PART 5

5. Denetleyici Türleri

Java dili , aşağıdakiler de dahil olmak üzere çok çeşitli denetleyici türlerine sahiptir.

- Java Erişim Kontrol Denetleyicileri (Access Control Modifiers)
- Erişim Harici Denetleyiciler (Non-Access Control Modifiers)

5.1 Erişim Kontrolü Denetleyicileri

Java sınıflar, değişkenler, metotlar ve yapılandırıcıların erişim düzeylerini ayarlamak için bir dizi erişim denetleyicisi sağlar. Bu 4 erişim seviyesi şunlardır :

- Paket için görünür (varsayılan değer). Denetleyiciye gerek yoktur.
- Sadece sınıf için görünür (private).
- Tüm nesnelere görünür (public).
- Paket ve bu paketten türetilen bütün altsınıflar için görünür (protected).

5.1.1 Private Erişim Denetleyicileri

Private olarak tanımlanan metotlar, değişkenler ve yapılandırıcılara yalnızca tanımlanmış olan sınıfın içinden erişilebilir. Private erişim denetleyicisi en kısıtlayıcı erişim düzeyidir. Sınıflar ve arayüzler private olarak tanımlanamaz. Eğer public tanımlanmış getter metotları sınıfın içinde mevcut ise, private olarak tanımlanan değişkenlere sınıfın dışından erişilebilinir. Private denetleyici kullanmak, bir nesnenin kendisini saklamasının ve dış dünyadan veri saklamanın ana yoludur

Örnek

Aşağıdaki sınıf private erişim denetleyicisini kullanıyor:

```
public class Logger {
   private String format;
   public String getFormat() {
      return this.format;
   }
   public void setFormat(String format) {
      this.format = format;
   }
}
```

Burada, *Logger* sınıfının *format* değişkeni private'dır, yani diğer sınıfların bu değere direkt olarak erişmesine imkan yoktur.Bu değişkeni erişilebilir kılmak için 2 tane metot oluşturduk. Format değerini geri döndüren *getFormat()* metodu ve format değerini düzenleyen *setFormat(String)* metodu.

5.1.2 Public Erişim Denetleyicisi

- Public olarak tanımlanmış bir sınıf,metot,yapılandırıcı veya arayüz diğer herhangi bir sınıftan erişilebilir. Bu yüzden public sınıf içindeki alanlar,metotlar,bloklar Java Evrenindeki herhangi başka bir sınıftan erişilebilir.
- Ancak; sınıf public olmasına rağmen başka bir paketten ulaşmaya çalışıyorsak, o public sınıf içeri alınmalıdır.
- Sınıf mirasından dolayı, sınıfın bütün public metotları ve değişkenleri alt sınıflar tarafından miras olarak alınmış olur.

Örnek

Aşağıdaki fonksiyon public erişim denetleyicisini kullanıyor:

```
public static void main(String[] arguments) {
   // ...
}
```

Uygulamanın main() metodu public olmalıdır. Aksi takdirde sınıfın çalıştırılması için Java yorumlayıcısı tarafından çağırılamaz.

5.1.3 Protected Erişim Denetleyicisi

- Süpersınıf içinde protected olarak tanımlanmış değişkenler,metotlar ve constructorlar sadece başka paketlerin içindeki alt sınıflardan veya paketin içindeki sınıfların protected üyeleri tarafından erişilebilir.
- Protected erişim denetleyicileri sınıflara ve arayüzlere uygulanamaz. Metotlar ve alanlar protected olarak tanımlanabilir, ancak bir arayüzdeki metotlar ve alanlar protected olarak tanımlanamaz.
- Protected erişim denetleyicisi alt sınıflara; kullanıcı tarafından yardımcı metotları veya değişkenleri kullanma şansı tanır. Bu sayede alakasız sınıfların kullanmaya çalışması engellenir.

Örnek

Aşağıdaki üst sınıf, alt sınıfının *openSpeaker()* metodunu geçersiz kılması için protected erişim denetleyicisini kullanır.

```
class AudioPlayer {
   protected boolean openSpeaker(Speaker sp) {
      // implementation details
   }
}

class StreamingAudioPlayer extends AudioPlayer {
   boolean openSpeaker(Speaker sp) {
      // implementation details
   }
}
```

5.2 Erişim Harici Denetleyiciler

5.2.1 “final” Denetleyicisi

- final Değişkenleri

- Bir final değişkenin ilk değeri sadece bir defa atanabilir. Değişkenin içindeki veri bir daha değiştirilemez
- *final* denetleyicisi sabit bir sınıf değişkeni oluşturmak için; değişkenlerle birlikte *static* olarak kullanılır.

Örnek

```
public class Test{
  final int value = 10;
  // The following are examples of declaring constants:
  public static final int BOXWIDTH = 6;
  static final String TITLE = "Manager";

  public void changeValue(){
     value = 12; //will give an error
  }
}
```

- # final Metotları

Herhangi bir final metodu hiç bir şekide bir alt sınıf tarafından geçersiz kılınamaz. Daha önce de bahsedildiği gibi final denetleyicisi bir metodun alt sınıf tarafından değiştirilmemesi için kullanılır.

Örnek

Örnekteki gibi final denetleyicisini kullanarak sınıf tanımlayarak metotlar oluşturun.

```
public class Test{
    public final void changeName(){
       // body of method
    }
}
```

- # final Sınıfları

Bir sınıfın final kullanılarak tanımlanmasının sebebi,o sınıftan bir alt sınıf oluşturulmasını engellemektir.

Örnek

```
public final class Test {
   // body of class
}
```

Bölüm 6

Java Operatörleri

Java, değişkenleri işlemek için çok sayıda operatör bulundurur. Java operatörlerini aşağıdaki gibi gruplandırabiliriz.

- Aritmetik operatörler
- Ilişkisel operatörler
- Binary operatörler
- Mantıksal operatörler
- Atama operatörleri
- Diğer Operatörler

6.1 Aritmetik Operatörler

Aşağıdaki tablo aritmetik operatörleri listeler. İnteger A değişkeninin 10, B değişkenin 20 değerini aldığını varsayalım.

Operator	Description	Example
+	Toplama: Operatörün iki tarafındaki değeri birbiriyle toplar	A + B 30 değerini verecektir.
-	Çıkartma: : Operatörün iki tarafındaki değeri birbirinden çıkartır.	A – B -10 değerini verecektir
*	Çarpma: Operatörün iki tarafındaki değeri birbiriyle çarpar	A * B 200 değerini verecektir
/	Bölme: Operatörün iki tarafındaki değeri birbirine böler	B / A 2 değerini verecektir.
%	Mod alma: Operatörün solundaki değişkenin; sağındakine göre modunu alıp kalanı verir.	B % A 0 değerini verecektir.
++	Arttırma: Değişkenin değerini 1 arttırır	B++ 21 değerini verecektir.

--	Azaltma: Değişkenin değerini 1 azaltacaktır.	B-- 19 değerini verecektir

6.2 İlişkisel Operatörler

Aşağıdaki ilişkisel operatörler Java tarafından desteklenmektedir. İnteger A değişkeninin 10, B değişkenin 20 değerini aldığını varsayalım.

Operator	Description	Example
==	İki değişkenin değerini kontrol eder,eğer eşitse true değerini döndürür.	(A == B) False değer döner..
!=	İki değişkenin değerini kontrol eder,eğer eşit değilse true değerini döndürür.	(A != B) True değer döner..
>	Operatörün solundaki değişkenin, sağındakinden büyük olup olmadığını kontrol eder. Eğer büyükse true değer döner.	(A > B) False değer döner.
<	Operatörün solundaki değişkenin, sağındakinden küçük olup olmadığını kontrol eder. Eğer küçükse true değer döner.	(A < B) true değer döner
>=	Operatörün solundaki değişkenin, sağındakine eşit veya büyük olup olmadığını kontrol eder. Eğer öyleyse true değer döner.	(A >= B) False değer döner.
<=	Operatörün solundaki değişkenin, sağındakine eşit veya küçük olup olmadığını kontrol eder. Eğer öyleyse true değer döner.	(A <= B) true değer döner

6.3 Binary operatörler

- Java integer,long,short,char ve byte tipleri için uygulanabilen birkaç Binary operatör sunar.
- Binary operatörler bitler üstünde çalışır ve bit-bit işlem yapar. a=60 ve b=13 diyelim. Bunların binary formları aşağıdaki gibidir.

```
a = 0011 1100
b = 0000 1101
```

```
a & b   = 0000 1100
a |  b   =  0011 1101
a ^ b   =  0011 0001
~a       = 1100 0011
```

Aşağıdaki tablo Binary operatörleri göstermektedir. int A'nın 60 , B'nin 13 değerlerini aldığını varsayalım.

Operator	Description	Example
&	Binary AND operatörü iki tarafta da bulunan biti sonuca kopyalar.	(A & B) 12 değerini verecektir. Bu da 0000 1100
|	Binary OR operatörü iki taraftan birinde bulunan biti sonuca kopyalar.	(A | B) 61 değerini verecektir. Bu da 0011 1101
^	Binary XOR operatörü bir tarafta bulunup diğer tarafta bulunmayan biti sonuca kopyalar.	(A ^ B) 49 değerini verecektir. Bu da 0011 0001
~	Binary 1'in tümleyeni (Ones Complement) operatörü. Sayının bütün 1 lerini sıfır, bütün sıfırlarını 1 yapar.	(~A) -60 değerini verecektir. Bu da 1100 0011
<<	Binary sola kaydırma operatörü. Operatörün solundaki değişkenin değeri sağdakine göre sola kaydırılır	A << 2 240 değerini verecektir. Bu da 1111 0000
>>	Binary sağa kaydırma operatörü. Operatörün solundaki değişkenin değeri soldakine göre sağa kaydırılır	A >> 2 15 değerini verecektir. Bu da 1111

6.4 Mantıksal Operatörler

Aşağıdaki tablo mantıksal operatörleri göstermektedir. boolean A'nın **true** , B'nin **false** değerlerini aldığını varsayalım.

Operator	Description	Example
&&	Mantıksal AND operatörü. Eğer iki taraftaki değişken de false değilse sonuç true döner	(A && B) false değer döner..
||	Mantıksal OR operatörü. Eğer operatörün iki tarafındaki değişkenlerden birisi true ise sonuç true döner.	(A || B) true değer döner.
!	Mantıksal NOT operatörü. Operasyonun mantıksal cevabını terse çevirir. Eğer true değer dönüyorsa, sonucu false değere çevirir.	!(A && B) true değer döner..

6.5 Atama Operatörleri

Aşağıdaki atama operatörleri Java dili tarafından desteklenmektedir.

Operator	Description	Example
=	Basit atama operatörüdür. Operatörün sağındaki değişkenin değeri solundaki(lere)'ne atanır.	C = A + B; A+B nin değerini C'ye atayacaktır.
+=	Operatörün sağındaki değeri solundaki değerle toplayıp tekrar solundaki değere atar.	C += A ; C = C + A değerine eşittir.
-=	Operatörün sağındaki değeri solundaki değerden çıkartıp tekrar solundaki değere atar.	C -= A; C = C - A değerine eşittir.
*=	Operatörün sağındaki değeri solundaki değerle çarpıp tekrar solundaki değere atar.	C *= A; C = C * A değerine eşittir.
/=	Operatörün sağındaki değeri solundaki değere bölüp tekrar solundaki değere atar.	C /= A; C = C / A değerine eşittir.
%=	Operatörün sağındaki değerin solundakine göre modunu alıp tekrar solundaki değere atar.	C %= A; C = C % A değerine eşittir.

6.6 Koşul Operatörü (? :)

Koşul operatörü aynı zamanda üçlü operatörler olarak da bilinir. Bu operatör 3 işlenen kullanır ve boolean ifadelerin çözümleri için kullanılır. Bu operatörün amacı değişkene hangi değerin atanacağına karar vermektir. Şu şekilde kodlanmaktadır:

```
variable x = (expression) ? value if true : value if false
```

ÖRNEK:

```
public class Test {

   public static void main(String args[]){
      int a , b;
      a = 10;
      b = (a == 1) ? 20: 30;
      System.out.println( "Value of b is : " +  b );

      b = (a == 10) ? 20: 30;
      System.out.println( "Value of b is : " + b );
   }
}
```

Aşağıdaki sonuç üretilecektir.

```
Value of b is : 30
Value of b is : 20
```

6.7 “instanceOf” Operatörü

Bu operatör sadece nesne referans değişkenleri için kullanılır. Nesnenin belirtilen tipte olup olmadığını kontrol eder.(sınıf tipi veya arayüz tipi.) instanceOf operatörü aşağıdaki gibi yazılır.

```
( Object reference variable ) instanceOf  (class/interface type)
```

ÖRNEK:

```
String name = = 'James';
boolean result = name instanceOf String;
// This will return true since name is type of String
```

PART 7

Java Döngüleri

Bazı durumlarda, bir kod bloğunu birkaç kere çalıştırmamız gerekir ve bu çoğu zaman döngü olarak adlandırılır. Java, çok esnek üç döngü mekanizmasına sahiptir. Aşağıdaki üç döngüden birini kullanabilirsiniz:

- while
- do...while
- for

Java 5 itibariyle ***geliştirilmiş for döngüsü (enhanced for loop)*** tanıtıldı. Bu döngü daha çok diziler için kullanılmaktadır.

7.1 while Döngüsü

While döngüsü, bir görevi belirli sayıda tekrarlarlamamız için olanak sağlayan bir kontrol yapısıdır.

```
while(Boolean_expression)
{
   //Statements
}
```

Çalıştırırken, *boolean_expression* ifadesinin sonucu true ise , döngü içindeki işlemler yürütülecektir. Bu işlem , ifadenin değeri true oldukça devam edecektir.

Burada while döngüsünün kilit noktası, döngünün hiç çalışmayabilir olmasıdır. İfade test edilir ve sonuç false olursa, döngü gövdesi geçilecektir.

Örnek:

```
public class Test {

   public static void main(String args[]) {
      int x = 10;

      while( x < 20 ) {
         System.out.print("value of x : " + x );
         x++;
         System.out.print("\n");
      }
   }
}
```

Bu aşağıdaki sonucu üretecektir:

```
value of x : 10
value of x : 11
value of x : 12
value of x : 13
value of x : 14
value of x : 15
value of x : 16
value of x : 17
value of x : 18
value of x : 19
```

7.2 do…while döngüsü

Bir do…while döngüsü, en az bir sefer çalışmayı garanti etmesi haricinde, while döngüsü ile birbirine benzerdir.

```
do
{
   //Statements
}while(Boolean_expression);
```

Dikkat ederseniz, Boolean ifade döngünün sonunda görülmektedir, bu yüzden döngünün içindeki komutlar Boolean test edilmeden önce bir kere çalışır.Boolean ifade true ise, kontrol akışı geri atlama yapar ve döngü içindeki komutlar tekrar çalışır. Bu işlem, Boolean ifade false olana kadar tekrarlanır.

Örnek:

```
public class Test {

   public static void main(String args[]){
      int x = 10;

      do{
         System.out.print("value of x : " + x );
         x++;
         System.out.print("\n");
      }while( x < 20 );
   }
}
```

Bu aşağıdaki sonucu üretecektir:

```
value of x : 10
value of x : 11
value of x : 12
value of x : 13
value of x : 14
value of x : 15
value of x : 16
value of x : 17
value of x : 18
value of x : 19
```

7.3 for Döngüsü

For döngüsü, belirli bir sayıda yürütülmesi gereken döngüyü, verimli bir şekilde yazmanıza olanak sağlayan bir tekrarlı kontrol yapısıdır. For döngüsü, bir görevin kaç defa tekrarlanacağını bildiğiniz zaman yararlıdır.

```
for(initialization; Boolean_expression; update)
{
   //Statements
}
```

Örnek:

```
public class Test {

   public static void main(String args[]) {

      for(int x = 10; x < 20; x = x+1) {
         System.out.print("value of x : " + x );
         System.out.print("\n");
      }
   }
}
```

Bu aşağıdaki sonucu üretecektir:

```
value of x : 10
value of x : 11
value of x : 12
value of x : 13
value of x : 14
value of x : 15
value of x : 16
value of x : 17
value of x : 18
value of x : 19
```

7.4 Geliştirilmiş(Enhanced) for Döngüsü

Java 5'ten itibaren ***geliştirilmiş for döngüsü(enhanced for loop)*** adlı yeni bir döngü yapısı kullanılmaya başlanmıştır. Bu döngü daha çok diziler için kullanılmaktadır.

Geliştirilmiş for döngüsünün syntax'i:

```
for(declaration : expression)
{
   //Statements
}
```

- **declaration.** Eriştiğiniz dizi elemanlarının türü ile uyumlu, yeni deklare edilmiş blok değişkeni.

- **expression.** Döngü boyunca ihtiyacınız olan dizi. İfade, bir dizi değişkeni veya bir dizi döndüren metot çağrısı olabilir.

Örnek:

```
public class Test {

   public static void main(String args[]){
      int [] numbers = {10, 20, 30, 40, 50};

      for(int x : numbers ){
         System.out.print( x );
         System.out.print(",");
      }
      System.out.print("\n");
      String [] names ={"James", "Larry", "Tom", "Lacy"};
      for( String name : names ) {
         System.out.print( name );
         System.out.print(",");
      }
   }
}
```

Bu aşağıdaki sonucu üretecektir:

```
10,20,30,40,50,
James,Larry,Tom,Lacy,
```

7.5 "break" anahtar sözcüğü

Break anahtar sözcüğü, tüm döngüyü durdurmak için kullanılır. Break anahtar sözcüğü, herhangi bir döngü içinde veya bir switch komutu içinde kullanılmalıdır.

Örnek

```
public class Test {

   public static void main(String args[]) {
      int [] numbers = {10, 20, 30, 40, 50};

      for(int x : numbers ) {
         if( x == 30 ) {
              break;
         }
         System.out.print( x );
         System.out.print("\n");
      }
   }
}
```

Bu aşağıdaki sonucu üretecektir:

```
10
20
```

7.6 “continue” anahtar sözcüğü

Continue anahtar sözcüğü, döngü kontrol yapılarından herhangi birinin içinde kullanılabilir. Döngünün derhal sonraki iterasyonuna atlamasına sebep olur.

Örnek:

```
public class Test {

   public static void main(String args[]) {
      int [] numbers = {10, 20, 30, 40, 50};

      for(int x : numbers ) {
         if( x == 30 ) {
              continue;
         }
         System.out.print( x );
         System.out.print("\n");
      }
   }
}
```

Bu aşağıdaki sonucu üretecektir:

```
10
20
40
50
```

PART 8

Karar Verme

8.1 “if” yapısı

if komutu, bir veya daha çok komut tarafından takip edilen bir Boolean ifadeden oluşmaktadır.

```
if(Boolean_expression)
{
   //Statements will execute if the Boolean expression is true
}
```

Eğer boolean ifade true olarak değerlendirilirse, o halde if komutu içindeki kod bloğu çalıştırılacaktır.

Örnek:

```
public class Test {

   public static void main(String args[]){
      int x = 10;

      if( x < 20 ){
         System.out.print("This is if statement");
      }
   }
}
```

Bu aşağıdaki sonucu üretecektir:

```
This is if statement
```

8.2 “if...else” yapısı

Bir if komutunun ardından, Boolean ifade false olduğu zaman çalışan isteğe bağlı bir else komutu gelebilir.

```
if(Boolean_expression){
   //Executes when the Boolean expression is true
}else{
   //Executes when the Boolean expression is false
}
```

Örnek:

```
public class Test {

   public static void main(String args[]){
      int x = 30;
```

```
      if( x < 20 ){
         System.out.print("This is if statement");
      }else{
         System.out.print("This is else statement");
      }
   }
}
```

Bu aşağıdaki sonucu üretecektir:

```
This is else statement
```

8.3 “if--else if--else” yapısı

Bir if komutunun ardından, isteğe bağlı bir else if…else komutu gelebilir. Çeşitli koşulları test etmek için, tek bir if..else if komutu kullanmak çok yararlıdır.

If…else ‘in syntax’i:

```
if(Boolean_expression 1){
   //Executes when the Boolean expression 1 is true
}else if(Boolean_expression 2){
   //Executes when the Boolean expression 2 is true
}else if(Boolean_expression 3){
   //Executes when the Boolean expression 3 is true
}else {
   //Executes when the none of the above condition is true.
}
```

Örnek

```
public class Test {
   public static void main(String args[]){
      int x = 30;

      if( x == 10 ){
         System.out.print("Value of X is 10");
      }else if( x == 20 ){
         System.out.print("Value of X is 20");
      }else if( x == 30 ){
         System.out.print("Value of X is 30");
      }else{
         System.out.print("This is else statement");
      }
   }
}
```

Bu aşağıdaki sonucu üretecektir:

```
Value of X is 30
```

8.4 İç içe “if…else”

If-else komutlarını iç içe koymak her zaman geçerlidir, yani; bir if veya else if komutunu başka bir if veya else if komutu içinde kullanabilirsiniz.

```
if(Boolean_expression 1){
   //Executes when the Boolean expression 1 is true
   if(Boolean_expression 2){
      //Executes when the Boolean expression 2 is true
   }
}
```

If komutunu içi içe koyduğumuz gibi, else if…else komutunu da iç içe koyabilirsiniz.

Örnek:

```
public class Test {

   public static void main(String args[]){
      int x = 30;
      int y = 10;

      if( x == 30 ){
         if( y == 10 ){
         System.out.print("X = 30 and Y = 10");
      }
   }
}
```

Bu aşağıdaki sonucu üretecektir:

```
X = 30 and Y = 10
```

8.5 “switch” yapısı

switch ifadesi, bir değişkenin değerler listesi karşısında eşitliğinin test edilmesini sağlar. Her bir değer “case” olarak adlandırılır ve değişken her bir case ile denetlendikten sonra bir ötekine geçer.

```
switch(expression){
    case value :
       //Statements
       break; //optional
    case value :
       //Statements
       break; //optional
    //You can have any number of case statements.
    default : //Optional
       //Statements
}
```

switch komutu için aşağıdaki kurallar uygulanır.

- Switch komutunda kullanılan değişken sadece byte, short, int veya char türünde olabilir.
- Bir switch içinde herhangi sayıda case komutu olabilir. Her bir case'in ardından karşılaştırılacak değer ve iki nokta üst üste gelir.
- Case için seçilen değer, switch içindeki değişkenle aynı veri tipinde olmalıdır.
- Switch içindeki değer ile case değeri birbirini tutuyorsa, case komutunun ardından gelen ifadeler, break komutuna kadar çalıştırılacaktır.
- Break komutuna ulaşıldığında, switch sonlanır, ve kontrol akışı switch komutunun ardından gelen satıra atlar.
- Bir switch komutunun isteğe bağlı bir default case'i olabilir ve bu case switch'in sonunda kullanılmalıdır. Default case, hiç bir case'in true olmama durumunda görevi yerine getirmek için kullanılır. Default case içinde break kullanımı gerekmez.

Örnek

```
public class Test {

   public static void main(String args[]){
      char grade = args[0].charAt(0);

      switch(grade)
      {
         case 'A' :
            System.out.println("Excellent!");
            break;
         case 'B' :
         case 'C' :
            System.out.println("Well done");
            break;
         case 'D' :
            System.out.println("You passed");
         case 'F' :
            System.out.println("Better try again");
            break;
         default :
            System.out.println("Invalid grade");
      }
      System.out.println("Your grade is " + grade);
   }
}
```

Yukardaki programı, çeşitli satır argümanları kullanarak derleyip, çalıştırın. Bu aşağıdaki sonucu üretecektir:

```
$ java Test a
Invalid grade
Your grade is a a
```

```
$ java Test A
Excellent!
Your grade is a A
$ java Test C
Well done
Your grade is a C
$
```