Object Oriented Programming

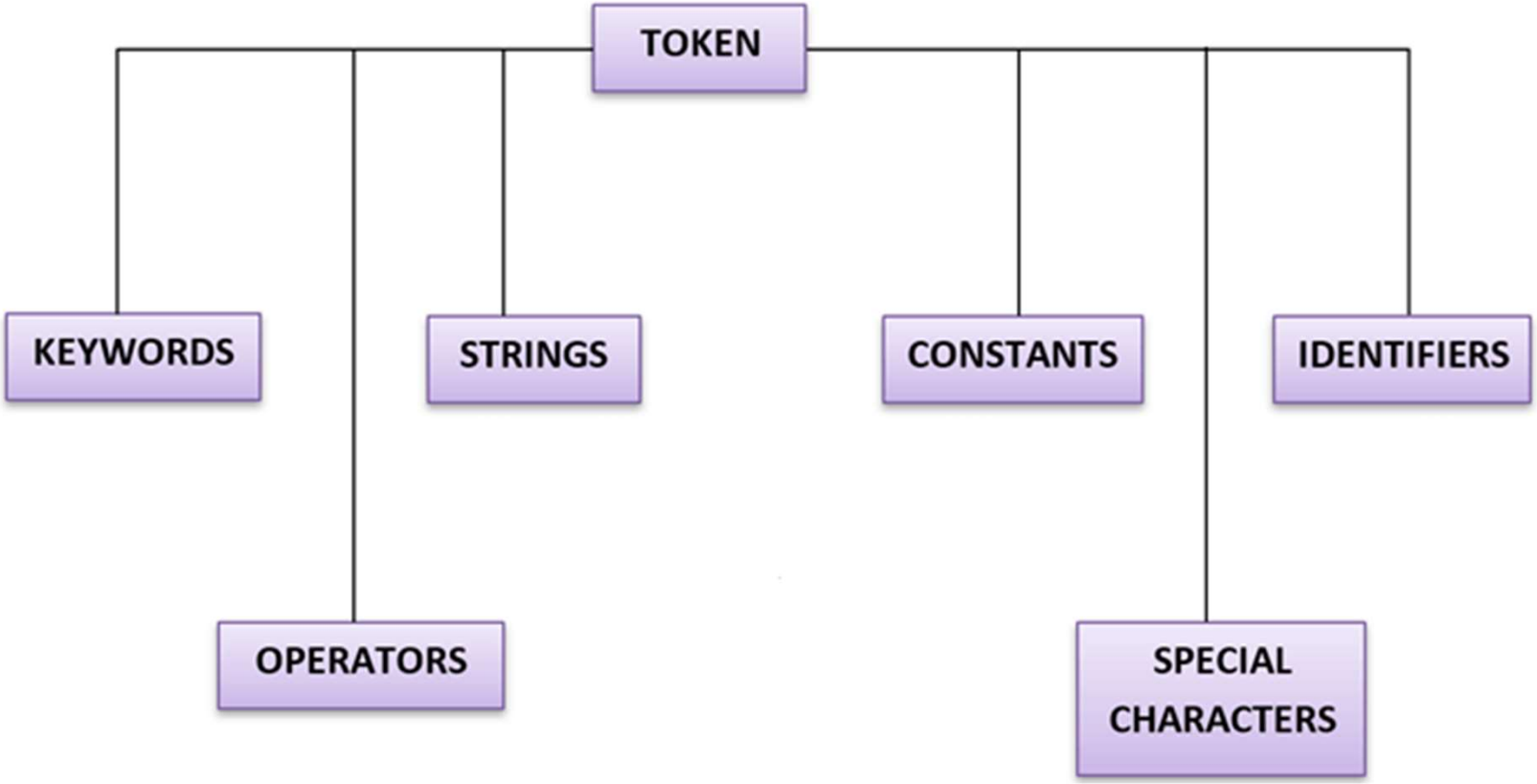
Unit-1

Tokens

- Separator is a **token** used to separate two individual **tokens** used in a **java** program.

- Tokens in Java are the small units of code which a Java compiler uses for constructing those statements and expressions. Java supports 5 types of tokens which are:
    - Keywords
    - Identifiers
    - Literals
    - Operators
    - Special Symbols

Fig. of Tokens



Keywords

- Keywords in Java are predefined or reserved words that have special meaning to the Java compiler.

- Each keyword is assigned a special task or function and cannot be changed by the user. You cannot use keywords as variables or identifiers as they are a part of Java syntax itself.

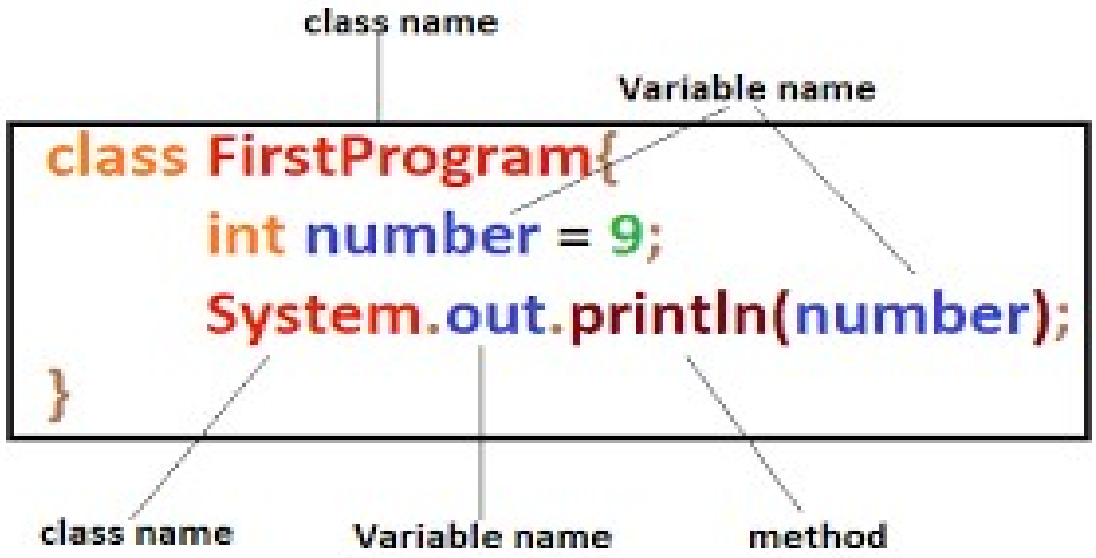
- A keyword should always be written in lowercase as Java is a case sensitive language. Java supports various keywords, some of them are listed below:

List of Keyword

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

Identifier

- Java Identifiers are the user-defined names of variables, methods, classes, arrays, packages, and interfaces. Once you assign an identifier in the Java program, you can use it to refer the value associated with that identifier in later statements. There are some de facto standards which you must follow while naming the identifiers such as:
- Identifiers must begin with a letter, dollar sign or underscore.
- Apart from the first character, an identifier can have any combination of characters.
- Identifiers in Java are case sensitive.
- Java Identifiers can be of any length.


class name
Variable name
class FirstProgram{
int number = 9;
System.out.println(number);
}
class name
Variable name
method

Constant

- A constant is a variable whose value **cannot change once it has been assigned**. Java doesn't have built-in support for constants.
- A constant can make our program more easily read and understood by others. In addition, a constant is cached by the JVM as well as our application, so using a constant can improve performance.
- To define a variable as a constant, we just need to add the keyword "**final**" in front of the variable declaration.

Syntax:-

```
final float pi = 3.14f;
```

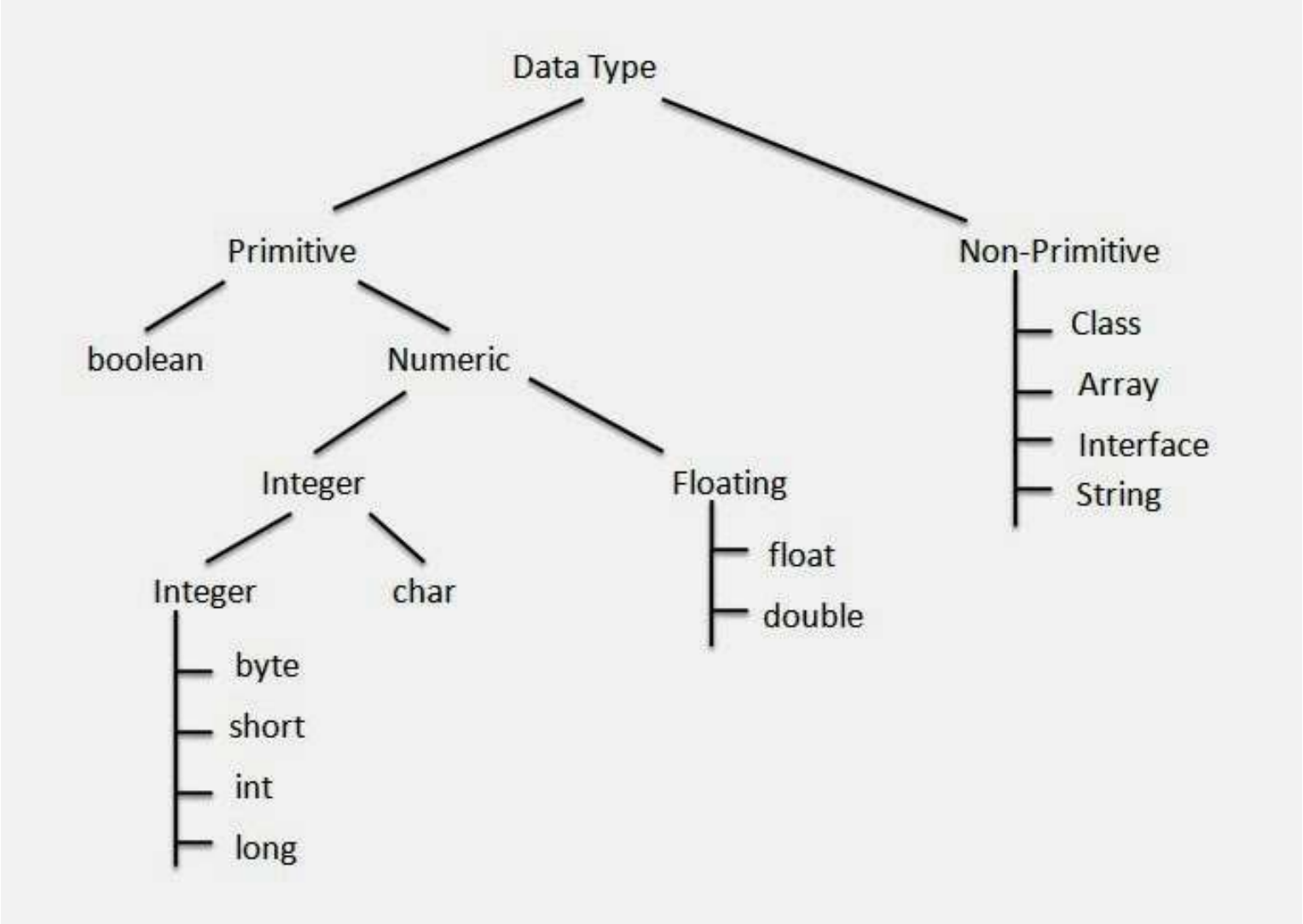
Data Type

- Data type specifies the size and type of variable values, and it has no additional methods.

Data types are divided into two groups:

- Primitive data types–
  includes byte, short, int, long, float, double,
  boolean and char
- Non-primitive data types –
  such as String, Arrays and Classes

Data Type


Data Type
Primitive
Non-Primitive
boolean
Numeric
Class
Array
Interface
String
Integer
Floating
Integer
char
float
double
byte
short
int
long

Type Casting

- A type cast is basically a conversion from one type to another. There are two types of type conversion:
- **Implicit Type Conversion**

  Also known as ‘automatic type conversion’.

  – Done by the compiler on its own, without any external trigger from the user.

  – Generally takes place when in an expression more than one data type is present. In such condition type conversion (type promotion) takes place to avoid lose of data.

  – All the data types of the variables are upgraded to the data type of the variable with largest data type.

Type Casting

- **Explicit Type Conversion**: This process is also called type casting and it is user-defined. Here the user can typecast the result to make it of a particular data type.

Program of Explicit

```
#include<iostream>
    using namespace std;
        int main()
        {
        int x=10;
        char y='A';
        int result;
        result=x+y;
        count<<result<<endl;
        }
```

Program of Implicit

```
#include<iostream>
Using namespace std;
    int main()
    {
    float x=5/2;
    cout<<x<<endl;
    }
```

```
#include<iostream>
Using namespace std;
    int main()
    {
    float x=(float)5/2;
    cout<<x<<endl;
    }
```

```
#include<iostream>
Using namespace std;
    int main()
    {
    float x=(float)(5/2);
    cout<<x<<endl;
    }
```

Interface

- An interface describes the behavior or capabilities of a C++ class without committing to a particular implementation of that class.

- The C++ interfaces are implemented using **abstract classes** and these abstract classes should not be confused with data abstraction which is a concept of keeping implementation details separate from associated data.

Program of Interface

```
Class box
{
Public:
// pure virtual function
virtual double getVolume()=0;
Private:
double length;
Double breadth;
Double height;
}
```

Operator and Expression

WHAT ARE OPERATORS?

The operators are the special type of functions that takes one or more parameters and gives new result. It is a symbol that tells the compiler to perform the mathematical and logical manipulations. The programming language like C or C++ is incomplete without the use of operators.

Operator

SOME OF THE BUILT IN OPERATORS ARE

- Arithmetic operators
- Relational operators
- Logical operators
- Assignment operators
- Bitwise operators

So, let us have a look all these operators briefly.

ARITHMETIC OPERATORS

- The arithmetic operators are used to perform the arithmetic operations on the operands. The operations can be addition, multiplication, subtraction and division.

Operator

RELATIONAL OPERATORS

- The relational operators are those operators that are used to compare the values of two operands. For example, by comparing two operands that their values are equal or not, or the value of one operand is greater than the other.

LOGICAL OPERATORS

- The logical operators are those operators that are used to combine two or more conditions. The logical operators are AND (&&) and OR (||). If we combine two statements using AND operator then only both the valid statements will be considered and if we combine two statements using OR operator then only either one of them will be considered.

Operator

BITWISE OPERATORS

- The bitwise operators are those are used to perform bit level operations on the operands. In the bitwise operators first operators are converted to bit level and then calculation is performed on the operands. Some of the operations which are performed are addition, subtraction, multiplication, division, etc.

Eg:-

```
#include <stdio.h>
 int main()
{
int a = 12, b = 25;
printf("Output = %d", a&b);
return 0;
}
```

12 = 00001100 (In Binary)
25 = 00011001 (In Binary)
Bit Operation of 12 and 25

```
    00001100
  & 00011001
  ----------
    00001000=8(In decimal)
```

Expression

WHAT ARE EXPRESSIONS?

- An expression is a sequence of operators and the operands. It is a form when you combine operands and operators.

THE EXPRESSIONS ARE OF THREE TYPES

- Arithmetic expression
- Relational expression
- Logical expression

ARITHMETIC EXPRESSION

- An arithmetic expression is that expression in which arithmetic operators are used. Like addition, multiplication, subtraction, division, etc.

Expression

RELATIONAL EXPRESSION

- A relational expression is that expression in which relational operators are used. The operators provided in relational expression are less than (<), greater than (>), less than equal to (<=), greater than equal to (>=), etc.

LOGICAL EXPRESSION

- A logical expression is that expression in which logical operators are used. Some of the logical operators are AND (&&), OR (||), NOT (!).

Operator precedence

Example:-

c=a+b*d

c=(a+b)*d

	Operator Precedence
1	! Logical not (Highest)
2	( ) Parenthesis
3	*, /, %
4	+, -
5	>, >=, <, <=
6	==, !=
7	&& (AND)
8	|| (OR)
9	= (Lowest)

Operator Precedence

- Example:-

  a,b=10,c=10,d=20

  a=b+c*d

  a=10+10*20

  a=10+200

  a=210

  a=(b+c)*d

  a=(10+10)*20

  a=20*20

  a=400

- a=b=c=d

Conditional Statement in C++

- Different form of implementation of if statement are:-
    - Simple if statement
    - If else statement
    - Nested if-else statement
    - Else if statement

Conditional Statement in C++

- Simple if Statement:-

This is used to execute the statement only if the condition is true.

Syntax:-

```
if(condition)
    {
      block of statement;
    }
```

Conditional Statement in C++

1.
```
if (20 > 18) {
  cout << "20 is greater than 18";
}
```

2.
```
int x = 20;
int y = 18;
if (x > y) {
  cout << "x is greater than y";
}
```

Conditional Statement in C++

- Print "Hello World" if x is **greater than** y

```
int x = 50;
int y = 10;
  ___(x___ y)
 {
cout << "Hello World";
}
```

Conditional Statement in C++

- If else Statement:-

Use the else statement to specify a block of code to be executed if the condition is false.

Syntax:-

```
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
}
```

Conditional Statement in C++

- ```
  int time = 20;
  if (time < 18) {
    cout << "Good day.";
  } else {
    cout << "Good evening.";
  }
  ```
  ```

Conditional Statement in C++

- Nested if Statement:-

A **nested if** in C++ is an **if statement** that is the target of another **if statement**. **Nested if statements** means an **if statement** inside another **if statement**.

Syntax:-

```
int main()
{
   int i = 10;
```

```
    if (i == 10)
  {
    // First if statement
    if (i < 15)
      cout<<"i is smaller than 15\n";

        if (i < 12)
      cout<<"i is smaller than 12 too\n";
    else
      cout<<"i is greater than 15";
  }

  return 0;
}
```

Conditional Statement in C++

- Else if Statement:-

Use the else if statement to specify a new condition if the first condition is false.

Syntax:-

```
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
}
```

Conditional Statement in C++

- ```
  int time = 22;
  if (time < 10) {
    cout << "Good morning.";
  } else if (time < 20) {
    cout << "Good day.";
  } else {
    cout << "Good evening.";
  }
  ```
  ```

Iterative Statement

- The statements that cause a set of statements to be executed repeatedly either for a specific number of times or until some condition is satisfied are known as **iteration statements**.
- That is, as long as the condition evaluates to True, the set of statement(s) is executed.
- The various iteration statements used in C++ *are*
  - for loop,
  - while loop and
  - do while loop.

Iterative Statement

- For Loop:-

The for loop is one of the most widely used loops in C++. The for loop is a deterministic loop in nature, that is, the number of times the body of the loop is executed.

Syntax:-

```
for(initialize; condition; inc/dec)
  {
  //body of the for loop
}
```

Iterative Statement

Example of for loop:-

```
#include<iostream>
using namespace std;
int main() {
  int n;
  for(n=l; n<=10; n++)
    cout<<n<<" ";
    return 0;
}
```

Iterative Statement

- While loop:-

The while loop is used to perform looping operations in situations where the number of iterations is not known in advance. That is, unlike the for loop, the while loop is non deterministic in nature.

The syntax of the while loop is:-

```
while(condition)
    {
     // body of while loop
      }
```

Iterative Statement

```
#include<iostream>
using namespace std;
int main() {
  int n,i,sum;
  cout<<" Enter the number of consecutive positive"<<
  "\n integers(starting from 1): ";
  cin>>n;
  sum=0;
  i=1;
  while (i<=n) {
   sum+=i;
   ++i;
}
cout<<"\nThe sum is "<<sum;
return 0;
}
```

Iterative Statement

```
#include<iostream>
using namespace std;
int main() {
  int n,i,sum;
  cout<<" Enter the number of consecutive
      positive"<<
  "\n integers(starting from 1): ";
  cin>>n;
  sum=0;
  i=1;
  while (i<=n) {
   sum+=i;
   ++i;
}
cout<<"\nThe sum is "<<sum;
return 0;
}
```

```
n=5
sum=0
i=1
1<=5
sum=sum+i
sum=0+1
sum=1
i=2    2<=5
sum=3
i=3   3<=5
sum=6
i=4       4<=5
sum=10
i=5     5<=5
sum=15
```

Iterative Statement

- In a while loop, the condition is evaluated at the beginning of the loop and if the condition evaluates to False, the body of the loop is not executed even once.
- If the body of the loop is to be executed at least once, no matter whether the initial state of the condition is True or False, the do-while loop is used.

Syntax:-

```
do {
    //body of do while loop
 }while(condition) ;
```

Iterative Statement

```
#include<iostream>
using namespace std;
int main () {
 int a,d,n,sum,term=0  cout<<"Enter the first term,
     common difference,"
 <<"and the number of terms to be summed"
 <<"respectively:\n";
 cin>>a>>d>>n;
 sum=0;
 int i=1;
 cout<<"\nThe terms are ";
 do //do-while loop {
  term= a+ (i-1)*d;
  sum+=term;
  cout<<term<<" “;
  ++i;
}
while (i<=n) ;
cout<<"\nThe sum of A.P. is "<<sum;
return 0;
}
```

Term=0
a=3
d=6
n=3
Sum=0, i=1
Term=a+(i-1)*d
=3+(1-1)*6
=3
Sum=sum+term
=0+3=3
i=2
Term=9
Sum=3+9=12
i=3
Term=15
Sum=12+15
=27