Dr. Faheem Akhtar Rajput

# Object Oriented Programming (JAVA)

Lecture 10 and 11

# Operators

- Arithmetic
- Bitwise
- Relational
- Logical

# Arithmetic Operators

| Operator | Result |
|---|---|
| + | Addition |
| – | Subtraction (also unary minus) |
| * | Multiplication |
| / | Division |
| % | Modulus |
| ++ | Increment |
| += | Addition assignment |
| –= | Subtraction assignment |
| *= | Multiplication assignment |
| /= | Division assignment |
| %= | Modulus assignment |
| – – | Decrement |

# Example - 1

```
// Demonstrate the basic arithmetic operators.
class BasicMath {
    public static void main(String args[]) { // arithmetic using integers
        System.out.println("Integer Arithmetic");
        int a = 1 + 1;     int b = a * 3;        int c = b / 4;       int d = c - a;     int e = -d;
        System.out.println("a = " + a);
        System.out.println("b = " + b);
        System.out.println("c = " + c);
        System.out.println("d = " + d);
        System.out.println("e = " + e);
        // arithmetic using doubles
        System.out.println("\nFloating Point Arithmetic");
        double da = 1 + 1; double db = da * 3; double dc = db / 4;
        double dd = dc - a; double de = -dd;
        System.out.println("da = " + da);        System.out.println("db = " + db);
        System.out.println("dc = " + dc);        System.out.println("dd = " + dd);
        System.out.println("de = " + de);
  }
}
```

# Arithmetic Operators

- Modulus operator:
  - %, returns the remainder of a division operation

    X=40.2; X % 10;
- Arithmetic Compound Assignment Operators
  - combine an arithmetic operation with an assignment

    *var = var op expression;*

    a = a + 4;

    a += 4;
- Increment and Decrement

  x = x + 1;  This can also rewrite as:  : x++;

  x = x - 1;  x--;

```
x = 42;

y = ++x;
```

**Same as**

```
x = x + 1;
y = x;
```

**However**

```
x = 42;
y = x++;
```

**Same as:**

```
y = x;
x = x + 1;
```

# Example - 2

```
// Demonstrate ++.
class IncDec {
   public static void main(String args[]) {
      int a = 1; int b = 2; int c; int d;
      c = ++b;
      d = a++;
      c++;
      System.out.println("a = " + a);
      System.out.println("b = " + b);
      System.out.println("c = " + c);
      System.out.println("d = " + d);
   }
}
```

**OUTPUT**

a= 2
b= 3
c= 4
d= 1

# Bitwise Operators

**Java defines several bitwise operators that can be applied to the integer types, long, int, short, char, and byte.**

| Operator | Result |
|---|---|
| ~ | Bitwise unary NOT |
| & | Bitwise AND |
| \| | Bitwise OR |
| ^ | Bitwise exclusive OR |
| >> | Shift right |
| >>> | Shift right zero fill |
| << | Shift left |
| &= | Bitwise AND assignment |
| \|= | Bitwise OR assignment |
| ^= | Bitwise exclusive OR assignment |
| >>= | Shift right assignment |
| >>>= | Shift right zero fill assignment |
| <<= | Shift left assignment |

# Logical Operations

- Bitwise logical operators are &, |, ^, and ~.

| A | B | A \| B | A & B | A ^ B | ~A |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 |

```
// Demonstrate the bitwise logical operators.
class BitLogic {
  public static void main(String args[]) {
    String binary[] = {
      "0000", "0001", "0010", "0011", "0100", "0101", "0110", "0111",
      "1000", "1001", "1010", "1011", "1100", "1101", "1110", "1111"
    };
    int a = 3; // 0 + 2 + 1 or 0011 in binary
    int b = 6; // 4 + 2 + 0 or 0110 in binary
    int c = a | b;
    int d = a & b;
    int e = a ^ b;
    int f = (~a & b) | (a & ~b);
    int g = ~a & 0x0f;

    System.out.println("      a = " + binary[a]);
    System.out.println("      b = " + binary[b]);
    System.out.println("    a|b = " + binary[c]);
    System.out.println("    a&b = " + binary[d]);
    System.out.println("    a^b = " + binary[e]);
    System.out.println("~a&b|a&~b = " + binary[f]);
    System.out.println("     ~a = " + binary[g]);
  }
}
```

**OUTPUT**

```
      a = 0011
      b = 0110
    a|b = 0111
    a&b = 0010
    a^b = 0101
~a&b|a&~b = 0101
     ~a = 1100
```

# Shift << Left

```
// Left shifting a byte value.
class ByteShift {
  public static void main(String args[]) {
    byte a = 64, b;
    int i;

    i = a << 2;
    b = (byte) (a << 2);

    System.out.println("Original value of a: " + a);
    System.out.println("i and b: " + i + " " + b);
  }
}
```

**OUTPUT**

Original value of a: 64
i and b: 256 0

# Relational Operators

- The relational operators determine the relationship that one operand has to the other.

| Operator | Result |
|---|---|
| == | Equal to |
| != | Not equal to |
| > | Greater than |
| < | Less than |
| >= | Greater than or equal to |
| <= | Less than or equal to |

# Valid & Not Valid in JAVA

```
int done;
// ...
if(!done) ... // Valid in C/C++
 if(done) ...          // but not in Java.
```

- In Java, these statements must be written like this:

```
if(done == 0) ... // This is Java-style.
if(done != 0) ...
```

# Boolean Logical Operators

| Operator | Result |
|---|---|
| & | Logical AND |
| \| | Logical OR |
| ^ | Logical XOR (exclusive OR) |
| \|\| | Short-circuit OR |
| && | Short-circuit AND |
| ! | Logical unary NOT |
| &= | AND assignment |
| \|= | OR assignment |
| ^= | XOR assignment |
| == | Equal to |
| != | Not equal to |
| ?: | Ternary if-then-else |

# Boolean Logical Operators

- The logical Boolean operators, &, |, and ^, operate on boolean values in the same way that they operate on the bits of an integer.

| A | B | A \| B | A & B | A ^ B | !A |
|---|---|---|---|---|---|
| False | False | False | False | False | True |
| True | False | True | False | True | False |
| False | True | True | False | True | True |
| True | True | True | True | False | False |

# Example

```
// Demonstrate the boolean logical operators.
class BoolLogic {
  public static void main(String args[]) {
    boolean a = true;
    boolean b = false;
    boolean c = a | b;
    boolean d = a & b;
    boolean e = a ^ b;
    boolean f = (!a & b) | (a & !b);
    boolean g = !a;

    System.out.println("       a = " + a);
    System.out.println("       b = " + b);
    System.out.println("     a|b = " + c);
    System.out.println("     a&b = " + d);
    System.out.println("     a^b = " + e);
    System.out.println("!a&b|a&!b = " + f);
    System.out.println("      !a = " + g);
  }
}
```

**OUTPUT**

```
       a = true
       b = false
     a|b = true
     a&b = false
     a^b = true
a&b|a&!b = true
      !a = false
```

# Short circuit (&&)

- Java will not bother to evaluate the right-hand operand when the outcome of the expression can be determined by the left operand alone.

```
if (denom != 0 && num / denom > 10)
```

- There is no risk of causing a run-time exception when denom is zero.

# The ? Operator

- ternary (three-way) operator that can replace certain types of **if-then-else** statements.

**expression1 ? expression2 : expression3**

- Expression that evaluates to a boolean value. If expression1 is true, then expression2 is evaluated; otherwise, expression3 is evaluated

# Example

```
// Demonstrate ?.
class Ternary {
  public static void main(String args[]) {
    int i, k;

    i = 10;
    k = i < 0 ? -i : i; // get absolute value of i
    System.out.print("Absolute value of ");
    System.out.println(i + " is " + k);

    i = -10;
    k = i < 0 ? -i : i; // get absolute value of i
    System.out.print("Absolute value of ");
    System.out.println(i + " is " + k);
  }
}
```

**OUTPUT**

```
Absolute value of 10 is 10
Absolute value of -10 is 10
```

# Control Statements

# Three Categories

- Java's control statements can be put into 3 categories:
  - Selection
  - Iteration
  - Jump

# Selection Statements

- Control the flow of the program only during run time.
  - if
  - switch

# If Statement

- Discussed in chapter 2, but here the power is discussed in detail:

```
if (condition) statement1;
else statement2;
```

Each statement may be a single statement or a compound statement (Block)

The if works like this: If the condition is true, then statement1 is executed. Otherwise, statement2 (if it exists) is executed.

In no case will both statements be executed.

```
int a, b;
// ...
if(a < b) a = 0;
else b = 0;
```

# If with Relational Op. & Boolean variables

- Most often, the expression used to control the if will involve the relational operators. However, this is not technically necessary. It is possible to control the if using a single boolean variable, as shown in this code fragment:

```
boolean dataAvailable;
// ...
if (dataAvailable)
    ProcessData();
else
   waitForMoreData();
```

```
int bytesAvailable;
// ...
if (bytesAvailable > 0) {
    ProcessData();
    bytesAvailable -= n;
} else
   waitForMoreData();
```

# Nested ifs

- A nested if is an if statement that is the target of another if or else.

```
if(i == 10) {
   if(j < 20)        a = b;
   if(k > 100)      c = d; // this if is
   else a = c; // associated with this else
}
else a = d;    // this else refers to if(i == 10)
```

**Ladder of if-else**

```
if(condition)
    statement;
else if(condition)
    statement;
else if(condition)
    statement;
.
.
.
else
statement;
```

# Example

```
// Demonstrate if-else-if statements.
class IfElse {
  public static void main(String args[]) {
    int month = 4; // April
    String season;

    if(month == 12 || month == 1 || month == 2)
      season = "Winter";
    else if(month == 3 || month == 4 || month == 5)
      season = "Spring";
    else if(month == 6 || month == 7 || month == 8)
      season = "Summer";
    else if(month == 9 || month == 10 || month == 11)
      season = "Autumn";
    else
      season = "Bogus Month”;
    System.out.println("April is in the " + season + ".");
  }
}
```

**OUTPUT**

April is in the Spring.

# switch

- Multiway branch statement.
- A better alternative than a large series of if-else-if statements.

```
switch (expression) {
case value1:
    // statement sequence
    break;
case value2:
    // statement sequence
    break;
.
.
.
case valueN:
    // statement sequence
    break;
default:
    // default statement sequence
}
```

```
// An improved version of the season program.
class Switch {
  public static void main(String args[]) {
    int month = 4;
    String season;
    switch (month) {
      case 12:
      case 1:
      case 2:
        season = "Winter";
        break;
      case 3:
      case 4:
      case 5:
        season = "Spring";
        break;
      case 6:
      case 7:
      case 8:
        season = "Summer";
        break;
      case 9:
      case 10:
      case 11:
        season = "Autumn";
        break;
      default:
        season = "Bogus Month";
    }
    System.out.println("April is in the " + season + ".");
  }
}
```

# Nested switch Statements

```
switch(count) {
      case 1:
           switch(target) { // nested switch
                case 0:
                     System.out.println("target is zero");
                     break;
                case 1: // no conflicts with outer switch
                          System.out.println("target is one");
                     break;
           }
      break;
case 2: // ...
```

# Question?

Sukkur IBA

Thanks…