Operators:

Reading:

- Java Tutorial: Section Operators under section Learning the Java Language

Operators are special symbols that perform specific operations on one, two, or three operands, and then return a result.
Operator precedence:
The closer to the top of the table an operator appears, the higher its precedence.

<table>
<thead>
<tr>
<th>Operators</th>
<th>Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>postfix</strong></td>
<td><code>expr++ expr--</code></td>
</tr>
<tr>
<td><strong>unary</strong></td>
<td><code>++expr --expr +expr -expr ~ !</code></td>
</tr>
<tr>
<td><strong>multiplicative</strong></td>
<td><code>* / %</code></td>
</tr>
<tr>
<td><strong>additive</strong></td>
<td><code>+ -</code></td>
</tr>
<tr>
<td><strong>shift</strong></td>
<td><code>&lt;&lt; &gt;&gt; &gt;&gt;&gt;</code></td>
</tr>
<tr>
<td><strong>relational</strong></td>
<td><code>&lt; &gt; &lt;= &gt;= instanceof</code></td>
</tr>
<tr>
<td><strong>equality</strong></td>
<td><code>== !=</code></td>
</tr>
<tr>
<td><strong>bitwise AND</strong></td>
<td><code>&amp;</code></td>
</tr>
<tr>
<td><strong>bitwise exclusive OR</strong></td>
<td><code>^</code></td>
</tr>
<tr>
<td><strong>bitwise inclusive OR</strong></td>
<td>`</td>
</tr>
<tr>
<td><strong>logical AND</strong></td>
<td><code>&amp;&amp;</code></td>
</tr>
<tr>
<td><strong>logical OR</strong></td>
<td>`</td>
</tr>
<tr>
<td><strong>ternary</strong></td>
<td><code>? :</code></td>
</tr>
<tr>
<td><strong>assignment</strong></td>
<td>`= += -= *= /= %= &amp;= ^=</td>
</tr>
</tbody>
</table>
Operators with higher precedence are evaluated before operators with relatively lower precedence.

Operators on the same line have equal precedence.

When operators of equal precedence appear in the same expression:
  – All binary operators except for the assignment operators are evaluated from left to right
  – Assignment operators are evaluated right to left.

Arithmetic Operators:

\[ \text{op1} + \text{op2} \]
\[ \text{op1} - \text{op2} \]
\[ \text{op1} \times \text{op2} \]
\[ \text{op1} / \text{op2} \]
\[ \text{op1} \% \text{op2} \]
## The Unary Operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Unary plus operator; indicates positive value (numbers are positive without this, however)</td>
</tr>
<tr>
<td>-</td>
<td>Unary minus operator; negates an expression</td>
</tr>
<tr>
<td>++</td>
<td>Increment operator; increments a value by 1</td>
</tr>
<tr>
<td>--</td>
<td>Decrement operator; decrements a value by 1</td>
</tr>
<tr>
<td>!</td>
<td>Logical complement operator; inverts the value of a boolean</td>
</tr>
</tbody>
</table>
Example:

class UnaryDemo {
    public static void main(String[] args) {
        int result = +1;
        // result is now 1
        System.out.println(result);

        result--;
        // result is now 0
        System.out.println(result);

        result++;  
        // result is now 1
        System.out.println(result);

        result = -result;
        // result is now -1
        System.out.println(result);

        boolean success = false;
        // false
        System.out.println(success);
        // true
        System.out.println(!success);
    }
}

The increment/decrement operators can be applied before (prefix) or after (postfix) the operand. The code result++; and ++result; will both end in result being incremented by one. The only difference is that the prefix version (result++)
evaluates to the incremented value, whereas the postfix version (result++) evaluates to the original value.

Another example:

```java
class PrePostDemo {
    public static void main(String[] args){
        int i = 3;
        i++;
        // prints 4
        System.out.println(i);
        ++i;
        // prints 5
        System.out.println(i);
        // prints 6
        System.out.println(++i);
        // prints 6
        System.out.println(i++);
        // prints 7
        System.out.println(i);
    }
}
```

Preactice (what does this output?)

```java
int[] arr = {1, 5, 10, 2, 4};
int i = 1;
System.out.println(arr[i++]);
System.out.println(i);
System.out.println(arr[++i]);
```
Equality, Relational, and Conditional Operators

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

Output?

class ComparisonDemo {
    public static void main(String[] args) {
        int value1 = 1;
        int value2 = 2;
        if (value1 == value2)
            System.out.println("value1 == value2");
        if (value1 != value2)
            System.out.println("value1 != value2");
        if (value1 > value2)
            System.out.println("value1 > value2");
        if (value1 < value2)
            System.out.println("value1 < value2");
        if (value1 <= value2)
            System.out.println("value1 <= value2");
    }
}

The Conditional Operators

&& Conditional - AND
|| Conditional - OR

class ConditionalDemo1 {
    public static void main(String[] args){
        int value1 = 1;
        int value2 = 2;
        if((value1 == 1) && (value2 == 2))
            System.out.println("value1 is 1 AND value2 is 2");
        if((value1 == 1) || (value2 == 1))
            System.out.println("value1 is 1 OR value2 is 1");
    }
}

Ternary Operator (?:)
Conditional operator (if-then-else)

class ConditionalDemo2 {
    public static void main(String[] args){
        int value1 = 1;
        int value2 = 2;
        int result;
        boolean someCondition = true;
        result = someCondition ? value1 : value2;
        System.out.println(result);
    }
}
The Type Comparison Operator `instanceof`:
- Compares an object to a specified type.
- You can use it to test if an object is an instance of a class, an instance of a subclass, or an instance of a class that implements a particular interface.
- Keep in mind that null is not an instance of anything.

class InstanceofDemo {
    public static void main(String[] args) {
        Parent obj1 = new Parent();
        Parent obj2 = new Child();

        System.out.println("obj1 instanceof Parent: " + (obj1 instanceof Parent));
        System.out.println("obj1 instanceof Child: " + (obj1 instanceof Child));
        System.out.println("obj1 instanceof MyInterface: " + (obj1 instanceof MyInterface));
        System.out.println("obj2 instanceof Parent: " + (obj2 instanceof Parent));
        System.out.println("obj2 instanceof Child: " + (obj2 instanceof Child));
        System.out.println("obj2 instanceof MyInterface: " + (obj2 instanceof MyInterface));
    }
}

class Parent {}
```java
class Child extends Parent implements MyInterface {}
interface MyInterface {}
```

**Output:**

```
obj1 instanceof Parent: true
obj1 instanceof Child: false
obj1 instanceof MyInterface: false
obj2 instanceof Parent: true
obj2 instanceof Child: true
obj2 instanceof MyInterface: true
```

**Bitwise and Bit Shift Operators**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>~</td>
<td>Unary bitwise complement</td>
</tr>
<tr>
<td>&lt;&lt;</td>
<td>Signed left shift</td>
</tr>
<tr>
<td>&gt;&gt;</td>
<td>Signed right shift</td>
</tr>
<tr>
<td>&gt;&gt;&gt;</td>
<td>Unsigned right shift</td>
</tr>
<tr>
<td>&amp;</td>
<td>Bitwise AND</td>
</tr>
<tr>
<td>^</td>
<td>Bitwise exclusive OR</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Control Flow Statements:**

**Reading:**

- Java Tutorial: Section *Operators* under section *Learning the Java Language*
- Textbook: Section 3.1 and 3.3
<table>
<thead>
<tr>
<th>Statement type</th>
<th>Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Looping</td>
<td>for, while, do-while</td>
</tr>
<tr>
<td>Decision making</td>
<td>if-else, switch-case</td>
</tr>
<tr>
<td>Exception Handling</td>
<td>try-catch-finally, throw</td>
</tr>
<tr>
<td>Branching</td>
<td>return, break, continue, label:</td>
</tr>
</tbody>
</table>

**IMPORTANT**: Go through all the examples in tutorial.