NESNEYE YÖNELİK PROGRAMLAMA
05.10.2017

Yrd. Doç.Dr. Pelin GÖRGEL

İstanbul Üniversitesi
Bilgisayar Mühendisliği Bölümü

Java'da Kavramlar

- Nesne (Object)
- Sınıf (Class)
- Soyutlama (Abstraction)
- Kalıtım (Inheritance)
- Çok Biçimlilik (Polymorphism)
- Bilgi Saklama (Encapsulation)

Kod 1

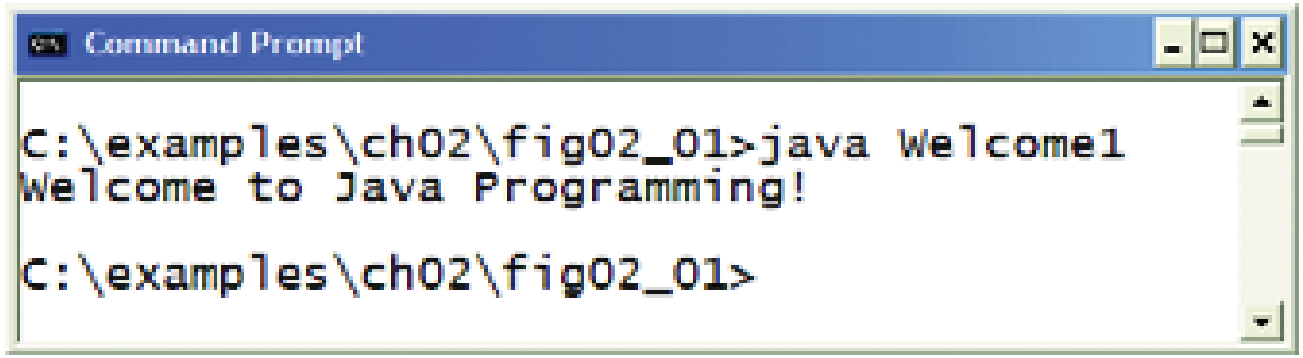
```
public class Welcome1
 {
public static void main( String args[] )
 {
 System.out.println( "Welcome to Java Programming!" );

 }
}
```

Output:Welcome to Java Programming!

Komut Satırından Çalıştırma

javac Welcome1.java



Kod 2

```
public class Welcome2
{
// main method begins execution of Java application
public static void main( String args[] )
{
 System.out.print( "Welcome to " );
System.out.println( "Java Programming!" );

} // end method main

} // end class Welcome2
```

Kod 3

- public class Welcome3
- {
- // main method begins execution of Java application
- public static void main(String args[])
- {
- System.out.println("Welcome to Java Programming!");
- } // end method main
- } // end class Welcome3

Kod 4:
İki tamsayının toplanması

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

public class Addition
{
   // main method begins execution of Java application
   public static void main( String args[] )
   {
      // create Scanner to obtain input from command window
      Scanner input = new Scanner( System.in );

      int number1; // first number to add
      int number2; // second number to add
      int sum; // sum of number1 and number2

      System.out.print( "Enter first integer: " ); // prompt
      number1 = input.nextInt(); // read first number from user

      System.out.print( "Enter second integer: " ); // prompt
      number2 = input.nextInt(); // read second number from user

      sum = number1 + number2; // add numbers

      System.out.printf( "Sum is %d\n", sum ); // display sum
   } // end method main
} // end class Addition
```

Java’da Operatör Öncelikleri

Operator	Description	Associativity
++ --	unary postfix increment unary postfix decrement	right to left
++ -- + - !	unary prefix increment unary prefix decrement unary plus unary minus unary logical negation	right to left
* / %	multiplication division remainder	left to right
+ -	addition or string concatenation subtraction	left to right
< <= > >=	less than less than or equal to greater than greater than or equal to	left to right
== !=	is equal to is not equal to	left to right
&	bitwise AND boolean logical AND	left to right
^	bitwise exclusive OR boolean logical exclusive OR	left to right

Java’da Operatör Öncelikleri - devam

Operator	Description	Associativity
|	bitwise inclusive OR boolean logical inclusive OR	left to right
&&	conditional AND	left to right
||	conditional OR	left to right
?:	conditional	right to left
= += -= *= /= %=	assignment addition assignment subtraction assignment multiplication assignment division assignment remainder assignment	right to left

Java'da Veri Tipleri

Type	Size in bits	Values
`boolean`		`true` or `false`
`char`	16	`'\u0000'` to `'\uFFFF'` (0 to 65535)
`byte`	8	–128 to +127 (-2^7 to $2^7 - 1$)
`short`	16	–32,768 to +32,767 (-2^{15} to $2^{15} - 1$)
`int`	32	–2,147,483,648 to +2,147,483,647 (-2^{31} to $2^{31} - 1$)
`long`	64	–9,223,372,036,854,775,808 to +9,223,372,036,854,775,807 (-2^{63} to $2^{63} - 1$)
`float`	32	*Negative range:* –3.4028234663852886E+38 to –1.40129846432481707e–45 *Positive range:* 1.40129846432481707e–45 to 3.4028234663852886E+38
`double`	64	*Negative range:* –1.7976931348623157E+308 to –4.94065645841246544e–324 *Positive range:* 4.94065645841246544e–324 to 1.7976931348623157E+308

Escape Dizileri

Escape sequence	Description
`\n`	Newline. Position the screen cursor at the beginning of the next line.
`\t`	Horizontal tab. Move the screen cursor to the next tab stop.
`\r`	Carriage return. Position the screen cursor at the beginning of the current line—do *not* advance to the next line. Any characters output after the carriage return overwrite the characters previously output on that line.
`\\`	Backslash. Used to print a backslash character.
`\"`	Double quote. Used to print a double-quote character. For example, `System.out.println( "\"in quotes\"" );` displays `"in quotes"`.

Kod 1:Sayı Karşılaştırma

```java
import java.util.Scanner; // program uses class Scanner
public class Comparison
{
   // main method begins execution of Java application
   public static void main( String args[] )
   {
      // create Scanner to obtain input from command window
      Scanner input = new Scanner( System.in );

      int number1; // first number to compare
      int number2; // second number to compare

      System.out.print( "Enter first integer: " ); // prompt
      number1 = input.nextInt(); // read first number from user

      System.out.print( "Enter second integer: " ); // prompt
      number2 = input.nextInt(); // read second number from user

      if ( number1 == number2 )
         System.out.printf( "%d == %d\n", number1, number2 );
      if ( number1 != number2 )
         System.out.printf( "%d != %d\n", number1, number2 );
      if ( number1 < number2 )
         System.out.printf( "%d < %d\n", number1, number2 );
      if ( number1 > number2 )
         System.out.printf( "%d > %d\n", number1, number2 );
      if ( number1 <= number2 )
         System.out.printf( "%d <= %d\n", number1, number2 );
      if ( number1 >= number2 )
         System.out.printf( "%d >= %d\n", number1, number2 );
   } // end method main
} // end class Comparison
```

Kod 2

```java
// Fig. 4.15: Increment.java
// Prefix increment and postfix increment operators.

public class Increment
{
   public static void main( String[] args )
   {
      int c;

      // demonstrate postfix increment operator
      c = 5; // assign 5 to c
      System.out.println( c );   // prints 5
      System.out.println( c++ ); // prints 5 then postincrements
      System.out.println( c );   // prints 6

      System.out.println(); // skip a line

      // demonstrate prefix increment operator
      c = 5; // assign 5 to c
      System.out.println( c );   // prints 5
      System.out.println( ++c ); // preincrements then prints 6
      System.out.println( c );   // prints 6
   } // end main
} // end class Increment
```

Uses current value, then increments c

Increments c then uses new value

Fig. 4.15 | Preincrementing and postincrementing. (Part 1 of 2.)

Kod 3

```java
// Fig. 5.1: WhileCounter.java
// Counter-controlled repetition with the while repetition statement.

public class WhileCounter
{
   public static void main( String[] args )
   {
      int counter = 1; // declare and initialize control variable

      while ( counter <= 10 ) // loop-continuation condition
      {
         System.out.printf( "%d  ", counter );
         ++counter; // increment control variable by 1
      } // end while

      System.out.println(); // output a newline
   } // end main
} // end class WhileCounter
```

Declares and initializes control variable **counter** to 1

Loop-continuation condition tests for **count**'s final value

Initializes **gradeCounter** to 1; indicates first grade about to be input

```
1  2  3  4  5  6  7  8  9  10
```

Fig. 5.1 | Counter-controlled repetition with the **while** repetition statement.

Kod 4

```java
// Fig. 5.2: ForCounter.java
// Counter-controlled repetition with the for repetition statement.

public class ForCounter
{
   public static void main( String[] args )
   {
      // for statement header includes initialization,
      // loop-continuation condition and increment
      for ( int counter = 1; counter <= 10; counter++ )
         System.out.printf( "%d  ", counter );

      System.out.println(); // output a newline
   } // end main
} // end class ForCounter
```

for statement's header contains everything you need for counter-controlled repetition

```
1  2  3  4  5  6  7  8  9  10
```

Fig. 5.2 | Counter-controlled repetition with the `for` repetition statement.

Kod 5

```java
// Fig. 5.5: Sum.java
// Summing integers with the for statement.

public class Sum
{
   public static void main( String[] args )
   {
      int total = 0; // initialize total

      // total even integers from 2 through 20
      for ( int number = 2; number <= 20; number += 2 )
         total += number;

      System.out.printf( "Sum is %d\n", total ); // display results
   } // end main
} // end class Sum
```

```
Sum is 110
```

Fig. 5.5 | Summing integers with the `for` statement.

Kod 6

```java
// Fig. 5.6: Interest.java
// Compound-interest calculations with for.

public class Interest
{
   public static void main( String[] args )
   {
      double amount; // amount on deposit at end of each year
      double principal = 1000.0; // initial amount before interest
      double rate = 0.05; // interest rate

      // display headers
      System.out.printf( "%s%20s\n", "Year", "Amount on deposit" );

      // calculate amount on deposit for each of ten years
      for ( int year = 1; year <= 10; year++ )
      {
         // calculate new amount for specified year
         amount = principal * Math.pow( 1.0 + rate, year );

```

Java treats floating-point literals as `double` values

Uses `static` method `Math.pow` to help calculate the amount on deposit

Fig. 5.6 | Compound-interest calculations with `for`. (Part 1 of 2.)

Kod 6-Devam

```
            // display the year and the amount
            System.out.printf( "%4d%,20.2f\n", year, amount );
         } // end for
      } // end main
   } // end class Interest
```

Comma in format specifier indicates that large numbers should be displayed with thousands separators

```
Year   Amount on deposit
   1            1,050.00
   2            1,102.50
   3            1,157.63
   4            1,215.51
   5            1,276.28
   6            1,340.10
   7            1,407.10
   8            1,477.46
   9            1,551.33
  10            1,628.89
```

Fig. 5.6 | Compound-interest calculations with `for`. (Part 2 of 2.)

Kod 7

```java
// Fig. 5.7: DoWhileTest.java
// do...while repetition statement.

public class DoWhileTest
{
   public static void main( String[] args )
   {
      int counter = 1; // initialize counter

      do
      {
         System.out.printf( "%d  ", counter );
         ++counter;
      } while ( counter <= 10 ); // end do...while

      System.out.println(); // outputs a newline
   } // end main
} // end class DoWhileTest
```

Condition tested at end of loop, so loop always executes at least once

```
1  2  3  4  5  6  7  8  9  10
```

Fig. 5.7 | `do...while` repetition statement.

Kod 8

```
1  // Fig. 5.12: BreakTest.java
2  // break statement exiting a for statement.
3  public class BreakTest
4  {
5     public static void main( String[] args )
6     {
7        int count; // control variable also used after loop terminates
8
9        for ( count = 1; count <= 10; count++ ) // loop 10 times
10       {
11          if ( count == 5 ) // if count is 5,
12             break;         // terminate loop
13
14          System.out.printf( "%d ", count );
15       } // end for
16
17       System.out.printf( "\nBroke out of loop at count = %d\n", count );
18    } // end main
19 } // end class BreakTest
```

Terminates the loop immediately and program control continues at line 17

```
1 2 3 4
Broke out of loop at count = 5
```

Fig. 5.12 | `break` statement exiting a `for` statement.

Kod 9

```java
// Fig. 5.13: ContinueTest.java
// continue statement terminating an iteration of a for statement.
public class ContinueTest
{
   public static void main( String[] args )
   {
      for ( int count = 1; count <= 10; count++ ) // loop 10 times
      {
         if ( count == 5 ) // if count is 5,
            continue; // skip remaining code in loop

         System.out.printf( "%d ", count );
      } // end for

      System.out.println( "\nUsed continue to skip printing 5" );
   } // end main
} // end class ContinueTest
```

Terminates current iteration of loop and proceeds to increment

```
1 2 3 4 6 7 8 9 10
Used continue to skip printing 5
```

Fig. 5.13 | `continue` statement terminating an iteration of a `for` statement.

Kod 10- While, If-Else Yapısı

```java
// Fig. 4.12: Analysis.java
// Analysis of examination results using nested control statements.
import java.util.Scanner; // class uses class Scanner

public class Analysis
{
   public static void main( String[] args )
   {
      // create Scanner to obtain input from command window
      Scanner input = new Scanner( System.in );

      // initializing variables in declarations
      int passes = 0; // number of passes
      int failures = 0; // number of failures
      int studentCounter = 1; // student counter
      int result; // one exam result (obtains value from user)

```

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

Kod 10-Devam

```
        // process 10 students using counter-controlled loop
        while ( studentCounter <= 10 )
        {
           // prompt user for input and obtain value from user
           System.out.print( "Enter result (1 = pass, 2 = fail): " );
           result = input.nextInt();

           // if...else is nested in the while statement
           if ( result == 1 )          // if result 1,
              passes = passes + 1;     // increment passes;
           else                        // else result is not 1, so
              failures = failures + 1; // increment failures

           // increment studentCounter so loop eventually terminates
           studentCounter = studentCounter + 1;
        } // end while

```

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

Kod 10-Devam

```
        // termination phase; prepare and display results
        System.out.printf( "Passed: %d\nFailed: %d\n", passes, failures );

        // determine whether more than 8 students passed
        if ( passes > 8 )
           System.out.println( "Bonus to instructor!" );
     } // end main
  } // 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
```