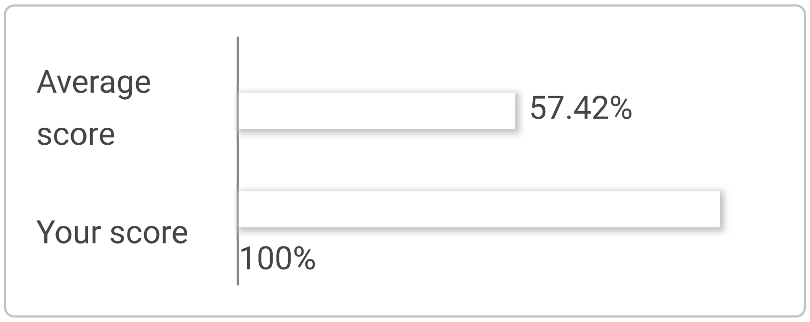
Java Operators Quiz Online Test – 1

Result

10 out of 10 questions were answered correctly.

Your score is 10 out of 10, (100%)



Review Answers

What will happen when you compile and run the following code?

```java
public class Test{

    public static void main(String[] args) {
        int i = 0;
        int j = i++ + ++i;
        System.out.println( j );
    }
}
```

1. 1
2. 3
3. 2
4. 4

Correct answer.

Option 3 is the correct choice. The i++ is a post increment which means that the value will be used first and incremented later, while ++i is a pre-increment which means that the value will be incremented first and used later. The statement i++ + ++i will be evaluated as (0) + ++(1) = 2.

What will happen when you compile and run the following code?

```
public class Test{

   public static void main(String[] args) {
       int i = 0;
       int j = ++i + i++;
       System.out.println( j );
   }
}
```

```
1.    3

2.    2

3.    4

4.    1
```

Correct answer.
Option 2 is the correct choice. The i++ is a post increment which means that the value will be used first and incremented later, while ++i is a pre-increment which means that the value will be incremented first and used later. The statement ++i + i++ will be evaluated as 1 + (1)++ = 1 + 1 = 2.

What will happen when you compile and run the following code?

```
public class Test{

   public static void main(String[] args) {
       int i = 100;
       int j = 10;
       System.out.println( i%j );
```

```
    }
}
```

```
1.    10

2.    0

3.    Compilation error

4.    None of the above
```

Correct answer.
Option 2 is the correct choice. The % is a modulus operator, means it returns remainder of division operation. The number 100 is divisible by 10 which yields remainder 0. The result of 10 % 3 will be 1 because if you divide 10 by 3, the remainder will be 1 (3 * 3 = 9 which leaves us 1).

What will happen when you compile and run the following code?

```
public class Test{

    public static void main(String[] args) {
        int i = 0;
        int j = 1;
        if(!i && j)
            System.out.println("1");
        else
            System.out.println("2");
    }
}
```

```
1.    2

2.    1

3.    Compilation error
```

Correct answer.

Option 3 is the correct choice. The ! operator is a unary logical complement operator which inverts the value of a boolean.

For example if value of boolean variable b is true, !b will make it false. The ! operator requires operand to be of type boolean. So code will give compilation error “The operator ! is undefined for the argument type(s) int”.

What will happen when you compile and run the following code?

```java
public class Test{

    public static void main(String[] args) {
        boolean b1 = true, b2 = false, b3 = true;
        System.out.println( b1&&b2&&!b3 );
    }
}
```

```
1.    true
2.    false
3.    Compilation error
```

Correct answer.
Option 2 is the correct choice. The && operator operates on boolean. It returns true if and only if both of the operands are true.

The ! operator is a complement operator. So if the operand is true, it makes it false. If the operand is false, it makes it true. The statement b1&&b2&&!b3 will be evaluated like (true && false && !(true)) which will equal to (true && false && true) = false.

What will happen when you compile and run the following code?

```
public class Test{

    public static void main(String[] args) {
        int i = 10, j = 12, k = 1;
        k += i++ - --j;
        System.out.println(k);
    }
}
```

1. -2
2. -1
3. 1
4. 0

Correct answer.
Option 4 is the correct choice. The expression i++ – –j will be evaluated like 10 – 11 = -1. The result is again added to the value of k because of “k +=”. So 1 + (-1) = 0 will be assigned to k.

What will happen when you compile and run the following code?

```
public class Test{

    public static void main(String[] args) {
        int i = 10;
        i++;i++;++i;
        int j = i++;
        System.out.println(j);
    }
}
```

1. 12
2. 11

```
3.     13

4.     14
```

Correct answer.
Option 3 is the correct choice. All three increments of the variable i are done before the final value is assigned to the variable j. The value of i after 3 increments will be 13. However, the value of j will be 13 as well because the value of i is assigned to j first and incremented later due to the post increment operator.

What will happen when you compile and run the following code?

```java
public class Test{

    public static void main(String[] args) {
        byte b1 = 2;
        b1 = b1 + 10;

        byte b2 = 2;
        b2 += 10;

        System.out.println(b1 + " " + b2);
    }
}
```

```
1.     10 12

2.     12 12

3.     12 10

4.     Compilation error
```

Correct answer.
Option 4 is the correct choice. The arithmetic operation automatically widens operands of byte, short and char types to int value. The b1 + 10 expression will be widened to int type which

cannot be assigned back to the byte type without an explicit cast.

In normal scenarios, a = a + b and a += b are similar, however there is a small difference. The += operator also does the casting if necessary. In above code, b2 += 10 will be executed as b2 = (byte) (b2 + 10). Since it already handles cast for you, there will be no compilation error for that statement.

What will happen when you compile and run the following code?

```java
public class Test{

    public static void main(String[] args) {
        int i = 19;
        int j = -5;
        System.out.println( i%j );
    }
}
```

```
1.    4

2.    -4

3.    3

4.    -3
```

Correct answer.
Option 1 is the correct choice. The % modulus operator returns remainder of the division operation. If the first operand of the modulus operator is positive, the remainder is also positive.

Hence, the code will print 4 when run.

What will happen when you compile and run the following code?

```
public class Test{

    public static void main(String[] args) {
        int i = -21;
        int j = 4;
        System.out.println( i%j );
    }
}
```

1. 1
2. -1
3. 5
4. -5

Correct answer.
Option 2 is the correct choice. The % modulus operator returns remainder of the division operation. If the first operand of the modulus operator is negative, the remainder is also negative.

So the code will print -1 when run.