Java Class and Objects :-

Java is a object - oriented Programming Language. The core concept of the object - oriented approach is to break complex problems into smaller objects.

An Object is any Entity that has a state and behavior. for Example, a bicycle is an object. It has

- States - idle, first gear, etc
- Behaviors - Braking, accelerating, etc.

Java Class :-

A class is a blueprint for the object. Before we create an object, we first need to define the class.

we can think of the class as a sketch (Prototype) of a house. It contain all the details about the floors, doors, windows, etc. Based on these descriptions we build the house. House is the object.

Create a Class in Java :-

we can create a class in Java using the class keyword.

```
Ex-   class ClassName {
          // fields
          // methods
      }
```

Here, fields (variables) and methods Represent the state and behaviors of the object Rebestively.

- fields are used to store data
- methods are used to perform some operations

for our bicycle object, we can create the class as

```
Class Bicycle
{
    // state or field
    Private int gear = 5;

    // behavior or method
    Public void braking ()
    {
System.out.println ("working of Braking");
    }
}
```

In the above example, we have created a class named bicycle. it contains a field named gear and a method named braking().

Here, Bicycle is a Prototype. Now, we can create any number of bicycles using the Prototype. and all the bicycles will share the fields and method of the Prototype.

Java Objects :-

An object is called an instance of a class.

Creating an object in Java :-

Here is How we can Create an object of a class.

```
ClassName object = new ClassName();

// for Bicycle class
Bicycle sportsBicycle = new Bicycle();
Bicycle touringBicycle = new Bicycle();
```

Example of java class and objects -

```
class Lamp
{
// stores the value for light
// true if light is on
// false if light is off
    Boolean isOn;

// method to turn on the light
    void turnOn()
    {
        isOn = true;
System.out.println("Light on? " + isOn);
    }
```

```
// Method to turnoff the light
    void turnoff ()
        {
            isOn = false;
System.out.println (" Light On? " + isOn);
        }
}

        Class main
            {
Public static void main ( String [] args )
            {

// Create objects led and halogen
Lamp led = new Lamp ();
Lamp halogen = new Lamp ();

// turn on the light by
// Calling method turnOn ()
        led.turnOn ();

// turn off the light by
// Calling method turnoff ()
        halogen.turnoff ();
        }
}
```

Output- Light On ? true
Light off ? false

Java methods :-

A method is a block of code that Performs a specific task.

Suppose you need to create a Program to create a circle and color it. you can create two methods to solve this Problem :

- A method to draw the circle
- A method to color the circle

Dividing a complex Problem into smaller chunks makes your Program Easy to Understand and Reusable.

In Java, there are two types of methods -

1. User - defined method :-

We can create our own method based on your Requirment.

2. Standard Library Methods :-

These are built - in methods in Java that are available to use.

Let's first learn about user - defined method -

Declaring a Java method :-

Syntax-

```
returnType methodName ( )
{
    // method body
}
```

This is the simple syntax of declaring a method. However, the complete syntax of declaring a method is -

```
modifier static returnType nameOfMethod (Parameter1,
        Parameter2, ....) {
    // method body
}
```

Calling a method in java :-

Here's is how we can call the addNumbers ( ) method.

```
// calls the method
addNumbers ( ) ;

int addNumbers ( ) {
    // codes
}
...
...
addNumbers ( ) ;
// code
```

Example of java method →

```
Class main
    {
    // Create a method
Public int addNumbers (int a, int b)
    {
        int sum = a + b;
    // Return value
        return sum;
    }
Public static void main (String [] args)
    {
        int num1 = 25;
        int num2 = 15;

// Create an object of main
    main obj = new main ();

// Calling method

int result = obj. addNumber (num1, num2);
system. out. println ("Sum is : " + result);
    }
}
```

output- sum is : 40 (in the above Example, we have created a method Named addNumber (). The method take two Parameters a and b. Notice line

int result = obj. addNumbers (num1, num2);

Java Constructors :-

A Constructor in Java is similar to a method that is invoked when a object of the class in created.

```
Class Test {
    Test () {
        // Constructor body
    }
}
```

Example of Java Constructor -

```
Class main {
Private String name ;
// Constructor
main () {
System.out.println (" Constructor called:");
name = " Programize";
}

Public static void main (String [] args) {

// Constructor is invoked while
// creating an object of the main class
main obj = new main ();
System.out.println (" The name is " + obj.name);
}
}
```

output- Constructor called :
The name is Programiz

Types of Constructor :-

In Java, Constructors can be devided into 3 types-

1. No - Arg Constructor
2. Parameterize Constructor
3. Default Constructor

1. Java NO - Arg Constructors :-

Similar to methods, a Java Constructor may or may not have any parameters (arguments).

```
Private Constructor ()
    {
        // body of the Constructor
    }
```

Example -

```
class main {
    int i;
// Constructor with no Parameter
    Private main () {
        i = 5;
System.out.println (" Constructor is called ");
    }

Public static void main (String [] args)
    {
// Calling the Constructor without any Parameter
    main obj = new main ();
```

```
        System.out.println ("Value of i :" + obj.i);
            }
        }
```

Output - Constructor is called
Value of i : 5

2. Java Parameterized Constructor :-

A java constructor can also accept one or more parameters. Such constructors are know as Parameterized Constructors.

Example -

```
class main {
    String languages;
// Constructor accepting single value
    main (String lang)
        {
            languages = lang;
        System.out.println (languages + " Programming Language")
            }

Public static void main ( String [] args)
            {
    // Call Constructor by passing a single value
        main obj1 = new main ("Java");
        main obj2 = new main ("Python");
        main obj3 = new main ("C");
            }
        }
```

Output - Java Programming Language
Python Programming Language
C Programming Language

3. Java default Constructor :-

if we do not create any constructor, the java Compiler Automatically create a no - arg constructor during the execution of the Program. This Constructor is called default Constructor.

Example-

```
Class main
{
    int a;
    boolean b;
Public static void main (String [.] args)
{
// A defaut Construction is called
main obj = new main ();

System.out.println (" default value :");
System.out.println (" a = " + obj.a);
System.out.println (" b = " + obj.b);
}
}
```

Output- a = 0
b = false

Some defaut value types - Boolen, float
byte, double
short, object
int
long
char

Java Strings :-

In Java, a string is a sequence of characters. for Example, "Hello" is a string containing a sequence of characters 'h', 'e', 'l', 'l', and 'o'.

We use double quotes to Repress a string in Java.

```
// Create a string
String type = "Java Programming";
```

Example -

```
Class main
{
Public static void main (String [] args)
{

// Create strings
String first = "Java";
String second = "Python";
String Third = "JavaScript";

// Print Strings
System.out.println (first);    // Print Java
System.out.println (second);   // Print Python
System.out.println (third);    // Print Javascript
```

Output -

```
Java
Python
JavaScript
```

Java this keyword :-

Ans Java this keyword is used to refer to the current object inside a method or a constructor.

Example-

```
Class main
{
    int instvar;
main (int instvar)
{
    this.instvar = instvar;
System.out.println ("this Reference =" + this);
}

Public static void main (String [] args)
{
    main obj = new main (8);
System.out.println ("object Reference =" + obj);
}
}
```

Output - this Reference = main@23fc625e
object Reference = main@23fc625e

Ans the above example, we created an object name obj of the class main. we then Print the Reference to the object obj and this keyword of the class.

Java final keyword :-

In java, the final keyword is used to donate constants. It can be used with variables, methods and classes.

One any entity (variable, method and class) is declared final, it can be assigned only once that is,

- the final variable cannot be Reinitialized with another value.
- the final method cannot be overridden
- the final class cannot be extended.

1. Java final variable -

In java, we can not change the value of a final variable.

2. Java final method -

In java, the final method cannot be overridden by the child class.

3. Java final class :-

In java, the final class cannot be inherited by another class.