

CHAPTER 5-INTRODUCTION TO CLASS IN JAVA,ACCESS SPECIFIERS, THIS AND STATIC IN JAVA

5.1 Objective

- Introduction to Classes
- Declaration of Classes
- Declaration of instance variables
- Declaration of methods
- Object and Object creation
- Access Modifiers
- Constructors
- Use of this Keyword
- Examples for “this” keyword
- static keyword

5.2 Course Content

5.2.1 Introduction to Classes

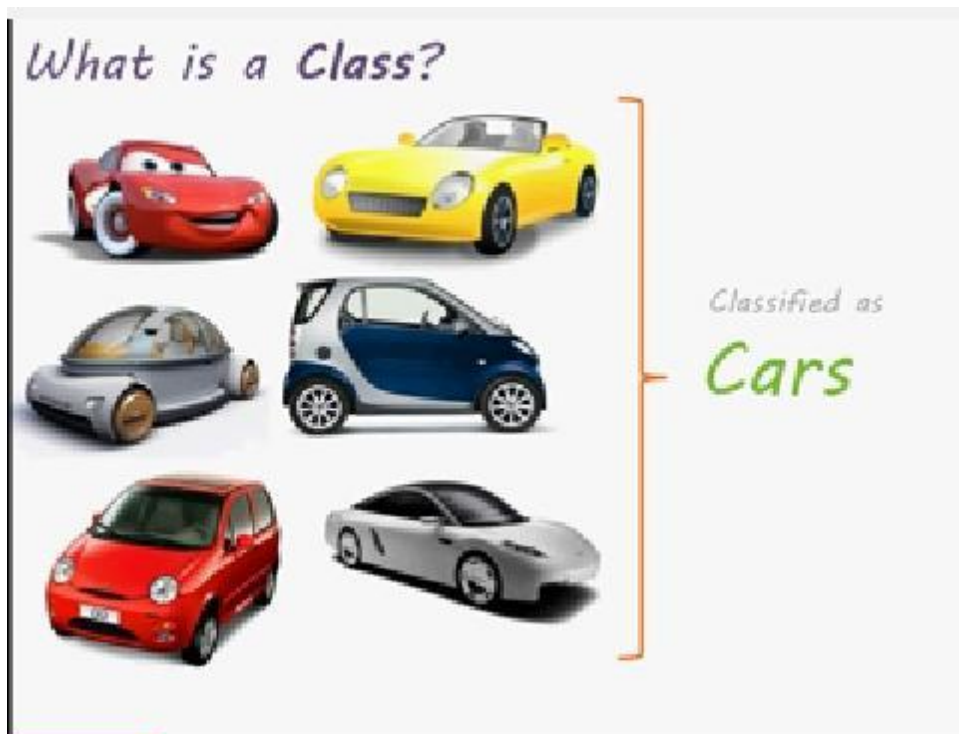
A class is a template, blueprint, or contract that defines what an object's data fields and methods will be. An object is an instance of a class. You can create many instances of a class. A Java class uses variables to define data fields and methods to define actions. The attributes and behaviours defined by a class are common to all objects of the same kind.

5.2.2 Declaration of Classes

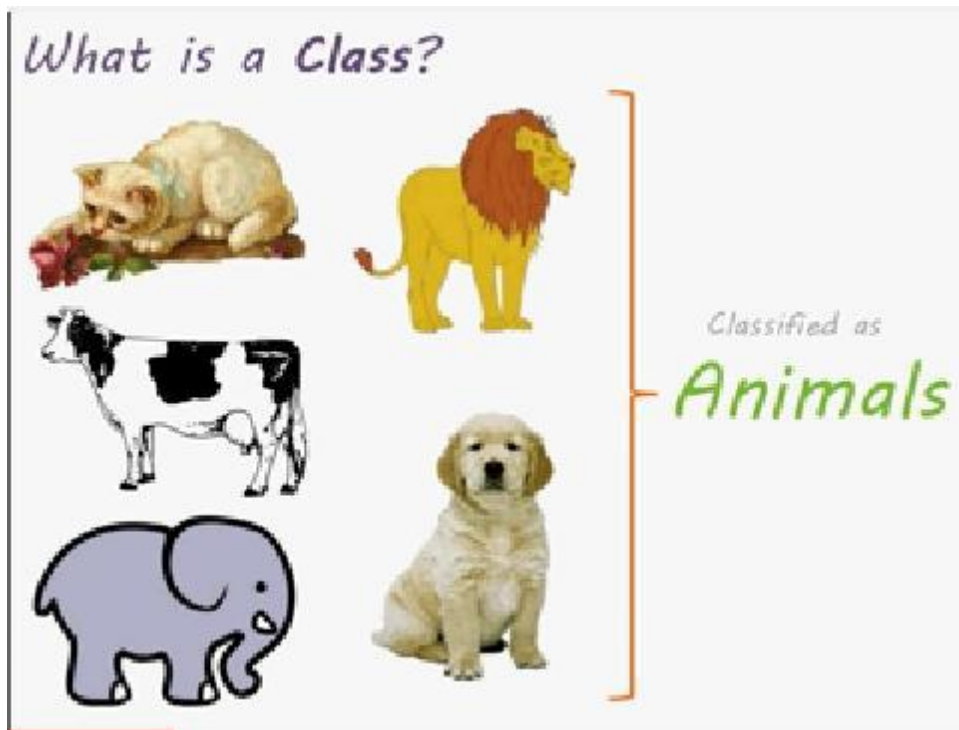
A class is declared by the use of the “**class**” keyword. Class body is enclosed between curly braces { and }. The data or variables defined within the class are called instance variables. The code is contained within methods. Collectively, the methods and variables defined in a class are called members of the class.

Finally we can say a class is a blue print of particular classification of objects.

Please refer to the below image for more details.



Here we can say there is a class **Car**.



Here we can say there is a class called **Animal**..

- ✓ A **Class** is a blue print of a particular classification of **Objects**
- ✓ An **Object** belongs to a **Class** of Objects
- ✓ An **Object** is an instance of a **Class**
- ✓ For example:
 - **myCar**, **petersCar** are instances of a class called **Car**
 - **myCat**, **petersDog** are instances of a class called **Animal**

Anatomy of Java Class

class accessibility level

keyword 'class'

class name

```
public class className {
```

```
// variables
```

These represent class attributes

```
// methods
```

These represent class behavior

```
}
```

All the code contained between { } is the class definition

Class Name Requirements

- You must follow these requirements:
 - A class name must begin with an alphabet.
 - A class name can contain only letters, digits, underscore, or dollar sign. Dollar sign is discouraged.
 - A class name can not be a Java reserved keyword such as public or void.

A class can contain any of the following variable types.

Local variables: Variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.

Instance variables: Instance variables are variables within a class but outside any method. These variables are instantiated when the class is loaded. Instance variables can be accessed from inside any method, constructor or blocks of that particular class.

Class variables: Class variables are variables declared within a class, outside any method, with the static keyword

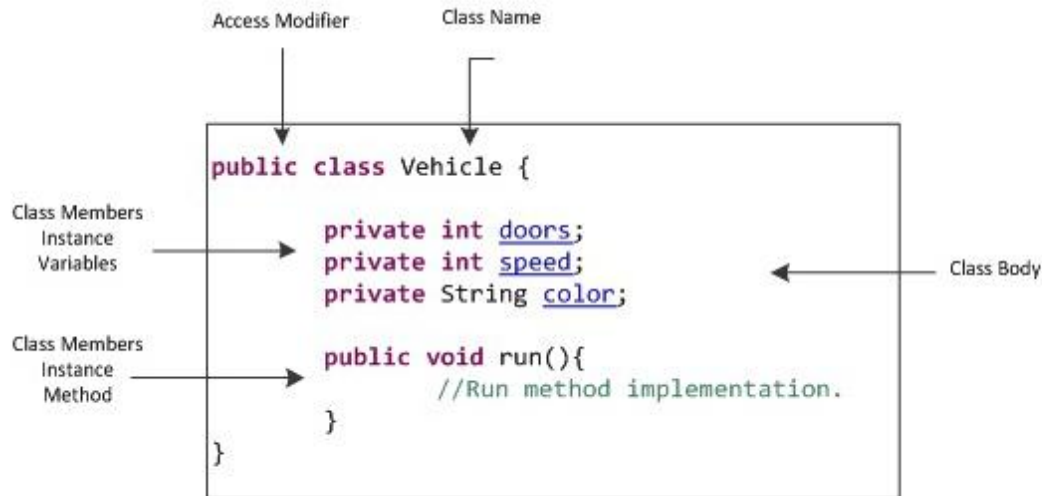
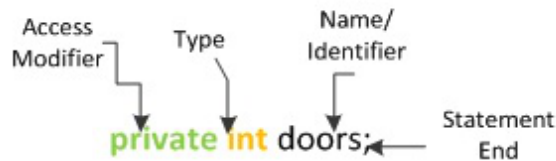


Figure 1

5.2.3 Declaration of Instance Variables

Variables defined within a class are called **instance variables** because each instance of the class (ie, each object of the class) contains its own copy of these variables. Thus, the data for one object is separate and unique from the data of another. Instance variables can be declared public or private or default (no modifier). When we do not want our variables value to be changed outside our class we should declare them private. Public variables can be accessed and changed from outside the class. We will have more information about this on OOP tutorial.

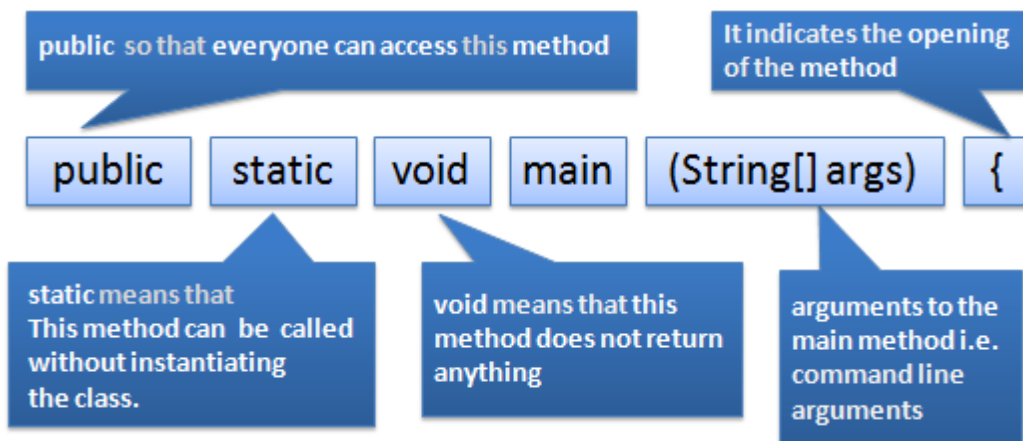
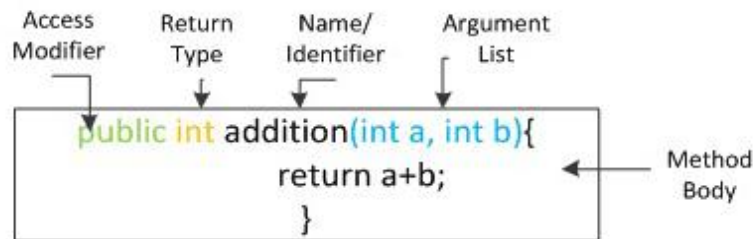
Below find the syntax for declaring an instance variable.



5.2.4 Declaration of Methods

A method is a program module that contains a series of statements that carry out a task. To execute a method, you invoke or call it from another method; the calling method makes a method call, which invokes the called method. Any class can contain an unlimited number of methods, and each method can be called an unlimited number of times.

The syntax for method declaration is as given below. Java main method is given as an example.



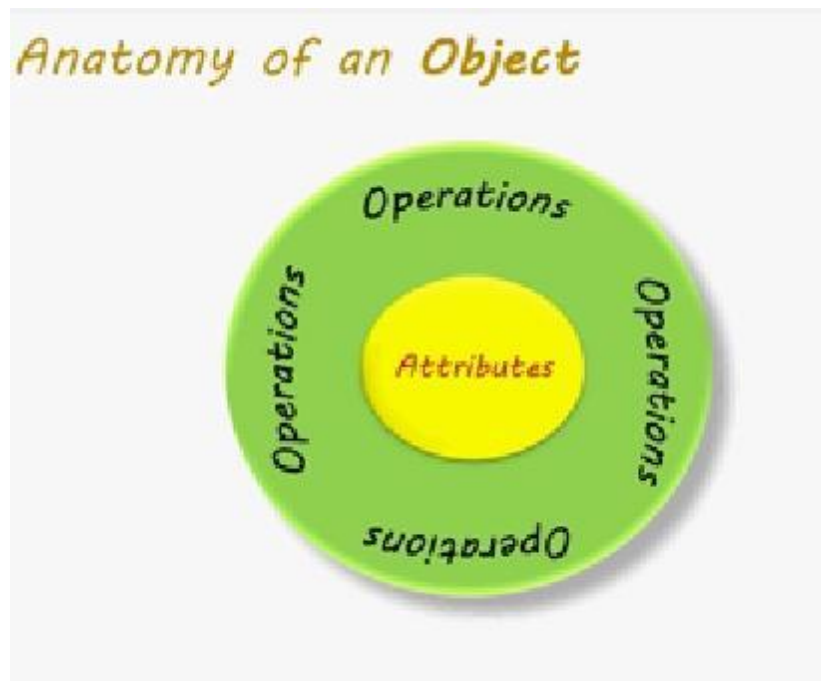
5.2.5 Objects or Instances



Consider Tiger from the above list,

Objects		
Object	Properties	Behavior
	weight height speed	eat attack run

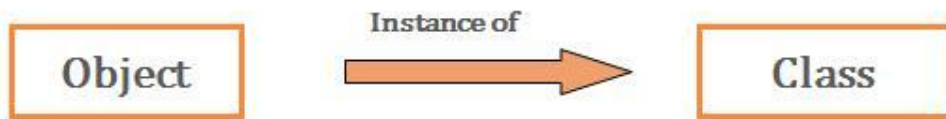
Objects are made up of attributes(properties) and methods(behaviours). Attributes are the characteristics that define an object; the values contained in attributes differentiate the objects of the same class from one another.



To understand this in a better way, let's take an example of **Mobile** as an object. Mobile has characteristics like model, manufacturer, cost, operating system etc. So if we create "Samsung" mobile object and "iPhone" mobile object, we can distinguish them from their characteristics. The values of the attributes of an object are also referred to as the **object's state**.

Objects need not be physical objects. It can be any entity. It can be account, an organization etc.

We can say --- *Object is an instance of a class.*



Class vs Object



Consider an object Car. What we know about a car defines its attributes. What a car can do defines its behaviours/methods.

An example

What do we know about a car?

- ✓ make
- ✓ model
- ✓ year
- ✓ color
- ✓ trim
- ✓ ...

Attributes



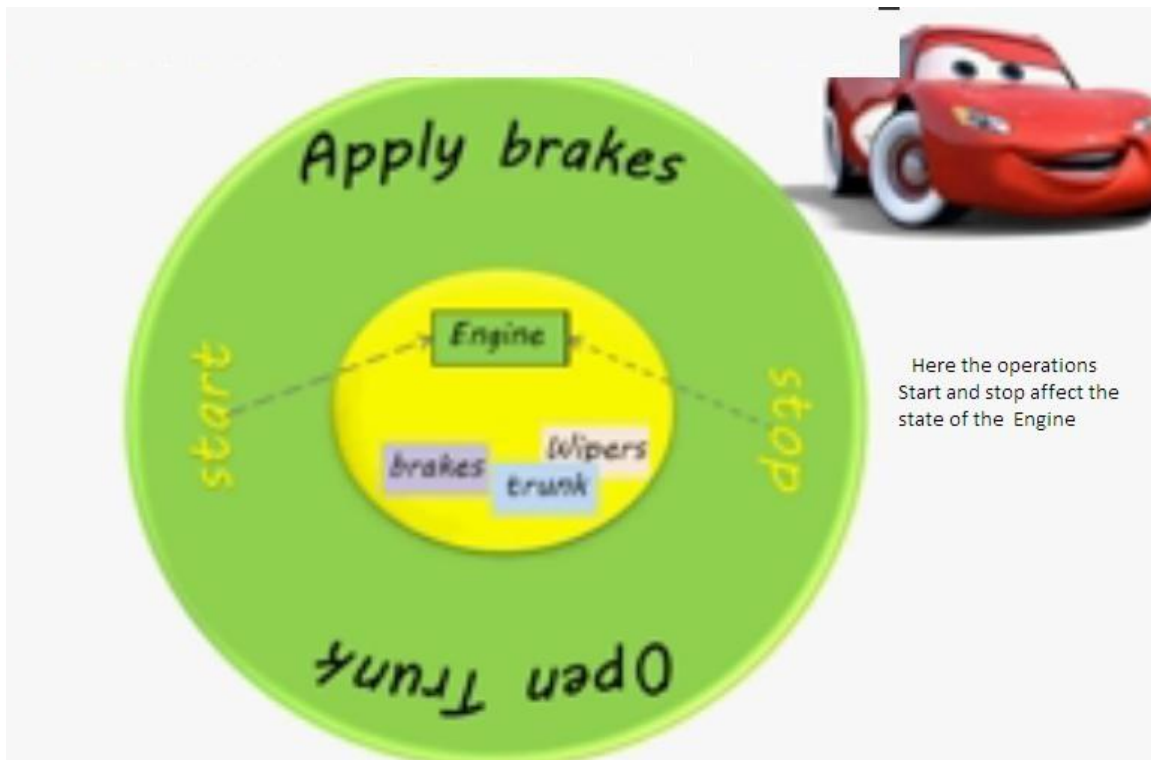
What can a car do?

- ✓ start
- ✓ stop
- ✓ apply brakes
- ✓ open the Trunk
- ✓ turn on wipers
- ✓ ...

Methods or Operations

Always operations can alter the state of the attributes.

Consider the above start operation for a Car. When the car is started ,it is affecting the state of the engine.



Creating an Object:



As mentioned previously, a class provides the blueprints for objects. So basically an object is created from a class. In Java, the new key word is used to create new objects.

There are three steps when creating an object from a class:

Declaration: A variable declaration with a variable name with an object type.

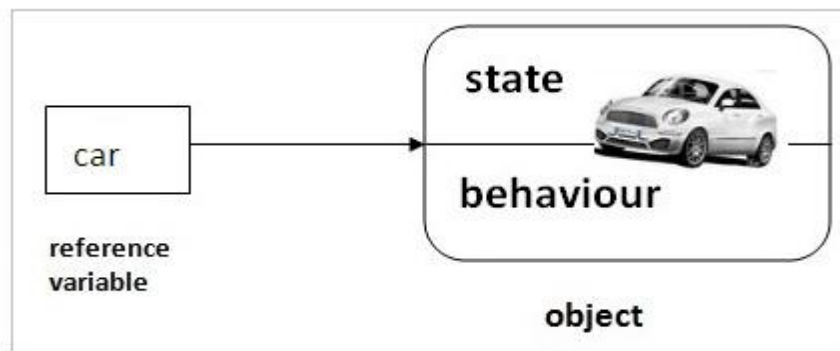
Instantiation: The 'new' key word is used to create the object.

Initialization: The 'new' keyword is followed by a call to a constructor. This call initializes the new object.

Example of creating an object is given below:

```
ClassName ObjectReferenceVariableName = new ConstructorName();
```

```
Vehicle car= new Vehicle();
```



Refer the below Vehicle class.

We are creating three objects truck, bike and a car from Vehicle class.

```
//creating a truck object from Vehicle class
Vehicle truck= new Vehicle("Tata","T200-Big", 4,500000);
```

```
//creating bike object from Vehicle class
Vehicle bike=new Vehicle("Suzuki" "RX500" 2, 300000);
```

```
//Create car object
```

```
.....
.....
```

Objects



Class

```
public class Vehicle {
    /* instance variable declaration */
    private String manufactureName;
    private String model;
    private int seatingCapacity;
    private double price;
    /*Constructor to set properties/characteristics
    of objects*/
    public Vehicle(String manufactureName,String model,
        int seatingCapacity, double price) {
        this.manufactureName = manufactureName;
        this.model = model;
        this.seatingCapacity = seatingCapacity;
        this.price = price;
    }
    /*method to get access to the manufactureName
    property of Object*/
    public String getManufactureName() {
        return manufactureName;
    }
    /* run behavior */
    public void run() {
        //run method implementation
    }
}
```

Accessing Instance Variables and Methods using Objects:

Dot Operator .

- Object variables & methods can be accessed using the dot operator

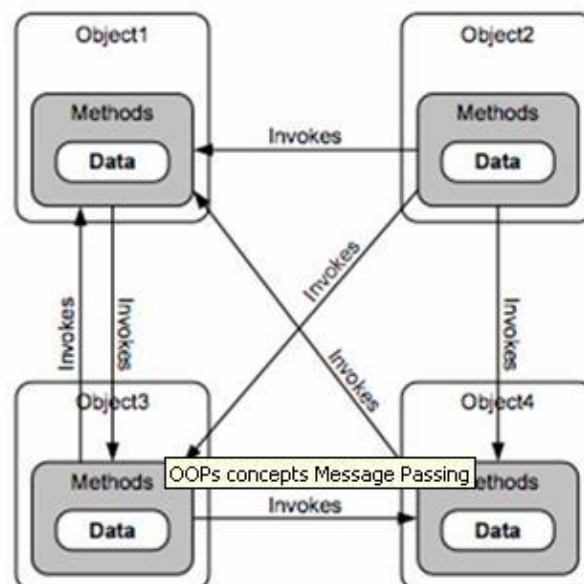
```
Car myCar= new Car();  
myCar.color="Red";  
myCar.start();
```

/* call a variable as follows */
ObjectReference.variableName;

/* Now you can call a class method as follows */
ObjectReference.MethodName();

How objects communicates each other?

One object invoking methods on another object is known as **Message passing**.
It is also referred to as **Method Invocation**.



5.2.6 Constructors

Java constructors are special methods which are used to initialize objects. Constructor method has the same name as that of class, they are called or invoked when an object of class is created and can't be called explicitly. They are designed to perform initializing actions such as initializing the data fields or objects.

A java constructor has the same name as the name of the class to which it belongs. Constructor's syntax does not include a return type, since constructors never return a value.

Constructors can be classified into two types, default constructors and parametrized constructors. If you don't define a constructor, then the compiler creates a default constructor. Default constructors do not contain any parameters. Default constructors are created only if there are no constructors defined by us. Java provides a default constructor which takes no arguments, when no explicit constructors are provided. Parametrized constructors are required to pass parameters on creation of objects.

```
class Vehicle {
    /* instance variable declaration */

    private String manufactureName;

    private String model;

    /*default constructor */
    public Vehicle(){
        System.out.println("Default Constructor called");
    }
    /* Constructor to set properties/characteristics of objects */
    public Vehicle(String manufactureName, String model){
        System.out.println("Constructor with arguments invoked");
        this.manufactureName=manufactureName;
        this.model=model;
    }

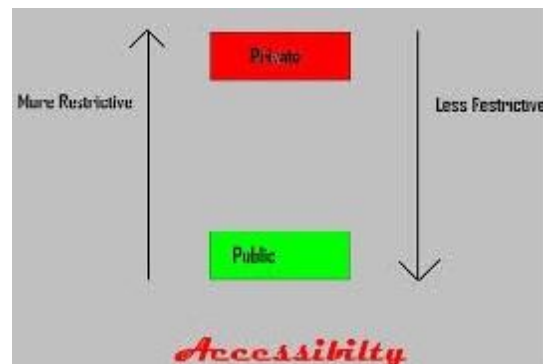
    /* main method.Execution starts from main method */
    public static void main(String[] args) {
        // creating vehicle object.Calling constructor to initialize
        // the instance variables.
        Vehicle car= new Vehicle();
        Vehicle bus= new Vehicle("Tata", "Air bus-T200");
    }
}
```



```
Console X
Default Constructor called
Constructor with arguments invoked
```

5.2.7 Access Specifiers

Each object has members (members can be variables or methods) which can be declared to have specific access.



Access Modifier - private:

Methods, Variables and Constructors that are declared private can only be accessed within the declared class itself. Private access modifier is the most restrictive access level.

Variables that are declared private can be accessed outside the class if public getter methods are present in the class. Using the private modifier is the main way that an object encapsulates itself and hide data from the outside world.

Access Modifier - public:

Members (variables, methods and constructors) declared public (least restrictive) within a public class are visible to any class in the java program, irrespective of whether these classes are in the package or in another package.

```

public class Flight {

    private String flightNumber;
    private String origin;
    private String destination;
    private int numberOfPassengers;

    public String getFlightNumber() {
        return flightNumber;
    }

    // other behaviors
}/* Flight */

```



I am not able to access the property numberOfPassengers in Flight class. It is a good practice to make your data as private. But that class should at least provide a public method which will give the data for me.?



Below program will give compilation Error.

```

public class Satellite {

    public static void main(String[] args) {
        /* Satellite class trying to access the numberOfPassengers in flight */
        Flight flight1= new Flight();
        int numberOfPassengers=flight1.numberOfPassengers;/* this property not accessible
        System.out.println("Number of passengers in flight1 is:"+numberOfPassengers);
    }
}

```

```

public class Flight {

    private String flightNumber;
    private String origin;
    private String destination;
    private int numberOfPassengers;

    public String getFlightNumber() {
        return flightNumber;
    }

    public int getNumberOfPassengers() {
        return numberOfPassengers;
    }

    // other behaviors
}/* Flight */

```

```

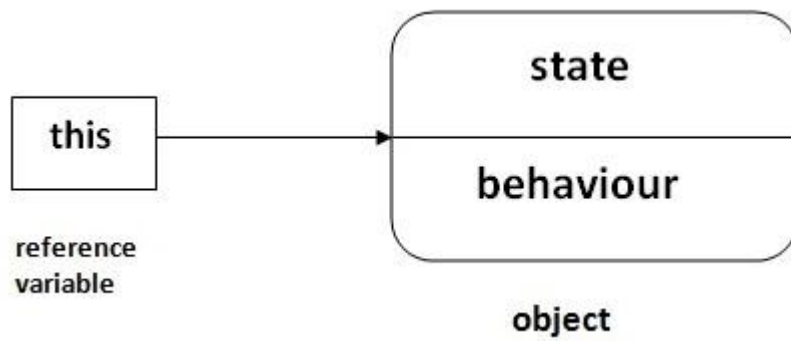
public class Satellite {

    public static void main(String[] args) {
        /* Satellite class trying to access the numberOfPassengers in flight */
        Flight flight1= new Flight();
        /*Now using getNumberOfPassengers() method Satellite can access
        numberOfPassengers in flight.Always make your data as
        private and provide getter and setter methods to access the data */
        int numberOfPassengers=flight1.getNumberOfPassengers();
        System.out.println("Number of passengers in flight1 is:"+numberOfPassengers);
    }
}

```

5.2.8 Use of this Keyword

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



5.2.9 Example on this keyword

If there is ambiguity between the instance variable and parameter, this keyword resolves the problem of ambiguity.

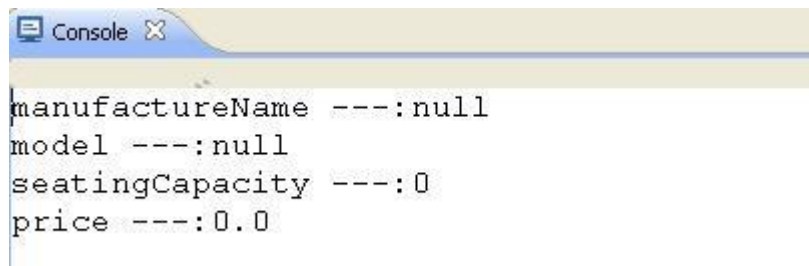
Understanding the problem without this keyword

Let's understand the problem if we don't use this keyword by the example given below:

```

class Vehicle {
    /* instance variable declaration */
    private String manufactureName;
    private String model;
    private int    seatingCapacity;
    private double price;
    //Constructor to set properties/characteristics of objects
    public Vehicle(String manufactureName,String model,
        int seatingCapacity, double price) {
        manufactureName = manufactureName;
        model = model;
        seatingCapacity = seatingCapacity;
        price = price;
    }
    /* main method.Execution starts from main method */
    public static void main(String[] args) {
        //creating a vehicle object.Calling constructor to initialize
        //the instance variables.
        Vehicle car= new Vehicle("Tata", "Indica Vista", 5, 450000);
        System.out.println("manufactureName ---:"+car.manufactureName);
        System.out.println("model ---:"+car.model);
        System.out.println("seatingCapacity ---:"+car.seatingCapacity);
        System.out.println("price ---:"+car.price);
    }
}

```



```

Console
manufactureName ---:null
model ---:null
seatingCapacity ---:0
price ---:0.0

```

In the above example, parameter (formal arguments) and instance variables are same that is why we are using this keyword to distinguish between local variable and instance variable.

Solution of the above problem by this keyword

```

class Vehicle {
    /* instance variable declaration */
    private String manufactureName;
    private String model;
    private int    seatingCapacity;
    private double price;
    /*Constructor to set properties/characteristics of objects */
    public Vehicle(String manufactureName,String model,
        int seatingCapacity, double price) {
        this.manufactureName = manufactureName;
        this.model = model;
        this.seatingCapacity = seatingCapacity;
        this.price = price;
    }
    /* main method.Execution starts from main method */
    public static void main(String[] args) {
        //creating a vehicle object.Calling constructor to initialize
        //the instance variables.
        Vehicle car= new Vehicle("Tata", "Indica Vista", 5, 450000);
        System.out.println("manufactureName ---:"+car.manufactureName);
        System.out.println("model ---:"+car.model);
        System.out.println("seatingCapacity ---:"+car.seatingCapacity);
        System.out.println("price ---:"+car.price);
    }
}

```

this keyword used

Console

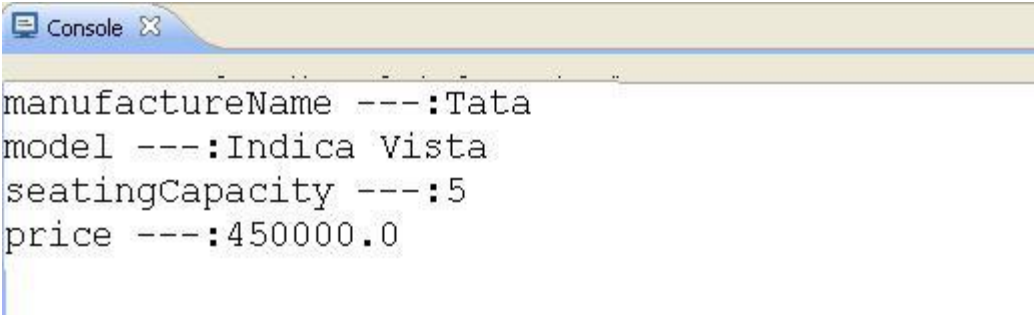
```

manufactureName ---:Tata
model ---:Indica Vista
seatingCapacity ---:5
price ---:450000.0

```


Solution of the above problem by with out using this keyword

```
class Vehicle {
    /* instance variable declaration */
    private String manufactureName;
    private String model;
    private int    seatingCapacity;
    private double price;
    /*Constructor to set properties/characteristics of objects */
    public Vehicle(String manufactureNameValue,String modelValue,
                   int seatingCapacityValue, double priceValue) {
        manufactureName = manufactureNameValue;
        model = modelValue;
        seatingCapacity = seatingCapacityValue;
        price = priceValue;
    }
    /* main method.Execution starts from main method */
    public static void main(String[] args) {
        //creating a vehicle object.Calling constructor to initialize
        //the instance variables.
        Vehicle car= new Vehicle("Tata", "Indica Vista", 5, 450000);
        System.out.println("manufactureName ---:"+car.manufactureName);
        System.out.println("model ---:"+car.model);
        System.out.println("seatingCapacity ---:"+car.seatingCapacity);
        System.out.println("price ---:"+car.price);
    }
}
```



The screenshot shows a console window with the following output:

```
manufactureName ---:Tata
model ---:Indica Vista
seatingCapacity ---:5
price ---:450000.0
```


5.2.10 Static Keyword

Static variables are also known as Class variables. 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.

Static variables are created when the program starts and destroyed when the program stops.

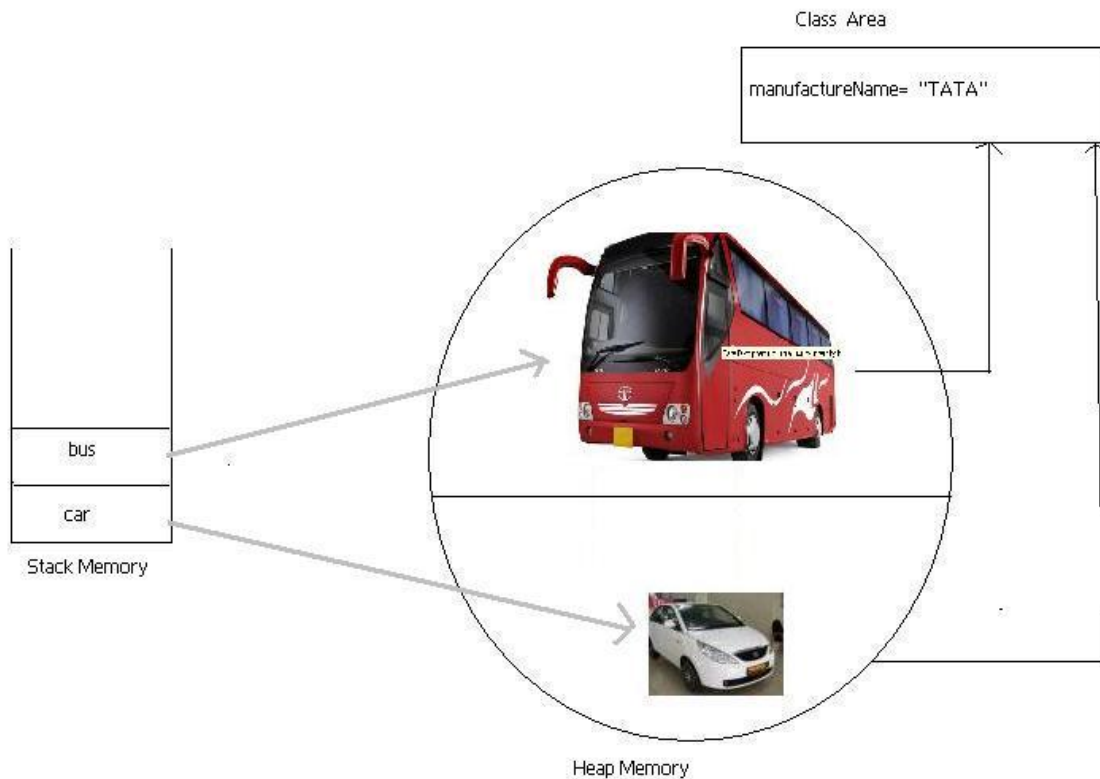
Visibility is similar to instance variables. However, most static variables are declared public since they must be available for users of the class.

Default values are same as instance variables. For numbers, the default value is 0; for Booleans, it is false; and for object references, it is null. Values can be assigned during the declaration or within the constructor.

Static variables can be accessed by calling with the class name . ClassName.VariableName.

Consider the below class Vehicle, here manufactureName is declared as static and getManufactureName() method also declared as static.

```
class Vehicle {
    private static String manufactureName="TATA"; /* class variable */
    private String model;
    private String color;
    private int seatingCapacity;
    private double price;
    public Vehicle(String model,String color,
        int seatingCapacity, double price) {
        this.model = model;
        this.color=color;
        this.seatingCapacity = seatingCapacity;
        this.price = price;
    }
    /*method getManufactureName() is declared as static.Class Behavior*/
    public static String getManufactureName(){
        return manufactureName;
    }
    public static void main(String[] args) {
        Vehicle car= new Vehicle("Indica Vista","White", 5, 450000);
        System.out.println("manufactureName ---:"+Vehicle.getManufactureName());
        System.out.println("model ---:"+car.model);
        Vehicle bus= new Vehicle("Tata Divo","Red", 30, 600000);
        System.out.println("manufactureName ---:"+Vehicle.getManufactureName());
        System.out.println("model ---:"+bus.model);
    }
}
```



Output

```
Console X
manufactureName ---:TATA
model ---:Indica Vista
manufactureName ---:TATA
model ---:Tata Divo
```

5.3 Test Your Knowledge

1. Keyword used to allocate dynamic memory

- a. New
- b. Create
- c. Allocate
- d. Put

Ans: a

2. Member of a class having same name as class that helps in the construction of the object

- a. Start method
- b. Constructor

TCS Internal

- c. Function
- d. Construct

Ans: b

3. Variables that are declared outside of method are called

- a. Inner variables
- b. Local variables
- c. Global variables
- d. Outer variables

Ans: c

4. What kind of data type is String

- a. Primitive type
- b. Local type
- c. Reference type
- d. None of the above

Ans: c

5. What kind of data type is char?

- a. Primitive type
- b. Local type
- c. Reference type
- d. None of the above

Ans: a

6. Modifier to prevent any method or attribute to be made invisible from outside object.

- a. Private
- b. Public
- c. Protected
- d. Default

Ans: a

7. Modifier to prevent any method or attribute to be made visible for any outside object.

- a. Private
- b. Public
- c. Protected
- d. Default

Ans: b

