Bölüm 2: Diziler

Veri Yapıları

Diziler

- Aynı türdeki verileri düzenli bir şekilde saklamak için kullanılır.
- Aynı isimle tanımlanan ve birden çok veriyi saklayabilen değişkenlerdir.
- Sabit boyuta sahip, elemanları her zaman değiştirilebilen yapılarıdır.
- Dizilerin boyutu başlangıçta belirlenir ve daha sonra değiştirilemez.
- Öğeler, bellekte her zaman bir arada paketlenir.
- Dizilerin içeriği değiştirilebilir; öğeler eklenebilir, silinebilir, güncellenebilir.
- Düşük seviye, arka planda çalışan ve sistem düzeyinde performansın kritik olduğu işlemler için tasarlanmıştır.
- Matris, resim ve ses verileri gibi yapıları temsil etmek için kullanılır.

Dizi Kullanımı

- Dizi elemanlarına, indeks numaraları ile erişilir.
- İlk elemanın indeksi 0'dan başlar.
- Elemanlara erişmek için dizinin adı ve indeksi kullanılır.

- **Örnek**:
- `Dizi = [10, 20, 30, 40, 50]`
- `İlkEleman = Dizi[0] # İlk eleman 10'dur.`

Dizi Avantajları

- Verileri düzenli ve eklenme sırasına göre saklar.
- Elemanlara hızlı erişim sağlar (indeksleme).
- Verileri gruplandırarak düzenler.
- Dizi türleri, alt türlerine genişletilebilir, yani kovaryans özelliğine sahiptir.
- Hem ilkel veri tiplerini (int, char, vb.) hem de nesne türlerini içerebilir.

Dizi Dezavantajları

- Sabit boyutludur, boyutu değiştirilemez.
- Öğe ekleme ve çıkarmalar zaman alabilir (tüm diziyi kaydırma gerekebilir).
- Statik tür kontrolü, tür hatalarını tespit edemez; tür kontrolü çalışma zamanında gerçekleşir.
- Java dizileri için içsel işlemler çok sınırlıdır. (sıralama, arama, gibi)

Dizi Türleri

- **Tek boyutlu** diziler: Elemanlar bir satırda saklanır.
- **İki boyutlu** diziler: Elemanlar satır ve sütunlarda saklanır.
- **Çok boyutlu** diziler: Daha karmaşık veri düzenlemelerini destekler.

Diziler

- Görsel, beş elemanlı bir diziyi temsil etmektedir.
- Dizideki her bir eleman, farklı bir hücre içinde bulunur.
- Dizinin elemanları bir arada ve düzenli bir şekilde saklanır.

indis	0	1	2	3	4
eleman	13	-1	4	2	7

Dizi Tanımlama

- `VeriTipi[] diziAdi = new VeriTipi[diziBoyutu];`

- **VeriTipi**: Dizide saklanacak verilerin türünü belirtir.
- **diziAdi**: Dizinin adını temsil eder.
- **diziBoyutu**: Dizinin kaç eleman içereceğini belirtir.

- **Örnek**:

```
int[] sayilar = new int[5];
```

Dizi Elemanlarına Erişim

- Dizi elemanlarına, indeks numaralarıyla erişilir.
- İndeksler 0'dan başlar.

- **Örnek**:

```
int ilkEleman = sayilar[0]; // İlk elemanı alır
int ikinciEleman = sayilar[1]; // İkinci elemanı alır
```

Dizi İlklendirme

- Dizi elemanlarına başlangıç değerleri atanabilir.
- İlklendirme sırasında dizi boyutunu belirtmeye gerek yoktur.
- İnitializers, bir dizinin ilk değerlerini atamak için kullanılır.

- **Örnek**:

```
int[] sayilar = {10, 20, 30, 40, 50};
```

Dizi Boyutunu Almak

- `length` özelliği ile dizinin boyutu alınabilir.
- Bu özellik dizinin kaç eleman içerdiğini verir.

- **Örnek**:

```
int diziBoyutu = sayilar.length; // Dizi boyutu: 5
```

Boyut Belirtilmediğinde

- Dizi boyutu belirtilmediğinde, ilklendiriciler boyutu belirler.
- İlklendirici sayısı, dizinin boyutunu belirler.

- **Örnek**:

```
// 5 başlatıcı olduğu için dizi 5 elemanlı bir dizi olur.
int[] dizi = {1, 2, 3, 4, 5};
```

Örnek Kod Parçası

```
public class DiziOrnegi {
    public static void main(String[] args) {
        // Bir diziyi tanımlama
        int[] sayilar = new int[5]; // 5 elemanlı bir dizi tanımlandı
        // Diziye ilk değerleri atama
        sayilar[0] = 10;
        sayilar[1] = 20;
        sayilar[2] = 30;
        sayilar[3] = 40;
        sayilar[4] = 50;
```

Örnek Kod Parçası

```
        // Dizinin elemanlarına erişme
        System.out.println("Dizinin birinci elemanı: " + sayilar[0]);
        System.out.println("Dizinin üçüncü elemanı: " + sayilar[2]);

        // Dizinin tüm elemanlarını döngü içerisinde yazdırma
        System.out.print("Dizi Elemanları: ");
        for (int i = 0; i < sayilar.length; i++) {
            System.out.print(sayilar[i] + " ");
        }
    }
}
```

OrtalamaHesaplama

```
public class OrtalamaHesaplama {
    public static void main(String[] args) {
        // Kullanıcıdan dizi boyutunu al
        Scanner scanner = new Scanner(System.in);
        System.out.print("Dizi boyutunu girin: ");
        int boyut = scanner.nextInt();
        // Dizi oluştur ve elemanları al
        double[] dizi = new double[boyut];
        for (int i = 0; i < boyut; i++) {
            System.out.print((i + 1) + ". elemanı girin: ");
            dizi[i] = scanner.nextDouble();
        }
```

OrtalamaHesaplama

```
        // Dizi elemanlarını topla
        double toplam = 0;
        for (int i = 0; i < boyut; i++) {
            toplam += dizi[i];
        } // Ortalama hesapla
        double ortalama = toplam / boyut;
        // Sonucu yazdır
        System.out.println("Dizi Elemanları: ");
        for (int i = 0; i < boyut; i++) {
            System.out.print(dizi[i] + " ");
        }
        System.out.println("\nOrtalama: " + ortalama);
    }
}
```

OrtancaDegerHesaplama

```
public class OrtaDegerHesaplama {
    public static void main(String[] args) {
        // Kullanıcıdan dizi boyutunu al
        Scanner scanner = new Scanner(System.in);
        System.out.print("Dizi boyutunu girin: ");
        int boyut = scanner.nextInt();
        // Kullanıcıdan elemanları al ve dizi oluştur
        double[] dizi = new double[boyut];
        for (int i = 0; i < boyut; i++) {
            System.out.print((i + 1) + ". elemanı girin: ");
            dizi[i] = scanner.nextDouble();
        }
        Arrays.sort(dizi); // Diziyi sırala
```

OrtancaDegerHesaplama

```
        if (boyut % 2 == 0) {  // Çift boyutlu dizi için ortanca hesabı
            int orta1 = boyut / 2 - 1;
            int orta2 = boyut / 2;
            ortanca = (dizi[orta1] + dizi[orta2]) / 2;
        } else {  // Tek boyutlu dizi için ortanca hesabı
            int orta = boyut / 2;
            ortanca = dizi[orta];
        } // Sonucu yazdır
        System.out.println("Dizi Elemanları (sıralı): ");
        for (int i = 0; i < boyut; i++) {
            System.out.print(dizi[i] + " ");
        }
        System.out.println("\nOrtanca Değer: " + ortanca);
    }
}
```

ModHesaplama

```
public class ModHesaplama {
    public static void main(String[] args) {
        // Kullanıcıdan dizi boyutunu al
        Scanner scanner = new Scanner(System.in);
        System.out.print("Dizi boyutunu girin: ");
        int boyut = scanner.nextInt();
        // Kullanıcıdan elemanları al ve dizi oluştur
        int[] dizi = new int[boyut];
        for (int i = 0; i < boyut; i++) {
            System.out.print((i + 1) + ". elemanı girin: ");
            dizi[i] = scanner.nextInt();
        }
        Arrays.sort(dizi); // Diziyi sırala
```

ModHesaplama

```
int enCokTekrarEden = dizi[0], enCokTekrarSayisi = 1;
int mevcutTekrarEden = dizi[0], mevcutTekrarSayisi = 1;

for (int i = 1; i < boyut; i++) {
    if (dizi[i] == dizi[i - 1]) {
        mevcutTekrarSayisi++;
    } else {
        mevcutTekrarSayisi = 1;
        mevcutTekrarEden = dizi[i];
    }
    if (mevcutTekrarSayisi > enCokTekrarSayisi) {
        enCokTekrarSayisi = mevcutTekrarSayisi;
        enCokTekrarEden = mevcutTekrarEden;
    }
}
```

ModHesaplama

```
        // Sonucu yazdır
        System.out.println("Dizi Elemanları (sıralı): " +
                                                Arrays.toString(dizi));
        System.out.println("Mod Değeri: " + enCokTekrarEden +
                           " (Tekrar Sayısı: " + enCokTekrarSayisi + ")");
        scanner.close();
    }
}
```

İki Boyutlu Diziler (Matrisler)

- Satır ve sütunlarla tanımlanan tablo benzeri yapılar oluşturur.
- Matrisler, satranç tahtası veya resim gibi iki boyutlu verileri temsil etmek için kullanışlıdır.

- **Örnek**:

```
int[][] matris = new int[3][3]; // 3x3 boyutunda bir matris
```

İki Boyutlu Dizilerin İlklendirilmesi

- İki boyutlu dizilere başlangıç değerleri atanabilir.
- İlklendirme sırasında matrisin boyutu ve başlangıç değerleri belirtilmelidir.

- **Örnek**:

```
int[][] matris = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9} };
```

CokBoyutluDiziOrnegi

```
public class CokBoyutluDiziOrnegi {
    public static void main(String[] args) {
        // 3x3 boyutunda iki boyutlu bir dizi (matris) tanımlama
        int[][] matris = new int[3][3];

        // Matris elemanlarına ilk değer atama
        for (int satir = 0; satir < 3; satir++) {
            for (int sutun = 0; sutun < 3; sutun++) {
                matris[satir][sutun] = satir * 3 + sutun + 1;
            }
        }
```

CokBoyutluDiziOrnegi

```
        // Matrisi ekrana yazdırma
        System.out.println("Matrisin İçeriği:");
        for (int satir = 0; satir < 3; satir++) {
            for (int sutun = 0; sutun < 3; sutun++) {
                System.out.print(matris[satir][sutun] + " ");
            }
            System.out.println(); // Yeni satıra geç
        }
    }
}
```

Üç Boyutlu Diziler

- Java'da üç boyutlu veya daha fazla boyutlu diziler kullanılabilir.
- Daha fazla boyut, daha karmaşık veri yapılarına izin verir.

- **Örnek**:

```
// 3x3x3 boyutunda üç boyutlu dizi
int[][][] üçBoyutluDizi = new int[3][3][3];
```

UcBoyutluDiziOrnegi

```
public class UcBoyutluDiziOrnegi {
    public static void main(String[] args) {
        // 3x3x3 boyutunda üç boyutlu bir dizi tanımlama ve başlatma
        int[][][] ucBoyutluDizi = new int[3][3][3];

        // Üç boyutlu dizi elemanlarına ilk değer atama
        for (int x = 0; x < 3; x++) {
            for (int y = 0; y < 3; y++) {
                for (int z = 0; z < 3; z++) {
                    ucBoyutluDizi[x][y][z] = x * 9 + y * 3 + z + 1;
                }
            }
        }
```

UcBoyutluDiziOrnegi

```
        // Üç boyutlu diziyi ekrana yazdırma
        System.out.println("Üç Boyutlu Dizinin İçeriği:");
        for (int x = 0; x < 3; x++) {
            for (int y = 0; y < 3; y++) {
                for (int z = 0; z < 3; z++) {
                    System.out.print(ucBoyutluDizi[x][y][z] + " ");
                }
                System.out.println(); // Yeni satıra geçme (y ekseni)
            }
            System.out.println(); // Yeni satıra geçme (x ekseni)
        }
    }
}
```

Varargs (Değişken Argümanlar)

- Çok sayıda parametreyi rahatça işlemeyi sağlar.

```
double ortalama(double... dizi) {
  // Metodun içinde dizi, bir double dizisidir.
  if (dizi.length == 0) {
    return Double.NaN;
  }
  var toplam = 0.0;
  for (var eleman: dizi) {
    toplam += eleman;
  }
  return toplam / dizi.length;
}
```

Dizi Kopyalama

- Java'da diziler de bir nesnedir.
- = işleci kullanıldığında, yalnızca referanslar kopyalanır.
- Dizi elemanlarını kopyalayarak yeni bir dizi oluşturmak için, Arrays.copyOf metodu kullanılabilir.

```
int[] dizi = {1, 2, 3, 4, 5};
int[] referans = dizi; // Sadece referans kopyalanır
int[] kopya = Arrays.copyOf(dizi, dizi.length);
```

Algoritma Karmaşıklığı

- Algoritmanın çalışma süresi / kaynak kullanımını analiz eden bir ölçüdür.
- Temel olarak, bir algoritmanın ne kadar hızlı veya verimli çalıştığını belirler.
- İşlem süresinin girdi boyutuna bağlı olarak değişimini gösterir.

Zaman Karmaşıklığı - Lineer Arama

- **Örnek**: Bir dizinin içinde belirli bir elemanı arama.
- Sıradan bir doğrusal arama algoritması.

```
public boolean lineerArama(int[] dizi, int hedef) {
    for (int eleman : dizi) {
        if (eleman == hedef) {
            return true;
        }
    }
    return false;
}
```

Zaman Karmaşıklığı - Lineer Arama

- Algoritma, en kötü durumda tüm diziyi dolaşır.
- Zaman karmaşıklığı O(n) olur (n dizinin boyutunu temsil eder).

- **İyi Durum**: Hedef erken bulunur, arama hızlı biter.
- **Kötü Durum**: Hedef dizinin sonunda bulunur veya hiç bulunmaz.

Zaman Karmaşıklığı - Eşleştir

- **Örnek**: İki dizi arasında çift elemanları eşleştirme.
- İç içe iki döngü kullanarak bir çift eleman arama algoritması.

```
public void eslestir(int[] dizi1, int[] dizi2) {
    for (int eleman1 : dizi1) {
        for (int eleman2 : dizi2) {
            if (eleman1 == eleman2) {
                System.out.println("Eşleşen Çift Eleman: " + eleman1);
            }
        }
    }
}
```

Zaman Karmaşıklığı - Eşleştir

- İç içe döngüler, her bir elemanı diğer dizide bulanan her bir eleman ile sırayla karşılaştırır.
- Zaman karmaşıklığı $O(n^2)$ olur (n dizilerin boyutunu temsil eder).
- $O(n^2)$ algoritmalar büyük veri setleri için verimsizdir.

Statik Diziler

- Statik diziler, yığında (stack) oluşturulur ve otomatik yaşam/bellek süresine sahiptir.
- İşlevleri sona erdiğinde bellekten atılırlar.
- Derleme zamanında belli olan sabit bir boyuta sahiptirler.

```
String[] plaka = {"", "Adana", "Adıyaman", "Afyon"};
System.out.println(plaka[1]);
```

Dinamik Diziler

- Dinamik bellek süresine sahiptir.
- Yığın yerine, heap denilen serbest bellek üzerinde depolanır.
- Çalışma zamanında istenilen boyuta sahip olabilirler
- Bellek tahsisi ve serbest bırakma işlemi otomatik gerçekleşmez.

```
int[] dinamikDizi = new int[10]; // Bellek tahsis edilir
// ...
dinamikDizi = null; // Bellek serbest bırakılır
```

Heap ve Yığın Belleği Karşılaştırması

- Heap belleği, dinamik veri yapıları ve nesneler için kullanılır.
- Yığın belleği, yerel değişkenler ve işlev çağrıları için kullanılır.
- Bellek yönetimi heap belleği için manuel, yığın belleği için otomatiktir.
- Bellek serbest bırakma işlemi (örneğin, delete veya System.gc() kullanarak) manuel olarak yapılabilir.

İndis Sınır Dışı Hatası (IndexOutOfBoundsException)

- Dizinin belirtilen sınırlarının dışına erişildiğinde ortaya çıkar.
- Bu hatalar programın çökmesine neden olabilir.

```
// Dizi uzunluğunu kontrol etme
if (index >= 0 && index < dizi.length) {
    int deger = dizi[index];
    // indis dışı hatayı önler
} else {
    // indis dışı hatayı ele alma veya hata mesajı gösterme
}
```

Dizi Elemanına Erişme Zaman Karmaşıklığı

- O(1), sabit zaman karmaşıklığıdır.
- İşlem süresi girdi boyutundan bağımsızdır.
- Dizi elemanlarının adresi hesaplanabilir.

```
int[] dizi = {10, 20, 30, 40, 50};
// Bir dizinin belirli bir elemanına erişme
int eleman = dizi[2]; // 30
// Bu işlem O(1) karmaşıklığına sahiptir.
```

Dizi Elemanı Arama Zaman Karmaşıklığı

- O(n), işlem süresi girdi boyutu ile doğru orantılıdır.
- Çünkü, dizideki her elemanı kontrol etmek gerekebilir.

```
int[] dizi = {10, 20, 30, 40, 50};
// Bir dizide belirli bir elemanı arama
int aranan = 30;
boolean bulundu = false;

for (int eleman : dizi) {
    if (eleman == aranan) {
        bulundu = true;
        break;
    }
}
```

Dizi Sıralama Zaman Karmaşıklığı

- Sıralama işleminin zaman karmaşıklığı, kullanılan algoritmaya bağlıdır.
- Kabarcık Sıralama (Bubble Sort): $O(n^2)$
- Seçimli Sıralama (Selection Sort): $O(n^2)$
- Eklemeli Sıralama (Insertion Sort): $O(n^2)$
- Hızlı Sıralama (Quick Sort): $O(n \log n)$
- Birleştirmeli Sıralama (Merge Sort): $O(n \log n)$

Seçimli Sıralama (Selection Sort)

- Her adımda dizideki en küçük elemanı bul, dizinin başına yerleştir.

```
int[] dizi = {64, 25, 12, 22, 11};

for (int i = 0; i < dizi.length - 1; i++) {
    int minIndex = i;
    for (int j = i + 1; j < dizi.length; j++) {
        if (dizi[j] < dizi[minIndex]) {
            minIndex = j;
        }
    }
    // Minimum eleman bulundu, swap işlemi yap
    yerDegistir(dizi[minIndex], dizi[i])
}
```

Alan Karmaşıklığı

- Bir algoritmanın/veri yapısının bellek ihtiyacını ifade eder.
- Bir dizinin alan karmaşıklığı, dizinin boyutu ve eleman tipine göre değişir.
- **Örnekler**:
  - Tamsayı dizisi (int [ ]) için, her bir eleman 4 byte gerektirir
  - Karakter dizisi (char [ ]) için, her bir eleman 2 byte gerektirir (Unicode).
  - Nesne dizisi (Object [ ]) için, her bir nesne referansı 4/8 byte (32/64 bit sistemlerde) gerektirir.
- Bir tamsayı dizisi, 100 elemandan oluşuyorsa ve her tamsayı 4 byte bellek gerektiriyorsa,
  - Alan karmaşıklığı = 100 (eleman sayısı) x 4 byte = 400 byte'dır.

Matris Transpozu

Matrisin, satırlarının sütunlara ve sütunlarının satırlara dönüştürülmesi.

```
public class MatrixTranspose {
    public static void main(String[] args) {
        int[][] matris = {
            {1, 2, 3},
            {4, 5, 6}
        };
        // Orijinal matrisi yazdır
        System.out.println("Orijinal Matris:");
        yazdirMatris(matris);
```

Matris Transpozu

```
int satirSayisi = matris.length;
int sutunSayisi = matris[0].length;
// Transpoz matrisi oluştur
int[][] transpozMatris = new int[sutunSayisi][satirSayisi];

for (int i = 0; i < satirSayisi; i++) {
    for (int j = 0; j < sutunSayisi; j++) {
        transpozMatris[j][i] = matris[i][j];
    }
}
```

Matris Transpozu

```
        // Transpoz matrisi yazdır
        System.out.println("Transpoz Matris:");
        yazdirMatris(transpozMatris);
    }
    // Matrisi ekrana yazdırmak için yardımcı fonksiyon
    public void yazdirMatris(int[][] matris) {
        for (int i = 0; i < matris.length; i++) {
            for (int j = 0; j < matris[i].length; j++) {
                System.out.print(matris[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

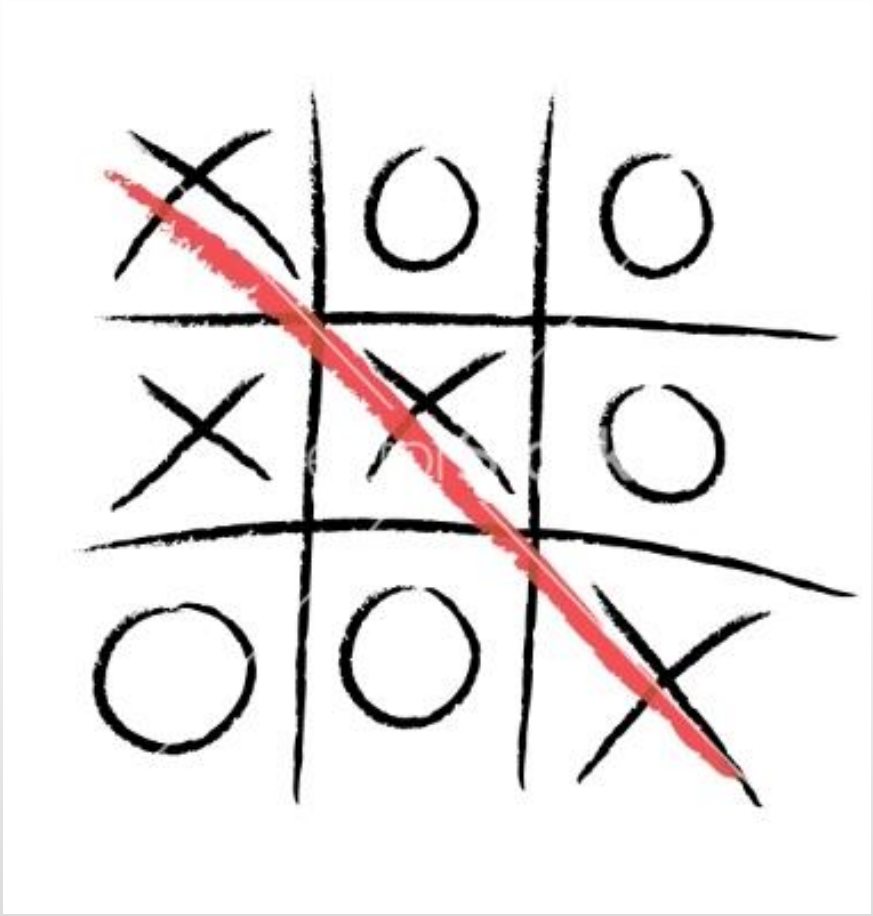
Diziyi Tersine Çevirme

```
public void tersineCevir(int[] dizi) {
    int baslangic = 0;
    int bitis = dizi.length - 1;

    while (baslangic < bitis) {
        // Baslangic ve bitis elemanlarini degistir
        int gecici = dizi[baslangic];
        dizi[baslangic] = dizi[bitis];
        dizi[bitis] = gecici;
        // Baslangic indeksi arttir, bitis indeksi azalt
        baslangic++;
        bitis--;
    }
}
```

Tic Tac Toe (X-Oyunu)

- 3x3'lük bir oyun tahtası kullanılır.
- İki oyuncuyla oynanır.
- Oyuncular sırayla hamle yapar.
- 3 elemanını yan yana ya da çapraz yerleştiren oyunu kazanır.
- Tahtada yer kalmadığında beraberlik olur.



Tic Tac Toe (X-Oyunu)

```
public class TicTacToe {
    public static void main(String[] args) {
        char[][] tahta = new char[3][3]; // 3x3'lük oyun tahtası
        char oyuncu = 'X'; // İlk oyuncu X ile başlar
        boolean oyunDevamEdiyor = true;
        tahtaDoldur(tahta); // Tahtayı başlangıç durumuyla doldur
        while (oyunDevamEdiyor) {
            tahtayiGoster(tahta);
            hamleYap(tahta, oyuncu);
            oyunDevamEdiyor = oyunDevamEdiyorMu(tahta, oyuncu);
            oyuncu = (oyuncu == 'X') ? 'O' : 'X'; // Oyuncu değişimi
        }
        tahtayiGoster(tahta);
    }
```

Tic Tac Toe (X-Oyunu)

```
public void tahtaDoldur(char[][] tahta) {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            tahta[i][j] = ' ';
        }
    }
}
```

Tic Tac Toe (X-Oyunu)

```
public void tahtayiGoster(char[][] tahta) {
    System.out.println("-------------");
    for (int i = 0; i < 3; i++) {
        System.out.print("| ");
        for (int j = 0; j < 3; j++) {
            System.out.print(tahta[i][j] + " | ");
        }
        System.out.println("\n-------------");
    }
}
```

Tic Tac Toe (X-Oyunu)

```
public void hamleYap(char[][] tahta, char oyuncu) {
    Scanner scanner = new Scanner(System.in);
    int satir, sutun;

    do {
        System.out.print("Sıra " + oyuncu + " oyuncusunda. Satır ve sütun seçin (1-3): ");
        satir = scanner.nextInt() - 1;
        sutun = scanner.nextInt() - 1;
    } while (satir < 0 || satir > 2 || sutun < 0 || sutun > 2 || tahta[satir][sutun] != ' ');

    tahta[satir][sutun] = oyuncu;
}
```

Tic Tac Toe (X-Oyunu)

```
public boolean oyunDevamEdiyorMu(char[][] tahta, char oyuncu) {
    // Kazanan durumlarını kontrol et
    if ((tahta[0][0] == oyuncu && tahta[0][1] == oyuncu && tahta[0][2] == oyuncu) ||
        (tahta[1][0] == oyuncu && tahta[1][1] == oyuncu && tahta[1][2] == oyuncu) ||
        (tahta[2][0] == oyuncu && tahta[2][1] == oyuncu && tahta[2][2] == oyuncu) ||
        (tahta[0][0] == oyuncu && tahta[1][0] == oyuncu && tahta[2][0] == oyuncu) ||
        (tahta[0][1] == oyuncu && tahta[1][1] == oyuncu && tahta[2][1] == oyuncu) ||
        (tahta[0][2] == oyuncu && tahta[1][2] == oyuncu && tahta[2][2] == oyuncu) ||
        (tahta[0][0] == oyuncu && tahta[1][1] == oyuncu && tahta[2][2] == oyuncu) ||
        (tahta[0][2] == oyuncu && tahta[1][1] == oyuncu && tahta[2][0] == oyuncu)) {
        System.out.println("Tebrikler, " + oyuncu + " oyuncusu kazandı!");
        return false;
    }
```

Tic Tac Toe (X-Oyunu)

```
// Berabere durumunu kontrol et
boolean berabere = true;
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
        if (tahta[i][j] == ' ') {
            berabere = false;
            break;
        }
    }
    if (!berabere) {
        break;
    }
}
```

Tic Tac Toe (X-Oyunu)

```
            if (berabere) {
                System.out.println("Oyun berabere bitti.");
                return false;
            }

            return true;
        }
}
```

SON