# OBJECT Oriented Programming

## * class

- a class in Java is a blueprint for creating objects
- it encapsulates data as field and behaviour as methods that can operate on That data.

- component of a class:
- Fields: variables that store the properties or Attributes of the Object
- methods: Functions that define the behaviour of the object and can manipulate fields or perform operations.
- Constructor; special methods that are called when a new object instance is created. used mainly to initialze objects.

Example of a Class;

```
Public class Car {
    // fields
    Private string color;
    private string model;

    // Constructor
    public car (string color, string model){
        This. color = color;
        This . model = model;
    }

   // method
   Public Void displayInformation ( ) {
       System.out.println ("car model :" +model + color);
   }
}
```

in The previous Example "car" is a class with:

- Two fields: color and model
- one Constructor that initializes the color and model of a new Car instance
- one method: displayInformation(). which prints out car's color and model.

## instance variables:

Definition: Variables that store data or state of an individual object. each object has it's own copy

charecteristics: Declared within a class but outside methods; have object specific values.

Access Control: can be controlled with access modifiers (eg. private, public)

## instance methods:

Definition: Function within a class that defines behavior of an object. They operate on and can access instance variables.

charecteristics: Called on individual objects; can manipulate object's state.

Access control: Access can be limited via access modifiers.

Example.

```
Public Class Car {

    //instance variables
    private String color;
    private String model;


    // constructor
    public Car (String Color, String model) {

        this.color = color;
        This.model = model;
    }

    //instance method
    ~~Public void display Information~~

    Public Void displayInformation () {
        System.out.println (model + Color);
    }
}.
```

## obJECT :

- an obJect is a fundamental entity that represents an instance of a class.
- a class provides the blueprint or the template from which obJects are created.
- obJect encapsulate both State (attributes or properties) and behaviour (methods or functions) and are the core building blocks of obJect-oriented programming (OOP)

### * State (Attributes or properties)

- An obJect's state is represented by it's attributes or fields.
- These are variables defined within the class but each obJect maintain it's own seperate copy of these variables.
- for example, if you have a 'Car' class with fields like 'Color' and 'model' each 'car' obJect can have different values for these fields.

  (e.g.' one car might be red and another blue)

Code Example:

```
Public Class Car {
        String Color; // field to store color
        string model; // field to store model

}
```

## * object Creation:

- objects are created using the "new" keyword followed by a ~~cto~~ call to a constructor, which is a special method designed to initialize new objects.

Example

$$\underbrace{\overset{\text{className}}{\text{Car}}\ \ \overset{\text{Reference variable (ObjectName)}}{\text{myCar}}}_{\text{Declaration}} = \underbrace{\overset{\text{NEW keyword}}{\text{new}}}_{\text{instantiation}}\ \underbrace{\overset{\text{constructorcall}}{\text{Car}(\,)}}_{\text{initialization}};\ \ //\ \text{creating a new car object.}$$

## * Object LifeCycle:

- The lifecycle of an object in Java involves it's creation, use, and then garbage collection.

- creation: an object is created in Java using the "new" keyword which allocates memory for the object in heap memory

- usage: once created, the object can be used to access it's methods and properties.

- Garbage Collection: Java has a built in garbage collector, which helps in automatically freeing up memory by destroying objects that are no longer in use or that cannot be reached.

Benefits of using Objects:

- Modularity : Encapsulation makes parts of the System independent.
- information Hiding : internal state is hidden; only expose methods.
- code reusability : inherits properties and behaviours from other classes.
- plugging and Debugging: Easy to replace and debug at the Object Level.

using objects:

Access properties using • operator like

```
myCar.displayInfo ( );
```

Class Vs Object :

- A class is a blueprint or Template from which objects are created
- classes Contain:

* Fields (also known as attributes or properties) that hold the state of objects of the class.

* Methods (functions) that define the behaviour of the objects

* Constructors for initializing new instances of the class.

* other nested classes, interfaces, and various access modifiers which define how the members (fields, methods) can be accessed.

Purpose : it provides a structured definition that encapsulates data and functionalities which are common to all objects that are instantiated from it.

Static : classes are static. this means the definition of a class is fixed at compile time; it's structure (fields, methods) doesn't change at runtime. Through the content of it's field can vary.

- An object is an instance of a class; when a class is instantiated, objects are created with ~~that~~ the defined properties and behaviour of the class.

Objects hold:

- the values for the fields defined by their class
- the ability to execute methods defined by their class.

Purpose; each object serves as a specific example of it's class with unique value and states. objects operate independently and can interact with other objects

Dynamic : objects are dynamic. The creation, manipulation, and destruction of objects can happen at runtime and they occupy memory space when created.

Example:

```
Public class Car {
 // fields
Private String color;
Private String model;

// constructor
Public Car (String Color, String model) {
        this. color = color;
        this. model = model;
}
// method
Public void displayInfo() {
      System.out. println ("car model: " + model + "color: "+color);
  }
}
```

• Here car is a class That defines The blueprint for Car Objects including a model and color attribute, a Constructor to initialize these attributes, and a method to display the car's info

```
Public class Main {
     Public static void main (String[] args) {
        // creating an object of Car
        Car myCar = new Car ("Red", "Toyota");
       // access the object properties using . operator
        myCar.displayInfo();
     }
}
```

## Constructor :

- a Constructor in Java is a special method that is called when an object is instantiated
- it has the same name as the class and does not have an return type not even void.
- Constructors are used to initialize the state of an object immedictly upon creation.
- They can take parameters to set initial values for the object's attributes based on the input provided at the time of instantiation.

## Charecteristics of constructors:

- Name is same as the class name
- Does not have a return type
- Can be overloaded (a class can have multiple constructors with different parameters)
- if no constructor is explicitely defined, Java provides a default constructor that takes no arguments and does nothing.

```
Public class Bicycle {

        Private int gear;
        private int speed;

    // constructor
    public Bicycle (int startGear, int startspeed){
        gear = startGear
        speed = startSpeed.
} }
```

```
Public class Car {

        Private String color;  }
        Private string model;  } instance variables.

                                      constructor parameters
            Public Car (String color, String model) {
            {
            {    This. Color = color; // Here This.color refers to the
constructor {                             field and color represents
            {                             Parameters.
            {    This. model = model;
            {
            {  }

            { Public Void updatemodel (String model) {
            {
method      {     this. model = model; //Distinguishes the model
            {                         parameter from the model
            {                         field.
            {  }
         }
```