# NESNEYE YÖNELİK PROGRAMLAMA 19.10.2017

Yrd. Doç.Dr. Pelin GÖRGEL

İstanbul Üniversitesi
Bilgisayar Mühendisliği Bölümü

# Kod 1-Diziler



```
1  // Fig. 7.2: InitArray.java
2  // Initializing the elements of an array to default values of zero.
3
4  public class InitArray
5  {
6     public static void main( String[] args )
7     {
8        int[] array; // declare array named array
9
10       array = new int[ 10 ]; // create the array object
11
12       System.out.printf( "%s%8s\n", "Index", "Value" ); // column headings
13
14       // output each array element's value
15       for ( int counter = 0; counter < array.length; counter++ )
16          System.out.printf( "%5d%8d\n", counter, array[ counter ] );
17    } // end main
18 } // end class InitArray
```

Variable `array` will refer to an array of `int` values.

Creates an array of `10` `int` elements, each with the value `0` by default

`for` statement iterates while `counter` is less than the array's `length`

Array-access expression gets the value at the index represented by `counter`

**Fig. 7.2** | Initializing the elements of an array to default values of zero. (Part 1 of 2.)

# Kod 2-Diziler



```
1  // Fig. 7.3: InitArray.java
2  // Initializing the elements of an array with an array initializer.
3
4  public class InitArray
5  {
6     public static void main( String[] args )
7     {
8        // initializer list specifies the value for each element
9        int[] array = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };
10
11       System.out.printf( "%s%8s\n", "Index", "Value" ); // column headings
12
13       // output each array element's value
14       for ( int counter = 0; counter < array.length; counter++ )
15          System.out.printf( "%5d%8d\n", counter, array[ counter ] );
16    } // end main
17 } // end class InitArray
```

Array initializer list for a 10-element `int` array

**Fig. 7.3** | Initializing the elements of an array with an array initializer. (Part 1 of 2.)

# Kod 3-Diziler

(4.6)

```
1  // Fig. 7.5: SumArray.java
2  // Computing the sum of the elements of an array.
3
4  public class SumArray
5  {
6     public static void main( String[] args )
7     {
8        int[] array = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
9        int total = 0;
10
11       // add each element's value to total
12       for ( int counter = 0; counter < array.length; counter++ )
13          total += array[ counter ];
14
15       System.out.printf( "Total of array elements: %d\n", total );
16    } // end main
17 } // end class SumArray
```

Adds each value in `array` to `total`, which is displayed when the loop terminates

```
Total of array elements: 849
```

**Fig. 7.5** | Computing the sum of the elements of an array.

# Kod 4-Diziler

(4.6)

```
1  // Fig. 7.6: BarChart.java
2  // Bar chart printing program.
3
4  public class BarChart
5  {
6     public static void main( String[] args )
7     {
8        int[] array = { 0, 0, 0, 0, 0, 0, 1, 2, 4, 2, 1 };
9
10       System.out.println( "Grade distribution:" );
11
12       // for each array element, output a bar of the chart
13       for ( int counter = 0; counter < array.length; counter++ )
14       {
15          // output bar label ( "00-09: ", ..., "90-99: ", "100: " )
16          if ( counter == 10 )
17             System.out.printf( "%5d: ", 100 );
18          else
19             System.out.printf( "%02d-%02d: ",
20                counter * 10, counter * 10 + 9  );
21
```

**Fig. 7.6** | Bar chart printing program. (Part 1 of 2.)

# Kod 4

(4.6)

```
22            // print bar of asterisks
23            for ( int stars = 0; stars < array[ counter ]; stars++ )
24               System.out.print( "*" );
25
26            System.out.println(); // start a new line of output
27         } // end outer for
28      } // end main
29   } // end class BarChart
```

```
Grade distribution:
00-09:
10-19:
20-29:
30-39:
40-49:
50-59:
60-69: *
70-79: **
80-89: ****
90-99: **
  100: *
```

Nested `for` loop uses the outer `for` loop's **`counter`** variable to determine which element of the array to access, then displays the appropriate number of asterisks

**Fig. 7.6** | Bar chart printing program. (Part 2 of 2.)

# Kod 5



```
1  // Fig. 7.7: RollDie.java
2  // Die-rolling program using arrays instead of switch.
3  import java.util.Random;
4
5  public class RollDie
6  {
7     public static void main( String[] args )
8     {
9        Random randomNumbers = new Random(); // random number generator
10       int[] frequency = new int[ 7 ]; // array of frequency counters
11
12       // roll die 6,000,000 times; use die value as frequency index
13       for ( int roll = 1; roll <= 6000000; roll++ )
14          ++frequency[ 1 + randomNumbers.nextInt( 6 ) ];
15
16       System.out.printf( "%s%10s\n", "Face", "Frequency" );
17
18       // output each array element's value
19       for ( int face = 1; face < frequency.length; face++ )
20          System.out.printf( "%4d%10d\n", face, frequency[ face ] );
21    } // end main
22 } // end class RollDie
```

**Fig. 7.7** | Die-rolling program using arrays instead of `switch`. (Part 1 of 2.)

# Kod 5

```
Face Frequency
   1    999690
   2    999512
   3   1000575
   4    999815
   5    999781
   6   1000627
```

**Fig. 7.7** | Die-rolling program using arrays instead of `switch`. (Part 2 of 2.)

# Kod 6-Enhanced For

(4.6)

```
1  // Fig. 7.12: EnhancedForTest.java
2  // Using the enhanced for statement to total integers in an array.
3
4  public class EnhancedForTest
5  {
6     public static void main( String[] args )
7     {
8        int[] array = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
9        int total = 0;
10
11       // add each element's value to total
12       for ( int number : array )
13          total += number;
14
15       System.out.printf( "Total of array elements: %d\n", total );
16    } // end main
17 } // end class EnhancedForTest
```

```
Total of array elements: 849
```

**Fig. 7.12** | Using the enhanced `for` statement to total integers in an array.

# Kod 7-Dizileri Parametre Olarak Yollama



```
1  // Fig. 7.13: PassArray.java
2  // Passing arrays and individual array elements to methods.
3
4  public class PassArray
5  {
6     // main creates array and calls modifyArray and modifyElement
7     public static void main( String[] args )
8     {
9        int[] array = { 1, 2, 3, 4, 5 };
10
11       System.out.println(
12          "Effects of passing reference to entire array:\n" +
13          "The values of the original array are:" );
14
15       // output original array elements
16       for ( int value : array )
17          System.out.printf( "   %d", value );
18
19       modifyArray( array ); // pass array reference
20       System.out.println( "\n\nThe values of the modified array are:" );
21
22       // output modified array elements
23       for ( int value : array )
24          System.out.printf( "   %d", value );
```

Passes the reference to `array` into method `modifyArray`

**Fig. 7.13** | Passing arrays and individual array elements to methods. (Part 1 of 3.)

# Kod 7

(4.6)

```
25
26          System.out.printf(
27             "\n\nEffects of passing array element value:\n" +
28             "array[3] before modifyElement: %d\n", array[ 3 ] );
29
30          modifyElement( array[ 3 ] ); // attempt to modify array[ 3 ]
31          System.out.printf(
32             "array[3] after modifyElement: %d\n", array[ 3 ] );
33       } // end main
34
35       // multiply each element of an array by 2
36       public static void modifyArray( int[] array2 )
37       {
38          for ( int counter = 0; counter < array2.length; counter++ )
39             array2[ counter ] *= 2;
40       } // end method modifyArray
41
42       // multiply argument by 2
43       public static void modifyElement( int element )
44       {
45          element *= 2;
46          System.out.printf(
47             "Value of element in modifyElement: %d\n", element );
48       } // end method modifyElement
49    } // end class PassArray
```

Passes a copy of `array[3]`'s `int` value into `modifyElement`

Method receives copy of an array's reference, which gives the method direct access to the original array in memory

Method receives copy of an `int` value; the method cannot modify the original `int` value in `main`

**ig. 7.13** | Passing arrays and individual array elements to methods. (Part 2 of 3.)

# Kod 7

(4.6)

```
Effects of passing reference to entire array:
The values of the original array are:
   1   2   3   4   5

The values of the modified array are:
   2   4   6   8   10

Effects of passing array element value:
array[3] before modifyElement: 8
Value of element in modifyElement: 16
array[3] after modifyElement: 8
```

**Fig. 7.13** | Passing arrays and individual array elements to methods. (Part 3 of 3.)

# Kod 8-Diziler

(4.6)

```
1  // Fig. 7.14: GradeBook.java
2  // GradeBook class using an array to store test grades.
3
4  public class GradeBook
5  {
6     private String courseName; // name of course this GradeBook represents
7     private int[] grades; // array of student grades
8
9     // two-argument constructor initializes courseName and grades array
10    public GradeBook( String name, int[] gradesArray )
11    {
12       courseName = name; // initialize courseName
13       grades = gradesArray; // store grades
14    } // end two-argument GradeBook constructor
15
16    // method to set the course name
17    public void setCourseName( String name )
18    {
19       courseName = name; // store the course name
20    } // end method setCourseName
21
```

Will refer to an array passed by the creator of the GradeBook

Receives the array from the GradeBook creator

Initializes the grades instance variable

**Fig. 7.14** | GradeBook class using an array to store test grades. (Part 1 of 7.)

# Kod 8

(4.6)

```
22      // method to retrieve the course name
23      public String getCourseName()
24      {
25         return courseName;
26      } // end method getCourseName
27
28      // display a welcome message to the GradeBook user
29      public void displayMessage()
30      {
31         // getCourseName gets the name of the course
32         System.out.printf( "Welcome to the grade book for\n%s!\n\n",
33            getCourseName() );
34      } // end method displayMessage
35
36      // perform various operations on the data
37      public void processGrades()
38      {
39         // output grades array
40         outputGrades();
41
42         // call method getAverage to calculate the average grade
43         System.out.printf( "\nClass average is %.2f\n", getAverage() );
44
```

**Fig. 7.14** | GradeBook class using an array to store test grades. (Part 2 of 7.)

# Kod 8

(4.6)

```
45          // call methods getMinimum and getMaximum
46          System.out.printf( "Lowest grade is %d\nHighest grade is %d\n\n",
47             getMinimum(), getMaximum() );
48
49          // call outputBarChart to print grade distribution chart
50          outputBarChart();
51       } // end method processGrades
52
53       // find minimum grade
54       public int getMinimum()
55       {
56          int lowGrade = grades[ 0 ]; // assume grades[ 0 ] is smallest
57
58          // loop through grades array
59          for ( int grade : grades )
60          {
61             // if grade lower than lowGrade, assign it to lowGrade
62             if ( grade < lowGrade )
63                lowGrade = grade; // new lowest grade
64          } // end for
65
66          return lowGrade; // return lowest grade
67       } // end method getMinimum
68
```

**Fig. 7.14** | GradeBook class using an array to store test grades. (Part 3 of 7.)

# Kod 8

(4.6)

```
69      // find maximum grade
70      public int getMaximum()
71      {
72         int highGrade = grades[ 0 ]; // assume grades[ 0 ] is largest
73
74         // loop through grades array
75         for ( int grade : grades )
76         {
77            // if grade greater than highGrade, assign it to highGrade
78            if ( grade > highGrade )
79               highGrade = grade; // new highest grade
80         } // end for
81
82         return highGrade; // return highest grade
83      } // end method getMaximum
84
```

**Fig. 7.14** | GradeBook class using an array to store test grades. (Part 4 of 7.)

# Kod 8

(4.6)

```
85      // determine average grade for test
86      public double getAverage()
87      {
88         int total = 0; // initialize total
89
90         // sum grades for one student
91         for ( int grade : grades )
92            total += grade;
93
94         // return average of grades
95         return (double) total / grades.length;
96      } // end method getAverage
97
98      // output bar chart displaying grade distribution
99      public void outputBarChart()
100     {
101        System.out.println( "Grade distribution:" );
102
103        // stores frequency of grades in each range of 10 grades
104        int[] frequency = new int[ 11 ];
105
106        // for each grade, increment the appropriate frequency
107        for ( int grade : grades )
108           ++frequency[ grade / 10 ];
```

Calculation is based on the length of the array used to initialize the GradeBook

**Fig. 7.14** | GradeBook class using an array to store test grades. (Part 5 of 7.)

# Kod 8

(4.6)

```
109
110        // for each grade frequency, print bar in chart
111        for ( int count = 0; count < frequency.length; count++ )
112        {
113           // output bar label ( "00-09: ", ..., "90-99: ", "100: " )
114           if ( count == 10 )
115              System.out.printf( "%5d: ", 100 );
116           else
117              System.out.printf( "%02d-%02d: ",
118                 count * 10, count * 10 + 9  );
119
120           // print bar of asterisks
121           for ( int stars = 0; stars < frequency[ count ]; stars++ )
122              System.out.print( "*" );
123
124           System.out.println(); // start a new line of output
125        } // end outer for
126     } // end method outputBarChart
127
```

**Fig. 7.14** | GradeBook class using an array to store test grades. (Part 6 of 7.)

# Kod 8

(4.6)

```
128     // output the contents of the grades array
129     public void outputGrades()
130     {
131        System.out.println( "The grades are:\n" );
132
133        // output each student's grade
134        for ( int student = 0; student < grades.length; student++ )
135           System.out.printf( "Student %2d: %3d\n",
136              student + 1, grades[ student ] );
137     } // end method outputGrades
138  } // end class GradeBook
```

**Fig. 7.14** | `GradeBook` class using an array to store test grades. (Part 7 of 7.)

# Kod 8

```
1  // Fig. 7.15: GradeBookTest.java
2  // GradeBookTest creates a GradeBook object using an array of grades,
3  // then invokes method processGrades to analyze them.
4  public class GradeBookTest
5  {
6     // main method begins program execution
7     public static void main( String[] args )
8     {
9        // array of student grades
10       int[] gradesArray = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
11
12       GradeBook myGradeBook = new GradeBook(
13          "CS101 Introduction to Java Programming", gradesArray );
14       myGradeBook.displayMessage();
15       myGradeBook.processGrades();
16    } // end main
17 } // end class GradeBookTest
```

**Fig. 7.15** | `GradeBookTest` creates a `GradeBook` object using an array of grades, then invokes method `processGrades` to analyze them. (Part 1 of 3.)

# Kod 8

(4.6)

```
Welcome to the grade book for
CS101 Introduction to Java Programming!

The grades are:

Student  1:  87
Student  2:  68
Student  3:  94
Student  4: 100
Student  5:  83
Student  6:  78
Student  7:  85
Student  8:  91
Student  9:  76
Student 10:  87

Class average is 84.90
Lowest grade is 68
Highest grade is 100
```

**Fig. 7.15** | `GradeBookTest` creates a `GradeBook` object using an array of grades, then invokes method `processGrades` to analyze them. (Part 2 of 3.)

# Kod 8

```
Grade distribution:
00-09:
10-19:
20-29:
30-39:
40-49:
50-59:
60-69: *
70-79: **
80-89: ****
90-99: **
  100: *
```

**Fig. 7.15** | `GradeBookTest` creates a `GradeBook` object using an array of grades, then invokes method `processGrades` to analyze them. (Part 3 of 3.)

# JAVA'DA KALITIM (INHERITANCE)

# Types of inheritance

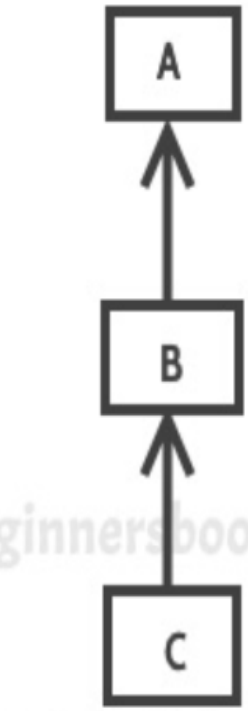
To learn types of inheritance in detail, refer: Types of Inheritance in Java.

**Single Inheritance**: refers to a child and parent class relationship where a class extends the another class.



Single Inheritance

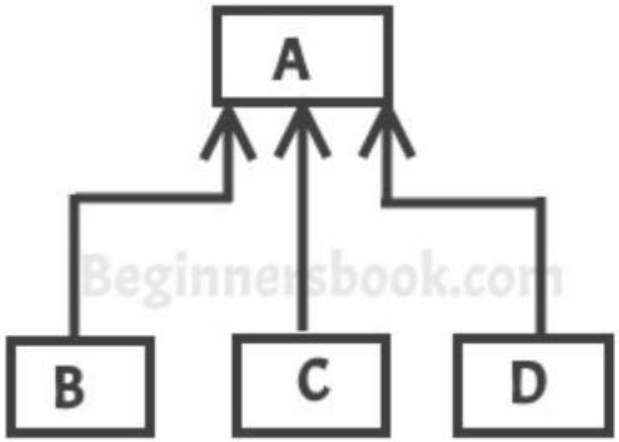
**Multilevel inheritance**: refers to a child and parent class relationship where a class extends the child class. For example class C extends class B and class B extends class A.





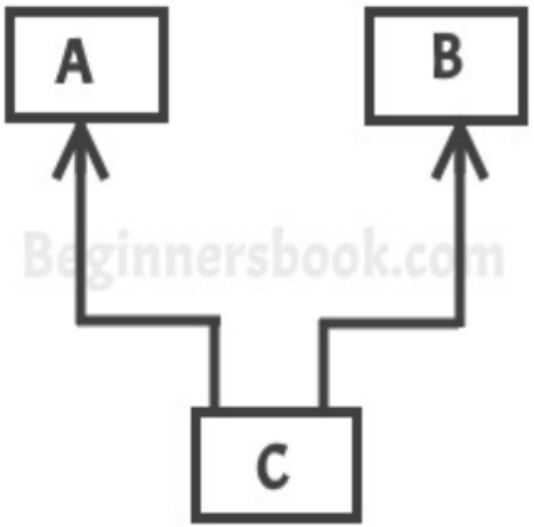
Multilevel Inheritance

**Hierarchical inheritance**: refers to a child and parent class relationship where more than one classes extends the same class. For example, classes B, C & D extends the same class A.




Hierarchical Inheritance

**Multiple Inheritance**: refers to the concept of one class extending more than one classes, which means a child class has two parent classes. For example class C extends both classes A and B. Java doesn't support multiple inheritance, read more about it here.




Multiple Inheritance

# Constructors and Inheritance

```
class ParentClass{
   //Parent class constructor
   ParentClass(){
        System.out.println("Constructor of Parent");
   }
}
class JavaExample extends ParentClass{
   JavaExample(){
        /* It by default invokes the constructor of parent class
         * You can use super() to call the constructor of parent.
         * It should be the first statement in the child class
         * constructor, you can also call the parameterized constructor
         * of parent class by using super like this: super(10), now
         * this will invoke the parameterized constructor of int arg
         */
        System.out.println("Constructor of Child");
   }
   public static void main(String args[]){
        //Creating the object of child class
        new JavaExample();
   }
}
```

Output:

```
Constructor of Parent
Constructor of Child
```

# Inheritance and Method Overriding

```
class ParentClass{
   //Parent class constructor
   ParentClass(){
        System.out.println("Constructor of Parent");
   }
   void disp(){
        System.out.println("Parent Method");
   }
}
class JavaExample extends ParentClass{
   JavaExample(){
        System.out.println("Constructor of Child");
   }
   void disp(){
        System.out.println("Child Method");
        //Calling the disp() method of parent class
        super.disp();
   }
   public static void main(String args[]){
        //Creating the object of child class
        JavaExample obj = new JavaExample();
        obj.disp();
   }
}
```

The output is :

```
Constructor of Parent
Constructor of Child
Child Method
Parent Method
```

# Kod 1:Kalıtım

```
class Ust{
   int xUst=1;
   void metodUst(){
      System.out.println("metodUst");
   }}
class Alt extends Ust{
   int xAlt=1;
   void metodAlt(){
      System.out.println("metodAlt");
   }}
public class Test {
  public static void main(String args[]) {
      Ust ust=new Ust();
      Alt alt=new Alt();
      System.out.println(ust.xUst);
      ust.metodUst();
      //System.out.println(ust.xAlt);  //ust.metodAlt();
       System.out.println(alt.xUst);
      alt.metodUst();  }}
```

*1*
*metodUst*
*1*
*metodUst*

# Kod 2: Kalıtım ve Override

```
class A{
    int xA=5;
    void metodA(){
       System.out.println("metodA");
    }
}
class B extends A{
    int xB=1;
    void metodB(){
       System.out.println("metodB");
    }
    @Override
    void metodA(){
       System.out.println("B
sınıfındaki metodA");
    }
}
```

```
public class Test {
   public static void main(String args[]) {
      A a=new A();
      B b=new B();
      System.out.println(a.xA);
      a.metodA();
    System.out.println(b.xA);
      b.metodA();
}}
```

*5*
*metodA*
*5*
*B sınıfındaki metodA*

# Kod 3:Kalıtımda Constructor

```
class B {
    int xB=10;
    B(){
       metodB();
       System.out.println("B nin constr. calisti");}
    void metodB(){
       System.out.println("metodB:"+xB);  }
    }
class C extends B {
    int xC=1;
    int xB=4;
    C() {
      System.out.println("C nin constr. calisti");
    }
    void metodC() {
       System.out.println("metodC");
    }
    @Override
    void metodB()  {
    System.out.println("C sınıfındaki metodB:"+xB);
    }  }
```

```
public class Test {
   public static void main(String args[]) {
       B b=new B();
       C c=new C();
       System.out.println(b.xB);
       b.metodB();
        System.out.println(c.xB);
        c.metodB();
}}
```

*metodB:10*
*B nin constr. calisti*

*C sınıfındaki metodB:0*
*B nin constr. calisti*
*C nin constr. calisti*

*10*
*metodB:10*
*4*
*C sınıfındaki metodB:4*

# Kod 4:Kalıtımda Constructor

```
class B {
    int xB=10;
    B(){
       metodB();
       System.out.println("B nin constr. calisti");}
    void metodB(){
       System.out.println("metodB:"+xB);  }
    }
class C extends B {
    int xC=1;
    int xB=4;
    C() {
      System.out.println("C nin constr. calisti");
    }
    void metodC() {
       System.out.println("metodC");
    }
    @Override
    void metodB()  {
        super.metodB();
       System.out.println("C sınıfındaki metodB:"+super.xB);
    } }
```

```
public class Test {
   public static void main(String args[]) {
       B b=new B();
       C c=new C();
       System.out.println(b.xB);
       b.metodB();
        System.out.println(c.xB);
        c.metodB();
}}
```

```
metodB:10
B nin constr. calisti

metodB:10
C sınıfındaki metodB:10
B nin constr. calisti
C nin constr. calisti

10
metodB:10
4
metodB:10
C sınıfındaki metodB:10
```