

Object Oriented Programming

Prepared by Dr. Yasser Abdelhamid

IT223
Object Oriented Programming

Lecture 03



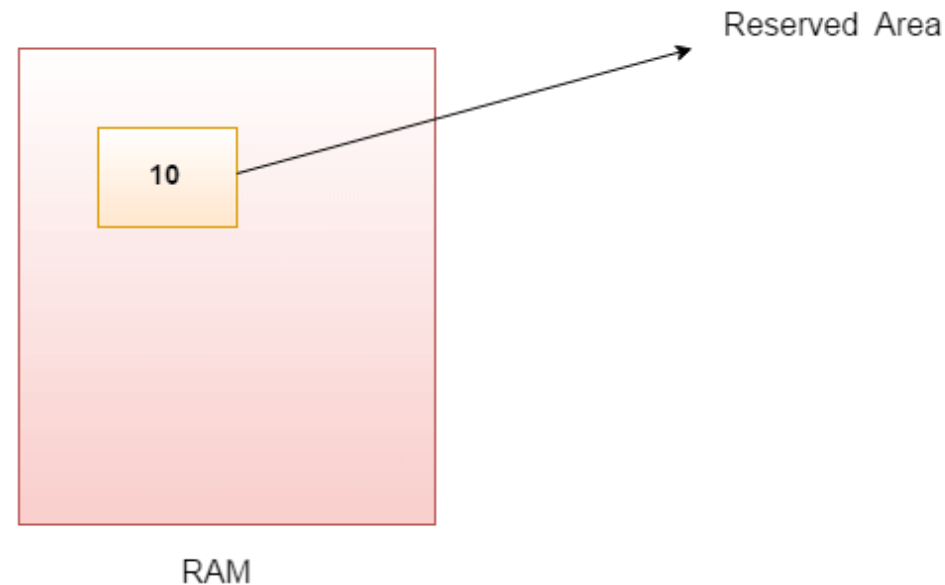
OUTLINE

- ◉ Java Programming
- ◉ Variables
- ◉ Types of variables
- ◉ Operators

VARIABLES

- ❖ A variable is a container which holds the value while the Java program is executed.
- ❖ It has name, data type, and value.

```
int data=10;
```



TYPES OF VARIABLES

- ❖ local variable
- ❖ instance variable
- ❖ static variable

```
public class A
{
    static int m=100; //static variable
    int data = 9; //instance variable
    void method1()
    {
        int n=90; //local variable
    }

    void method2()
    {
        int n=77; //local variable
    }
}
```

Ex-3-1

DATA TYPES

❖ Primitive data types - include

- byte,
- short,
- int,
- long,
- float,
- double,
- boolean and
- char

❖ Non-primitive data types - such as String, Arrays and Classes

PRIMITIVE DATA TYPES

❖ A primitive data type specifies the size and type of variable values, and it has no additional methods.

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

NON-PRIMITIVE DATA TYPES

- ❖ Primitive types are predefined (already defined) in Java.
- ❖ Non-primitive types are created by the programmer and is not defined by Java (except for String).
- ❖ Non-primitive types can be used to call methods to perform certain operations, while primitive types cannot.
- ❖ A primitive type has always a value, while non-primitive types can be **null**.
- ❖ **Conventionally**, primitive type **starts with a lowercase letter**, while **non-primitive types starts with an uppercase letter**.

WIDENING CASTING

❖ Widening Casting (**automatically**) - converting a smaller type to a larger type size

- byte -> short -> char -> int -> long -> float -> double

```
public class MyClass {  
    public static void main(String[] args) {  
        int myInt = 9;  
        double myDouble = myInt;  
        // Automatic casting: int to double  
  
        System.out.println(myInt);    // Outputs 9  
        System.out.println(myDouble); // Outputs 9.0  
    }  
}
```

Ex-3-2

NARROWING CASTING

❖ Narrowing Casting (manually) - converting a larger type to a smaller size type

- double -> float -> long -> int -> char -> short -> byte

```
public class MyClass{  
    public static void main(String[] args) {  
        double myDouble = 9.78d;  
        int myInt = (int) myDouble;  
        // Manual casting: double to int  
  
        System.out.println(myDouble); // Outputs 9.78  
        System.out.println(myInt);    // Outputs 9  
    }  
}
```

EXERCISE 4

What is the output?

```
public class FundamentalEx3 {  
    public static void main(String args[])  
    {  
        int c=0x12;//ok  
        int d=0x1E;//ok  
        int e=0X1F;//ok  
        System.out.println("c="+c);//18  
        System.out.println("d="+d);//30  
        System.out.println("e="+e);//31  
    }  
}
```

EXPLANATION:

When we prefix 0x or 0X, Java treats them as a **hexadecimal** integer literal representation

EXERCISE 7

What is the output?

```
public class FundamentalEx6 {  
    public static void main(String args[])  
    {  
        byte b1=127;//ok  
        int i1=b1;//ok  
        System.out.println("b1="+b1);  
        System.out.println("i1="+i1);  
    }  
}
```

EXPLANATION:

- ❖ The two types- int and byte are compatible and here we are putting the byte into an int means that the destination type is larger than source type.
- ❖ So, compiler is ok with this conversion.

EXERCISE 8

❖ What is the output?

```
public class FundamentalEx7 {  
    public static void main(String args[])  
    {  
  
        byte b1=127;  
        int i1=b1;//ok: small to big  
        b1=i1;//Error: big to small  
        System.out.println("b1="+b1);  
        System.out.println("i1="+i1);  
    }  
}
```

EXPLANATION

- ❖ Here the destination type is smaller than source type.
- ❖ So, compiler is raising the error.

EXERCISE 9

❖ What is the output?

```
public class FundamentalEx8 {  
    public static void main(String args[])  
    {  
        int i=2147483647;  
        System.out.println("i="+i);  
        int j=++i;  
        System.out.println("Now i is="+i);  
        System.out.println("j="+j);  
    }  
}
```

EXPLANATION

❖ The maximum value of integer is **2,147,483,647**

❖ the minimum value is **-2,147,483,648**.

Here in j (with pre increment of i), we have crossed the maximum limit of an integer.

OPERATORS

TYPES OF OPERATORS IN JAVA

- ❖ Unary Operator,
- ❖ Arithmetic Operator,
- ❖ Shift Operator,
- ❖ Relational Operator,
- ❖ Bitwise Operator,
- ❖ Logical Operator,
- ❖ Ternary Operator and
- ❖ Assignment Operator.

ARITHMETIC OPERATORS

❖ We all know arithmetic operators which are:

- + for addition
- - for subtraction
- / for division
- * for multiplication
- % for modulus (remainder of integer division)

All these operators are (**Binary operators**) because it needs two operands.

Operator precedence?

PREFIX, POSTFIX, AND INFIX NOTATION

❖ Infix notation : Operator between operands.

- $2 * 3$
- $a + b$

❖ Prefix notation: Operator before operands.

- $* 2 3$
- $+ a b$

❖ Postfix notation: Operator after operands.

- $2 3 *$
- $a b +$

UNARY OPERATORS

❖ Unary operators require only one operand

- ++ : increment the value of the variable by 1
- -- : Decrement the value of the variable by 1
- ! : Invert boolean value.

EXAMPLE

```
public class OperatorExample{  
    public static void main(String args[]){  
        boolean c=true;  
        boolean d=false;  
        System.out.println(!c);//false (opposite of boolean value)  
        System.out.println(!d);//true  
    }  
}
```

EXAMPLE

```
public class Ex9
{
    public static void main(String args[])
    {
        int i=5;
        int j=i++;
        System.out.println("j now="+j);
        System.out.println("i now="+i);
        int k=++j;
        System.out.println("j="+j);
        System.out.println("k="+k);
    }
}
```

EXPLANATION

- ❖ $++i$ is a pre-increment, that means, increment is done before accessing the variable
- ❖ $i++$ is a post-increment, that means, increment is done after accessing the current value of the variable.
- ❖ $--$ operator is treated the same.

EXAMPLE

```
public class Ex12 {  
    public static void main(String args[])  
    {  
        char c1=65;  
        char c2='a'+3;  
        System.out.println("c1="+c1++);  
        System.out.println("c2="+c2);  
        System.out.println("next to c2 =" + ++c2);  
    }  
}
```

EXPLANATION

- ❖ ASCII value of A is 65 and ASCII value of a is 97.
- ❖ $97+3=100$ which is the ASCII value of d.
- ❖ We used the increment operator to get the next character of the value of c2

EXERCISE 14

❖ What is the output?

```
public class Ex13
{
    public static void main(String args[])
    {
        int x = 10;
        int result=++x*5;
        System.out.println(" The result is : "+ result);
    }
}
```

EXERCISE 15

❖ What is the result?

```
public class Ex17 {  
    public static void main(String args[])  
    {  
        int x=24;  
        int y=11;  
        int z=100;  
        //int result= ++x * y--; //275  
        int result= ++x * --y %z;  
        System.out.println(" Result is : "+ result);  
        System.out.println(" y now : "+ y);  
    }  
}
```

EXERCISE 16

❖ What is the output?

```
public class Ex18 {  
    public static void main(String args[])  
    {  
        int x=10;  
        int y=4;  
        double result= ++y*x/y;  
        System.out.println(" Result is : "+ result);  
        System.out.println(" y now : "+ y);  
    }  
}
```

EXERCISE 17

❖ What is the output?

```
public class Ex19 {  
    public static void main(String args[])  
    {  
        int a=7,b=12;  
        System.out.println(a+b);//19  
        System.out.println("a+b=" +a+b);//a+b=712  
        System.out.println(a+b+"=a+b=" +a+b);//19=a+b=712  
    }  
}
```

SHIFT OPERATORS

- ❖ The Java left shift operator `<<` is used to shift all of the bits in a value to the left side of a specified number of times.
- ❖ The Java right shift operator `>>` is used to shift all of the bits in a value to the right side of a specified number of times.

```
public class OperatorExample{  
    public static void main(String args[]){  
        // left shift  
        System.out.println(10<<2);//10*2^2=10*4=40  
        System.out.println(10<<3);//10*2^3=10*8=80  
        System.out.println(20<<2);//20*2^2=20*4=80  
        System.out.println(15<<4);//15*2^4=15*16=240  
  
        //right shift  
        System.out.println(10>>2);//10/2^2=10/4=2  
        System.out.println(20>>2);//20/2^2=20/4=5  
        System.out.println(20>>3);//20/2^3=20/8=2  
    }  
}
```

LOGICAL OPERATORS

❖ && : Logical AND operator

- Returns true if all operands are true

❖ || : Logical OR operator

- Returns true if at least one of its operands are true.

❖ ! : Logical NOT operator

- A unary operator that returns true if its operand is false, and returns false otherwise.

LOGICAL AND OPERATOR && VS BITWISE &

❖ The logical && operator doesn't check the second condition if the first condition is false. It checks the second condition only if the first one is true.

```
public class OperatorExample{
    public static void main(String args[]){
        int a=10;
        int b=5;
        int c=20;
        System.out.println(a<b&&a++<c); //false && true = false
        System.out.println(a); //10 because second condition is not checked
        System.out.println(a<b&a++<c); //false && true = false
        System.out.println(a); //11 because second condition is checked
    }
}
```

LOGICAL || AND BITWISE |

- ❖ The logical || operator doesn't check the second condition if the first condition is true. It checks the second condition only if the first one is false.
- ❖ The bitwise | operator always checks both conditions whether first condition is true or false.

```
public class OperatorExample{  
    public static void main(String args[]){  
        int a=10;  
        int b=5;  
        int c=20;  
        System.out.println(a>b || a++<c); //true || true = true  
        System.out.println(a); //10 because second condition is not checked  
        System.out.println(a>b | a++<c); //true | true = true  
        System.out.println(a); //11 because second condition is checked  
    }  
}
```

JAVA TERNARY OPERATOR

❖ Ternary operator is a one-line conditional statement.

❖ Syntax:

- <condition> ? <if true> : <if false>

```
public class OperatorExample{  
    public static void main(String args[]){  
        int a=2;  
        int b=5;  
        int min=(a<b)?a:b;  
        System.out.println(min);  
    }  
}
```

JAVA ASSIGNMENT OPERATORS

❖ Instead of writing $x=x+1$ we write it $x+=1$

```
public class OperatorExample{
    public static void main(String[] args){
        int a=10;
        a+=3;//10+3
        System.out.println(a);
        a-=4;//13-4
        System.out.println(a);
        a*=2;//9*2
        System.out.println(a);
        a/=2;//18/2
        System.out.println(a);
    }
}
```

CODING CHALLENGE

GRADING

- ❖ Write a java program to determine the grade of the student according to his score.
 - F is less than 60.
 - D is from 60 to less than 70
 - C is from 70 to less than 80
 - B is from 80 to less than 90
 - A is from 90.
- ❖ Write this program in the least number of coding lines.

PRIZE CALCULATOR

- ❖ I decided to give prizes to my students as follows:
 - For students with grade D, I'll give them \$1
 - For students with grade D+, I'll give them as much as D students + \$5
 - For students with grade C, I'll give them as much as D+ students + \$10
 - For students with grade C+, I'll give them as much as C students + \$15
 - For students with grade B, I'll give them as much as C+ students + \$20
 - For students with grade B+, I'll give them as much as B students + \$25
 - For students with grade A, I'll give them as much as B+ students + \$30
 - For students with grade A+, I'll give them as much as A students + \$50
- ❖ Write a program to calculate the prize in the least number of coding lines.



SUMMARY

- ◉ Java Programming
- ◉ Variables
- ◉ Types of variables
- ◉ Operators



مجموعة جامعة
الجميع الذكية
للتعليم القابضة

MIDOCEAN
UNIVERSITY
BY EVERYONE'S SMART UNIVERSITY



Thank You

www.midocean.ae

