CSE310: Programming in Java

Topic: Operators in Java

Outlines

- Introduction
- Assignment Operator
- Arithmetic Operator
- Relational Operator
- Bitwise Operator
- Conditional Operator
- Unary Operator

Introduction

- Operators are special symbols that perform specific operations on one, two, or three operands, and then return a result.

Assignment Operator

- One of the most common operators is the simple assignment operator "=".
- This operator assigns the value on its right to the operand on its left.

Example:

```
int salary = 25000;          double speed = 20.5;
```

Arithmetic Operators

- Java provides operators that perform addition, subtraction, multiplication, and division.

Operator	Description
+	**Additive operator (also used for String concatenation)**
-	**Subtraction operator**
*	**Multiplication operator**
/	**Division operator**
%	**Remainder operator**

Example of arithmetic operators

```
// To show the working of arithmetic operators
class Example
    {
      public static void main(String args[])
      {
      int a=10;
      int b=5;
      System.out.println(a+b);//15
      System.out.println(a-b);//5
      System.out.println(a*b);//50
      System.out.println(a/b);//2
      System.out.println(a%b);//0
      }
    }
```

Compound Assignments

- Arithmetic operators are combined with the simple assignment operator to create compound assignments.
- Compound assignment operators are +=, -=, *=, /=, %=
- For example, x+=1; and x=x+1; both increment the value of x by 1.

Relational Operators

- Relational operators determine if one operand is greater than, less than, equal to, or not equal to another operand.

- It always returns boolean value i.e true or false.

Relational Operators

Operator	Description
==	equal to
!=	not equal to
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to

Example of relational operator

```
// To show the working of relational operators
class Example
    {
      public static void main(String args[])
      {
      int a=10;
      int b=5;
      System.out.println(a>b);//true
      System.out.println(a<b);//false
      System.out.println(a==b);//false
      System.out.println(a!=b);//true
      }
    }
```

CODERINDEED

Unary Operators

➢ The unary operators require only one operand.

Operator	Description
+	Unary plus operator; indicates positive value
-	Unary minus operator; negates an expression
++	Increment operator; increments a value by 1
--	Decrement operator; decrements a value by 1
!	Logical complement operator; inverts the value of a boolean

Examples

```
// To show the working of  ++ and -- operator
class Example
       {
         public static void main(String args[])
         {
         int x=10;
         System.out.println(x++);//10 (11)
         System.out.println(++x);//12
         System.out.println(x--);//12 (11)
         System.out.println(--x);//10
         }
       }
```

Boolean Logical Operators

- The Boolean logical operators shown here operate only on boolean operands.

Operator	**Result**
&	Logical AND
|	Logical OR
^	Logical XOR (exclusive OR)
||	Short-circuit OR
&&	Short-circuit AND
!	Logical unary NOT

- The following table shows the effect of each logical operation:

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

Short-Circuit Logical Operators (&& and ||)

- These are secondary versions of the Boolean AND and OR operators, and are known as short-circuit logical operators.

- OR (||) operator results in true when A is true , no matter what B is. Similarly, AND (&&) operator results in false when A is false, no matter what B is.

Example of logical and short-circuited operators

```
// To show the working of logical and shortcircuited operators
class Example
      {
        public static void main(String args[])
        {
        int a=10;
        int b=5;
        int c=20;
        System.out.println(a>b||a++<c);// 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
        }
      }
```

```
// To show the working of  short circuited && and Logical & operator
class Example
      {
        public static void main(String args[])
        {
        int a=10;
        int b=5;
        int c=20;
        System.out.println(a<b&&a++<c);//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
        }
      }
```

The ? Operator

- Java includes a special ternary (three-way)operator, ?, that can replace certain types of if-then-else statements.
- The ? has this general form:

  expression1 ? expression2 : expression3

- Here,expression1 can be any expression that evaluates to a boolean value.

- If expression1 is true , then expression2 is evaluated; otherwise, expression3 is evaluated.

- Both expression2 and expression3 are required to return the same type, which can't be void.

int ratio = denom == 0 ? 0 : num / denom ;

- When Java evaluates this assignment expression, it first looks at the expression to the left of the question mark.

- If denom equals zero, then the expression between the question mark and the colon is evaluated and used as the value of the entire ? expression.

- If denom does not equal zero, then the expression after the colon is evaluated and used for the value of the entire ? expression.

- The result produced by the? operator is then assigned to ratio.

Bitwise Operators

These operators act upon the individual bits of their operands.

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

Example: Bitwise operators

```
// To show the working of  & | ^ operator
class Example
    {
     public static void main(String args[])
     {
     byte a=4; // 00000100
     byte b=5; // 00000101
     System.out.println(a&b);//(00000100)4
     System.out.println(a|b);//(00000101)5
     System.out.println(a^b);//(00000001)1
     }
    }
```

Representation of –ve number in java[2’s Complement form]

Example:

-10&-20

10(00001010)

Taking 2's complement:

11110101

+   1

11110110(-10)

20(00010100)

Taking 2's Complement:

11101011

+   1

11101100(-20)

Taking Bitwise &

11110110

11101100

11100100[Here MSB is 1 so answer will be -ve]

Taking 2's complement again to get the final result

00011011

+   1

00011100(28)-->Final answer -28[As MSB was already observed to be 1, hence 28 will be represented as -28]

The Left Shift Operator

- The left shift operator,<<, shifts all of the bits in a value to the left a specified number of times.

value << num

- Example:

00000110	6<< 2
00011000	24

The Right Shift Operator

- The right shift operator, >>, shifts all of the bits in a value to the right a specified number of times.

  value >> num

- It is also known as signed right shift.

- Example:

00001000	8>> 2
00000010	2

For positive numbers

```
// To show the working of  << and >> operator
class Example
    {
      public static void main(String args[])
      {
      byte x=10;
      System.out.println(x<<2);// 10*2^2=40
      System.out.println(x>>2);// 10/2^2=2
      }
    }
```

For negative numbers

```
// To show the working of  << and >> operator
class Example
      {
        public static void main(String args[])
        {
        byte x=-10;
        System.out.println(x<<2);// 10*2^2=-40
        System.out.println(x>>2);// 10/2^2 -1=-3[-1 will be added if not completely
      divisible,otherwise Number/2^no.of bits]
        }
      }
```

The Unsigned Right Shift

- In these cases, to shift a zero into the high-order bit no matter what its initial value was. This is known as an unsigned shift.

- To accomplish this, we will use Java’s unsigned, shift-right operator, >>>, which always shifts zeros into the high-order bit.

- Example:
  - 11111111 11111111 11111111 11111111 –1 in binary as an int
  - >>>24
  - 00000000 00000000 00000000 11111111 255 in binary as an int

Unsigned right shift example

Take example of -1(which will be represented as 2’s Complement of 1)

00000000 00000000 00000000 00000001(1)[32 bit representation]

Taking 2's Complement:

```
11111111 11111111 11111111 11111110
+                                 1
--------------------------------------------------------
11111111 11111111 11111111 11111111(-1)
```

>>>24[Unsigned right shift][Shifting by 24 bits]

00000000 00000000 00000000 11111111(255)[Here higher order bits are replaced with 0[No matter what the sign was][Here no need to take 2’s complement again to get final answer, it will be 255]

But if we use:

>>24

It will be:

11111111 11111111 11111111 11111111[Here we need to take 2’s complement

Taking 2’s Complement again, and it wil be -1

CODERINDEED

Operator Precedence

Highest						
++ (postfix)	-- (postfix)					
++ (prefix)	-- (prefix)	~	!	+ (unary)	– (unary)	(*type-cast*)
*	/	%				
+	–					
>>	>>>	<<				
>	>=	<	<=	instanceof		
==	!=					
&						
^						
|						
&&						
||						
?:						
–>						
=	op=					
Lowest						

Q1

What will be the output of following code?

```
public class First
{
public static void main(String[] args)
{
System.out.println(20+2%3*5-10/5);
}
}
```

A. 5

[✓ B. 28]

C. 10

D. 0

Q2

What will be the output of following code?

```
public class First
{
public static void main(String[] args)
{
int a=6,b=3,c=2;
System.out.println(a>b+c++);
}
}
```

[✓ A. true]

B. false

C. 6

D. 5

Q3

What will be the output of following code?

```
public class First
{
public static void main(String[] args)
{
int a=100;
boolean b=false;
System.out.println(++a>100&!b);
}
}
```

[✓ A. true]

B. false

C. 100

D. -1

Q4

What will be the output of following code?

```
public class First
{
public static void main(String[] args)
{
int a=6,b=7;
boolean c;
c=++a==b||b++>=8;
System.out.println(c+" "+b);
}
}
```

A. true 8

B. false 7

[✓ C. true 7]

D. false 8

Q5

What will be the output of the following code snippets?

```
public class First
{
public static void main(String[] args)
{
System.out.println(12^3);
}
}
```

A. 0

[✓ B. 15]

C. 36

D. 9

Q6

CODERINDEED

What will be the output of following code?

```
public class First
{
public static void main(String[] args)
{
System.out.print(2>1||4>3?false:true);
}
}
```

A. true

[✓ B. false]

C. -1

D. Error

Q7

What will be the output of following code?

```
public class First
{
public static void main(String[] args)
{
byte b=14;
System.out.println(b>>3);
}
}
```

A. 112

[✓ B. 1]

C. 0

D. 17

Q8

What will be the output of followng code?

```
public class Test {
    public static void main(String args[])  {
        System.out.println(10  +  20 + "Hello");
        System.out.println("Hello" + 10 + 20);
    }
}
```

A. 30Hello

Hello30

B. 1020Hello

Hello1020

[✓ C. 30Hello

Hello1020]

D. 1020Hello

Hello30

CODERINDEED

Q9

In Java, after executing the following code what are the values of x, y and z?

int x,y=10; z=12; x=y++ + z++;

A. x=22, y=10, z=12

B. x=24, y=10, z=12

C. x=24, y=11, z=13

[✓ D. x=22, y=11, z=13]