Unit - I

Java Editions

We have following java editions available:

1.Java SE→ It is also known as java standard edition.

2.Java EE→ It is also known as java enterprise edition. We will use java EE while learning concepts like JSP, Servlet, EJB, Spring, JPA, etc as it helps in developing web applications and enterprise applications.

3.Java ME→ It is also known as java micro edition.

4.JavaFX→ It is java technology used to develop desktop applications and rich internet applications.

Introduction

History and Evolution of Java, Java Buzzwords, An Overview of Java, Object-Oriented Programming, First Simple Program, Data Types, Variables, Type Conversion and Casting, Arrays, Operators, Control Statements.

Classes and Methods

Class Fundamentals, Declaring Objects, Methods and Constructors, Overloading Methods, Overloading Constructors, Recursion, this Keyword, Access Control, Understanding static, Nested and Inner Classes, Exploring the String Class, Command-Line Arguments, garbage collection.

History and Evolution of Java

- The history of Java is very interesting. Java was originally designed for interactive television, but it was too advanced technology for the digital cable television industry at the time.

- The history of Java starts with the Green Team.

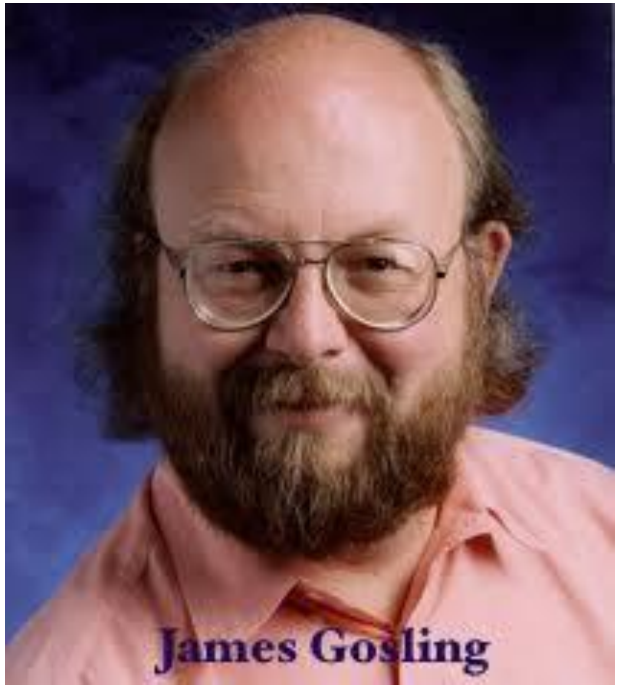
- Java team members (also known as Green Team), initiated this project to develop a language for digital devices such as set-top boxes, televisions, etc.

- However, it was best suited for internet programming. Later, Java technology was incorporated by Netscape.

Green Team

- Java was developed by James Gosling, who is known as the father of Java, in 1995.

- Currently, Java is used in
  - Mobile App Development.
  - Desktop GUI Applications.
  - Web-based Applications.
  - Gaming Applications.
  - Big Data Technologies.
  - Distributed Applications.
  - Cloud-based Applications.
  - IoT Applications.



1. Initially it was designed for small, embedded systems in electronic appliances like set-top boxes.

2. Firstly, it was called **"Greentalk"** by James Gosling, and the file extension was .gt

3. After that, it was called ”**Oak”** and was developed as a part of the Green project.

Why Java was named as "Oak"?

- Oak is a symbol of strength and chosen as a national tree of many countries like the U.S.A., France, Germany, Romania, etc.

- In 1995, Oak was renamed as **"Java"** because it was already a trademark by Oak Technologies.

Why Java Programming named "Java"?

- Why had they chose the name Java for Java language?

The team gathered to choose a new name. The suggested words were "dynamic", "revolutionary", "Silk", "jolt", "DNA", etc.

- They wanted something that reflected the essence of the technology: revolutionary, dynamic, lively, cool, unique, and easy to spell, and fun to say.

- According to James Gosling, "Java was one of the top choices along with **Silk**". Since Java was so unique, most of the team members preferred Java than other names.

- Java is an island in Indonesia where the first coffee was produced (called Java coffee). It is a kind of espresso bean. Java name was chosen by James Gosling while having a cup of coffee nearby his office.

- Notice that Java is just a name, not an acronym.

- Initially developed by James Gosling at Sun Microsystems.

- In 1995, Time magazine called **Java one of the Ten Best Products of 1995**.

Versions of java

- JDK 1.0 was released on January 23, 1996. After the first release of Java, there have been many additional features added to the language.
- JDK Alpha and Beta (1995)
- JDK 1.0 (23rd Jan 1996)
- JDK 1.1 (19th Feb 1997)
- J2SE 1.2 (8th Dec 1998)
- J2SE 1.3 (8th May 2000)
- J2SE 1.4 (6th Feb 2002)
- J2SE 5.0 (30th Sep 2004)
- Java SE 6 (11th Dec 2006)
- Java SE 7 (28th July 2011)

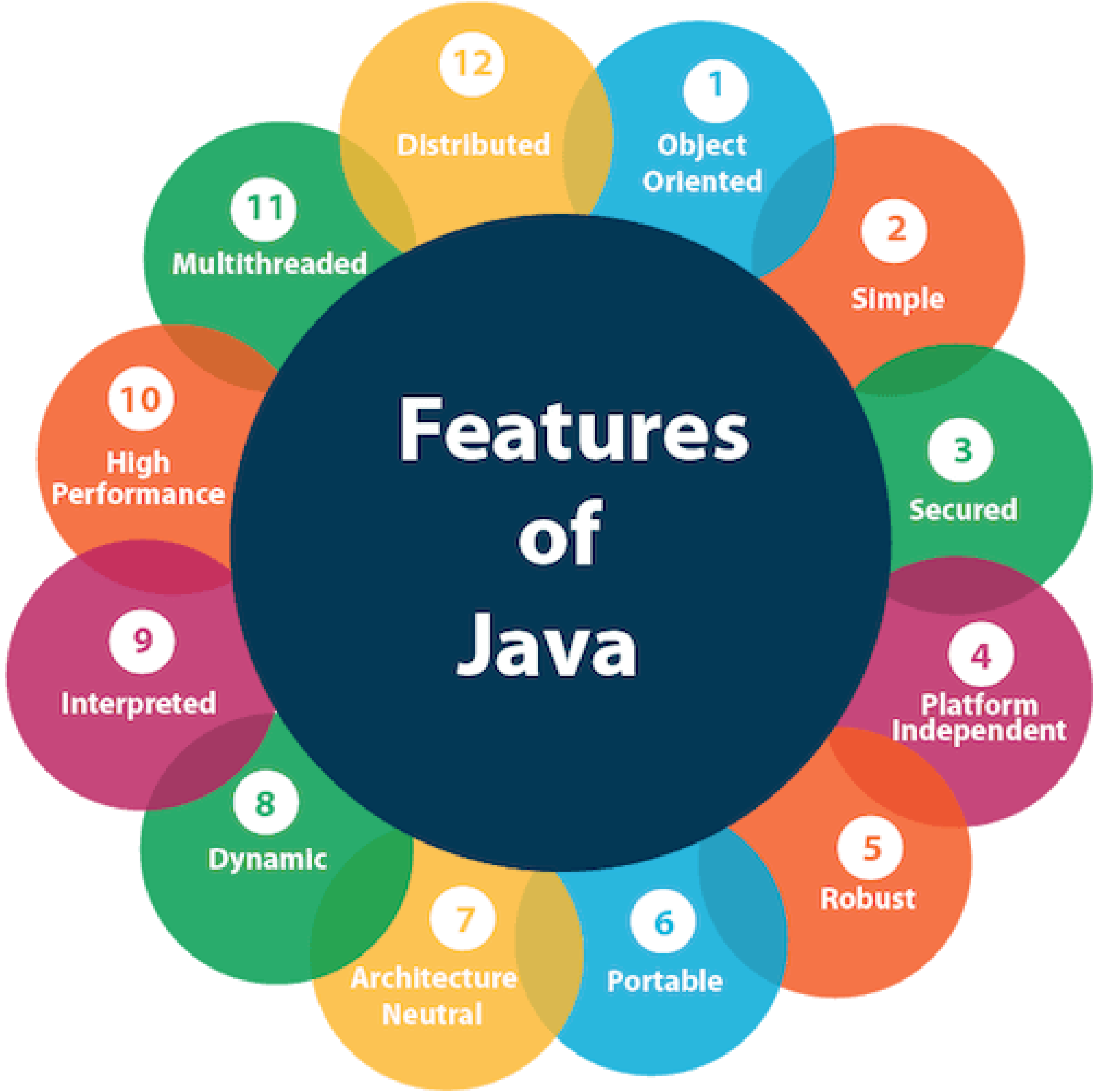
Versions of java…

- Java SE 8 (18th Mar 2014)
- Java SE 9 (21st Sep 2017)
- Java SE 10 (20th Mar 2018)
- Java SE 11 (September 2018)
- Java SE 12 (March 2019)
- Java SE 13 (September 2019)
- Java SE 14 (Mar 2020)
- Java SE 15 (September 2020)
- Java SE 16 (Mar 2021)
- Java SE 17 (September 2021)
- Java SE 18 (to be released by March 2022)
- Since Java SE 8 release, the Oracle corporation follows a pattern in which every even version is release in March month and an odd version released in September month.

Java Buzzwords or Features of Java

The Java programming language is a high-level language that can be characterized by all of the following buzzwords:

buzzword: A trendy word or phrase that is used more to impress than explain

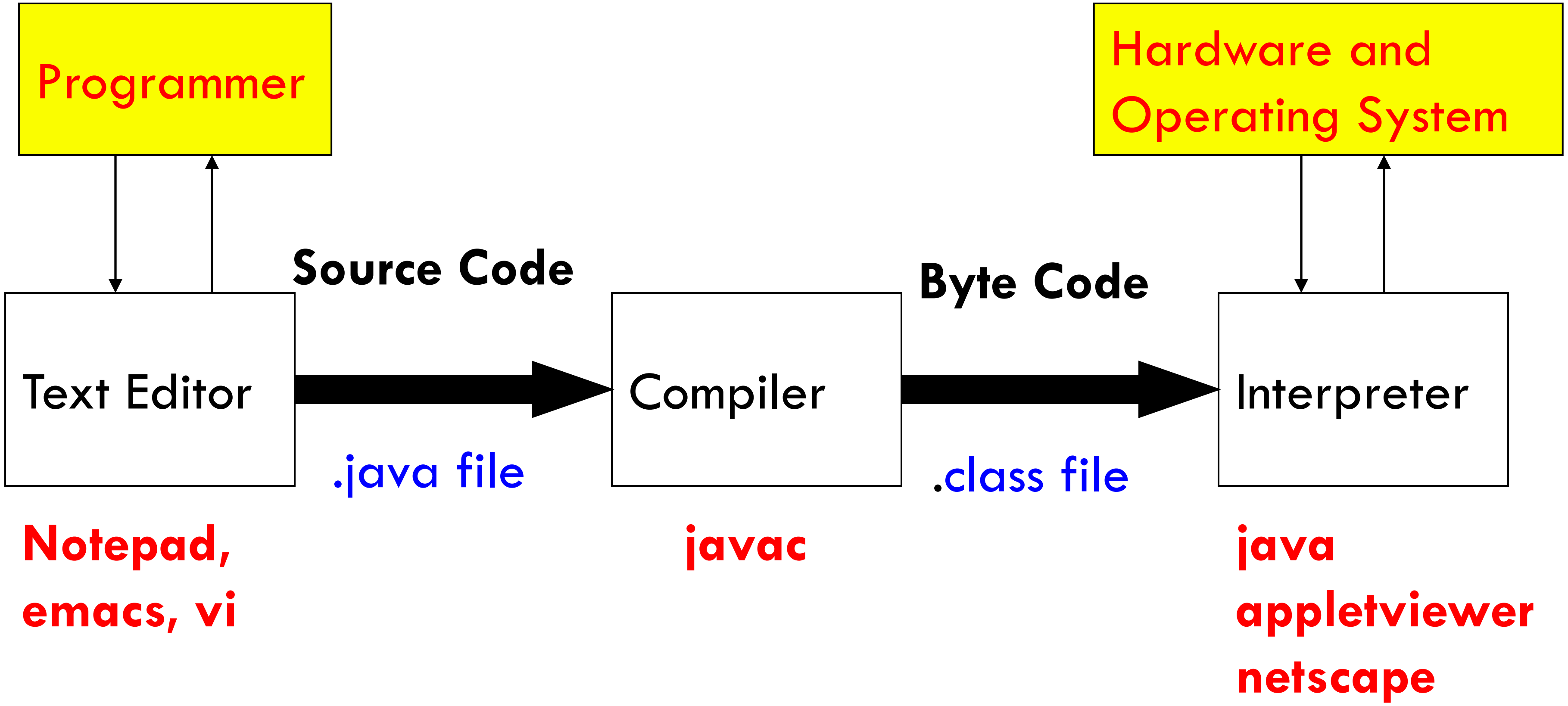


"Java is a simple, object-oriented, distributed,
interpreted, robust, secure, architecture-neutral,
portable, high performance, multi-threaded, and
dynamic language."

Java Attributes

- Familiar, Simple, Small
- Compiled and Interpreted
- Platform-Independent and Portable
- Object-Oriented
- Robust and Secure
- Distributed
- Multithreaded and Interactive
- High Performance
- Dynamic and Extensible

Java is Compiled and Interpreted



Programmer

Hardware and Operating System

Source Code

Byte Code

Text Editor

Compiler

Interpreter

.java file

.class file

Notepad, emacs, vi

javac

java
appletviewer
netscape

Java is both ***compiled*** and an ***interpreted*** lang. First the java compiler translates source code into the byte code instructions. In the next stage the java interpreter converts the byte code instructions to Machine Code.

1. Simple

- Java was designed to be easy for a professional programmer to learn and use effectively.
- It’s simple and easy to learn if you already know the basic concepts of Object Oriented Programming.
- Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.

2. Object Oriented

- Java is a true object-oriented programming language.
- Almost the **“Everything is an Object”** paradigm. All program code and data reside within objects and classes.
- The object model in Java is simple and easy to extend.
- Java comes with an extensive set of classes, arranged in packages that can be used in our programs through inheritance.

3. Distributed

- Java is designed to create distributed applications on networks.
- Java applications can access remote objects on the Internet as easily as they can do in the local system.
- Java enables multiple programmers at multiple remote locations to collaborate and work together on a single project.

4. Compiled and Interpreted

- Usually, a computer language is either compiled or Interpreted. Java combines **both** this approach and makes it a two-stage system.
- **Compiled:** Java enables the creation of cross-platform programs by compiling them into an intermediate representation called Java Bytecode.
- **Interpreted:** Bytecode is then interpreted, which generates machine code that can be directly executed by the machine that provides a Java Virtual machine.

5. Robust

- It provides many features that make the program execute reliably in a variety of environments.
- Java is a strictly typed language. It checks code both at compile time and runtime.

6. Secure

- Java provides a “firewall” between a networked application and your computer.
- When a Java Compatible Web browser is used, downloading can be done safely without fear of viral infection or malicious intent.

7. Architecture Neutral

- Java language and Java Virtual Machine helped in achieving the goal of **“write once; run anywhere, any time, forever.”**
- Changes and upgrades in operating systems, processors and system resources will not force any changes in Java Programs.

8. Portable

- Java Provides a way to download programs dynamically to all the various types of platforms connected to the Internet.
- Java is portable because of the Java Virtual Machine (JVM). The JVM provides a consistent environment for Java programs to run on, regardless of the underlying hardware and operating system. This means that a Java program can be written on one device and run on any other device with a JVM installed, without any changes or modifications.

9. High Performance

- Java performance is high because of the use of bytecode.
- The bytecode was used so that it can be easily translated into native machine code.

10. Multithreaded

- Multithreaded Programs handled **multiple tasks simultaneously**, which was helpful in creating interactive, networked programs.
- Java run-time system comes with tools that support multiprocess synchronization used to construct smoothly interactive systems.

11. Dynamic

- Java is capable of linking in new class libraries, methods, and objects.
- Java programs carry with them substantial amounts of run-time type information that is used to verify and resolve accesses to objects at runtime. This makes it possible to dynamically link code in a safe and expedient manner.

Java Comments

Why do we use comments in a code?

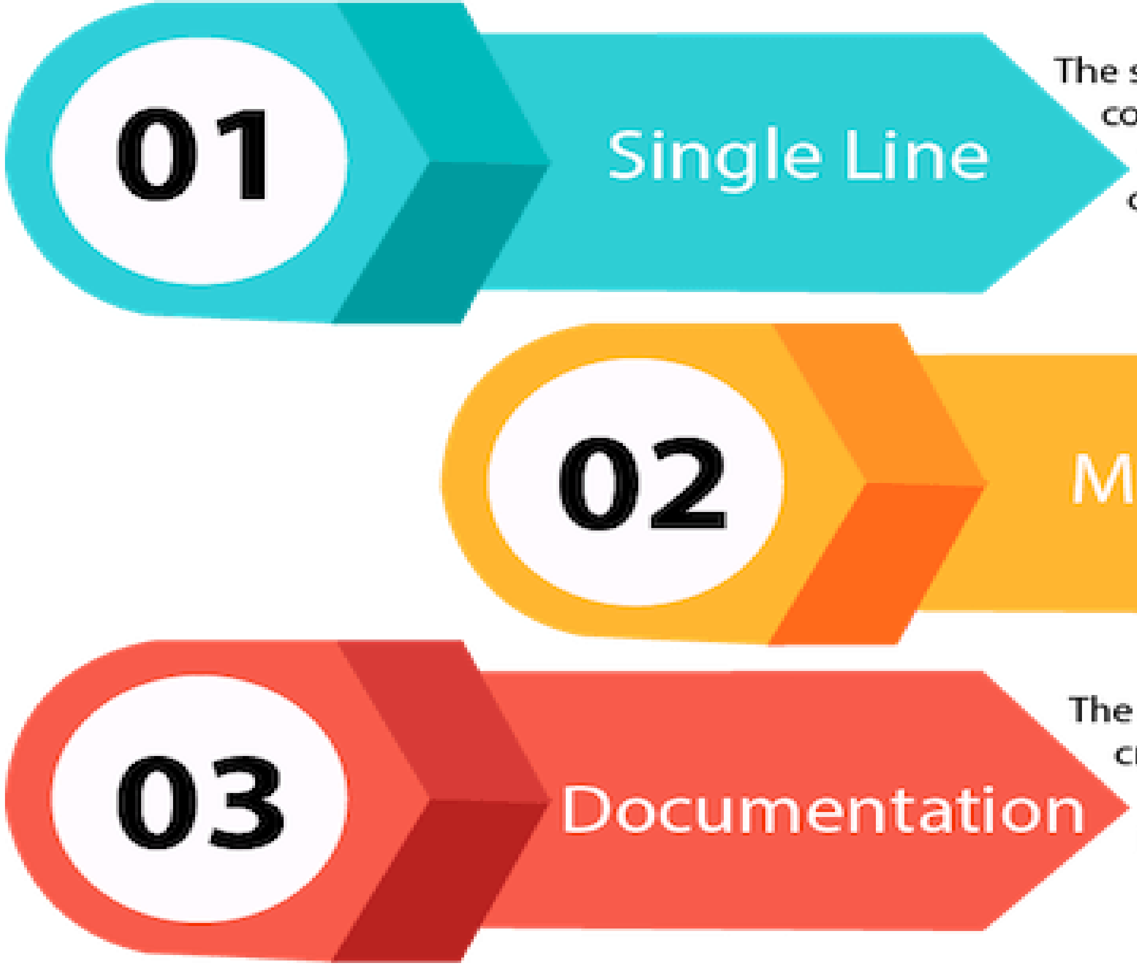
- Comments are used to make the program more readable by adding the details of the code.
- It makes easy to maintain the code and to find the errors easily.
- The comments can be used to provide information or explanation about the variable, method, class, or any statement.
- It can also be used to prevent the execution of program code while testing the alternative code.

Types of Java Comments

There are three types of comments in Java.

1. Single Line Comment
2. Multi Line Comment
3. Documentation Comment

Types of Java Comments



01 Single Line

The single line comment is used to comment only one line.

02 Multi Line

The multi line comment is used to comment multiple lines of code.

03 Documentation

The documentation comment is used to create documentation API. To create documentation API, you need to use javadoc tool.

1) Java Single Line Comment

The single-line comment is used to comment only one line of the code. It is the widely used and easiest way of commenting the statements.

Single line comments starts with two forward slashes **(//)**. Any text in front of // is not executed by Java.

Syntax:

```
//This is single line comment
```

2) Java Multi Line Comment

The multi-line comment is used to comment multiple lines of code. It can be used to explain a complex code snippet or to comment multiple lines of code at a time (as it will be difficult to use single-line comments there).

Multi-line comments are placed between /* and */. Any text between /* and */ is not executed by Java.

Syntax:

```
/*
This
is
multi line
comment
*/
```

3) Java Documentation Comment

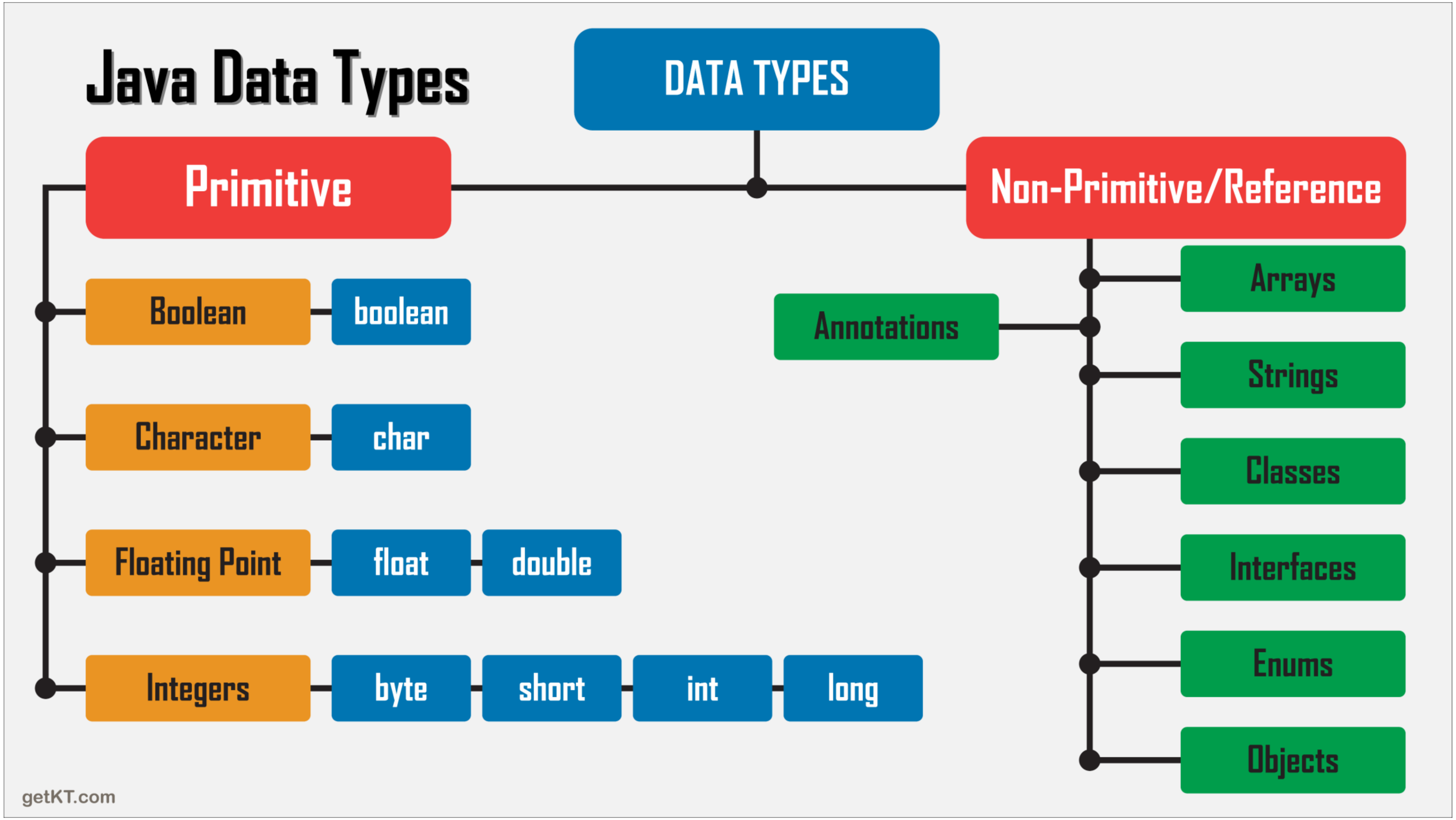
Documentation comments are usually used to write large programs for a project or software application as it helps to create documentation API. These APIs are needed for reference, i.e., which classes, methods, arguments, etc., are used in the code.

To create documentation API, we need to use the **javadoc tool**. The documentation comments are placed between /** and */.

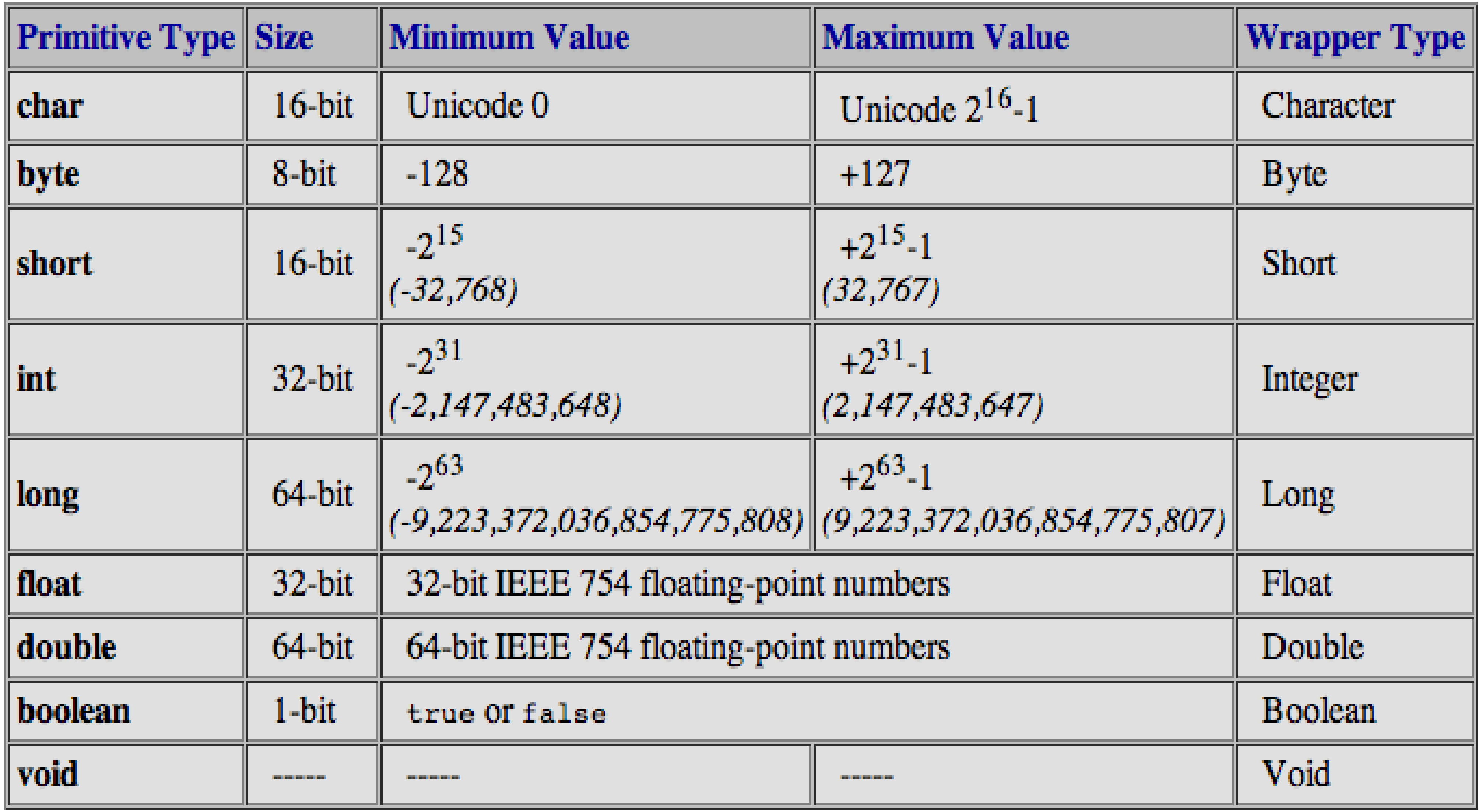
Syntax:

```
/**
*
*We can use various tags to depict the parameter
*or heading or author name
*We can also use HTML tags
*
*/
```

Data Types in Java



Java is a statically-typed programming language. It means, all variables must be declared before its use. That is why we need to declare variable's type and name.



Primitive Type	Size	Minimum Value	Maximum Value	Wrapper Type
char	16-bit	Unicode 0	Unicode 2^{16}-1	Character
byte	8-bit	-128	+127	Byte
short	16-bit	-2^{15} *(-32,768)*	$+2^{15}$-1 *(32,767)*	Short
int	32-bit	-2^{31} *(-2,147,483,648)*	$+2^{31}$-1 *(2,147,483,647)*	Integer
long	64-bit	-2^{63} *(-9,223,372,036,854,775,808)*	$+2^{63}$-1 *(9,223,372,036,854,775,807)*	Long
float	32-bit	32-bit IEEE 754 floating-point numbers		Float
double	64-bit	64-bit IEEE 754 floating-point numbers		Double
boolean	1-bit	`true` or `false`		Boolean
void	-----	-----	-----	Void

Enumerated Data Type:

enum - reference data type whenever you have fixed no of objects at that time we can use enum.

Example :Number of Days – 7, Number of Months – 12

1. enum may have constructor
2. enum may have methods

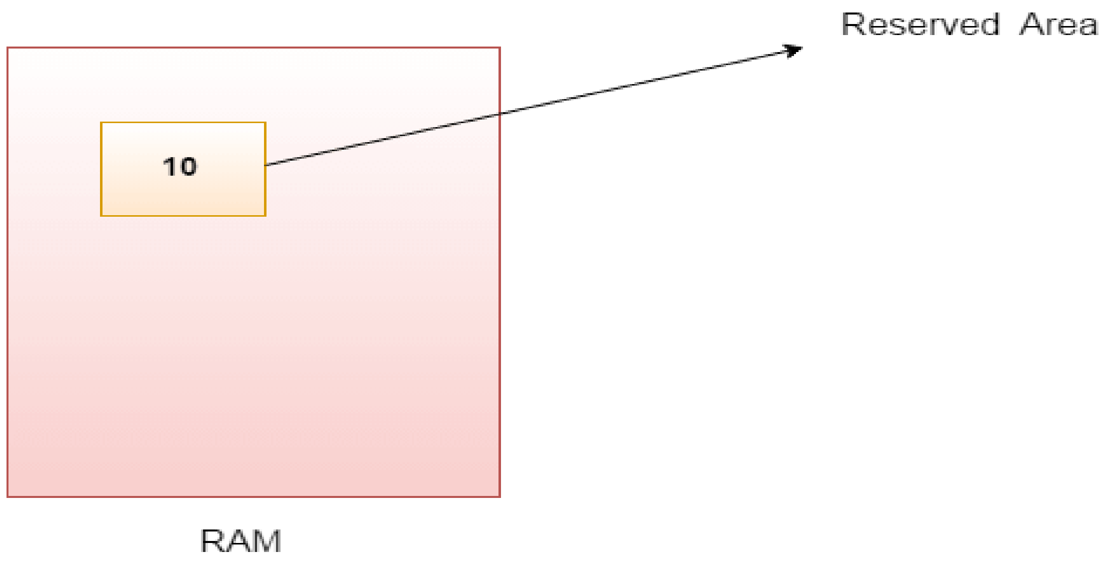
- ➢ name() - It returns name of instance
- ➢ values() - It returns array of all the instance available in enum
- ➢ ordinal() - It returns the position of instance. Index startwith 0.

Java Variables

A variable is a container which holds the value while the Java program is executed. A variable is assigned with a data type.

Variable is a name of memory location. There are three types of variables in java: **local, instance and static.**

There are two types of data types in Java: primitive and non-primitive.



Variable

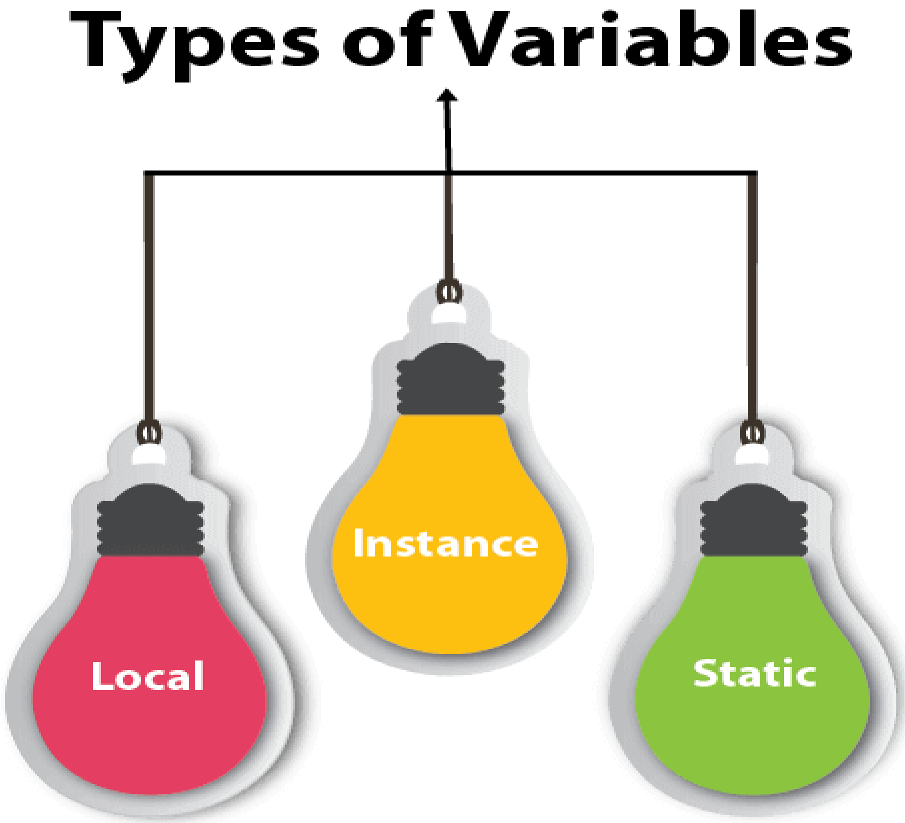
A variable is the name of a reserved area allocated in memory. In other words, it is a name of the memory location. It is a combination of "vary + able" which means its value can be changed.

```
int data=50;//Here data is variable
```

Types of Variables

There are three types of variables in Java:

- local variable
- instance variable
- static variable



1) Local Variable

- A variable declared **inside the body of the method** is called local variable.

- You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.

- A local variable **cannot** be defined with "**static**" keyword.

2) Instance Variable

- A variable declared **inside the class but outside the body of the method**, is called an instance variable. It is not declared as static.

- It is called an instance variable because its value is **instance-specific** and is not shared among instances.

Java Variable Example: Add Two Numbers

```
public class Simple{
public static void main(String[] args){
int a=10;
int b=10;
int c=a+b;
System.out.println(c);
}
}
```

Output:

```
20
```

3) Static variable

A variable that is declared as static is called a static variable. It cannot be local. You can create a single copy of the static variable and share it among all the instances of the class. Memory allocation for static variables happens only once when the class is loaded in the memory.

Example to understand the types of variables in java

```
public class A
{
    static int m=100;//static variable
             int data=50;//instance variable
        public static void main(String args[])
    {
          int n=10;//local variable
    }
}//end of class
```

Java Variable Example: Widening(Type conversion)

```
public class Simple{
public static void main(String[] args){
int a=10;
float f=a;
System.out.println(a);
System.out.println(f);
}
}
```

Output:

```
10
10.0
```

If a data type is automatically converted into another data type at compile time is known as type conversion.

Java Variable Example: Narrowing (Type casting)

```
public class Simple{
public static void main(String[] args){
float f=10.5f;
//int a=f;//Compile time error
int a=(int)f;
System.out.println(f);
System.out.println(a);
}
}
```

When a data type is converted into another data type by a programmer or user while writing a program code of any programming language, the mechanism is known as **type casting**.

Output:

```
10.5
10
```

Java Variable Example: Overflow(byte range -128 to +127)

```
class Simple{
public static void main(String[] args){
//Overflow
int a=130;
byte b=(byte)a;
System.out.println(a);
System.out.println(b);
}
}
```

Output:

```
130
-126
```

Understanding First Java Program:

Let's see what is the meaning of class, public, static, void, main, String[], System.out.println().

- **class** keyword is used to declare a class in java.
- **public** keyword is an access modifier which represents visibility, it means it is visible to all.
- **static** is a keyword, if we declare any method as static, it is known as static method. The core advantage of static method is that there is **no need to create object to invoke the static method**. The main method is executed by the JVM, so it doesn't require to create object to invoke the main method. So, it saves memory.
- **void** is the return type of the method; it means it doesn't return any value.
- **main** represents startup of the program.
- **String[] args** is used for command line argument. We will learn it later.
- **System.out.println()** is used print statement. We will learn about the internal working of System.out.println statement later.

How many ways can we write a java program

There are many ways to write a java program. The modifications that can be done in a java program are given below:

1) By changing sequence of the modifiers, method prototype is not changed.

Let's see the simple code of main method.

1. static public void main(String args[])

2) subscript notation in java array can be used after type, before variable or after variable.

Let's see the different codes to write the main method.

1. public static void main(String[] args)
2. public static void main(String []args)
3. public static void main(String args[])

3) You can provide var-args support to main method by passing 3 ellipses (dots)

Let's see the simple code of using var-args in main method.

```
public static void main(String... args)
```

4) Having semicolon at the end of class in java is optional.

Let's see the simple code.

```
class A{
static public void main(String... args){
System.out.println("hello java4");
}
};
```

Valid java main method signature

```
public static void main(String[] args)
public static void main(String []args)
public static void main(String args[])
public static void main(String... args)
static public void main(String[] args)
public static final void main(String[] args)
final public static void main(String[] args)
final strictfp public static void main(String[] args)
```

Invalid java main method signature

```
public void main(String[] args)
static void main(String[] args)
public void static main(String[] args)
abstract public static void main(String[] args)
```

Complete the code segment **to find the perimeter and area of a circle given a value of radius**.

```
import java.util.Scanner;
public class Exercise1_1 {
        public static void main(String[] args) {
Scanner s = new Scanner(System.in);
        double radius= s.nextDouble();
        double perimeter;
        double area;
//Calculate the perimeter
  //Calculate the area

if(radius < 0)
  System.out.print(" please enter non zero positive number ");
else
{
  perimeter = 2*Math.PI*radius;
  area = Math.PI*radius*radius;
    System.out.println(perimeter);
  System.out.print(area);
}
  }
}
```

Java Arrays

Normally, an array is a collection of similar type of elements which has contiguous memory location.

- **Java array** is an object which contains elements of a similar data type. Additionally, The elements of an array are stored in a contiguous memory location.
- It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.
- Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.

Advantages

- **Code Optimization:** It makes the code optimized, we can retrieve or sort the data efficiently.
- **Random access:** We can get any data located at an index position.

Disadvantages

- **Size Limit:** We can store only the fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in Java which grows automatically.

Instantiation of an Array in Java

```
arrayRefVar=new datatype[size];
```

```
class Testarray{
public static void main(String args[]){
int a[]=new int[5];//declaration and instantiation
a[0]=10;//initialization
a[1]=20;
a[2]=70;
a[3]=40;
a[4]=50;
//traversing array
for(int i=0;i<a.length;i++)//length is the property of
 array
System.out.println(a[i]);
}}
```

Example to instantiate Multidimensional Array in Java

```
int[][] arr=new int[3][3];//3 row and 3 column
```

Example to initialize Multidimensional Array in Java

```
1.arr[0][0]=1;
2.arr[0][1]=2;
3.arr[0][2]=3;
4.arr[1][0]=4;
5.arr[1][1]=5;
6.arr[1][2]=6;
7.arr[2][0]=7;
8.arr[2][1]=8;
9.arr[2][2]=9;
```

```
int arr[][]={{1,2,3},{4,5,6},{7,8,9}};
```

Jagged Array in Java

If we are creating odd number of columns in a 2D array, it is known as a jagged array. In other words, it is an array of arrays with different number of columns.

```
//Java Program to illustrate the jagged array
class TestJaggedArray{
    public static void main(String[] args){
        //declaring a 2D array with odd columns
        int arr[][] = new int[3][];
        arr[0] = new int[3];
        arr[1] = new int[4];
        arr[2] = new int[2];
        //initializing a jagged array
        int count = 0;
        for (int i=0; i<arr.length; i++)
            for(int j=0; j<arr[i].length; j++)
                arr[i][j] = count++;
```

```
//printing the data of a jagged array
        for (int i=0; i<arr.length; i++){
            for (int j=0; j<arr[i].length; j++){
                System.out.print(arr[i][j]+" ");
            }
            System.out.println();//new line
        }    }          }
```

Output:

```
0 1 2
3 4 5 6
7 8
```

Java Command Line Arguments

The java command-line argument is an argument i.e. passed at the time of running the java program.

The arguments passed from the console can be received in the java program and it can be used as an input.

So, it provides a convenient way to check the behavior of the program for the different values. You can pass **N** (1,2,3 and so on) numbers of arguments from the command prompt.

Ex:

```
class CommandLineExample{
public static void main(String args[]){
System.out.println("Your first argument is: "+args[0]);

}
}
```

```
compile by > javac CommandLineExample.java
run by > java CommandLineExample hello
```

Example of command-line argument that prints all the values

```
class A{
public static void main(String args[]){

for(int i=0;i<args.length;i++)
System.out.println(args[i]);

}
}
```

```
compile by > javac A.java
run by > java A Hari Sai 1 3 abc
```

Output: Hari Sai 1 3 abc

Program to read values from the keyboard

```
import java.util.Scanner;
public class Display {
       public static void main(String[] args) {
         Scanner s = new Scanner(System.in);
            System.out.println("Enter rollno:");
        int rollno= s.nextInt();
         System.out.println("Enter name:");
String name= s.next();
 System.out.println("Enter marks:");
  double marks= s.nextDouble();
  System.out.println("My rollno is:" +rollno);
   System.out.println("My name is:" +name);
  System.out.print("My marks :" +marks);
       }
}
```

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.

Operator Type	Category	Precedence
Unary	postfix	*expr*++ *expr*--
	prefix	++*expr* --*expr* +*expr* -*expr* ~ !
Arithmetic	multiplicative	* / %
	additive	+ -
Shift	shift	<< >> >>>
Relational	comparison	< > <= >= instanceof
	equality	== !=
Bitwise	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	|
Logical	logical AND	&&
	logical OR	||
Ternary	ternary	? :
Assignment	assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Java Unary Operator

```
public 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
}}
```

Java Arithmetic Operator Example

```
public 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 Left Shift Operator Example

```
public 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 Right Shift Operator Example

```
public 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 AND Operator Example: Logical && and Bitwise &

The logical && operator doesn't check the second condition if the first condition is false. It checks the second condition only if the first one is true.

The bitwise & operator always checks both conditions whether first condition is true or false.

```
public 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

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 checked
}}
```

Java OR Operator Example: Logical || and Bitwise |

The logical || operator doesn't check the second condition if the first condition is true. It checks the second condition only if the first one is false.

The bitwise | operator always checks both conditions whether first condition is true or false.

```
public 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

System.out.println(a>b||a++<c);//true || true = true
System.out.println(a);//10 because second condition is not checked
System.out.println(a>b|a++<c);//true | true = true
System.out.println(a);//11 because second condition is checked

}}
```

Java Ternary Operator

Java Ternary operator is used as one line replacement for if-then-else statement and used a lot in Java programming. It is the only conditional operator which takes three operands.

Java Ternary Operator Example

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

Java Assignment Operator

Java assignment operator is one of the most common operators. It is used to assign the value on its right to the operand on its left.

```
public class OperatorExample{
public static void main(String args[]){
int a=10;
a+=3;// a=a+3 (10+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 Control Statements | Control Flow in Java

Java compiler **executes** the code from **top to bottom**. The statements in the code are executed according to the order in which they appear.

However, Java provides statements that can be used to control the flow of Java code. Such statements are called control flow statements.

It is one of the fundamental features of Java, which provides a smooth flow of program.
Java provides three types of control flow statements.

1.Decision Making statements

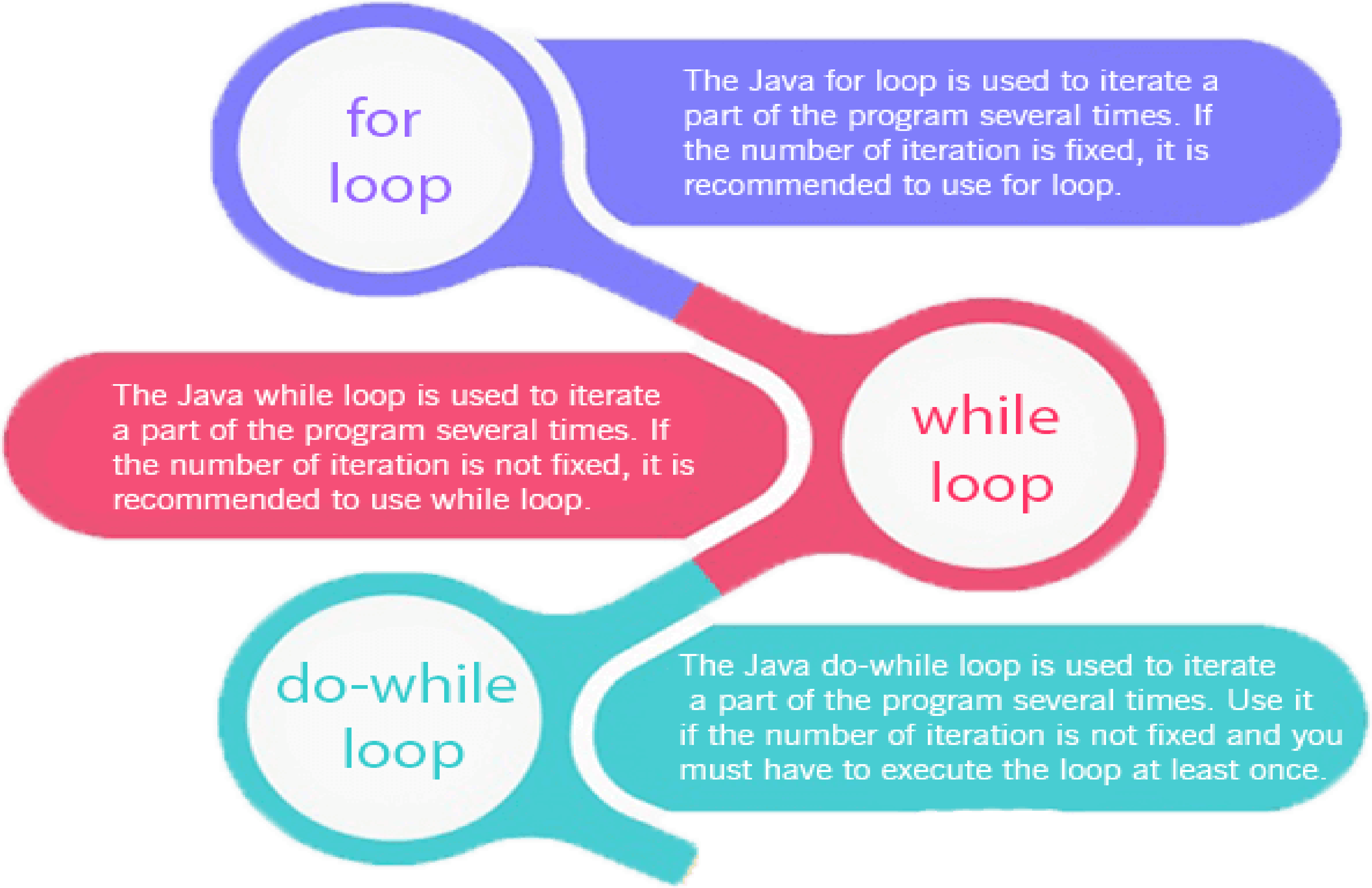
1. if statements
2. switch statement

2.Loop statements

1. do while loop
2. while loop
3. for loop
4. for-each loop

3.Jump statements

1. break statement
2. continue statement



for loop

The Java for loop is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop.

while loop

The Java while loop is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.

do-while loop

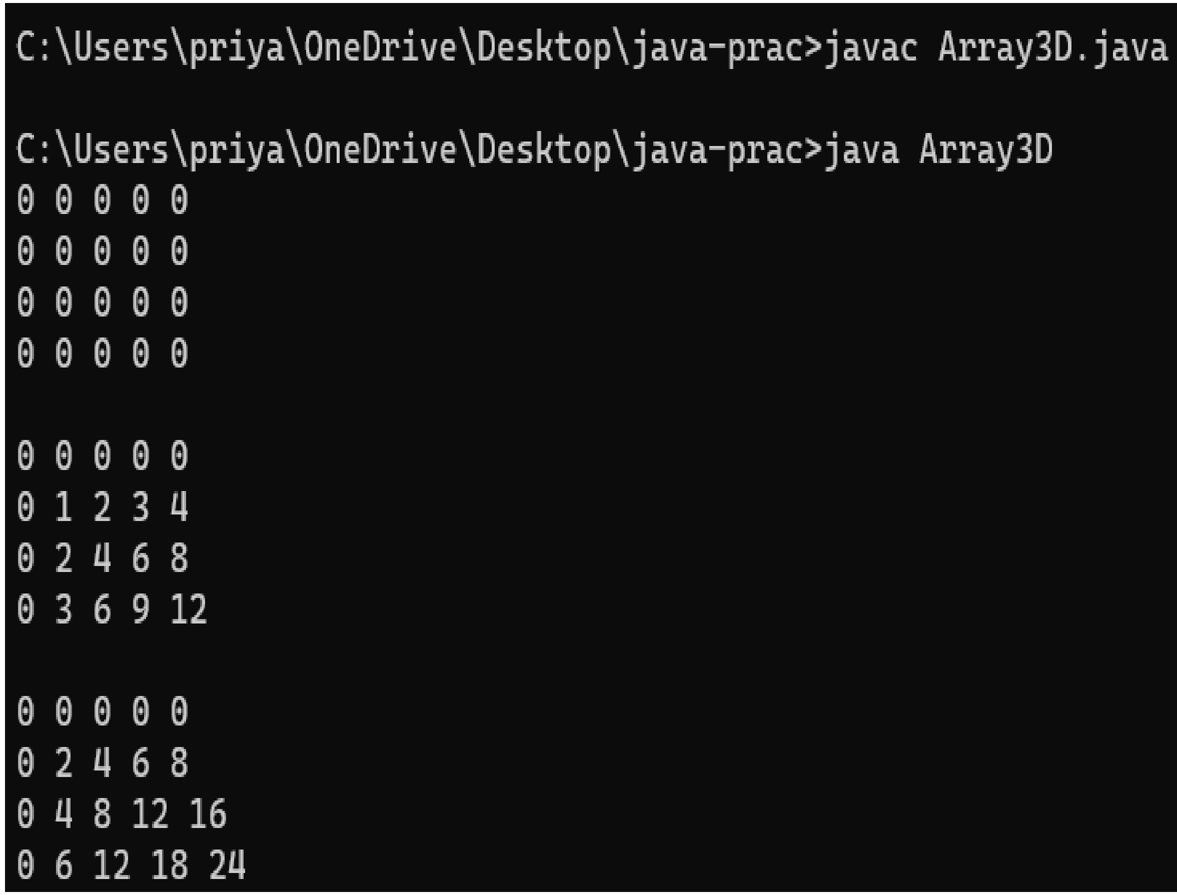
The Java do-while loop is used to iterate a part of the program several times. Use it if the number of iteration is not fixed and you must have to execute the loop at least once.

```
public class Student {
public static void main(String[] args) {
String address = "Telangana, India";

if(address.endsWith("India"))
{
    if(address.contains("Hyderabad")) {
    System.out.println("Your city is Hyderabad");
            }
    else if(address.contains("Medak")) {
    System.out.println("Your city is Medak");
                 }
    else {
    System.out.println(address.split(",")[0]);
                }
}
else {
System.out.println("You are not living in India");
                             }
              }
}
```

1.Simple if statement
2.if-else statement
3.if-else-if ladder
4.Nested if-statement

Array3D.java

```
class Array3D {
     public static void main(String args[]) {
       int my3DArray [ ][ ][ ] = new int [3][4][5];
       int i, j, k;
    for(i=0; i<3; i++)
          for(j=0; j<4; j++)
                 for(k=0; k<5; k++)
                      my3DArray[i][j][k] = i * j * k;

    for(i=0; i<3; i++) {
          for(j=0; j<4; j++) {
                for(k=0; k<5; k++)
             System.out.print(my3DArray[i][j][k] + " ");
             System.out.println();
                }
                 System.out.println();
          }          }     }
```



```
C:\Users\priya\OneDrive\Desktop\java-prac>javac Array3D.java

C:\Users\priya\OneDrive\Desktop\java-prac>java Array3D
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0

0 0 0 0 0
0 1 2 3 4
0 2 4 6 8
0 3 6 9 12

0 0 0 0 0
0 2 4 6 8
0 4 8 12 16
0 6 12 18 24
```

Java for-each Loop

The for-each loop is used to **traverse array or collection in Java**.

It is easier to use than simple for loop because we don't need to increment value and use subscript notation.

It works on the basis of elements and not the index.

It returns element one by one in the defined variable.

ForEachExample.java

```
public class ForEachExample {
public static void main(String[] args)
{
   //Declaring an array
   int arr[]={12,23,44,56,78};
          //Printing array using for-each loop
   for(int i:arr){
      System.out.println(i);
   }
}
}
```

Java While Loop

- The Java *while loop* is used to iterate a part of the program repeatedly until the specified Boolean condition is true.
- As soon as the Boolean condition becomes false, the loop automatically stops.
- The while loop is considered as a repeating if statement. If the number of iteration is not fixed, it is recommended to use the while loop.

Example Program:

```
public class WhileExample {
public static void main(String[] args)
{
    int i=1;
    while(i<=10){
        System.out.println(i);
    i++;
    }
}
}
```

Java do-while Loop

Java do-while loop is called an **exit control loop**.

Therefore, unlike while loop and for loop, the do-while check the condition at the end of loop body.

The Java *do-while loop* is executed at least once because condition is checked after loop body.

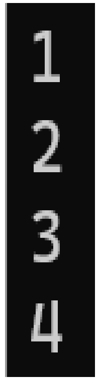
Example Program:

```
public class DoWhileExample {
public static void main(String[] args) {

    int i=1;
    do{
        System.out.println(i);
    i++;
    }while(i<=10);
}
}
```

Java Break and Continue Statement

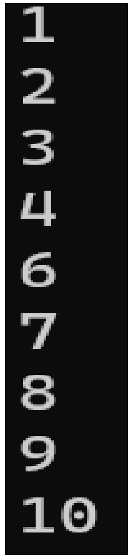
```
public class BreakExample {
public static void main(String[] args) {
    //using for loop
    for(int i=1;i<=10;i++){
        if(i==5){
            break;
            //continue;
        }
        System.out.println(i);
    }
}
}
```

Break output



```
1
2
3
4
```

Continue output



```
1
2
3
4
6
7
8
9
10
```

Java Continue Statement with Labelled For Loop

```
//Java Program to illustrate the use of continue statement
//with label inside an inner loop to continue outer loop
public class ContinueExample3 {
public static void main(String[] args) {
        aa:
        for(int i=1;i<=3;i++){
            bb:
            for(int j=1;j<=3;j++){
              if(i==2&&j==2){
                //using continue statement with label
                continue aa;
              }
              System.out.println(i+" "+j);
            }
        }
}
}
```

Naming Conventions of the Different Identifiers

Identifiers Type	Naming Rules	Examples

Class	It should start with the uppercase letter.
	It should be a noun such as Color, Button, System, Thread, etc.
	Use appropriate words, instead of acronyms.
		public class **Employee**
		{
		//code snippet
		}

Interface	It should start with the uppercase letter.
	It should be an adjective such as Runnable, Remote, ActionListener.
	Use appropriate words, instead of acronyms.
		interface **Printable**
		{
		//code snippet
		}

Method	It should start with lowercase letter.
	It should be a verb such as main(), print(), println().
	If the name contains multiple words, start it with a lowercase letter followed by an uppercase letter such as actionPerformed().
		class Employee
		{
		// method
		void **draw()**
		{
		//code snippet
		}
		}

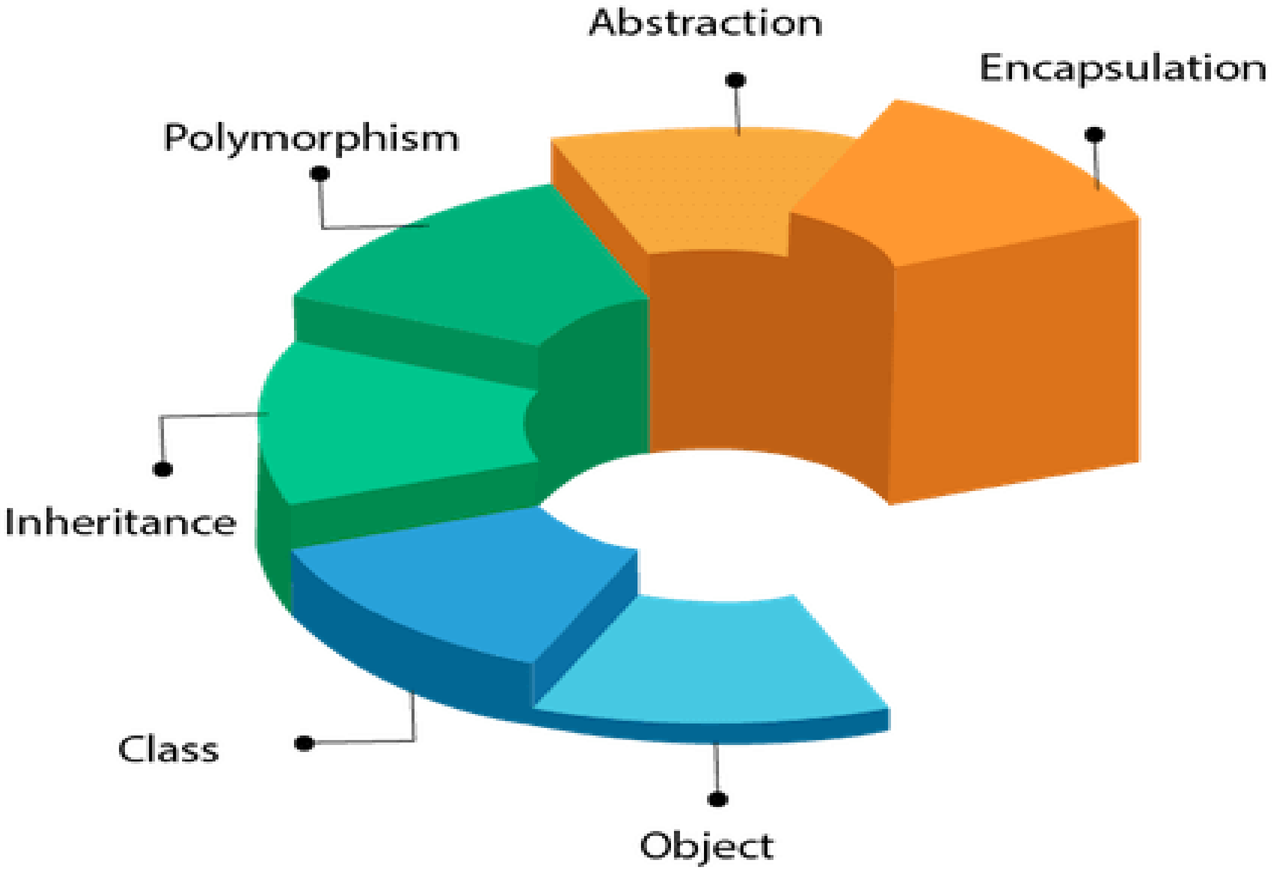
Identifiers Type	Naming Rules	Examples

Variable	It should start with a lowercase letter such as id, name.
	It should not start with the special characters like & (ampersand), $ (dollar), _ (underscore).
	If the name contains multiple words, start it with the lowercase letter followed by an uppercase letter such as firstName, lastName.
	Avoid using one-character variables such as x, y, z.
		class Employee
		{
		// variable
		int **id**;
		//code snippet
		}

Package	It should be a lowercase letter such as java, lang.
	If the name contains multiple words, it should be separated by dots (.) such as java.util, java.lang.
		//package
		package **com.javatpoint;**
		class Employee
		{
		//code snippet
		}

Constant	It should be in uppercase letters such as RED, YELLOW.
	If the name contains multiple words, it should be separated by an underscore(_) such as MAX_PRIORITY.
	It may contain digits but not as the first letter.
		class Employee
		{
		//constant
		static final int **MIN_AGE** = 18;
		//code snippet
		}

OOPs (Object-Oriented Programming System)



Class

Collection of objects is called class. It is a logical entity.

A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.

Object

Any entity that has state and behavior is known as an object.
For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.
An Object can be defined as an instance of a class. An object contains an address and takes up some space in memory. Objects can communicate without knowing the details of each other's data or code.

Constructors in Java

- In Java, a constructor is a block of codes similar to the method.
- It is called when an instance of the class is created.
- At the time of calling constructor, memory for the object is allocated in the memory.
- It is a special type of method which is used to initialize the object.
- Every time an object is created using the new() keyword, at least one constructor is called.
- It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

Rules for creating Java constructor

The rules for defining the constructor.

1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type
3. A Java constructor cannot be abstract, static, final, and synchronized

Types of Java constructors

There are two types of constructors in Java:

1. Default constructor (no-arg constructor): The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.

2. Parameterized constructor: The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

Constructor Overloading in Java

Constructor overloading in Java is a technique of having more than one constructor with different parameter lists.

They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types.

```
class ConstructorEx{
    String name;
    int rno,age;

    ConstructorEx(){
        name="ABC";
        rno=123;
             age=30;
  }
     ConstructorEx(String st,int i){
        name=st;
        rno=i;
 }
 ConstructorEx(String st,int i,int j){
        name=st;
        rno=i;
             age=j;
    }
```

```
void display()
{System.out.println(name+" "+rno+" "+age);}

    public static void main(String ag[])
    {
        ConstructorEx c1= new ConstructorEx();
 ConstructorEx c2= new ConstructorEx("DEF",456);
ConstructorEx c3= new ConstructorEx("GHI",789,20);
        c1.display();
        c2.display();
        c3.display();
            }
}
```

Output

```
ABC 123 30
DEF 456 0
GHI 789 20
```

Difference between constructor and method in Java

There are many differences between constructors and methods. They are given below.

Java Constructor	Java Method
A constructor is used to initialize the state of an object.	A method is used to expose the behavior of an object.
A constructor must not have a return type.	A method must have a return type.
The constructor is invoked implicitly.	The method is invoked explicitly.
The Java compiler provides a default constructor if you don't have any constructor in a class.	The method is not provided by the compiler in any case.
The constructor name must be same as the class name.	The method name may or may not be same as the class name.

Recursion in Java

Recursion in java is a process in which a method calls itself continuously.

A method in java that calls itself is called recursive method.

It makes the code compact but complex to understand.

Syntax:

```
returntype methodname(){
//code to be executed
methodname();//calling same method
}
```

Java Recursion Example 1: Infinite times

```
public class RecursionExample1 {
static void p(){
System.out.println("hello");
p();
}

public static void main(String[] args) {
p();  }
}
```

Output

```
hello
hello
...
java.lang.StackOverflowError
```

Java Recursion : Finite times

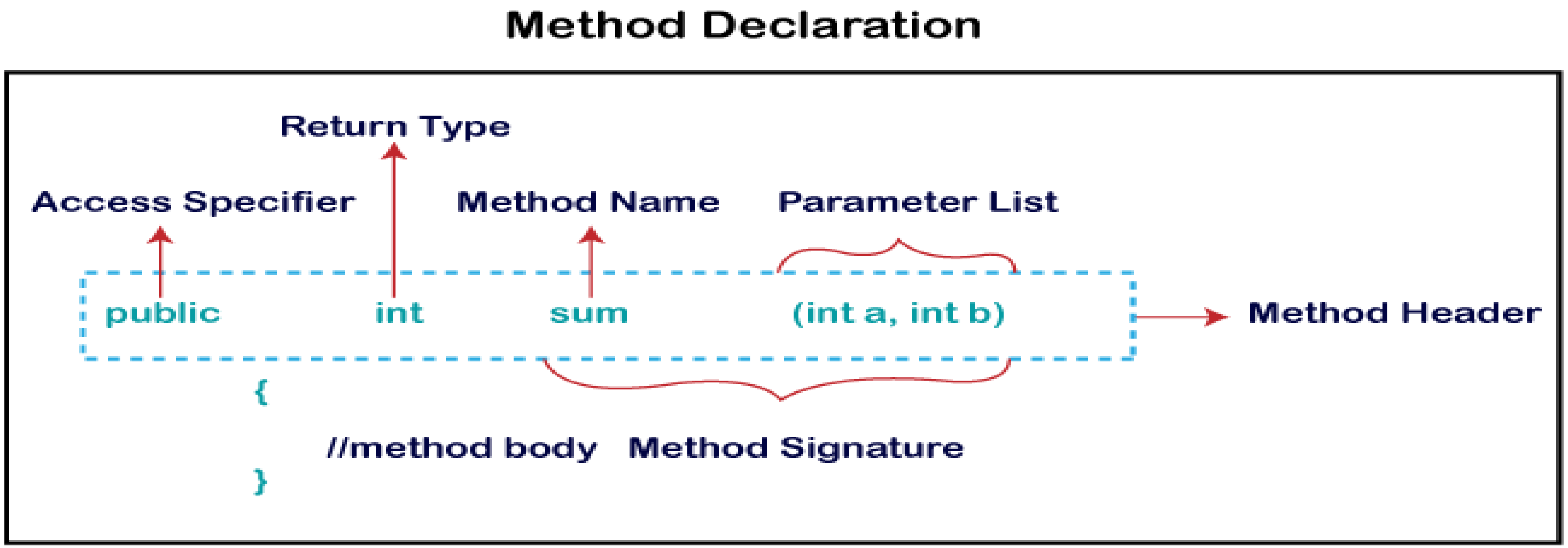
```
public class RecursionExample2 {
static int count=0;
static void p(){
count++;
if(count<=5){
System.out.println("hello "+count);
p();
}
}
public static void main(String[] args) {
p();
}
}
```

Output:

```
hello 1
hello 2
hello 3
hello 4
hello 5
```

Method in Java

- A **method** is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation.
- It is used to achieve the **reusability** of code. We write a method once and use it many times.
- We do not require to write code again and again. It also provides the **easy modification** and **readability** of code, just by adding or removing a chunk of code.
- The method is executed only when we call or invoke it.
- The most important method in Java is the **main()** method.



Method Signature: Every method has a method signature. It is a part of the method declaration. It includes the **method name** and **parameter list**.

Access Specifier: Access specifier or modifier is the access type of the method. It specifies the visibility of the method. Java provides **four** types of access specifier:

- **Public:** The method is accessible by all classes when we use public specifier in our application.

- **Private:** When we use a private access specifier, the method is accessible only in the classes in which it is defined.

- **Protected:** When we use protected access specifier, the method is accessible within the same package or subclasses in a different package.

- **Default:** When we do not use any access specifier in the method declaration, Java uses default access specifier by default. It is visible only from the same package.

Naming a Method

While defining a method, remember that the method name must be a **verb** and start with a **lowercase** letter. If the method name has more than two words, the first name must be a verb followed by adjective or noun. In the multi-word method name, the first letter of each word must be in **uppercase** except the first word.

For example:

Single-word method name: sum(), area()

Multi-word method name: areaOfCircle(), stringComparision()

It is also possible that a method has the same name as another method name in the same class, it is known as **method overloading**.

Types of Method

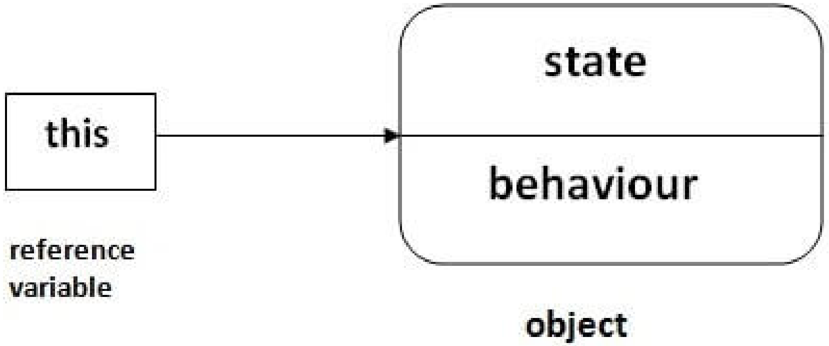
There are two types of methods in Java:

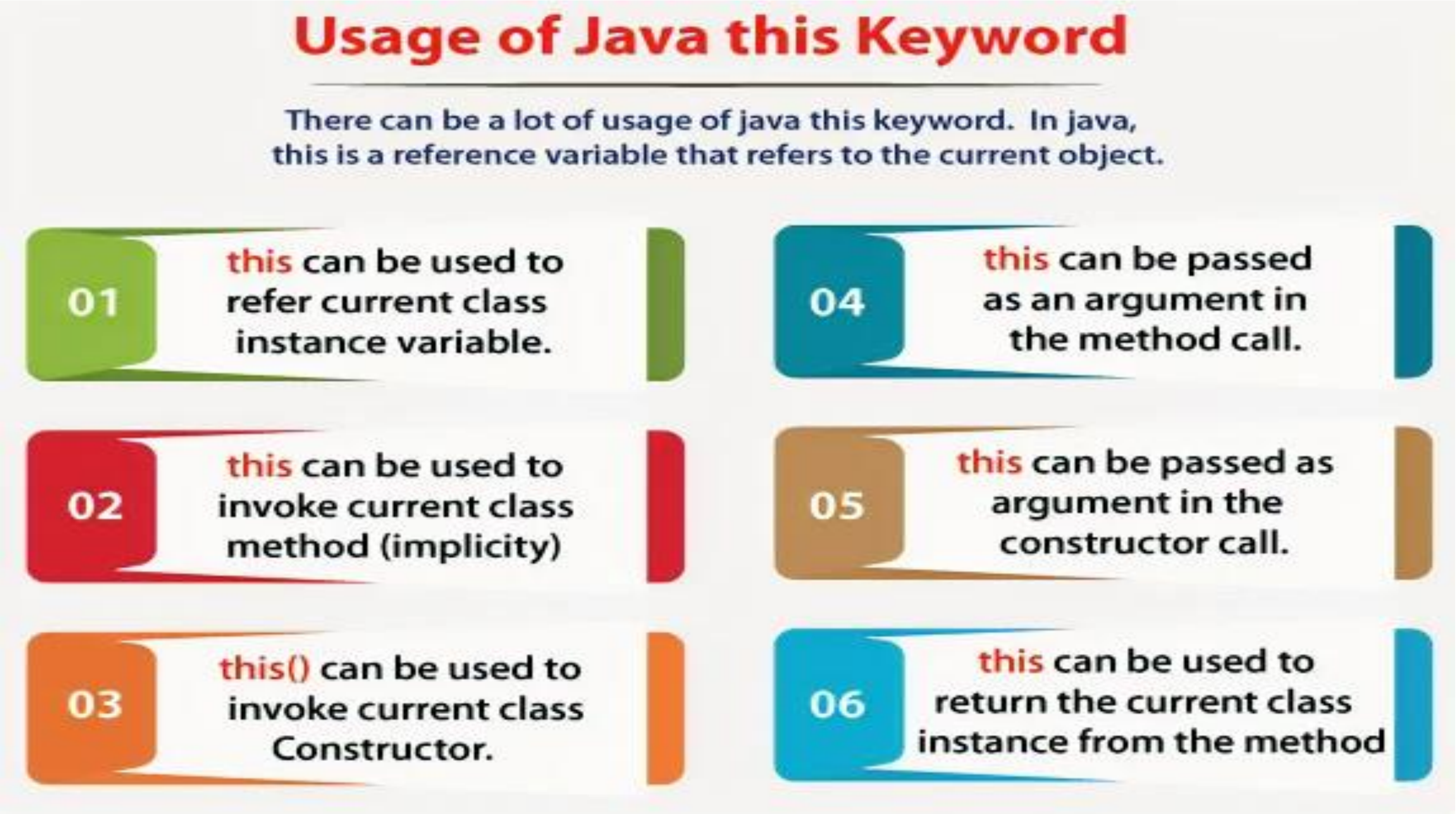
•Predefined Method (length(), equals(), compareTo())

•User-defined Method

this keyword in Java

There can be a lot of usage of **Java this keyword**. In Java, this is a **reference variable** that refers to the current object.





Usage of Java this Keyword

There can be a lot of usage of java this keyword. In java, this is a reference variable that refers to the current object.

01 this can be used to refer current class instance variable.

02 this can be used to invoke current class method (implicity)

03 this() can be used to invoke current class Constructor.

04 this can be passed as an argument in the method call.

05 this can be passed as argument in the constructor call.

06 this can be used to return the current class instance from the method

1) this: to refer current class instance variable

```
class Student{
int rollno;
String name;
float fee;
Student(int rollno,String name,float fee){
rollno=rollno;
name=name;
fee=fee;  }
void display(){System.out.println(rollno+" "+name+" "+fee);}
  }
 public class TestThis1{
public static void main(String args[]){
Student s1=new Student(111,"ABC",5000f);
Student s2=new Student(112,“DEF",6000f);
s1.display();
s2.display();
}}
```

Output:(with out this keyword)

```
0 null 0.0
0 null 0.0
```

Output:(with this keyword)

```
111 ABC 5000.0
112 DEF 6000.0
```

b) this: difference between local and instance variables

```
class Student{
int i=10;
void display(){
            int i=20;
System.out.println(i);
                         }
            }
 public class TestThis{
public static void main(String args[]){
s1.display();
}}
```

Output: without this

```
20
```

Output: with this

```
10
```

2) this: to invoke current class method

```
class A{
void m(){System.out.println("hello m");}
void n(){
System.out.println("hello n");
//m();//same as this.m()
this.m();
}    }
class TestThis4{
public static void main(String args[]){
A a=new A();
a.n();   }
      }
```

- You may invoke the method of the current class by using the this keyword.

- If you don't use the this keyword, compiler automatically adds this keyword while invoking the method.

Output:

```
hello n
hello m
```

3) this() : to invoke current class constructor

The this() constructor call can be used to invoke the current class constructor. It is used to reuse the constructor. In other words, it is used for constructor chaining.

a) Calling default constructor from parameterized constructor:

```
class A{

A(){System.out.println("hello a");}

A(int x){
this();
System.out.println(x);
}
}

class TestThis{
public static void main(String args[]){
A a=new A(10);
}}
```

Output:

```
hello a
10
```

b) Calling parameterized constructor from default constructor:

```
class A{

A(){
this(5);
System.out.println("hello a");
}

A(int x){
System.out.println(x);
}  }
class TestThis{
public static void main(String args[]){
A a=new A();
}}
```

Output:

```
5
hello a
```

Real usage of this() constructor call

- The this() constructor call should be used to reuse the constructor from the constructor.
- It maintains the chain between the constructors i.e. it is used for constructor chaining.

```
class Student{
int rollno;
String name,course;
float fee;
Student(int rollno,String name,String course){
this.rollno=rollno;
this.name=name;
this.course=course;
}
Student(int rollno,String name,String course,
float fee){
this(rollno,name,course);//reusing constructor
this.fee=fee;
}
```

```
void  display(){System.out.println(rollno+"  "+name
+" "+course+" "+fee);}
              }
class TestThis7{
public static void main(String args[]){
Student s1=new Student(111,"ankit","java");
Student s2=new Student(112,"sumit","java",6000f);

s1.display();
s2.display();
}}
```

4) this: to pass as an argument in the method

The this keyword can also be passed as an argument in the method. It is mainly used in the event handling.

```
class S2{
  void m(S2 obj){
  System.out.println("method is invoked");
  }
  void p(){
  m(this);
  }
  public static void main(String args[]){
  S2 s1 = new S2();
  s1.p();
  }
}
```

5) this: to pass as argument in the constructor call

We can pass the this keyword in the constructor also. It is useful if we have to use one object in multiple classes.

```
class B{
  A obj;

  B(A obj){
    this.obj=obj;
  }

  void display(){
    System.out.println(obj.data);//using data mem
ber of A class
  }
}
```

```
class A{
  int data=10;

  A(){
   B b=new B(this);
   b.display();
  }

  public static void main(String args[]){
   A a=new A();
  }
}
```

Java Enums

The **Enum in Java** is a data type which contains a fixed set of constants.

It can be used for days of the week (SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, and SATURDAY) , directions (NORTH, SOUTH, EAST, and WEST), season (SPRING, SUMMER, WINTER, and AUTUMN or FALL), colors (RED, YELLOW, BLUE, GREEN, WHITE, and BLACK) etc. (represents group of named constants)

According to the Java naming conventions, we should have all constants in capital letters. So, we have enum constants in capital letters.

Java Enums can be **thought of as classes** which have a fixed set of constants (a variable that does not change). The Java enum constants are static and final implicitly. It is available since JDK 1.5.

Enums are used to create **our own data type** like classes. The enum data type (also known as Enumerated Data Type) is used to define an enum in Java. Unlike C/C++, enum in Java is more powerful. Here, ***we can define an enum either inside the class or outside the class.***

Defining Java Enum

The enum can be defined within or outside the class because it is similar to a class. The semicolon (;) at the end of the enum constants are optional.

```
enum Season { WINTER, SPRING, SUMMER, FALL }
```

Or,

```
enum Season { WINTER, SPRING, SUMMER, FALL; }
```

Both the definitions of Java enum are the same.

Java Enum Example: Defined outside class

```
enum Season { WIN, SUM, RAIN }
class EnumEx{
public static void main(String[] args) {
Season s=Season.WIN;
System.out.println(s);
}}
```

Java Enum Example: Defined inside class

```
class EnumEx1{
enum Season {WIN, SUM, RAIN }
public static void main(String[] args) {
Season s=Season.WIN;//enum type is required to access WINTER
System.out.println(s);
}}
```

The purpose of the values() method in the enum

The Java compiler internally adds the values() method when it creates an enum. The values() method returns an array containing all the values of the enum.

The purpose of the valueOf() method in the enum

The Java compiler internally adds the valueOf() method when it creates an enum. The valueOf() method returns the value of given constant enum.

The purpose of the ordinal() method in the enum

The Java compiler internally adds the ordinal() method when it creates an enum. The ordinal() method returns the index of the enum value.

Example of Enum : Inside the class

```
class EnumInClass{

public enum Season { SPRING, WIN, SUM, RAIN;}

public enum Week { SUN,MON,TUE}

public static void main(String[] args) {
   System.out.println(Season.WIN);
for (Week w : Week.values())
System.out.println(w);

}}
```

Example of Enum : Outside the class

```
enum Month {
    JAN(31), FEB(28),MAR(31), APR(30),
MAY(31),JUN(30),JUL(31),AUG(31),SEP(30),
OCT(31),NOV(30),DEC(31);
    int nod;
    Month(int nod)
    {    this.nod=nod;}

    int getNod()
        {   return nod;      }

            }//enum close
```

```
class EnumOutClass{
public static void main(String[] args) {
    Month m=Month.AUG;//return name
    System.out.println(m.name()+ " "+m.getNod());
    for (Month m1 : Month.values())
System.out.println(m1.ordinal()+" "+m1.name()+" "
+m1.getNod());
}
}
```

Java String

In Java, string is basically an object that represents sequence of char values. An array of characters works same as Java string. For example:

```
char[] ch={'j','a','v','a','t','p','o','i','n','t'};
String s=new String(ch);
```

is same as:

```
String s="javatpoint";
```

How to create a string object?

There are two ways to create String object:

1. By string literal
2. By new keyword

1) String Literal

Java String literal is created by using double quotes. For Example:

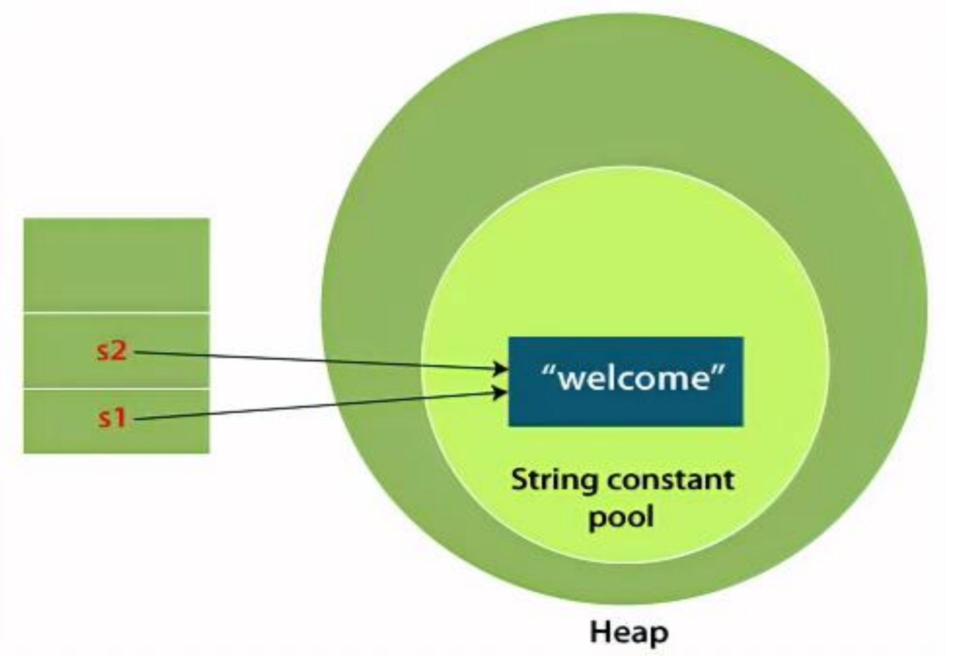
1.String s="welcome";

Each time you create a string literal, the JVM checks the "**string constant pool**" first. If the string already exists in the pool, a reference to the pooled instance is returned. If the string doesn't exist in the pool, a new string instance is created and placed in the pool.

For example:

1.String s1="Welcome";
2.String s2="Welcome";//It doesn't create a new instance

Note: String objects are stored in a special memory area known as the "string constant pool".



2) By new keyword

```
String s=new String("Welcome");//creates two objects and one reference variable
```

In such case, JVM will create a new string object in normal (non-pool) heap memory, and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in a heap (non-pool).

StringExample.java

```
public class StringExample{
public static void main(String args[]){
String s1="java";//creating string by Java string literal
char ch[]={'s','t','r','i','n','g','s'};
String s2=new String(ch);//converting char array to string
String s3=new String("example");//creating Java string by new keyword
System.out.println(s1);
System.out.println(s2);
System.out.println(s3);
}}
```

Java String class methods

The java.lang.String class provides many useful methods to perform operations on sequence of char values.

No.	Method	Description
1	char charAt(int index)	It returns char value for the particular index
2	int length()	It returns string length
3	static String format(String format, Object... args)	It returns a formatted string.
4	static String format(Locale l, String format, Object... args)	It returns formatted string with given locale.
5	String substring(int beginIndex)	It returns substring for given begin index.
6	String substring(int beginIndex, int endIndex)	It returns substring for given begin index and end index.
7	boolean contains(CharSequence s)	It returns true or false after matching the sequence of char value.
8	static String join(CharSequence delimiter, CharSequence... elements)	It returns a joined string.

No.	Method	Description
9	static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)	It returns a joined string.
10	boolean equals(Object another)	It checks the equality of string with the given object.
11	boolean isEmpty()	It checks if string is empty.
12	String concat(String str)	It concatenates the specified string.
13	String replace(char old, char new)	It replaces all occurrences of the specified char value.
14	String replace(CharSequence old, CharSequence new)	It replaces all occurrences of the specified CharSequence.
15	static String equalsIgnoreCase(String another)	It compares another string. It doesn't check case.
16	String[] split(String regex)	It returns a split string matching regex.
17	String[] split(String regex, int limit)	It returns a split string matching regex and limit.
18	String intern()	It returns an interned string.

No.	Method	Description
19	int indexOf(int ch)	It returns the specified char value index.
20	int indexOf(int ch, int fromIndex)	It returns the specified char value index starting with given index.
21	int indexOf(String substring)	It returns the specified substring index.
22	int indexOf(String substring, int fromIndex)	It returns the specified substring index starting with given index.
23	String toLowerCase()	It returns a string in lowercase.
24	String toLowerCase(Locale l)	It returns a string in lowercase using specified locale.
25	String toUpperCase()	It returns a string in uppercase.
26	String toUpperCase(Locale l)	It returns a string in uppercase using specified locale.
27	String trim()	It removes beginning and ending spaces of this string.
28	static String valueOf(int value)	It converts given type into string. It is an overloaded method.

The charAt() returns char value for the particular index

```
class Main {
  public static void main(String[] args) {
    String str1 = "Java Programming";
    // returns character at index 2
    System.out.println(str1.charAt(2));
  }
}
```

The **concat()** method concatenates (joins) two strings and returns it.

```
class Main {
  public static void main(String[] args) {
    String str1 = "Java";
    String str2 = "Programming";

    // concatenate str1 and str2
    System.out.println(str1.concat(str2));
  }
}
```

The indexOf(s) returns the specified substring index.

```
class Main {
  public static void main(String[]
args) {
    String str1 = "Java is fun";
    int result;
    // getting index of character 's'
    result = str1.indexOf('s');
    System.out.println(result);
  }
}
```

The replace('b', 'c’) replaces all occurrences of the specified char value with given value.

```
class Main {
  public static void main(String[] args) {
    String str1 = "bat ball";

    // replace b with c
    System.out.println(str1.replace('b', 'c'));

  }
}
```