# OPERATORS

CSC 111

# 1. TYPES OF OPERATORS

- There are five types of operators
  - Assignment
  - Arithmetic
  - Increment/Decrement
  - Relational
  - Logical

# 2. ASSIGNMENT OPERATORS

- When a variable is declared, a memory space is allocated for it according to its type.
- However, it does NOT have a value yet.
- Assignment operators are used to "assign" (give) values to variables.

| | |
|---|---|
| SYNTAX | variable = literal; |
| Example 1 | x = 10; // x is previously declared |
| SYNTAX | variable1 = variable2; |
| Example 2 | x = 10; // x is previously declared<br>y = 15; // y is previouslydeclared<br>x = y; // x=15 y=15 |
| SYNTAX | variable = expression; |
| Example 3 | x = 10; // x is previously declared<br>y = 15; // y is previously declared<br>x = x * y; // x=10*15 ❾ x=150 |

# 2. ASSIGNMENT OPERATORS

## PROGRAM

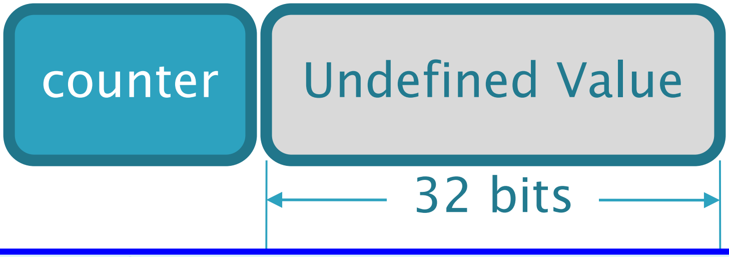
- What is the memory state of the following program:

```
public class AssignmentOperator
{
public static void main (String[] args)
      {
         // Declaration section: to declare needed variables
            int counter;
         // Input section: to enter values of used variables
            counter = 0;
         // Processing section: processing statements
            counter = counter + 1;
         // Output section: display program output
            System.out.println (“counter= “ + counter);
      } // end main
} // end class
```

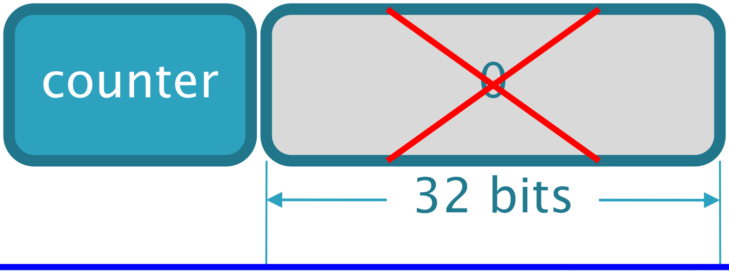
# 2. ASSIGNMENT OPERATORS

## PROGRAM – SOLUTION

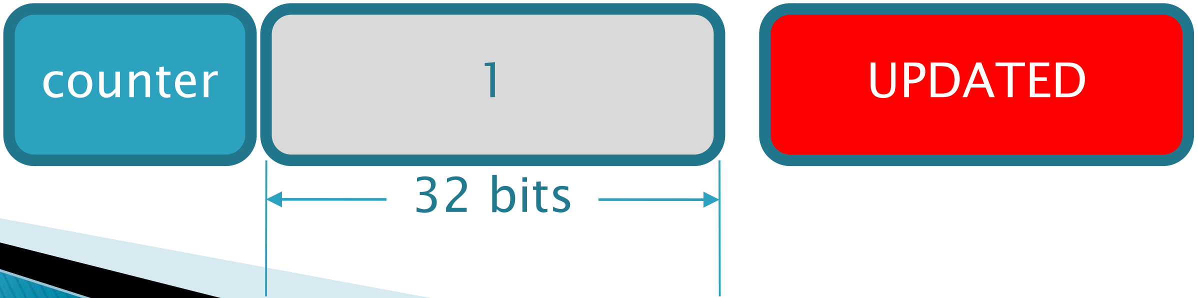
```
int counter;
```



```
counter = 0;
```



```
counter = counter + 1; // update the value of counter
```



# 3. ARITHMETIC OPERATORS

- There are 5 arithmetic operators in Java
  - Addition (+)
  - Subtraction (-)
  - Multiplication (*)
  - Division (/)
  - Modulus (%)

SYNTAX

```
variable = operand1 operator operand2;
```

Example 1

```
x = 10 + 5;        // 10 and 5 are called operands
```

Example 2

```
x = 10;
y = x * 10;        // x and 10 are the operands
```

Example 3

```
x = 10;
y = x * 10 + 5;   // x and 10 are the operands of *
                  // x*10 and 5 are the operands of +
```

# 3. ARITHMETIC OPERATORS

## DIVISION

Example 1

```
int x = 15;
int y = 2;
int z = x / y;   // z = 7 (the decimal part is truncated)
```

Example 2

```
int x = 15;
int y = 2;
double  z = x / y;   // z = 7.0 (since x & y are integers)
```

Example 3

```
int x = 15;
double y = 2.0;
double  z = x / y;   // z = 7.5 (since y is double)
```

Example 4

```
double x = 15.0;
double y = 2.0;
double  z = x / y;   // z = 7.5 (since x & y are double)
```

The division of two integers truncate the decimal part of the result

# 3. ARITHMETIC OPERATORS

## MODULUS

- Modulus is the remainder of the division of two numbers

| | |
|---|---|
| Example 1 | `x = 5 %  2;       // x = 1      (1 < 2)` |
| Example 2 | `x = 24 % 2;       // x = 0      (0 < 2)` |
| Example 3 | `x = 21 % 7;       // x = 0      (0 < 7)` |
| Example 4 | `x = 8 % 3;        // x = 2      (2 < 3)` |

**The operands of a mod operator must be of integer type**

# 3. ARITHMETIC OPERATORS

## ORDER OF PRECEDENCE

- The order of precedence of the arithmetic operators is as follows:
  - Parenthesis have the highest priority: they are evaluated from inside to outside.
  - Multiplication, Division, and Modulus have the same priority rules. They are evaluated from left to right.
  - Addition and Subtraction have the same priority rules. They are evaluated from left to right.

| | | |
|---|---|---|
| Example 1 | 3 * 7 – 6 + 2 * 5 / 4 + 6 | No parenthesis, look for * / % |
| Step 1 | 3 * 7 – 6 + 2 * 5 / 4 + 6 | Evaluate from left to right |
| Step 2 | 21 – 6 + 10 / 4 + 6 | This is an integer division |
| Step 3 | 21 – 6 + 2 + 6 | Evaluate from left to right |
| Step 4 | 15 + 2 + 6 | Evaluate from left to right |
| Step 5 | 17 + 6 | Evaluate from left to right |
| Step 6 | 23 | Final result |

This is equivalent to: (((3*7)–6)+((2*5)/4))+6

# 3. ARITHMETIC OPERATORS

## ORDER OF PRECEDENCE

| | | |
|---|---|---|
| Example 2 | 3 *(7 – 6) + 2 * 5 / (4 + 6) | Evaluate parenthesis first |
| Step 1 | 3 * 1 + 2 * 5 / 10 | Look for * / % |
| Step 2 | 3 * 1 + 2 * 5 / 10 | Evaluate from left to right |
| Step 3 | 3 + 10 / 10 | Evaluate division |
| Step 4 | 3 + 1 | Evaluate addition |
| Step 5 | 4 | Final result |

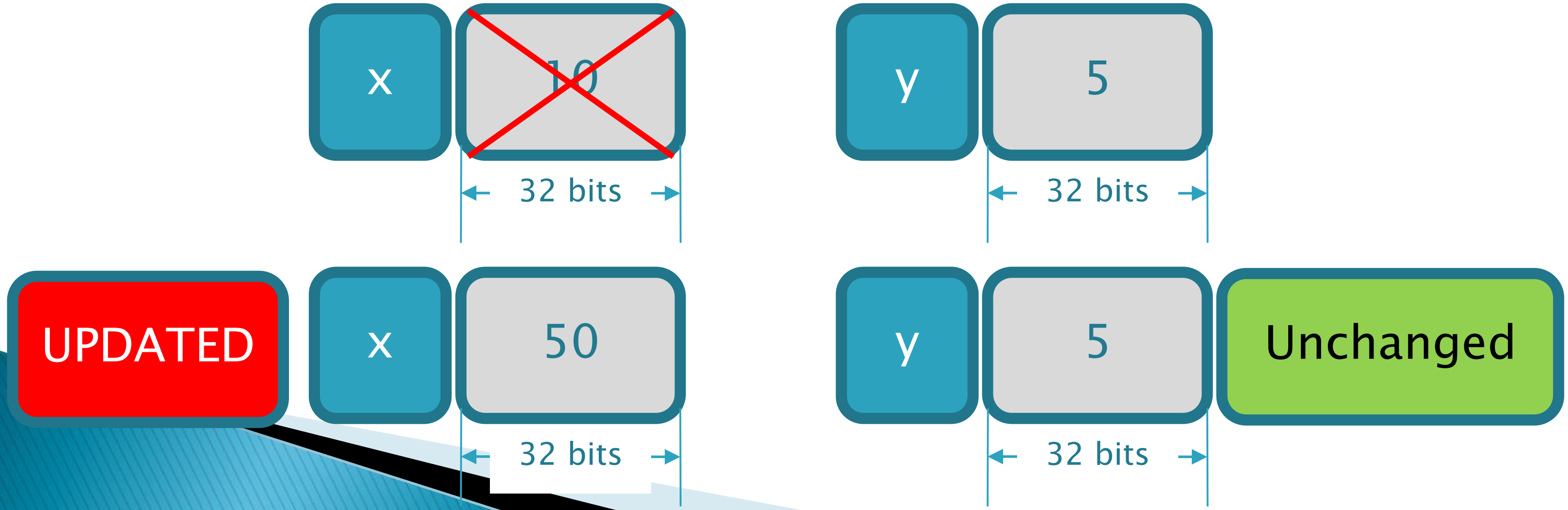
This is equivalent to: (3*(7–6))+((2*5)/(4+6))

# 4. COMPOUND OPERATORS

- Java provides five compound operators:
  - += (x += y equivalent to x = x + y)
  - -= (x -= y equivalent to x = x - y)
  - *= (x *= y equivalent to x = x * y)
  - /= (x /= y equivalent to x = x / y)
  - %= (x %= y equivalent to x = x % y)

Example 1

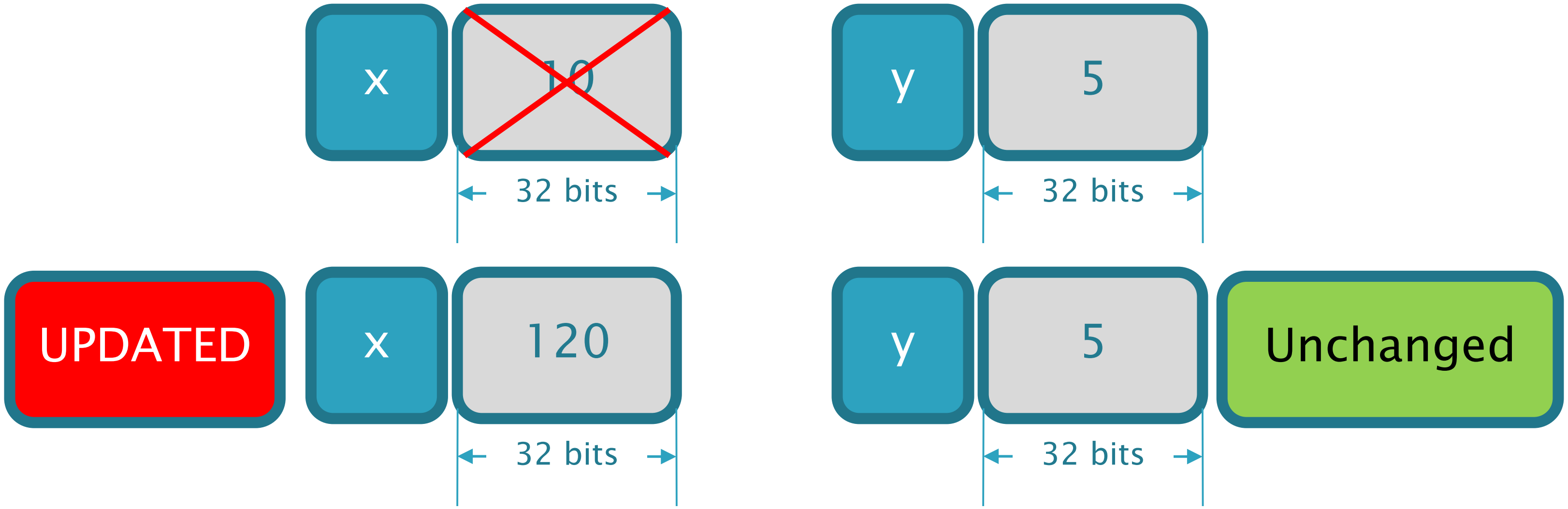
```
int x = 10;
int y = 5;
x*= y;              //x = x * y
```



# 4. COMPOUND OPERATORS

Example 2

```
int x = 10;
int y = 5;
x*= y + 7;          //x = x * (y + 7)
```



x ~~10~~

32 bits

y 5

32 bits

UPDATED

x 120

32 bits

y 5

32 bits

Unchanged

# ٥. INCREMENT/DECREMENT

- Two increment operators are used

| | | |
|---|---|---|
| Pre-increment | ++x | increment then act |
| Post-increment | x++ | act then increment |

- Two decrement operators are used

| | | |
|---|---|---|
| Pre-decrement | --x | decrement then act |
| Post-decrement | x-- | act then decrement |

# ٥. INCREMENT OPERATOR

## PRE-INCREMENT (INCREMENT THEN ACT)

SYNTAX

```
++variable;
```

Example 1

```
counter = 0;      // counter is previously declared
++counter;        // counter = counter + 1
                  // counter should previously have a value
```

After this example, counter = 1

Example 2

```
x = 3;            // x is previously declared
y = ++x;          // y is previously declared
                  // x already has a value
```

- In Example 2, (y = ++x) is equivalent to the following statements in this order:

```
1   x = x + 1;          //increment
2   y = x;              //act (assign)
```

After this example, x = 4 and y = 4

# ٥. INCREMENT OPERATOR

## PRE-INCREMENT (INCREMENT THEN ACT)

Example 3

```
a = 7;                                      // a is previously declared
System.out.println (++a);                   //increment then act (print)
```

- In Example 3, System.out.println (++a) is equivalent to the following two statements in this order:

```
1  a = a + 1;                   //increment
2  System.out.println (a);      //act (print)
```

The program output is 8

Example 4

```
a = 5;              // a is previously declared
b = ++a % 2;        // b is previously declared
                    // a has already a value
```

- In Example 4, b = ++a % 2 is equivalent to the following two statements in this order:

```
1  a = a + 1;                   //increment
2  b = a % 2;                   //act (mod)
```

After this example, a = 6 and b = 0

# ٥. INCREMENT OPERATOR

## POST-INCREMENT (ACT THEN INCREMENT)

SYNTAX

```
variable++;
```

Example 1

```
counter = 0;       // counter is previously declared
counter++;         // counter = counter + 1
                   // counter should previously have a value
```

After this example, counter = 1

Example 2

```
x = 3;             // x is previously declared
y = x++;           // y is previously declared
                   // x already has a value
```

- In Example 2, (y = x++) is equivalent to the following statements in this order:

```
1  y = x;                  //act (assign)
2  x = x + 1;              //increment
```

After this example, x = 4 and y = 3

# ٥. INCREMENT OPERATOR

## POST-INCREMENT (ACT THEN INCREMENT)

Example 3

```
a = 7;                              // a is previously declared
System.out.println (a++);           //act (print) then increment
```

- In Example 3, System.out.println (a++) is equivalent to the following two statements in this order:

```
1  System.out.println (a);    //act (print)
2  a = a + 1;
```

The program output is 7

Example 4

```
a = 5;            // a is previously declared
b = a++ % 2;      // b is previously declared
                  // a already has a value
```

- In Example 4, b = a++ % 2 is equivalent to the following two statements in this order:

```
1  b = a % 2;                 //act (mod)
2  a = a + 1;                 //increment
```

After this example, a = 6 and b = 1

# ٥. INCREMENT OPERATOR

## NOTES

- When a variable is used by itself, there is no difference between the post-increment and the pre-increment:
  - counter++;
  - ++counter;

- The following statements have all the same effect:
  - counter = counter + 1;
  - counter++;
  - ++counter;
  - counter +=1;

# ٦. DECREMENT OPERATOR

## PRE-DECREMENT (DECREMENT THEN ACT)

- The same rules of the increment operator. However, it decrements rather than increments.

SYNTAX

```
--variable;
```

Example 1

```
x = 3;          // x is previously declared
y = --x;        // y is previously declared
                // x already has a value
```

- In Example 2, (y = --x) is equivalent to the following statements in this order:

```
1  x = x - 1;          //decrement
2  y = x;              //act (assign)
```

After this example, x = 2 and y = 2

# ٦. DECREMENT OPERATOR

## POST-DECREMENT (ACT THEN DECREMENT)

SYNTAX

```
variable--;
```

Example 1

```
x = 3;          // x is previously declared
y = x--;        // y is previously declared
                // x already has a value
```

- In Example 2, (y = x--) is equivalent to the following statements in this order:

```
1  y = x;          //act (assign)
2  x = x -1;       //decrement
```

After this example, x = 2 and y = 3

# ٦. DECREMENT OPERATOR

## NOTES

- When a variable is used by itself, there is no difference between the post-increment and the pre-increment:
  - counter--;
  - --counter;

- The following statements have all the same effect:
  - counter = counter – 1;
  - counter--;
  - --counter;
  - counter -=1;

# ٧. RELATIONAL OPERATORS

- Relational operators are used to compare items.
- Java uses the following relational operators:

| | |
|---|---|
| == | equal to |
| != | not equal to |
| < | less than |
| <= | less than or equal |
| > | greater than |
| >= | greater than or equal |

- Logical expressions use relational operators.
- Logical expressions evaluate to either true or false.
- The result of a logical expression is stored in a variable of type boolean.

```
int x = 10, y = 15, z = 10;
char ch1 = ‘a’, ch2 = ‘z’; //Unicode(‘a’) = 97, Unicode(‘z’) = 122
boolean result;
result = (x < y);          //result = true
result = (x <= y);         //result = true
result = (ch1 >= ch2);     //result = false since Unicode of ‘a’ is less
result = (x != z);         //result = false
result = (x > y);          //result = false
```

# ٨. LOGICAL OPERATORS

- Logical operators are used to construct compound logical expressions.
- Java uses the following logical operators:

| | |
|---|---|
| ! | not |
| && | and |
| \|\| | or |

- Logical operators take only boolean values as operands.
- The logical expressions evaluate to either true or false according to the following truth tables:

| NOT | |
|---|---|
| Operand | Result |
| true | false |
| false | true |

| AND | | |
|---|---|---|
| Operand1 | Operand2 | Result |
| true | true | true |
| true | false | false |
| false | true | false |
| false | false | false |

| OR | | |
|---|---|---|
| Operand1 | Operand2 | Result |
| true | true | true |
| true | false | true |
| false | true | true |
| false | false | false |

# ٨. LOGICAL OPERATORS

## EXAMPLE

```
int x = 24, y = 35, z = 20;
char ch1  = ‘a’;                          //Unicode (‘a’) = 97
char ch2  = ‘A’;                          //Unicode (‘A’) = 65
char ch3  = ‘<‘;                          //Unicode (‘<‘) = 60
char ch4  = ‘5’;                          //Unicode (‘5’) = 53
boolean result;
result = (x >= y) && (ch1  < ch3);// false && false ➔ result = false
result = (ch2 == ch4) || (x > z); // false || true ➔ result = true
result = !(ch1  < ch2);                   // !(false) ➔ true
```

The complete Unicode table is in lecture W2.2 Identifiers, slide 11

# ٩. ORDER OF PRECEDENCE

| | |
|---|---|
| Parenthesis ( ) | inside-out |
| Increment (++), Decrement (--) | from left to right |
| * / % | from left to right |
| + - | from left to right |
| < > <= >= | from left to right |
| == != | from left to right |
| && | from left to right |
| \|\| | from left to right |
| = += -= *= /= %= | from left to right |

# ١٠ . TYPE CASTING

- Type casting is the conversion from a data type to another.

SYNTAX

(dataTypeName) expression

Examples

```
(int) (7.9);                              // = 7
(double) (25);                            // = 25.0
(double) (5 + 3);                         // = (double) (8) = 8.0
(double) (15) / 2;                        // = 15.0 / 2   = 7.5
(double) (15 / 2);                        // = (double) (7)   = 7.0
(int) (7.8 + (double) (15) / 2);
                                  //=(int)(7.8+15.0/2)=(int)(7.8+7.5) = 15
(int) (7.8 + (double) (15 / 2));          //(int)(7.8+7.0) = 14
```

- In Java, arithmetic expressions may have mixed data types. In this case, Java performs implicit type coercion (automatic casting). However, implicit type coercion may generate unexpected results.

Avoid using expressions with mixed data types without explicit type coercion (explicit type casting).

# ١٠ . TYPE CASTING

Examples

```
char ch = ‘a’;                //Unicode of ‘a’ = 97
int unicode;
unicode = (int)(ch);          //unicode = 97
System.out.println (unicode);
```

```
97
```

Examples

```
int x = 98;
char ch;
ch = (char)(x);
System.out.println (ch);
```

```
b
```

# Self-Check Exercises (1)

- What is the memory state of the following program

```
public class Accumulator
{
  public static void main (String[] args)
    {
          int a, sum;
          a = 10;
          sum = 0;
          sum = sum + a;
          System.out.println (“sum = “ + sum);
    }
}
```

- Evaluate the following expressions:
  - 23 + 7 % 2 – 3
  - 15.0 + 3.0 * 2.0 / 5.0
  - 30.0 % 6 + 1

# Self-Check Exercises (2)

- Given x=5, y=7 and z=10. Evaluate the following expressions:
  - y *= 2 * x + 5 – z;
  - z %= x;

- Write a program that exchanges the values of two numbers. (This is known as swapping). For example, if x = 5 and y = 7; after swapping, they become x = 7 and y = 5.

# Self-Check Exercises (1)

- What is the output of the following program

```
public class Increment
{
   public static void main (String[] args)
      {
            int sum = 7;
            System.out.println (sum);
            System.out.println (sum++);
            System.out.println (sum);
            System.out.println (++sum);
            System.out.println (sum);
      }
}
```

# Self-Check Exercises (2)

- Assume x, y, and z are int variables; with x = 9, y = 10, and z = 8. What are the values of the three values after the execution of each of the following statements:
  - x++;
  - System.out.println (--y);
  - z *= ++x;

- Assume x = 7 and y = 9, and z = 22.2. Evaluate the following expressions accordingly:
  - total = x + y + int(z);
  - z /= x;
  - y += (int) z – x;
  - x *= 2 * y + (int) z;

# Self-Check Exercises (3)

- Trace the following statements by filling the table below:

```
1   public class Trace
2   {
3   public static void main (String[] args)
4       {
5           int firstNum;
6           int secondNum;
7           char ch;
8           double z;
9           firstNum = 4;
10          secondNum = 2 + firstNum * 6;
11          z = (firstNum + 1) / 2.0;
12          ch = ‘A’;
13          firstNum = (int) (z) + 8;
14          firstNum = secondNum--;
15      }
16  }
```

| Line | firstNum | secondNum | ch | z |
|---|---|---|---|---|