Object Oriented Programming with Java (Subject Code: BCS-403)

Unit 1

Lecture 3

Lecture 3

- Tokens of Java:
- Operators
- Variables
- Data types
- Comments

Java Tokens

In Java, **Tokens** are the smallest elements of a program that is meaningful to the compiler. They are also known as the fundamental building blocks of the program.

Tokens can be classified as follows:

- Keywords
- Identifiers
- Constants
- Special Symbols
- Operators
- Comments
- Separators

1. Keyword:

Keywords are pre-defined or reserved words in a programming language. Each keyword is meant to perform a specific function in a program. Since keywords are referred names for a compiler, they can’t be used as variable names because by doing so, we are trying to assign a new meaning to the keyword which is not allowed.

abstract	assert	boolean
break	byte	case
catch	char	class
const	continue	default

2. Identifiers:

Identifiers are used as the general terminology for naming of variables, functions and arrays. These are user-defined names consisting of an arbitrarily long sequence of letters and digits with either a letter or the underscore (_) as a first character. Identifier names must differ in spelling and case from any keywords.

Examples of valid identifiers:

MyVariable

MYVARIABLE

myvariable

x

i

x1

i1

3. Constants/Literals:

Constants are also like normal variables. But the only difference is, their values cannot be modified by the program once they are defined.

final data_type variable_name;

4. Special Symbols:

The following special symbols are used in Java having some special meaning and thus, cannot be used for some other purpose.

[] () {}, ; * =

5. Operators:

Java provides many types of operators which can be used according to the need. They are classified based on the functionality they provide. Some of the types are-

- Arithmetic Operators
- Unary Operators
- Assignment Operator
- Relational Operators
- Logical Operators
- Ternary Operator
- Bitwise Operators
- Shift Operators

6. Comments:

In Java, Comments are the part of the program which are ignored by the compiler while compiling the Program. They are useful as they can be used to describe the operation or methods in the program.

The Comments are classified as follows:

- Single Line Comments
- Multiline Comments

```
// This is a Single Line Comment
/*
This is a Multiline Comment
*/
```

7. Separators:

Separators are used to separate different parts of the codes. It tells the compiler about completion of a statement in the program. The most commonly and frequently used separator in java is semicolon (;).

Operators in java

Operator in java is a symbol that is used to perform operations. For example: +, -, *, / etc.

There are many types of operators in java which are given below:

- Unary Operator,
- Arithmetic Operator,
- shift Operator,
- Relational Operator,
- Bitwise Operator,
- Logical Operator,
- Ternary Operator and
- Assignment Operator.

Java Unary Operator Example: ++ and --

```
class OperatorExample{
public static void main(String args[]){
int x=10;
System.out.println(x++);//10 (11)
System.out.println(++x);//12
System.out.println(x--);//12 (11)
System.out.println(--x);//10
}}
```

Output:

```
10
12
12
10
```

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=10;
System.out.println(a++ + ++a);//10+12=22
System.out.println(b++ + b++);//10+11=21
 }}
```

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
System.out.println(a+b);//15
System.out.println(a-b);//5
System.out.println(a*b);//50
System.out.println(a/b);//2
System.out.println(a%b);//0
}}
```

Java AND Operator Example: Logical && and Bitwise &

- The logical && operator doesn't check second condition if first condition is false. It checks second condition only if first one is true.
- The bitwise & operator always checks both conditions whether first condition is true or false.

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
int c=20;
System.out.println(a<b&&a<c);//false && true = false
System.out.println(a<b&a<c);//false & true = false
}}
```

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
int c=20;
System.out.println(a<b&&a++<c);//false && true = false
System.out.println(a);//10 because second condition is not
    checked
System.out.println(a<b&a++<c);//false & true = false
System.out.println(a);//11 because second condition is chec
    ked
}}
```

Java OR Operator Example: Logical || and Bitwise |

- The logical || operator doesn't check second condition if first condition is true. It checks second condition only if first one is false.
- The bitwise | operator always checks both conditions whether first condition is true or false.

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
int c=20;
System.out.println(a>b||a<c);//true || true = true
System.out.println(a>b|a<c);//true | true = true
//|| vs |
System.out.println(a>b||a++<c);//true || true = true
System.out.println(a);//10 because second condition is no
    t checked
System.out.println(a>b|a++<c);//true | true = true
System.out.println(a);//11 because second condition is ch
    ecked
}}
```

Java Ternary Operator

```
class OperatorExample{
public static void main(String args[]){
int a=2;
int b=5;
int min=(a<b)?a:b;
System.out.println(min);
}}
```

Output:

```
2
```

Java Assignment Operator

```
class OperatorExample{
public static void main(String[] args){
int a=10;
a+=3;//10+3  a=a+3
System.out.println(a);
a-=4;//13-4
System.out.println(a);
a*=2;//9*2
System.out.println(a);
a/=2;//18/2
System.out.println(a);
}}
```

Java Shift Operator Example: Left Shift

```
class OperatorExample{
public static void main(String args[]){
System.out.println(10<<2);//10*2^2=10*4=40
System.out.println(10<<3);//10*2^3=10*8=80
System.out.println(20<<2);//20*2^2=20*4=80
System.out.println(15<<4);//15*2^4=15*16=240

}}
```

Java Shift Operator Example: Right Shift

```
class OperatorExample{
public static void main(String args[]){
System.out.println(10>>2);//10/2^2=10/4=2
System.out.println(20>>2);//20/2^2=20/4=5
System.out.println(20>>3);//20/2^3=20/8=2
}}
```

Java Variable Types

Variable is name of reserved area allocated in memory.

There are three kinds of variables in Java:

- Local variables
- Instance variables
- Class/static variables

```
class A{
int data=50;//instance variable
static int m=100;//static variable
void method(){
int n=90;//local variable
}
}//end of class
```

Local variables :

- 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.
- Local variables are visible only within the declared method, constructor or block.
- Local variables are implemented at stack level internally.
- 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 variables :

- Instance variables are declared in a class, but outside a method, constructor or any block.
- When a space is allocated for an object in the heap a slot for each instance variable value is created.
- Instance variables are created when an object is created with the use of the key word '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 through out the class.

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 stored in static memory. It is rare to use static variables other than declared final and used as either public or private constants.