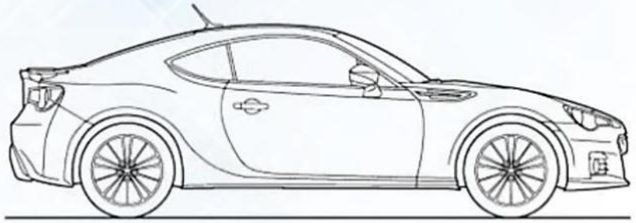
Objects and Classes in Java

Classes

- A class in java is a blueprint which includes all the data.
- It describes the state and behavior of a specific object.

Example :



Syntax :

```
Public class Car {

Color
Model
Price
speedUp();
gearChange();
}
```

Class name

Variables

Methods

Class

What is a class in Java

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

A class in Java can contain:

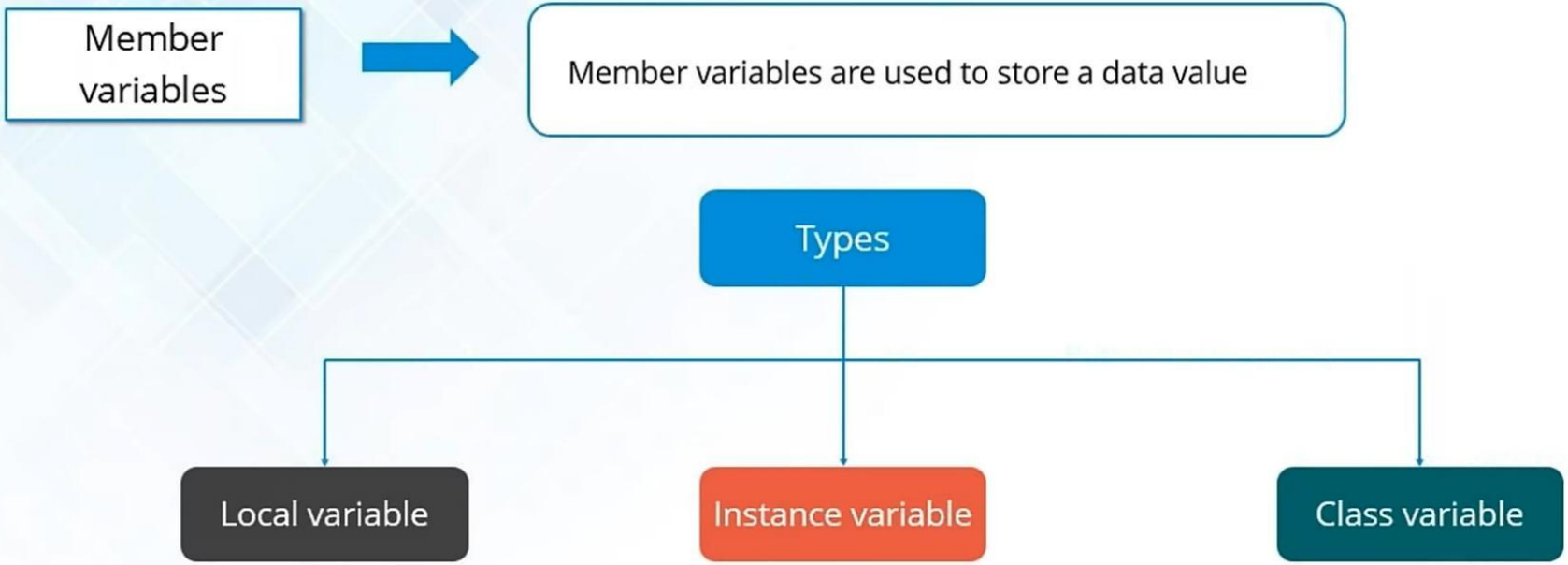
- **Fields**
- **Methods**
- **Constructors**
- **Blocks**
- **Nested class and interface**

Syntax to declare a class:

```
class <class_name>{
    fields;
    methods;
}
class Book
{
   int b_id;
   String name;
Static int x=12;
   void getdata()
    {
   int a;
     }
  void display()
    {
     }
```

Fields/instance variable

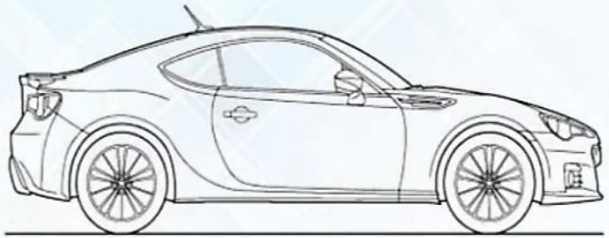
Methods/ behavior


Member variables
Member variables are used to store a data value
Types
Local variable
Instance variable
Class variable

Local variable	Instance variable	Class variable
Local variables are declared within the method of a class	Instance variable are declared in a class but outside a method, constructor or any block	Class / static variable has only one copy that is shared by all the different objects of a class

Objects

- It is an entity that has state and behavior
- Instance of a class which can access data



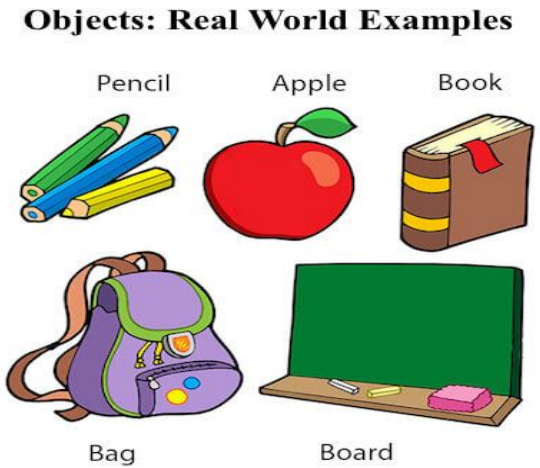
Class - Car
(Blueprint)

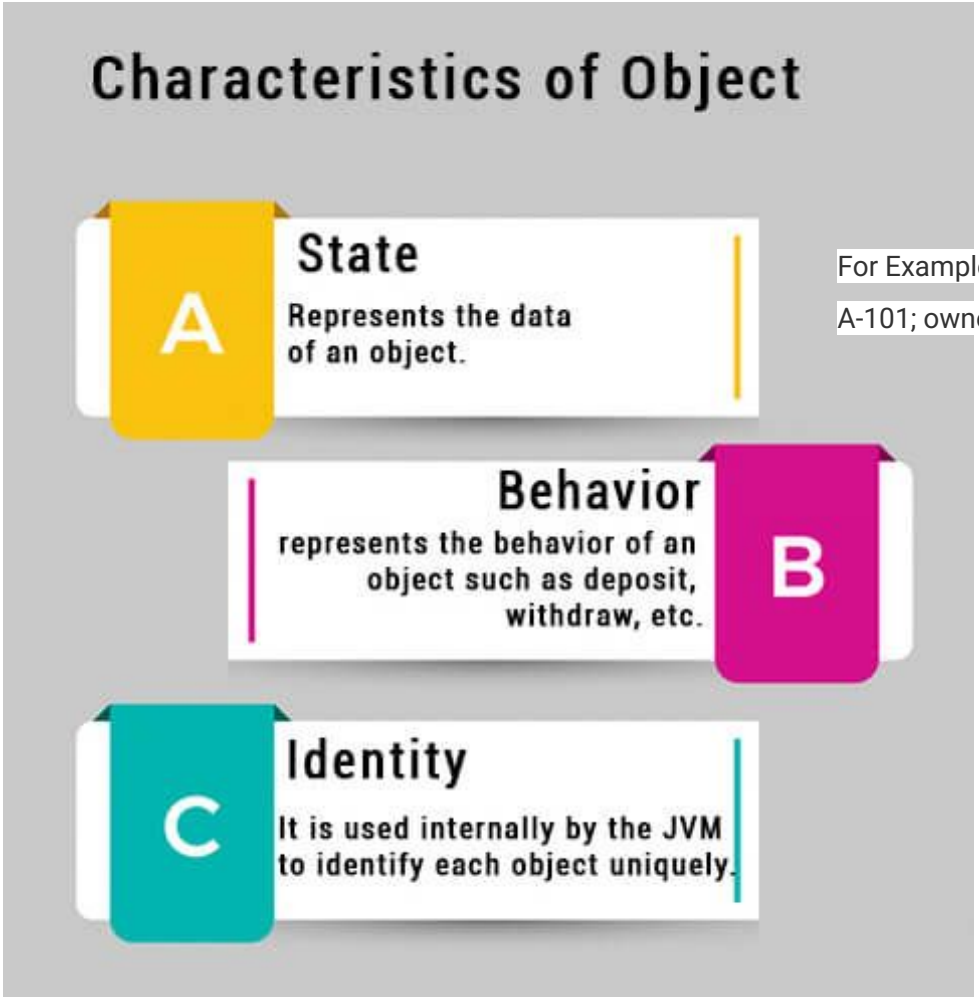
- **States of a Car :** Color - Orange
  Model - 1
  Price - 50000
- **Behavior of Car :** Speed up
  Change gear

What is an object in Java

An object in Java is the **physical as well as a logical entity**, whereas, a class in Java is a **logical entity** only.

An entity that has state and behavior is known as an object e.g., chair, bike, marker, pen, table, car, etc. It can be physical or logical (tangible and intangible). The example of an intangible object is the banking system.





For Example, account is an object. Its id is A-101; owner is “abc”, known as its state.

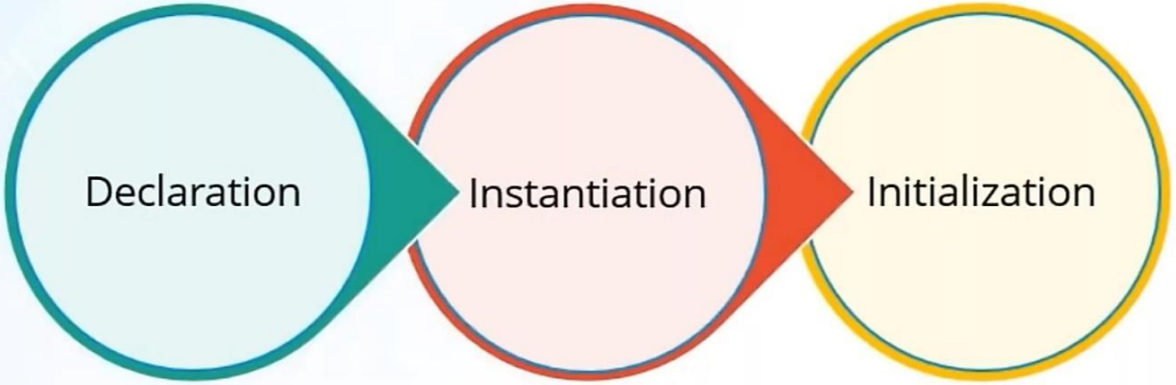
Object

An object is an instance of a class. A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.

Object Definitions:

- An object is *a real-world entity*.
- An object is *a runtime entity*.
- The object is *an entity which has state and behavior*.
- The object is *an instance of a class*.

How can you create an object?



Object Instantiation

- Objects to a class is created by using the keyword “**new”.**
- **new** allocates memory for the object.
- syntax:
  <class_name> <object name>=new <constructor_name>();
  OR
  <class_name> <object name>
  <object_name>=new <constructor_name>();

e.g. Student s1= new Student();
OR
Student s1;
s1=new Student();

new keyword in Java

The new keyword is used to allocate memory at runtime. All objects get memory in Heap memory area.

Instance variable in Java

A variable which is created inside the class but outside the method is known as an instance variable. **Instance variable doesn't get memory at compile time. It gets memory at runtime when an object or instance is created.** That is why it is known as an instance variable.

Method in Java

In Java, a method is like a function which is used to expose the behavior of an object.

Advantage of Method

- Code Reusability
- Code Optimization

Object and Class Example:

```
//Creating Student class.
class Student{
 int id;
 String name;
}
//Creating another class TestStudent1 which contains the main method
class TestStudent1{
  public static void main(String args[]){
  Student s1=new Student();
  System.out.println(s1.id);
  System.out.println(s1.name);
 }
}
```

Object and Class Example:

```
//Creating Student class.
class Student{
 int id;
 String name;
}
//Creating another class TestStudent1 which contains the main method
class TestStudent1{
  public static void main(String args[]){
  Student s1=new Student();
  System.out.println(s1.id);
  System.out.println(s1.name);
 }
}
```

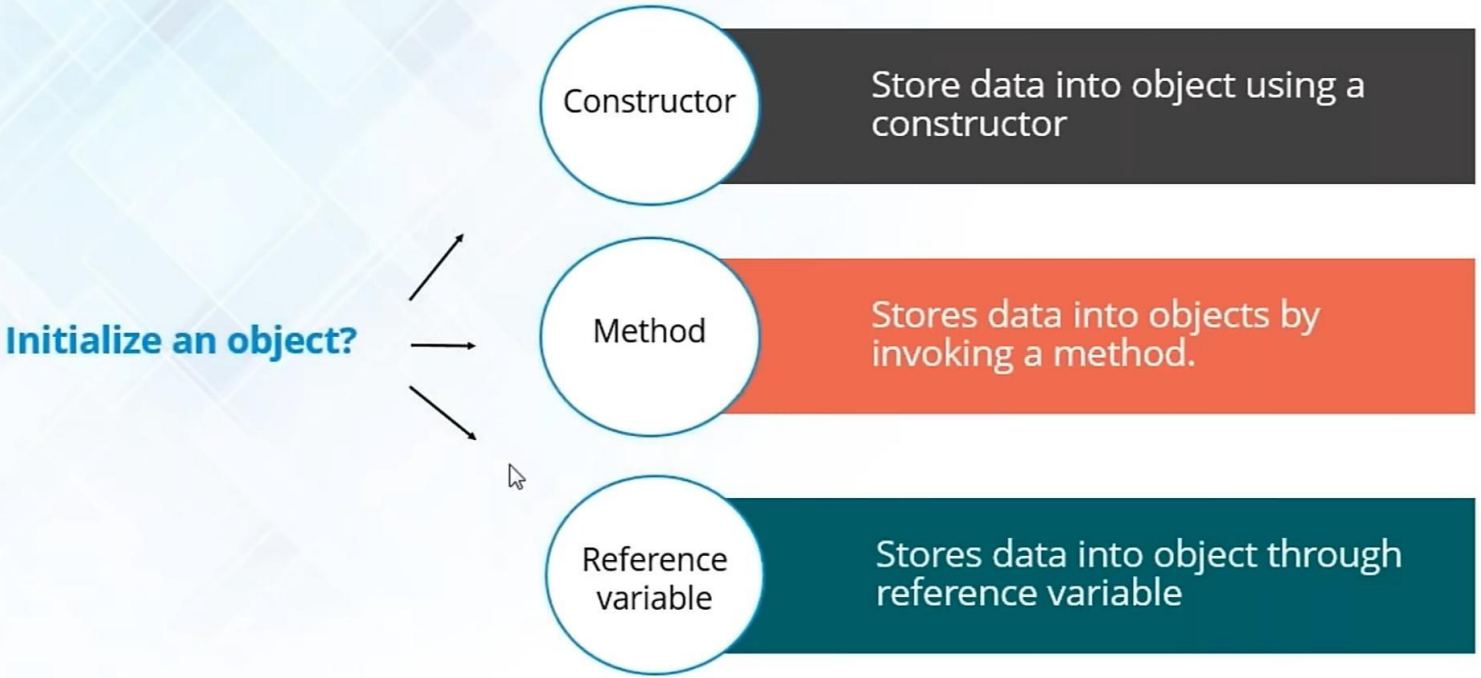
```
class Student
{
  int r_no;
  String name;

}

public class Test1
{
  public static void main(String args[])
  {
    Student s= new Student();

    System.out.println(s.r_no);
    System.out.println(s.name);
   }
}
```

3 Ways to initialize an object



By reference variable

```
class Student
{
 int r_no;
 String name;
 }

 public class StudInitRef
 {
   public static void main(String args[])
   {
      Student s1=new Student();
      s1.r_no= 12;
      s1.name="abc";
      System.out.println("roll no:="+s1.r_no);
      System.out.println("name:="+s1.name);
   }
 }
```

By method

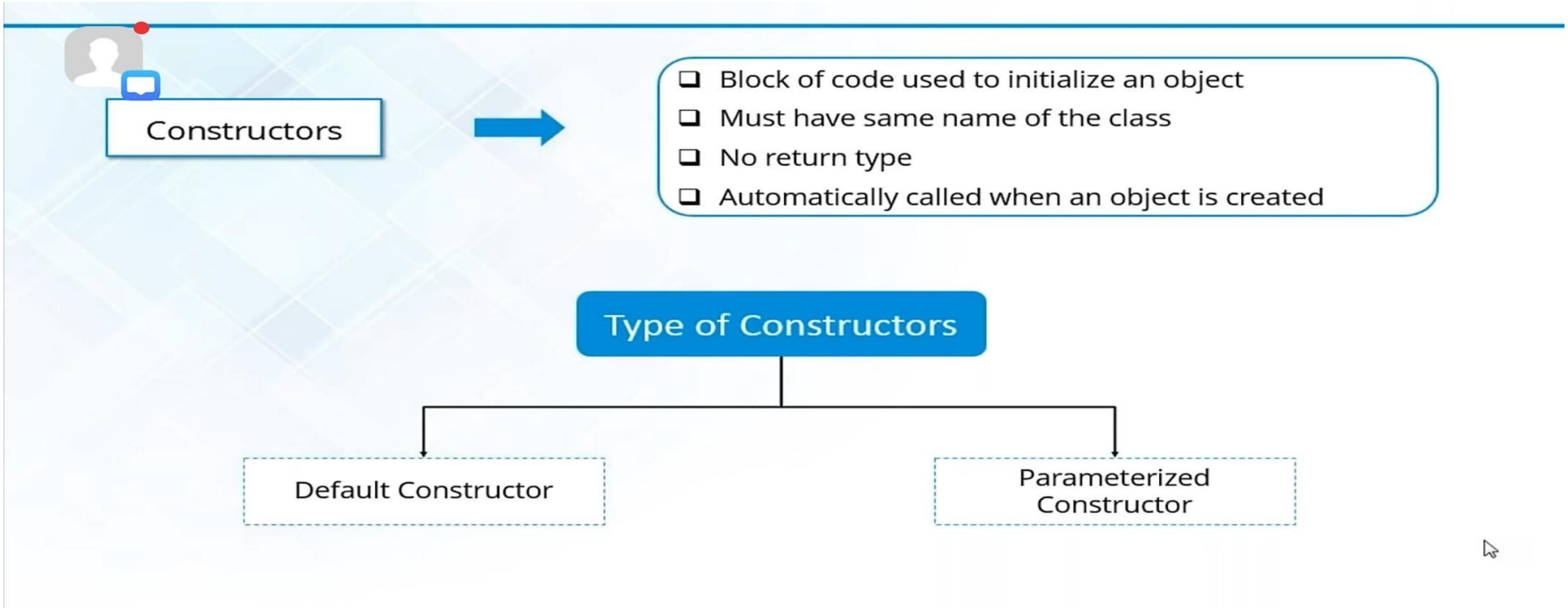
```
class Student
{
        int id;
        String name;

        void insert_stud(int s_id,String s_name)
        {
          id=s_id;
          name=s_name;
        }
        void display_data()
        {
           System.out.println(id+" "+name);
        }
}
```

```
class StudInitMethod
{
  public static void main(String args[])
  {
  //Creating objects
  Student s1=new Student();

  //Initializing objects
  s1.insert_stud(101,"Jack");
  s1.display_data();
  }
}
```

By Constructor



Constructor

- 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 heap 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.

- There are rules defined for 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

- There are two types of constructors in Java:
  Parameterized constructor
  Default constructor

Default Constructor

- The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.

Default Constructor

- A constructor is called “non Parametrized Constructor" when it doesn't have any parameter.
- Syntax of non parametrized constructor is
- <class_name>(){}

```
class Student
{
Student()
    {
      System.out.println(“New student object is created");
   }
}
public class Const
{
    public static void main( String args[])
    {
        Student s=new Student();
     }
}
```

Constructors

- Java Parameterized Constructor
- A constructor which has a specific number of parameters is called a parameterized constructor.

- Constructor overloading
- In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods.
- Constructor overloading 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.

Parametrized Constructor

```
class Book
{
      int b_id;
      String name;

 Book()
 {
   System.out.println("Book is created");
   b_id=101;
   name="abc";
 }
 Book(int book_id,String b_name)
 {
     b_id=book_id;
     name=b_name;
 }
 void display()
 {
     System.out.println(b_id+" "+ name);
 }
}
```

```
public class Const1
{
 public static void main( String args[])
 {
    Book b=new Book();
    Book b1=new Book(102,"Java");
    b.display();
    b1.display();
 }
}
```

Constructor Overloading

```
class Book
{
      int b_id;
      String name;

 Book()
 {
   System.out.println("Book is created");
   b_id=101;
   name="abc";
 }
 Book(int book_id,String b_name)
 {
     b_id=book_id;
     name=b_name;
 }
Book(int id)
 {
     b_id= id;
 }
 void display()
 {
     System.out.println(b_id+" "+ name);
 }
}
```

```
public class Const1
{
 public static void main( String args[])
 {
    Book b=new Book();
    Book b1=new Book(102,"Java");
    Book b2=new Book(103);
    b.display();
    b1.display();
    b2.display();
 }
}
```

- this keyword
  - use of this,
  - constructor overloading using this keyword,
  - access current class method using this keyword
- Static variables and method
- Methods and method overloading
- Working with Multiple classes

this keyword in Java

The `this` keyword refers to the current object in a method or constructor.
The most common use of the `this` keyword is eliminate the confusion between class to attributes and parameters with the same name

Program without this

```
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);}
}
class TestThis1{
public static void main(String args[]){
Student s1=new Student(111,"ankit",5000f);
Student s2=new Student(112,"sumit",6000f);
s1.display();
s2.display();
}}
```

Using this

```
class Student{
int rollno;
String name;
float fee;
Student(int rollno,String name,float fee){
this.rollno=rollno;
this.name=name;
this.fee=fee;
}
void display(){System.out.println(rollno+" "+name+" "+fee);}
}

class TestThis2{
public static void main(String args[]){
Student s1=new Student(111,"ankit",5000f);
Student s2=new Student(112,"sumit",6000f);
s1.display();
s2.display();
}}
```

Calling parameterized constructor from default constructor:

```
class A{
A(){
this(5);
System.out.println("hello a");
}
A(int x){
System.out.println(x);
}
}
class TestThis6{
public static void main(String args[]){
A a=new A();
}}
```

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();
}}
```

this() : to invoke current class constructor

```
class A{
A(){System.out.println("hello a");}
A(int x){
this();
System.out.println(x);
}
}
class TestThis5{
public static void main(String args[]){
A a=new A(10);
}}
```

Static variables and methods

The **static keyword** in Java is **used for memory management** mainly. We can apply static keyword with variables, methods, blocks and nested classes.

The static can be:

1. Variable (also known as a class variable)
2. Method (also known as a class method)
3. Block
4. Nested classes

Java static variable

If you declare any variable as static, it is known as a static variable.
The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc.
The static variable gets memory only once in the class area at the time of class loading.

Advantages of static variable

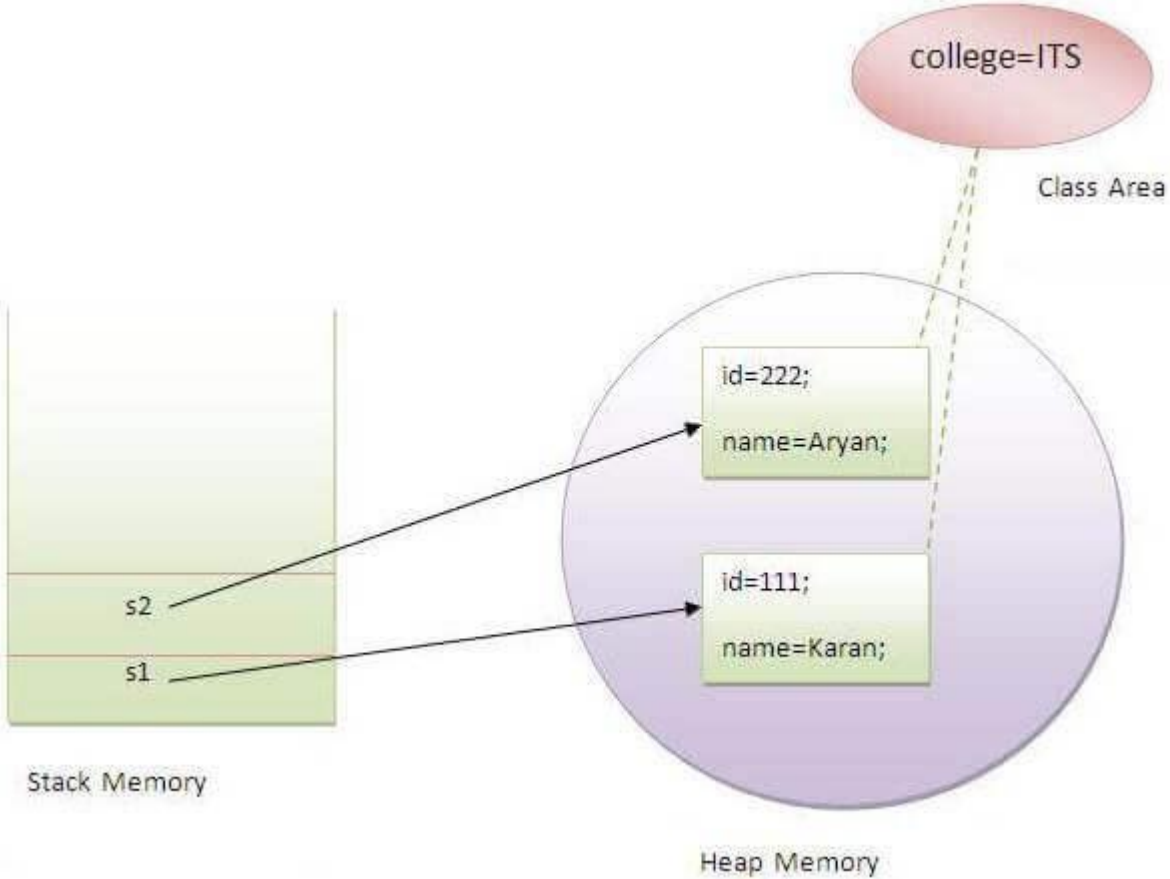
It makes your program **memory efficient** (i.e., it saves memory).

```
static String company_name=“XYZ”;
static int x=10;
```

Static Variables

```
class Student{
    int rollno;
    String name;
    String college=“ITS";
}
```

Suppose there are 500 students in college, now all instance data members will get memory each time when the object is created. All students have its unique rollno and name, so instance data member is good in such case. Here, "college" refers to the common property of all objects. If we make it static, this field will get the memory only once.


college=ITS
Class Area
id=222;
name=Aryan;
id=111;
name=Karan;
s2
s1
Stack Memory
Heap Memory

- The Java Memory Allocation is divided into following sections :
  Heap
  Stack
  Code
  Static
- This division of memory is required for its effective management.
  The **code** section contains your **bytecode**.
  The **Stack** section of memory contains **methods, local variables, and reference variables.**
  The **Heap** section contains **Objects** (may also contain reference variables).
  The **Static** section contains **Static data/methods**.

Java static block

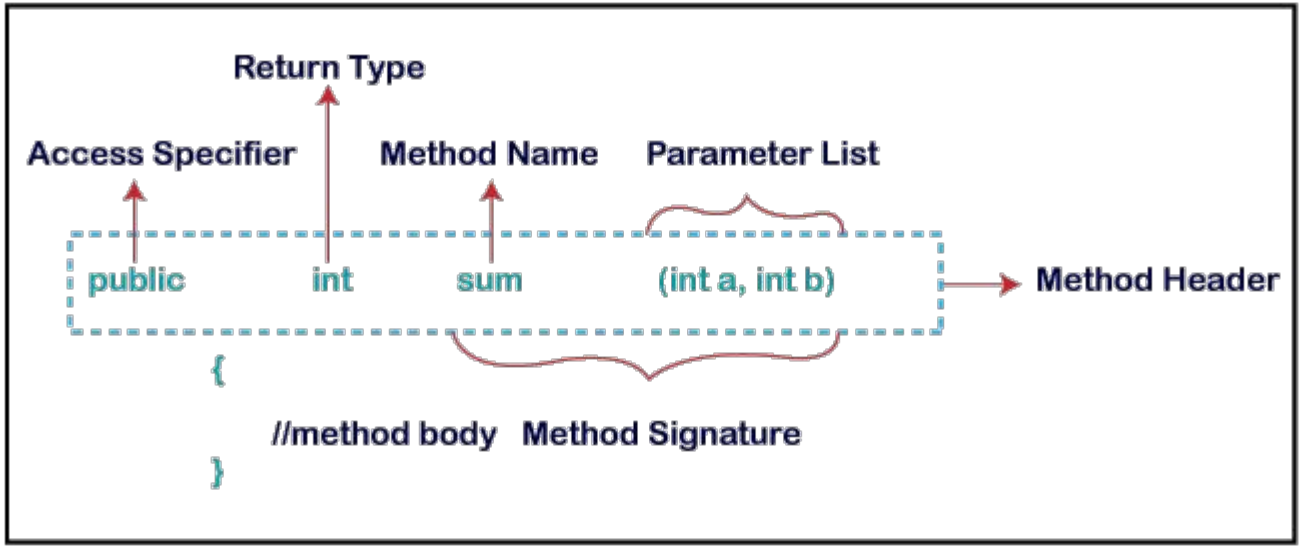
Java static block

Is used to initialize the static data member.

It is executed before the main method at the time of classloading.

```
class A2{
  static{System.out.println("static block is invoked");}
  public static void main(String args[]){
   System.out.println("Hello main");
  }
}
```

Method Declaration



Static Variable and Method

```
class Add{
 static int p=10;
 static int add(int x,int y)
 {
  return x+y;
 }
 int mul(int x1,int y1)
 {
       return x1*y1;
 }
}
```

```
class StaticVar3
  {
    static int a=20;
         static void abc()
         {
                  System.out.println("non static method");
         }
         public static void main(String args[])
         {
           //StaticVar3 s=new StaticVar3();
           Add m=new Add();
           System.out.println(a);
           abc();
           System.out.println("static variable of class add"+Add.p);
           int c=Add.add(2,3);
           System.out.println("addition is"+c);
           int mult=m.mul(4,5);
           System.out.println("addition is"+mult);
          }
  }
```

Access Modifiers:

- **Public, Protected, and Package-private Access Modifiers:**
- **Public**: Accessible from any other class.
- **Protected:** Accessible within the same package and by subclasses.
- **Package-private** (default): Accessible only within the same package.

```
public class Example
{
    public int publicVar;
    protected int protectedVar;
    int packagePrivateVar;
    private int privateVar;
}
```

Nested class

- In Java, it is possible to define a class within another class, such classes are known as *nested* classes. They enable you to logically group classes that are only used in one place, thus this increases the use of encapsulation, and creates more readable and maintainable code.
- The scope of a nested class is bounded by the scope of its enclosing class.
- Class A

{

//instance variable, method, class

}

- **Member Inner Classes:**
- A member inner class is a non-static class that is defined within another class. It can access the outer class’s members, including private ones.

Nested class syntax

```
class OuterClass
{
...
    class NestedClass
    {
        ...
    }
}
```

Nested Class

```
class OuterClass
{
 int a=10;
 void show()
 {
   System.out.println(a);
 }
 class InnerClass
 {
   void display()
   {
     System.out.println("display is InnerClass
method");
   }
 }
}
```

```
public class NestedClass
{
  public static void main(String args[])
  {
     OuterClass o=new OuterClass();
     OuterClass.InnerClass o1=o.new   InnerClass();
     o.show();
     o1.display();
  }
}
```

Static Nested Classes:

- Static Nested Classes: A static nested class is a class declared inside another class with the static modifier. Unlike inner classes, static nested classes do not have access to the instance variables and methods of the outer class.

```
public class OuterClass
{
   static class StaticNestedClass
      {
          void display()
      {
          System.out.println("Static nested class.");
       }
   }
}
```

For loop variation

```
int[] numbers = {1, 2, 3, 4, 5};
for (int num : numbers) {
System.out.println(num);
}
```

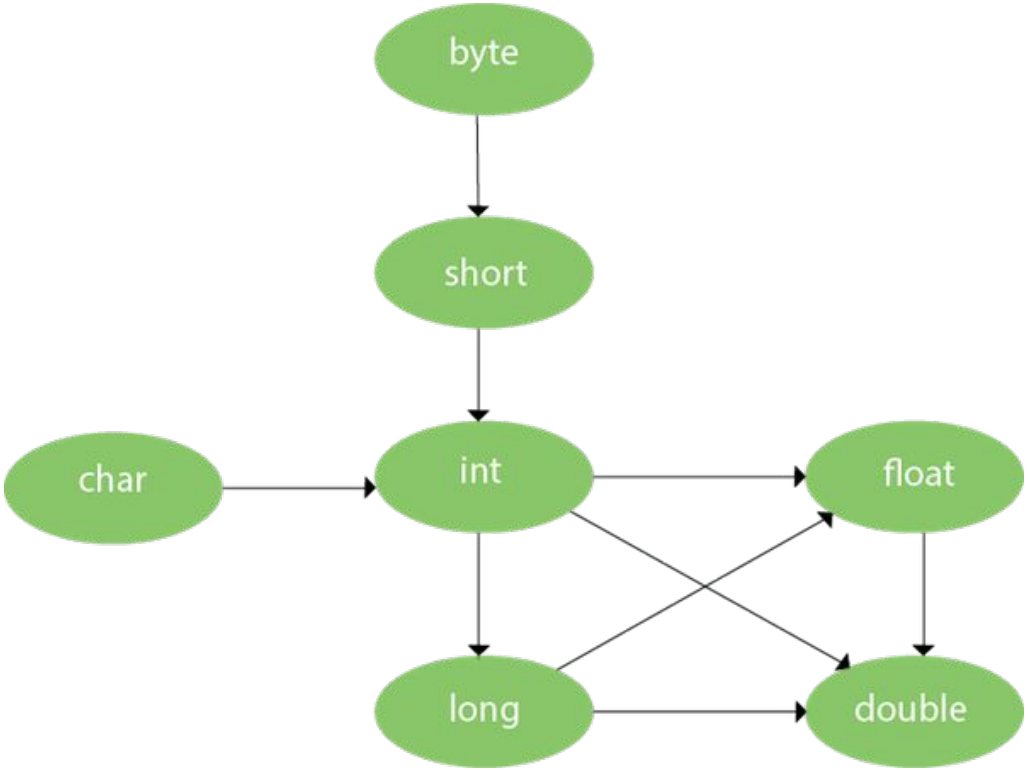
Method Overloading: changing no. of arguments

```
class Adder{
static int add(int a,int b){return a+b;}
static int add(int a,int b,int c){return a+b+c;}
}
class TestOverloading1{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(11,11,11));
}}
```

Method Overloading: changing data type of arguments

```
class Adder{
static int add(int a, int b){return a+b;}
static double add(double a, double b){return a+b;}
}
class TestOverloading2{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(12.9f,12.8f));
}}
```

Method Overloading and Type Promotion



Example of Method Overloading with TypePromotion

```
class OverloadingCalculation1{
  void sum(int a,long b){System.out.println(a+b);}
  void sum(int a,int b,int c){System.out.println(a+b+c);}

  public static void main(String args[]){
  OverloadingCalculation1 obj=new OverloadingCalculation1();
  obj.sum(20,10);//now second int literal will be promoted to long
  obj.sum(20,20,20);

  }
}
```

Example of Method Overloading with Type Promotion if matching found

If there are matching type arguments in the method, type promotion is not performed.

```
class OverloadingCalculation2{
  void sum(int a,int b){System.out.println("int arg method invoked");}
  void sum(long a,long b){System.out.println("long arg method invoked");}

  public static void main(String args[]){
  OverloadingCalculation2 obj=new OverloadingCalculation2();
  obj.sum(20,20);//now int arg sum() method gets invoked
  }
}
```

Example of Method Overloading with Type Promotion in case of ambiguity

```
class OverloadingCalculation3{
  void sum(int a,long b){System.out.println("a method invoked");}
  void sum(long a,int b){System.out.println("b method invoked");}

  public static void main(String args[]){
  OverloadingCalculation3 obj=new OverloadingCalculation3();
  obj.sum(20,20);//now ambiguity
  }
}
```

Method Overloading

```
class DisplayMethod
{
  void display(int a,int b)
  {
    System.out.println("two int args passed");
  }
  void display(int a,int b,int c)
  {
    System.out.println("three int args passed");
  }
  void display(String s1,String s2)
        {
    System.out.println("three int args passed");
  }
  void display(int a,long b)
  {
    System.out.println("one int and one long args passed");
  }
  void display(long a,int b)
  {
    System.out.println("one long and one int args passed");
  }
  void display(double a,double b)
  {
    System.out.println("two double args passed");
  }
}
```

```
public class MethodOverloading1
{
 public static void main(String args[])
 {
    DisplayMethod d=new DisplayMethod();
       d.display(2,5);
       d.display(3,7,5);
       d.display("abc","xyz");
       d.display(4.7f,5.6f);

 }
```

Create a class DisplayMethod

1. Add method - display(int a,int b)// print -2 int args passed
2. Add method -display(int a,int b,int c)// print -3 int args passed

3. Add method- display(string s1,string s2)// print -2 string args passed

4. Comment to 1 and add
   add method- display(int a, long b)//print - one int and one long args

5. display(long a, int b)//print - one long and one int args

6. Remove comment of 1 method

7. Add method -display(double a, double b)//print 2 double args passed
   access display in main method using display(2.3f,4.5f)

Working with more than one classes

- Create 2 classes class car(name, price) and class bike(brand,price)
- Add get_data(c_name ,c_price) and print_data() methods
- Create objects for the classes in main method.