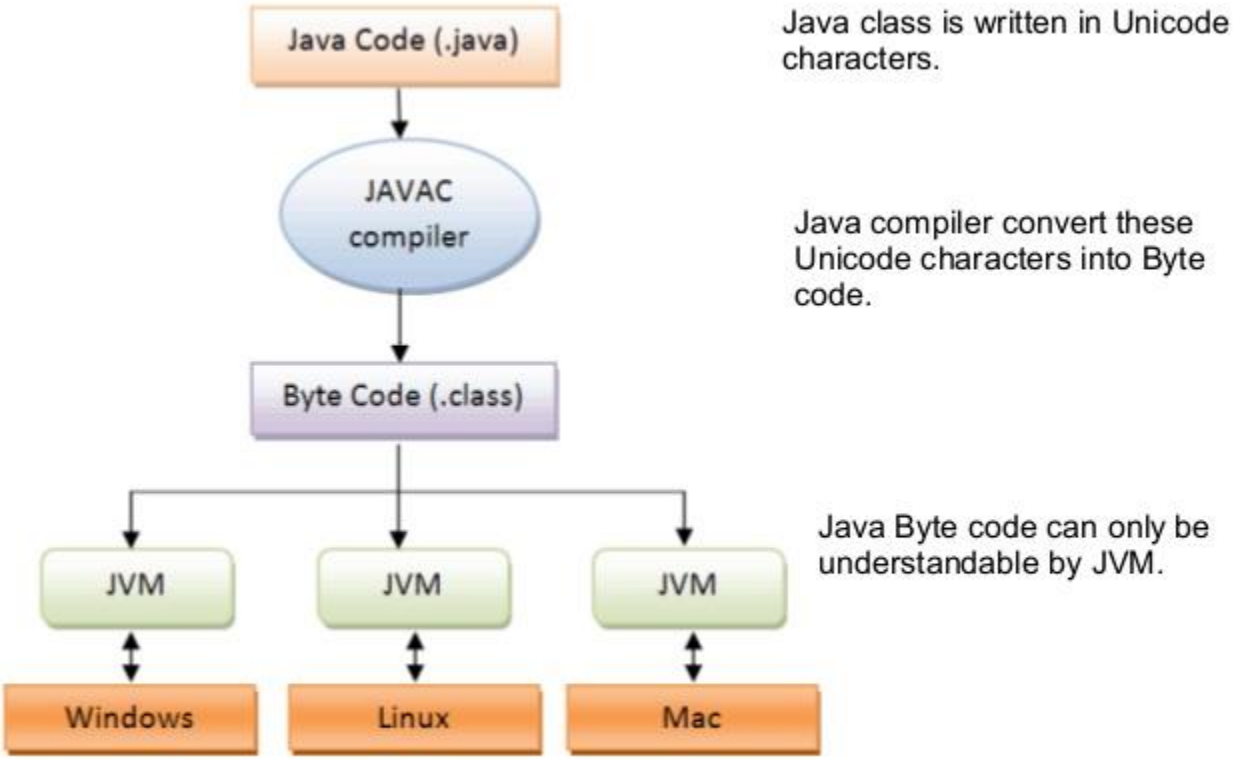
Java is platform independent language



The Java Virtual machine (JVM) is the virtual machine that runs on actual machine (your computer) and executes Java byte code. The JVM doesn't understand Java source code, that's why we need to have javac compiler that compiles *.java files to obtain *.class files that contain the byte codes understood by the JVM.
JVM makes java portable (write once, run anywhere). Each operating system has different JVM, however the output they produce after execution of byte code is same across all operating systems.

Types of variables in java

There are three types of variables in java.

1. Local variable
2. Static (or class) variable
3. Instance variable

Local Variable

These variables are declared inside the method of the class. Their scope is limited to the method which means that You can't change their values and access them outside of the method.
In this example, I have declared the instance variable with the same name as local variable, this is to demonstrate the scope of local variables.

Example of Local variable

```
public class LocalVariableExample
{
        // instance variable
        public String myVariable = "This is an Instance Variable";

        public static void main(String args[])
        {
            // Creating object
            VariableExample obj1 = new LocalVariableExample();

            /* We are calling the method, that changes the
             * value of myVariable. We are displaying myVariable again after
             * the method call, to demonstrate that the local
             * variable scope is limited to the method itself.
             */
             System.out.println("Calling Method");
            obj1.myMethod();
            System.out.println(obj1.myVariable);
         }

        public void myMethod()
        {
            // local variable
            String myVariable = "Inside Method";
            System.out.println(myVariable);
        }
}
```

Output:

```
Calling Method
Inside Method
This is an Instance Variable
```

Static (or class) Variable

Static variables are also known as class variables because they are associated with the class and common for all the instances of class. For example, if I create three objects of a class and access this static variable, it would be common for all, the changes made to the variable using one of the object would reflect when you access it through other objects.

Example of static variable

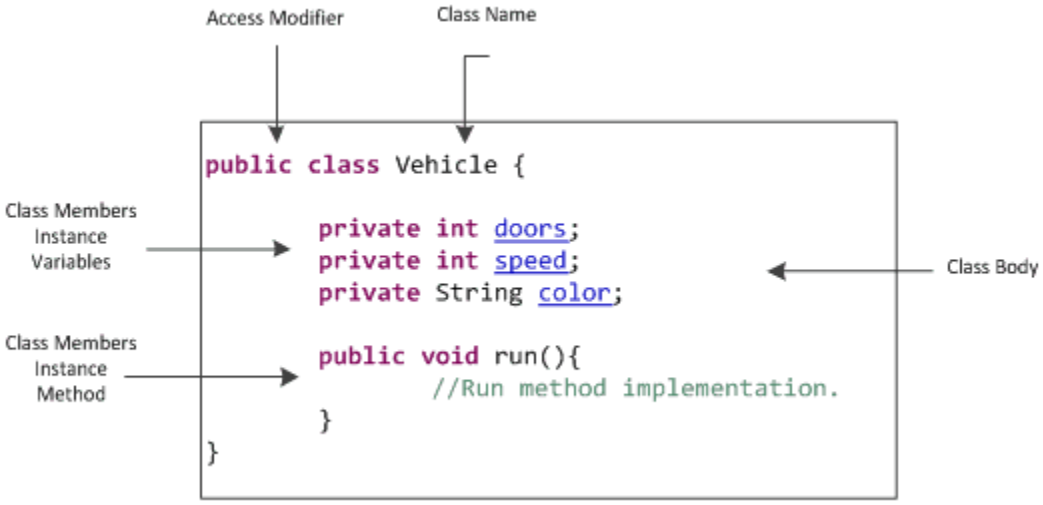
```
public class StaticVariableExample
{
        public static String myStaticVar= ”This will be common to all the objects created”;
        public static void main(String args[])
        {
                StaticVariableExample obj1 = new StaticVariableExample();
                StaticVariableExample obj2 = new StaticVariableExample();
                StaticVariableExample obj3 = new StaticVariableExample();

                System.out.println(“Original value ➔ “ +obj1.myStaticVar);
                System.out.println(“Original value ➔ “ +obj2.myStaticVar);
                System.out.println(“Original value ➔ “ +obj3.myStaticVar);

                /* Let’s change the value of the static variable (myStaticVar) using obj3 */
                obj3.myStaticVar = “New Value of the Static Variable changed by obj3”;

                /* All 3 objects will display the changed value */
                System.out.println(“Changed value ➔ “ +obj1.myStaticVar);
                System.out.println(“Changed value ➔ “ +obj2.myStaticVar);
                System.out.println(“Changed value ➔ “ +obj3.myStaticVar);
        }
}
```

Output of the above program:
Original value ➔ This will be common to all the objects created
Original value ➔ This will be common to all the objects created
Original value ➔ This will be common to all the objects created
Changed value ➔ New Value of the Static Variable changed by obj3
Changed value ➔ New Value of the Static Variable changed by obj3
Changed value ➔ New Value of the Static Variable changed by obj3

Instance variable

Instance variables are used by Objects to store their states. Variables which are defined without the STATIC keyword and are outside any method declaration are Object specific are are known as Instance variables. They are called so because their values are instance specific and are not shared among instances.

Example of Instance variable



```
public class VariablesInJava
{
      String firstName;              /* Instance variable */
      String lastName;               /* Instance variable */
      static String homeAddress = "1234 Somebodys Lane, Carrollton, TX 75010";

      public String Greetings(String fName, String lName)
      {
            String fullName = fName + ' ' + lName;
            return "Hi There " +fullName;
      }

      public int sum(int value1, int value2)
      {
            return (value1+value2);
      }

     public int subtract(int value1, int value2)
     {
            return (value1-value2);
     }

    public static void main(String args[])
    {
             Int x = 50;       /* local variable */
             Int y = 65;       /* local variable */

           /* create object for the VariablesInJava Class */
           VariablesInJava obj1 = new VariablesInJava();

           /* now obj1 can access the instance variable as follows */
           obj1.firstName = "James";
           obj1.lastName = "Bond";

          /* we can also call the object's method as follows */
          String person = Greetings(obj1.firstName,obj.lastName);
          System.out.println(person);
          Int addResult = sum(x,y);
          Int subResult = subtract(x,y);
          System.out.println("Result of addition     ➔ " +addResult);
          System.out.println("Result of subtraction ➔ " +addResult);
}
```

Loops

Loops are used to execute a set of statements repeatedly until a particular condition is satisfied. In Java, there are 3 types of basic loops:

For loop
While loop
Do-while loop

Syntax of for loop:

```
For(initial value; Condition; increment/decrement)
{
        Statement(s);
}
```

Step 1: Initial value happens first and only once. (executes only once)

Step 2: Condition is evaluated on each iteration. If the condition is true then the statements inside the for loop body gets executed. If the condition returns false, the statement in the body of the loop does not get executed and the control returns to the next statement after the loop.

Step 3: After every execution of the loop's body statements, the increment/decrement part of the loop executes which updates the loop counter.

Step 4: After the 3rd step, the control jumps to Step 2 (where the condition is re-evaluated).

For Loop Example:

```
Class ForLoopExample
{
        Public static void main(String args[])
        {
                For(int x = 0; x < 10; x++)
                {
                        System.out.println("The value of x  is:  "+x);
                }
        }
}


Class ForLoopExample2
{
        Public static void main(String args[])
        {
                For(int i = 10; i >  1; i--)
                {
                        System.out.println("The value of i  is:  "+i);
                }
        }
}
```

```
Class ForLoopExample3
{
	Public static void main(String args[])
	{
		For(int i = 1; i >=  1; i++)
		{
			System.out.println("The value of i  is:  "+i);
		}
	}
}
```

Syntax of while Loop:

```
while(condition)
{
	Statement(s);
}
```

while Loop Example:

```
Class WhileLoopExample
{
	Public static void main(String args[])
	{
		Int z = 15;
		while(z > 1)
		{
			System.out.println("The value of z is:  "+z);
			z--;
		}
	}
}
```

Syntax of do-while loop:

```
do
{
	Statement(s);
}(while condition);
```

do-while Loop Example:

```
Class DoWhileLoopExample
{
	Public static void main(String args[])
	{
		Int x = 15;
		do
		{
			System.out.println("The value of x is:  "+x);
			i--;
		}while(x > 1);
	}
}
```