NESNEYE YÖNELİK PROGRAMLAMA 12.10.2017

Yrd. Doç.Dr. Pelin GÖRGEL

İstanbul Üniversitesi
Bilgisayar Mühendisliği Bölümü

Kod 10- While, If-Else Yapısı

```
1  // Fig. 4.12: Analysis.java
2  // Analysis of examination results using nested control statements.
3  import java.util.Scanner; // class uses class Scanner
4
5  public class Analysis
6  {
7     public static void main( String[] args )
8     {
9        // create Scanner to obtain input from command window
10       Scanner input = new Scanner( System.in );
11
12       // initializing variables in declarations
13       int passes = 0; // number of passes
14       int failures = 0; // number of failures
15       int studentCounter = 1; // student counter
16       int result; // one exam result (obtains value from user)
17
```

Fig. 4.12 | Analysis of examination results using nested control statements. (Part 1 of 4.)

Kod 10-Devam

```
18          // process 10 students using counter-controlled loop
19          while ( studentCounter <= 10 )
20          {
21             // prompt user for input and obtain value from user
22             System.out.print( "Enter result (1 = pass, 2 = fail): " );
23             result = input.nextInt();
24
25             // if...else is nested in the while statement
26             if ( result == 1 )           // if result 1,
27                passes = passes + 1;      // increment passes;
28             else                         // else result is not 1, so
29                failures = failures + 1; // increment failures
30
31             // increment studentCounter so loop eventually terminates
32             studentCounter = studentCounter + 1;
33          } // end while
34
```

Fig. 4.12 | Analysis of examination results using nested control statements. (Part 2 of 4.)

Kod 10-Devam

```
35          // termination phase; prepare and display results
36          System.out.printf( "Passed: %d\nFailed: %d\n", passes, failures );
37
38          // determine whether more than 8 students passed
39          if ( passes > 8 )
40             System.out.println( "Bonus to instructor!" );
41       } // end main
42    } // end class Analysis
```

```
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Passed: 9
Failed: 1
Bonus to instructor!
```

Fig. 4.12 | Analysis of examination results using nested control statements. (Part 3 of 4.)

Kod 11: Sınıf ve Nesnelere Giriş

```java
public class GradeBook
{
   // display a welcome message to the GradeBook user
   public void displayMessage()
   {
      System.out.println( "Welcome to the Grade Book!" );
   } // end method displayMessage
} // end class GradeBook
```

```java
public class GradeBookTest
{
   // main method begins program execution
   public static void main( String args[] )
   {
      // create a GradeBook object and assign it to myGradeBook
      GradeBook myGradeBook = new GradeBook();

      // call myGradeBook's displayMessage method
      myGradeBook.displayMessage();
   } // end main
} // end class GradeBookTest
```

Nesne Kurucular (Constructors)

```java
public class GradeBook
{
   private String courseName; // course name for this GradeBook

   // constructor initializes courseName with String supplied as argument
   public GradeBook( String name )
   {
      courseName = name; // initializes courseName
   } // end constructor

   // method to set the course name
   public void setCourseName( String name )
   {
      courseName = name; // store the course name
   } // end method setCourseName

   // method to retrieve the course name
   public String getCourseName()
   {
      return courseName;
   } // end method getCourseName

   // display a welcome message to the GradeBook user
   public void displayMessage()
   {
      // this statement calls getCourseName to get the
      // name of the course this GradeBook represents
      System.out.printf( "Welcome to the grade book for\n%s!\n",
         getCourseName() );
   } // end method displayMessage

} // end class GradeBook
```

Kod 12

Kod 12

```
public class GradeBookTest
{
   // main method begins program execution
   public static void main( String args[] )
   {
      // create GradeBook object
      GradeBook gradeBook1 = new GradeBook(
         "CS101 Introduction to Java Programming" );
      GradeBook gradeBook2 = new GradeBook(
         "CS102 Data Structures in Java" );

      // display initial value of courseName for each GradeBook
      System.out.printf( "gradeBook1 course name is: %s\n",
         gradeBook1.getCourseName() );
      System.out.printf( "gradeBook2 course name is: %s\n",
         gradeBook2.getCourseName() );
   } // end main

} // end class GradeBookTest
```

Kod 13

```
import java.util.Scanner; // program uses class Scanner

public class GradeBook
{
   private String courseName; // name of course this GradeBook represents

   // constructor initializes courseName
   public GradeBook( String name )
   {
      courseName = name; // initializes courseName
   } // end constructor

   // method to set the course name
   public void setCourseName( String name )
   {
      courseName = name; // store the course name
   } // end method setCourseName
```

Kod 13-Devam

```
public String getCourseName()
{
   return courseName;
} // end method getCourseName

// display a welcome message to the GradeBook user
public void displayMessage()
{
   // getCourseName gets the name of the course
   System.out.printf( "Welcome to the grade book for\n%s!\n\n",
      getCourseName() );
} // end method displayMessage

// determine class average based on 10 grades entered by user
public void determineClassAverage()
{
   // create Scanner to obtain input from command window
   Scanner input = new Scanner( System.in );

   int total; // sum of grades entered by user
   int gradeCounter; // number of the grade to be entered next
```

Kod 13-Devam

```
44          int grade; // grade value entered by user
45          int average; // average of grades
46
47          // initialization phase
48          total = 0; // initialize total
49          gradeCounter = 1; // initialize loop counter
50
51          // processing phase uses counter-controlled repetition
52          while ( gradeCounter <= 10 ) // loop 10 times
53          {
54             System.out.print( "Enter grade: " ); // prompt
55             grade = input.nextInt(); // input next grade
56             total = total + grade; // add grade to total
57             gradeCounter = gradeCounter + 1; // increment counter by 1
58          } // end while
59
60          // termination phase
61          average = total / 10; // integer division yields integer result
62
63          // display total and average of grades
64          System.out.printf( "\nTotal of all 10 grades is %d\n", total );
65          System.out.printf( "Class average is %d\n", average );
66       } // end method determineClassAverage
67    } // end class GradeBook
```

Fig. 4.6 | GradeBook class that solves class-average problem using counter-controlled repetition. (Part 3 of 3.)

Kod 13-Devam

```
1  // Fig. 4.10: GradeBookTest.java
2  // Create GradeBook object and invoke its determineClassAverage method.
3
4  public class GradeBookTest
5  {
6     public static void main( String[] args )
7     {
8        // create GradeBook object myGradeBook and
9        // pass course name to constructor
10       GradeBook myGradeBook = new GradeBook(
11          "CS101 Introduction to Java Programming" );
12
13       myGradeBook.displayMessage(); // display welcome message
14       myGradeBook.determineClassAverage(); // find average of grades
15    } // end main
16 } // end class GradeBookTest
```

Fig. 4.10 | `GradeBookTest` class creates an object of class `GradeBook` (Fig. 4.9) and invokes its `determineClassAverage` method. (Part 1 of 2.)

Kod 14

```
1  // Fig. 4.9: GradeBook.java
2  // GradeBook class that solves the class-average problem using
3  // sentinel-controlled repetition.
4  import java.util.Scanner; // program uses class Scanner
5
6  public class GradeBook
7  {
8     private String courseName; // name of course this GradeBook represents
9
10    // constructor initializes courseName
11    public GradeBook( String name )
12    {
13       courseName = name; // initializes courseName
14    } // end 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
```

Kod 14-Devam

```
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      // determine the average of an arbitrary number of grades
37      public void determineClassAverage()
38      {
39         // create Scanner to obtain input from command window
40         Scanner input = new Scanner( System.in );
41
42         int total; // sum of grades
43         int gradeCounter; // number of grades entered
```

Kod 14-Devam

```
44          int grade; // grade value
45          double average; // number with decimal point for average
46
47          // initialization phase
48          total = 0; // initialize total
49          gradeCounter = 0; // initialize loop counter
50
51          // processing phase
52          // prompt for input and read grade from user
53          System.out.print( "Enter grade or -1 to quit: " );
54          grade = input.nextInt();
55
56          // loop until sentinel value read from user
57          while ( grade != -1 )
58          {
59             total = total + grade; // add grade to total
60             gradeCounter = gradeCounter + 1; // increment counter
61
62             // prompt for input and read next grade from user
63             System.out.print( "Enter grade or -1 to quit: " );
64             grade = input.nextInt();
65          } // end while
```

Fig. 4.9 | GradeBook class that solves the class-average problem using sentinel-controlled repetition. (Part 3 of 4.)

Kod 14-Devam

```
66
67          // termination phase
68          // if user entered at least one grade...
69          if ( gradeCounter != 0 )
70          {
71             // calculate average of all grades entered
72             average = (double) total / gradeCounter;
73
74             // display total and average (with two digits of precision)
75             System.out.printf( "\nTotal of the %d grades entered is %d\n",
76                gradeCounter, total );
77             System.out.printf( "Class average is %.2f\n", average );
78          } // end if
79          else // no grades were entered, so output appropriate message
80             System.out.println( "No grades were entered" );
81       } // end method determineClassAverage
82    } // end class GradeBook
```

Fig. 4.9 | GradeBook class that solves the class-average problem using sentinel-controlled repetition. (Part 4 of 4.)

Kod 14-Devam

```
1  // Fig. 4.10: GradeBookTest.java
2  // Create GradeBook object and invoke its determineClassAverage method.
3
4  public class GradeBookTest
5  {
6     public static void main( String[] args )
7     {
8        // create GradeBook object myGradeBook and
9        // pass course name to constructor
10       GradeBook myGradeBook = new GradeBook(
11          "CS101 Introduction to Java Programming" );
12
13       myGradeBook.displayMessage(); // display welcome message
14       myGradeBook.determineClassAverage(); // find average of grades
15    } // end main
16 } // end class GradeBookTest
```

Fig. 4.10 | `GradeBookTest` class creates an object of class `GradeBook` (Fig. 4.9) and invokes its `determineClassAverage` method. (Part 1 of 2.)

Kod 15

```
1  // Fig. 5.9: GradeBook.java
2  // GradeBook class uses switch statement to count letter grades.
3  import java.util.Scanner; // program uses class Scanner
4
5  public class GradeBook
6  {
7     private String courseName; // name of course this GradeBook represents
8     // int instance variables are initialized to 0 by default
9     private int total; // sum of grades
10    private int gradeCounter; // number of grades entered
11    private int aCount; // count of A grades
12    private int bCount; // count of B grades
13    private int cCount; // count of C grades
14    private int dCount; // count of D grades
15    private int fCount; // count of F grades
16
17    // constructor initializes courseName;
18    public GradeBook( String name )
19    {
20       courseName = name; // initializes courseName
21    } // end constructor
22
```

Fig. 5.9 | `GradeBook` class uses `switch` statement to count letter grades. (Part 1 of 7.)

Kod 15-Devam

```
23      // method to set the course name
24      public void setCourseName( String name )
25      {
26         courseName = name; // store the course name
27      } // end method setCourseName
28
29      // method to retrieve the course name
30      public String getCourseName()
31      {
32         return courseName;
33      } // end method getCourseName
34
35      // display a welcome message to the GradeBook user
36      public void displayMessage()
37      {
38         // getCourseName gets the name of the course
39         System.out.printf( "Welcome to the grade book for\n%s!\n\n",
40            getCourseName() );
41      } // end method displayMessage
42
```

Fig. 5.9 | `GradeBook` class uses `switch` statement to count letter grades. (Part 2 of 7.)

Kod 15-Devam

```
43      // input arbitrary number of grades from user
44      public void inputGrades()
45      {
46         Scanner input = new Scanner( System.in );
47
48         int grade; // grade entered by user
49
50         System.out.printf( "%s\n%s\n   %s\n   %s\n",
51            "Enter the integer grades in the range 0-100.",
52            "Type the end-of-file indicator to terminate input:",
53            "On UNIX/Linux/Mac OS X type <Ctrl> d then press Enter",
54            "On Windows type <Ctrl> z then press Enter" );
55
```

Fig. 5.9 | `GradeBook` class uses `switch` statement to count letter grades. (Part 3 of 7.)

Kod 15-Devam

```
56          // loop until user enters the end-of-file indicator
57          while ( input.hasNextInt () )
58          {
59             grade = input.nextInt(); // read grade
60             total += grade; // add grade to total
61             ++gradeCounter; // increment number of grades
62
63             // call method to increment appropriate counter
64             incrementLetterGradeCounter( grade );
65          } // end while
66       } // end method inputGrades
67
```

Kod 15-Devam

```
68      // add 1 to appropriate counter for specified grade
69      private void incrementLetterGradeCounter( int grade )
70      {
71         // determine which grade was entered
72         switch ( grade / 10 )
73         {
74            case 9:  // grade was between 90
75            case 10: // and 100, inclusive
76               ++aCount; // increment aCount
77               break; // necessary to exit switch
78
79            case 8: // grade was between 80 and 89
80               ++bCount; // increment bCount
81               break; // exit switch
82
83            case 7: // grade was between 70 and 79
84               ++cCount; // increment cCount
85               break; // exit switch
86
87            case 6: // grade was between 60 and 69
88               ++dCount; // increment dCount
89               break; // exit switch
```

Fig. 5.9 | `GradeBook` class uses `switch` statement to count letter grades. (Part 5 of 7.)

Kod 15-Devam

```
90
91             default: // grade was less than 60
92                ++fCount; // increment fCount
93                break; // optional; will exit switch anyway
94             } // end switch
95          } // end method incrementLetterGradeCounter
96
97          // display a report based on the grades entered by the user
98          public void displayGradeReport()
99          {
100            System.out.println( "\nGrade Report:" );
101
102            // if user entered at least one grade...
103            if ( gradeCounter != 0 )
104            {
105               // calculate average of all grades entered
106               double average = (double) total / gradeCounter;
107
108               // output summary of results
109               System.out.printf( "Total of the %d grades entered is %d\n",
110                  gradeCounter, total );
111               System.out.printf( "Class average is %.2f\n", average );
```

Fig. 5.9 | `GradeBook` class uses `switch` statement to count letter grades. (Part 6 of 7.)

Kod 15-Devam

System.out.printf(“%s\n%s%d\n%s%d\n%s%d\n%s%d\n%s%d\n”,

```
112
113             "Number of students who received each grade:",
114             "A: ", aCount,   // display number of A grades
115             "B: ", bCount,   // display number of B grades
116             "C: ", cCount,   // display number of C grades
117             "D: ", dCount,   // display number of D grades
118             "F: ", fCount ); // display number of F grades
119       } // end if
120       else // no grades were entered, so output appropriate message
121          System.out.println( "No grades were entered" );
122    } // end method displayGradeReport
123 } // end class GradeBook
```

Fig. 5.9 | `GradeBook` class uses `switch` statement to count letter grades. (Part 7 of 7.)

Kod 15-Devam

```
1  // Fig. 5.10: GradeBookTest.java
2  // Create GradeBook object, input grades and display grade report.
3
4  public class GradeBookTest
5  {
6     public static void main( String[] args )
7     {
8        // create GradeBook object myGradeBook and
9        // pass course name to constructor
10       GradeBook myGradeBook = new GradeBook(
11          "CS101 Introduction to Java Programming" );
12
13       myGradeBook.displayMessage(); // display welcome message
14       myGradeBook.inputGrades(); // read grades from user
15       myGradeBook.displayGradeReport(); // display report based on grades
16    } // end main
17 } // end class GradeBookTest
```

Fig. 5.10 | Create `GradeBook` object, input grades and display grade report. (Part 1 of 3.)

Method	Description	Example

`abs( x )`	absolute value of x	`abs( 23.7 )` is `23.7`
		`abs( 0.0 )` is `0.0`
		`abs( -23.7 )` is `23.7`

`ceil( x )`	rounds x to the smallest integer not less than x	`ceil( 9.2 )` is `10.0` `ceil( -9.8 )` is `-9.0`
`cos( x )`	trigonometric cosine of x (x in radians)	`cos( 0.0 )` is `1.0`
`exp( x )`	exponential method e^x	`exp( 1.0 )` is `2.71828` `exp( 2.0 )` is `7.38906`
`floor( x )`	rounds x to the largest integer not greater than x	`floor( 9.2 )` is `9.0` `floor( -9.8 )` is `-10.0`
`log( x )`	natural logarithm of x (base e)	`log( Math.E )` is `1.0` `log( Math.E * Math.E )` is `2.0`
`max( x, y )`	larger value of x and y	`max( 2.3, 12.7 )` is `12.7` `max( -2.3, -12.7 )` is `-2.3`
`min( x, y )`	smaller value of x and y	`min( 2.3, 12.7 )` is `2.3` `min( -2.3, -12.7 )` is `-12.7`

Fig. 6.2 | `Math` class methods. (Part 1 of 2.)

Method	Description	Example
`pow( x, y )`	x raised to the power y (i.e., x^y)	`pow( 2.0, 7.0 )` is `128.0` `pow( 9.0, 0.5 )` is `3.0`
`sin( x )`	trigonometric sine of x (x in radians)	`sin( 0.0 )` is `0.0`
`sqrt( x )`	square root of x	`sqrt( 900.0 )` is `30.0`
`tan( x )`	trigonometric tangent of x (x in radians)	`tan( 0.0 )` is `0.0`

Fig. 6.2 | `Math` class methods. (Part 2 of 2.)

Kod 16: Random Sayı Üretme

```
1  // Fig. 6.6: RandomIntegers.java
2  // Shifted and scaled random integers.
3  import java.util.Random; // program uses class Random
4
5  public class RandomIntegers
6  {
7     public static void main( String[] args )
8     {
9        Random randomNumbers = new Random(); // random number generator
10       int face; // stores each random integer generated
11
12       // loop 20 times
13       for ( int counter = 1; counter <= 20; counter++ )
14       {
15          // pick random integer from 1 to 6
16          face = 1 + randomNumbers.nextInt( 6 );
17
18          System.out.printf( "%d  ", face ); // display generated value
19
20          // if counter is divisible by 5, start a new line of output
21          if ( counter % 5 == 0 )
22             System.out.println();
23       } // end for
24    } // end main
25 } // end class RandomIntegers
```

Fig. 6.6 | Shifted and scaled random integers. (Part 1 of 2.)

Kod 17

(4.6)

```
1  // Fig. 6.7: RollDie.java
2  // Roll a six-sided die 6,000,000 times.
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
11       int frequency1 = 0; // maintains count of 1s rolled
12       int frequency2 = 0; // count of 2s rolled
13       int frequency3 = 0; // count of 3s rolled
14       int frequency4 = 0; // count of 4s rolled
15       int frequency5 = 0; // count of 5s rolled
16       int frequency6 = 0; // count of 6s rolled
17
18       int face; // most recently rolled value
19
20       // tally counts for 6,000,000 rolls of a die
21       for ( int roll = 1; roll <= 6000000; roll++ )
22       {
23          face = 1 + randomNumbers.nextInt( 6 ); // number from 1 to 6
24
```

Fig. 6.7 | Roll a six-sided die 6,000,000 times. (Part 1 of 3.)

Kod 17

(4.6)

```
25            // determine roll value 1-6 and increment appropriate counter
26            switch ( face )
27            {
28               case 1:
29                  ++frequency1; // increment the 1s counter
30                  break;
31               case 2:
32                  ++frequency2; // increment the 2s counter
33                  break;
34               case 3:
35                  ++frequency3; // increment the 3s counter
36                  break;
37               case 4:
38                  ++frequency4; // increment the 4s counter
39                  break;
40               case 5:
41                  ++frequency5; // increment the 5s counter
42                  break;
43               case 6:
44                  ++frequency6; // increment the 6s counter
45                  break; // optional at end of switch
46            } // end switch
47         } // end for
48
```

Fig. 6.7 | Roll a six-sided die 6,000,000 times. (Part 2 of 3.)

Kod 17

(4.6)

```
49          System.out.println( "Face\tFrequency" ); // output headers
50          System.out.printf( "1\t%d\n2\t%d\n3\t%d\n4\t%d\n5\t%d\n6\t%d\n",
51             frequency1, frequency2, frequency3, frequency4,
52             frequency5, frequency6 );
53       } // end main
54    } // end class RollDie
```

```
Face    Frequency
1       999501
2       1000412
3       998262
4       1000820
5       1002245
6       998760
```

```
Face    Frequency
1       999647
2       999557
3       999571
4       1000376
5       1000701
6       1000148
```

Fig. 6.7 | Roll a six-sided die 6,000,000 times. (Part 3 of 3.)

Method Overloading (Metod Aşırı Yükleme)

```
class Calculation{
    void sum(int a,int b){
        System.out.println(a+b);}
    void sum(int a,int b,int c){
        System.out.println(a+b+c);}
    public static void main (String args[]){
  Calculation obj=new Calculation();
        obj.sum(10,10,10);
        obj.sum(20,20);
 } }
```

Kod 18

(4.6)

```
1  // Fig. 6.9: Scope.java
2  // Scope class demonstrates field and local variable scopes.
3
4  public class Scope
5  {
6     // field that is accessible to all methods of this class
7     private static int x = 1;
8
9     // method main creates and initializes local variable x
10    // and calls methods useLocalVariable and useField
11    public static void main( String[] args )
12    {
13       int x = 5; // method's local variable x shadows field x
14
15       System.out.printf( "local x in main is %d\n", x );
16
17       useLocalVariable(); // useLocalVariable has local x
18       useField(); // useField uses class Scope's field x
19       useLocalVariable(); // useLocalVariable reinitializes local x
20       useField(); // class Scope's field x retains its value
21
22       System.out.printf( "\nlocal x in main is %d\n", x );
23    } // end main
```

Fig. 6.9 | Scope class demonstrates field and local variable scopes. (Part 1 of 3.)

Kod 18

(4.6)

```
24
25      // create and initialize local variable x during each call
26      public static void useLocalVariable()
27      {
28         int x = 25; // initialized each time useLocalVariable is called
29
30         System.out.printf(
31            "\nlocal x on entering method useLocalVariable is %d\n", x );
32         ++x; // modifies this method's local variable x
33         System.out.printf(
34            "local x before exiting method useLocalVariable is %d\n", x );
35      } // end method useLocalVariable
36
37      // modify class Scope's field x during each call
38      public static void useField()
39      {
40         System.out.printf(
41            "\nfield x on entering method useField is %d\n", x );
42         x *= 10; // modifies class Scope's field x
43         System.out.printf(
44            "field x before exiting method useField is %d\n", x );
45      } // end method useField
46   } // end class Scope
```

Fig. 6.9 | Scope class demonstrates field and local variable scopes. (Part 2 of 3.)

Kod 19

```java
// Fig. 6.10: MethodOverload.java
// Overloaded method declarations.

public class MethodOverload
{
   // test overloaded square methods
   public static void main( String[] args )
   {
      System.out.printf( "Square of integer 7 is %d\n", square( 7 ) );
      System.out.printf( "Square of double 7.5 is %f\n", square( 7.5 ) );
   } // end main
```

Kod 19

```
13      // square method with int argument
14      public static int square( int intValue )
15      {
16         System.out.printf( "\nCalled square with int argument: %d\n",
17            intValue );
18         return intValue * intValue;
19      } // end method square with int argument
20
21      // square method with double argument
22      public static double square( double doubleValue )
23      {
24         System.out.printf( "\nCalled square with double argument: %f\n",
25            doubleValue );
26         return doubleValue * doubleValue;
27      } // end method square with double argument
28   } // end class MethodOverload
```

```
Called square with int argument: 7
Square of integer 7 is 49

Called square with double argument: 7.500000
Square of double 7.5 is 56.250000
```

Kod 20

```
1  // Fig. 6.3: MaximumFinder.java
2  // Programmer-declared method maximum with three double parameters.
3  import java.util.Scanner;
4
5  public class MaximumFinder
6  {
7     // obtain three floating-point values and locate the maximum value
8     public static void main( String[] args )
9     {
10       // create Scanner for input from command window
11       Scanner input = new Scanner( System.in );
12
13       // prompt for and input three floating-point values
14       System.out.print(
15          "Enter three floating-point values separated by spaces: " );
16       double number1 = input.nextDouble(); // read first double
17       double number2 = input.nextDouble(); // read second double
18       double number3 = input.nextDouble(); // read third double
19
20       // determine the maximum value
21       double result = maximum( number1, number2, number3 );
22
```

Fig. 6.3 | Programmer-declared method `maximum` with three `double` parameters. (Part 1 of 3.)

Kod 20

```
23          // display maximum value
24          System.out.println( "Maximum is: " + result );
25       } // end main
26
27       // returns the maximum of its three double parameters
28       public static double maximum( double x, double y, double z )
29       {
30          double maximumValue = x; // assume x is the largest to start
31
32          // determine whether y is greater than maximumValue
33          if ( y > maximumValue )
34             maximumValue = y;
35
36          // determine whether z is greater than maximumValue
37          if ( z > maximumValue )
38             maximumValue = z;
39
40          return maximumValue;
41       } // end method maximum
42    } // end class MaximumFinder
```

Fig. 6.3 | Programmer-declared method `maximum` with three `double` parameters. (Part 2 of 3.)

Kod 21 : Garbage Colector

```
1  // Fig. 8.12: Employee.java
2  // Static variable used to maintain a count of the number of
3  // Employee objects in memory.
4
5  public class Employee
6  {
7     private String firstName;
8     private String lastName;
9     private static int count = 0; // number of Employees created
10
11    // initialize Employee, add 1 to static count and
12    // output String indicating that constructor was called
13    public Employee( String first, String last )
14    {
15       firstName = first;
16       lastName = last;
17
18       ++count;  // increment static count of employees
19       System.out.printf( "Employee constructor: %s %s; count = %d\n",
20          firstName, lastName, count );
21    } // end Employee constructor
22
```

Fig. 8.12 | `static` variable used to maintain a count of the number of `Employee` objects in memory. (Part 1 of 2.)

Kod 21

```
23      // get first name
24      public String getFirstName()
25      {
26         return firstName;
27      } // end method getFirstName
28
29      // get last name
30      public String getLastName()
31      {
32         return lastName;
33      } // end method getLastName
34
35      // static method to get static count value
36      public static int getCount()
37      {
38         return count;
39      } // end method getCount
40   } // end class Employee
```

Fig. 8.12 | `static` variable used to maintain a count of the number of `Employee` objects in memory. (Part 2 of 2.)

Kod 21

```
1  // Fig. 8.13: EmployeeTest.java
2  // static member demonstration.
3
4  public class EmployeeTest
5  {
6     public static void main( String[] args )
7     {
8        // show that count is 0 before creating Employees
9        System.out.printf( "Employees before instantiation: %d\n",
10          Employee.getCount() );
11
12       // create two Employees; count should be 2
13       Employee e1 = new Employee( "Susan", "Baker" );
14       Employee e2 = new Employee( "Bob", "Blue" );
15
16       // show that count is 2 after creating two Employees
17       System.out.println( "\nEmployees after instantiation: " );
18       System.out.printf( "via e1.getCount(): %d\n", e1.getCount() );
19       System.out.printf( "via e2.getCount(): %d\n", e2.getCount() );
20       System.out.printf( "via Employee.getCount(): %d\n",
21          Employee.getCount() );
22
```

Fig. 8.13 | `static` member demonstration. (Part 1 of 3.)

Kod 21

```
23         // get names of Employees
24         System.out.printf( "\nEmployee 1: %s %s\nEmployee 2: %s %s\n",
25            e1.getFirstName(), e1.getLastName(),
26            e2.getFirstName(), e2.getLastName() );
27
28         // in this example, there is only one reference to each Employee,
29         // so the following two statements indicate that these objects
30         // are eligible for garbage collection
31         e1 = null;
32         e2 = null;
33      } // end main
34   } // end class EmployeeTest
```

Fig. 8.13 | `static` member demonstration. (Part 2 of 3.)

Kod 21

```
Employees before instantiation: 0
Employee constructor: Susan Baker; count = 1
Employee constructor: Bob Blue; count = 2

Employees after instantiation:
via e1.getCount(): 2
via e2.getCount(): 2
via Employee.getCount(): 2

Employee 1: Susan Baker
Employee 2: Bob Blue
```

Fig. 8.13 | `static` member demonstration. (Part 3 of 3.)