Operators

Arithmetic Operators

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo Operation (Remainder after division)

+, -, and * operators to compute addition, subtraction, and multiplication operations.

Assignment Operators

Operator	Example	Same as
=	a = b;	a = b;
+=	a += b;	a = a + b;
-=	a -= b;	a = a - b;
*=	a *= b;	a = a * b;
/=	a /= b;	a = a / b;
%=	a %= b;	a = a % b;

Relational Operators

Operator	Description
==	Is Equal To
!=	Not Equal To
>	Greater Than
<	Less Than
>=	Greater Than or Equal To
<=	Less Than or Equal To

Logical Operators

Operator/Example	Meaning
&& (Logical AND)/expression1 && expression2	true only if both expression1 and expression2 are true
|| (Logical OR) expression1 || expression2	true if either expression1 or expression2 is true
! (Logical NOT) ! expression	true if expression is false and vice versa

Unary Operators

Operator Meaning

+ Unary plus: not necessary to use since numbers are positive without using it

- Unary minus: inverts the sign of an expression

++ Increment operator: increments value by 1

-- Decrement operator: decrements value by 1

! Logical complement operator inverts the value of a Boolean

Conditional Statements

Java has the following conditional statements:
if - Use if to specify a block of code to be executed, if a specified condition is true

```
if (condition) {
  // block of code to be executed if the condition is true
}
```

else - Use else to specify a block of code to be executed, if the same condition is false

```
if (condition) {
  // block of code to be executed if the condition is true
} else {
  // block of code to be executed if the condition is false
}
```

else if - Use else if to specify a new condition to test, if the first condition is false

```
if (condition1) {
  // block of code to be executed if condition1 is true
} else if (condition2) {
```

```
  // block of code to be executed if the condition1 is false and condition2 is true
} else {
  // block of code to be executed if the condition1 is false and condition2 is false
}
```

Switch Statement

Use the switch statement to select one of many code blocks to be executed. Use switch to specify many alternative blocks of code to be executed.

Difference between If Else and Switch Statements

- if-else statement test for equality as well as for logical expression. switch statement test only for equality.
- if statement evaluates integer, character, pointer or floating-point type or Boolean type.
- switch statement evaluates only character or integer value.
- You can have multiple if statement for multiple choice of statements. In switch, you only have one expression for the multiple choices.

```
public class ExplainingConditionalStatements {
	public static void main(String[] args) {
		// TODO Auto-generated method stub
//
//		int a = 30 ;
//		int b = 20;
//		int c=  30;
//		int d = 40;
//		if((a>b) && (d == 40))
//			{
//				System.out.println(" A is greatest value");
//			}
//		else if(b>c)
//		{
//			System.out.println(" B is greatest value");
//		}else
//		{
//			System.out.println(" C is greatest value");
//		}

//	------------------------------------------------------------
		//Integers or Characters



//	------------------------------------------------------------Switch ------------------------
		int i = 1;
	switch (i) {
		  case 1:
		    System.out.println("Monday");
		    break;
		  case 2:
		    System.out.println("Tuesday");
```

```
                    break;
                  case 3:
                          System.out.println("Wednesday");
                  default:
                    System.out.println("Weekoff");
                 }


//-----------------------------------------------If else ------------
 if(i ==1) {
           System.out.println("Monday");
         }else if (i ==2)
         {
                  System.out.println("Tuesday");
         }else if (i == 3)
         {
                  System.out.println("Wednesday");
         }else
         {
                  System.out.println("Weekoff");
         }
 }
```

Loops

Loops can execute a block of code until a specified condition is reached.
The while loop loops through a block of code until a specified condition is true

```
while (condition) {
  // code block to be executed
}
```

The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop until the condition is true

```
do {
  // code block to be executed
}
while (condition);
```

When you know exactly how many times you want to loop through a block of code, use the for loop instead of a while loop

```
for (statement 1; statement 2; statement 3) {
  // code block to be executed
}
```

Statement 1 is executed (one time) before the execution of the code block.
Statement 2 defines the condition for executing the code block.
Statement 3 is executed (every time) after the code block has been executed.
The break statement can also be used to jump out of a loop.
The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

Variables

There are three types of variables in Java:

1. local variable
2. Instance Variable
3. Static/Class Variable

Local variable - Local variables are declared in methods, constructors, or blocks.
Local variables are created when the method, constructor or block is entered, and the variable will be destroyed once it exits the method, constructor, or block.

- *Access modifiers cannot be used for local variables.*
- There is no default value for local variables, so local variables should be declared, and an initial value should be assigned before the first use.

Instance variable - Instance variables are declared in a class, but outside a method, constructor, or any block.

- Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.
- Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object's state that must be present throughout the class.
- Access modifiers can be given for instance variables.
- The instance variables are visible for all methods, constructors, and block in the class.
- Instance variables have default values. For numbers, the default value is 0, for Booleans it is false, and for object references it is null.
- Instance variables can be accessed directly by calling the variable name inside the class. ObjectReference.VariableName.

static variable- Class/Static Variables - Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor, or a block.

- There would only be one copy of each class variable per class, regardless of how many objects are created from it.
- Static variables are created when the program starts and destroyed when the program stops.
- Visibility is like instance variables. However, most static variables are declared public since they must be available for users of the class.
- Default values are same as instance variables. For numbers, the default value is 0; for Booleans, it is false; and for object references, it is null. Values can be assigned during the declaration or within the constructor. Additionally, values can be assigned in special static initializer blocks.
- Static variables can be accessed by calling with the class name ClassName.VariableName.

Differences between the Instance variable Vs. the Static variables

- Each object will have its copy of the instance variable, whereas We can only have one copy of a static variable per class irrespective of how many objects we create.
- *Changes made in an instance variable using one object will not be reflected in other objects as each object has its own copy of the instance variable. In the case of static, changes will be reflected in other objects as static variables are common to all objects of a class.*
- We can access instance variables through object references, and Static Variables can be accessed directly using the class name.