ANAHTARI

1	D
2	A
3	C
4	B
5	A
6	Doğru
7	Yanlış
8	Doğru

MODÜL DEĞERLENDİRMENİN CEVAP ANAHTARI

1	A
2	B
3	D
4	C
5	B
6	B
7	B
8	C
9	D
10	C

KAYNAKÇA

- SHARP John, **Adım Adım Microsoft Visual Studio C#** 2008, Arkadaş Yayınevi, Ankara, 2009.
- ALGAN Sefer, **Her Yönüyle C#**, Pusula Yayınevi, İstanbul, 2010.