Java Keywords, Syntax, Expression, Statement & Block

Java Keywords

Java keywords are also known as reserved words. Keywords are particular words which acts as a key to a code. These are predefined words by Java so it cannot be used as a variable or object name.

abstract	assert	boolean	break	byte
case	catch	char	class	const
continue	default	do	double	else
enum	extends	final	finally	float
for	goto	if	implements	import
instanceof	int	interface	long	native
new	package	private	protected	public
return	short	static	strictfp	super
switch	synchronized	this	throw	throws
transient	try	void	volatile	while

Basic Syntax

- **Case Sensitivity** − Java is case sensitive, which means an identifier
    - Hello and
    - hello

  would have a different meaning in Java.

- **Class Names** − For all class names the first letter should be in Upper Case. If several words are used to form a name of the class, each inner word's first letter should be in Upper Case.

  Example: *class MyFirstJavaClass*

- **Method Names** − All method names should start with a Lower Case letter. If several words are used to form the name of the method, then each inner word's first letter should be in Upper Case.

  Example: *public void myMethodName()*

- **Program File Name** − Name of the program file should exactly match the class name.

  When saving the file, you should save it using the class name (Remember Java is case sensitive) and append '.java' to the end of the name (if the file name and the class name do not match, your program will not compile).

  But please make a note that in case you do not have a public class present in the file then file name can be different than class name. It is also not mandatory to have a public class in the file.

Example: Assume 'MyFirstJavaProgram' is the class name. Then the file should be saved as *'MyFirstJavaProgram.java'*

- **public static void main(String args[])** − Java program processing starts from the main() method which is a mandatory part of every Java program.

Java Comments

- Traditional comments – They are also known as block comments, they are multi-line comments and expand across **multiple lines**.

  They start with /* and ends with */

- End of line comment – This type starts with // and usually **extends to the end of the line**.

Example

```
public class MyFirstJavaProgram {

   /* This is my first java program.
    * This will print 'Hello World' as the output
    * This is an example of multi-line comments.
    */

   public static void main(String []args) {
      // This is an example of single line comment
      /* This is also an example of single line comment. */
      System.out.println("Hello World");
   }
}
```

Identifiers in Java

Java Identifiers are the names of **packages**, **classes, methods**, and **variables** from which we identify the type of class, variable and method used in it. They are basically the **name of any element**.

Identifiers in Java are **case-sensitive**, there are some basic conventions that we have to follow while naming any identifier, they contain,

- Any Unicode character, which is a letter or a digit
- Special signs such as an underscore
- A $ sign (a currency sign)

- An *identifier* is the name of a class, variable, field, method, or constructor.
    - Cannot be a Java keyword
    - Can contain letters, digits, underscores, and dollar signs
    - Cannot start with a digit
- Valid identifier examples: **HelloWorld**, **args**, **i**, **Car**, **$myVar**, **employeeList2**
- Invalid identifier examples: **1st**, **byte**, **my-Arg**, ***z**
- Java naming conventions for identifiers:
    - Use *CamelCase* for most identifiers (classes, interfaces, variables, and methods).
    - Use an initial capital letter for classes and interfaces, and a lower case letter for variables and methods.
    - For named constants, use all capital letters separated by underscores.
    - Avoid using **$** characters in identifiers.

Identifiers in our **HelloWorld** program are in bold:

```
public class HelloWorld {
   public static void main(String[] args) {
      System.out.println("Hello World!");
   }
}
```

Expressions

An *expression* is a construct made up of variables, operators, and method invocations, which are constructed according to the syntax of the language, that evaluates to a single value. You've already seen examples of expressions, illustrated in bold below:

```
int cadence = 0;
anArray[0] = 100;
System.out.println("Element 1 at index 0: " + anArray[0]);

int result = 1 + 2; // result is now 3
if (value1 == value2)
    System.out.println("value1 == value2");


x + (y / 100) // unambiguous, recommended
```

When writing compound expressions, be explicit and indicate with parentheses which operators should be evaluated first. This practice makes code easier to read and to maintain.

Statements

Statements are roughly equivalent to sentences in natural languages. A *statement* forms a complete unit of execution. The following types of expressions can be made into a statement by terminating the expression with a semicolon (;).

- Assignment expressions
- Any use of ++ or --
- Method invocations
- Object creation expressions

Such statements are called *expression statements*. Here are some examples of expression statements.

```
// assignment statement
aValue = 8933.234;
// increment statement
aValue++;
// method invocation statement
```

```
System.out.println("Hello World!");
// object creation statement
Bicycle myBike = new Bicycle();
```

In addition to expression statements, there are two other kinds of statements: *declaration statements* and *control flow statements*. A *declaration statement* declares a variable. You've seen many examples of declaration statements already:

```
// declaration statement
double aValue = 8933.234;
```

Finally, *control flow statements* regulate the order in which statements get executed.

Blocks

A *block* is a group of zero or more statements between balanced braces and can be used anywhere a single statement is allowed. The following example, BlockDemo, illustrates the use of blocks:

```
class BlockDemo {
    public static void main(String[] args) {
        boolean condition = true;
        if (condition) { // begin block 1
            System.out.println("Condition is true.");
        } // end block one
        else { // begin block 2
            System.out.println("Condition is false.");
        } // end block 2
    }
     }
```