➢ Objects and Classes in Java

Objects

- 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).
- An object has three characteristics:
- **State:** represents the data (value) of an object.
- **Behavior:** represents the behavior (functionality) of an object such as deposit, withdraw, etc.
- **Identity:** An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.
- For Example, Pen is an object.

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*.

Creating an Object in Java

- className object = new className();

Classes

- 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>{
field;
method;  }
```

➢ Method in Java

A **method** is a way to perform some task. Similarly, the **method in Java** is a collection of instructions that performs a specific task.

Advantages of methods

• Program development and debugging are easier
• Increases code sharing and code reusability
• Increases program readability
• It makes program modular and easy to understanding
• It shortens the program length by reducing code redundancy

Types of methods

There are two types of methods in Java programming:

• Standard library methods (built-in methods or predefined methods):

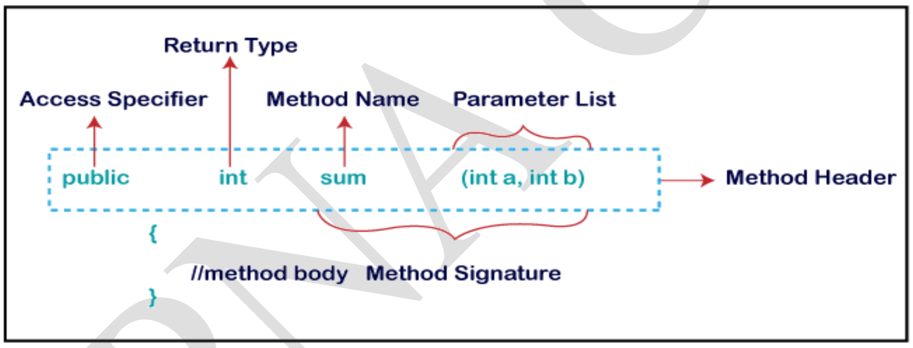
The standard library methods are built-in methods in Java programming to handle tasks such as mathematical computations, I/O processing, graphics, string handling etc. These methods are already defined and come along with Java class libraries, organized in packages.
eg. println( ), sqrt( )

• **User defined methods:**
User-defined methods
The methods created by user are called user defined methods. eg. Sum( ).

Every method has the following.
• Method declaration (also called as method signature or method prototype)
• Method definition (body of the method)
• Method call (invoke/activate the method)

Method Declaration:
Syntax:
return_type method_name(parameter_list);
Here, the return_type specifies the data type of the value returned by method. It will be void if the method returns nothing. method_name indicates the unique name assigned to the method. parameter_list specifies the list of values accepted by the method.

Method Definition
Method definition provides the actual body of the method. The instructions to complete a specific task are written in method definition. The syntax of method is as follows:
Syntax:
```
Modifier/Access specifier return_type method_name(parameter_list)
{
// body of the method
}
```



• **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 only.

• **Return Type:** Return type is a data type that the method returns. It may have a primitive data type, object, collection, void, etc. If the method does not return anything, we use void keyword.

• **Method Name:** It is a unique name that is used to define the name of a method. It must be corresponding to the functionality of the method. Suppose, if we are creating a method for subtraction of two numbers, the method name must be **subtraction().** A method is invoked by its name.

- **Parameter List:** It is the list of parameters separated by a comma and enclosed in the pair of parentheses. It contains the data type and variable name. If the method has no parameter, left the parentheses blank.
- **Method Body:** It is a part of the method declaration. It contains all the actions to be performed. It is enclosed within the pair of curly braces.

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**.

Method Call

A method gets executed only when it is called. The syntax for method call is.
Syntax:
method_name(parameters); or
object_name .method_name()
When a method is called, the program control transfers to the method definition where the actual code gets executed and returns back to the calling point. The number and type of parameters passed in method call should match exactly with the parameter list mentioned in method prototype.

- **Instance method**

The method of the class is known as an **instance method**. It is a **non-static** method defined in the class. Before calling or invoking the instance method, it is necessary to create an object of its class.

InstanceMethodExample.java

```
public class Instance  {
public static void main(String [] args)  {
Instance obj = new Instance ();
System.out.println("The sum is: "+obj.add(12, 13));  }
int s;
public int add(int a, int b)  {
s = a+b;
return s;
}
}
```

Output: The sum is: 25

Object and Class Example: main within the class

```
class HelloWorld {
  int id=12;
```

```
 boolean a= true;
  public static void main(String[] args) {
 HelloWorld h1=new HelloWorld();
   System.out.println(h1.id);
    System.out.println(h1.a);           }
 }
```

Object and Class Example: main outside the class

```
class HelloWorld {
  int id=12;
 boolean a= true; }
  class Test{
  public static void main(String[] args) {
 HelloWorld h1=new HelloWorld();
   System.out.println(h1.id);
    System.out.println(h1.a);           }  }
```

Ways to initialize object

There are 3 ways to initialize object in Java.

- By reference variable
- By method
- By constructor

- **Initialization through reference variable**
- Initializing an object means storing data into the object. Let's see a simple example where we are going to initialize the object through a reference variable.

```
class Student{
int id;
String name;}
class TestStudent2{
public static void main(String args[]){
Student s1=new Student();
s1.id=101;
s1.name="Sonoo";
System.out.println(s1.id+" "+s1.name);//printing members with a white space}}
```

Output:

```
101 Sonoo
```

- **Initialization through method**
- In this example, we are creating the two objects of Student class and initializing the value to these objects by invoking the insertRecord method. Here, we are displaying the state (data) of the objects by invoking the displayInformation() method.

```
class Student{
int rollno;
String name;
void insertRecord(int r, String n){
rollno=r;
name=n; }
void displayInformation(){System.out.println(rollno+" "+name);} }
class TestStudent4{
public static void main(String args[]){
Student s1=new Student();
Student s2=new Student();
s1.insertRecord(111,"Karan");
s2.insertRecord(222,"Aryan");
s1.displayInformation();
s2.displayInformation(); }
}
```

Output:

```
111 Karan
222 Aryan
```

- **Java static keyword**

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 keyword belongs to the class than an instance (object) of the class.

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.
- Java static property is shared to all objects.

Example

```
class Student{
int roll_no;//instance variable
String name;
static String college ="SIPNA";//static variable
//constructor
Student(int r, String n){
roll_no = r;
name = n;
}
//method to display the values
void display (){System.out.println (rollno+" "+name+" "+college);}
}
//Test class to show the values of objects
public class TestStaticVariable1{
public static void main(String args[]){
Student s1 = new Student(111,"Karan");
Student s2 = new Student(222,"Aryan");
s1.display();
s2.display();
}
}
```

Output:111 karan SIPNA

```
222 Aryan SIPNA
```

Java static method

If you apply static keyword with any method, it is known as static method.

o A static method belongs to the class rather than the instance (object) of a class.

o A static method can be invoked without the need for creating an instance of a class.

o A static method can access static data member and can change the value of it.

- **Syntax to declare the static method:**

  Access_modifier static void methodName() { // Method body. }

- **Syntax to call a static method:**

  className.methodName();

Example of static method

```
//Java Program to demonstrate the use of a static method.
class Student{
int rollno;
String name;
static String college = "ABC";
//static method to change the value of static variable
static void change(){
college = "SIPNA"; }
//constructor to initialize the variable
Student(int r, String n){
rollno = r;
name = n; }
//method to display values
void display(){System.out.println (rollno+" "+name+" "+college);} }
//Test class to create and display the values of object
public class Test{
public static void main(String args[]){
Student.change();//calling (static)change method
//creating objects
Student s1 = new Student(111,"Karan");
Student s2 = new Student(222,"Aryan");
Student s3 = new Student(333,"Sonoo");
//calling display method
s1.display();
s2.display();
s3.display(); } }
```

Output:

```
111 Karan SIPNA
 222 Aryan SIPNA
 333 Sonoo SIPNA
```

NOTE:

Why is the Java main method static?

Ans) It is because the object is not required to call a static method. If it were a non-static method, JVM creates an object frst then call main() method that will lead the problem of extra memory allocation.

Can we execute a program without main() method?
Ans) No, one of the ways was the static block, but it was possible till JDK 1.6. Since JDK 1.7, it is not possible to execute a Java class without the main method.

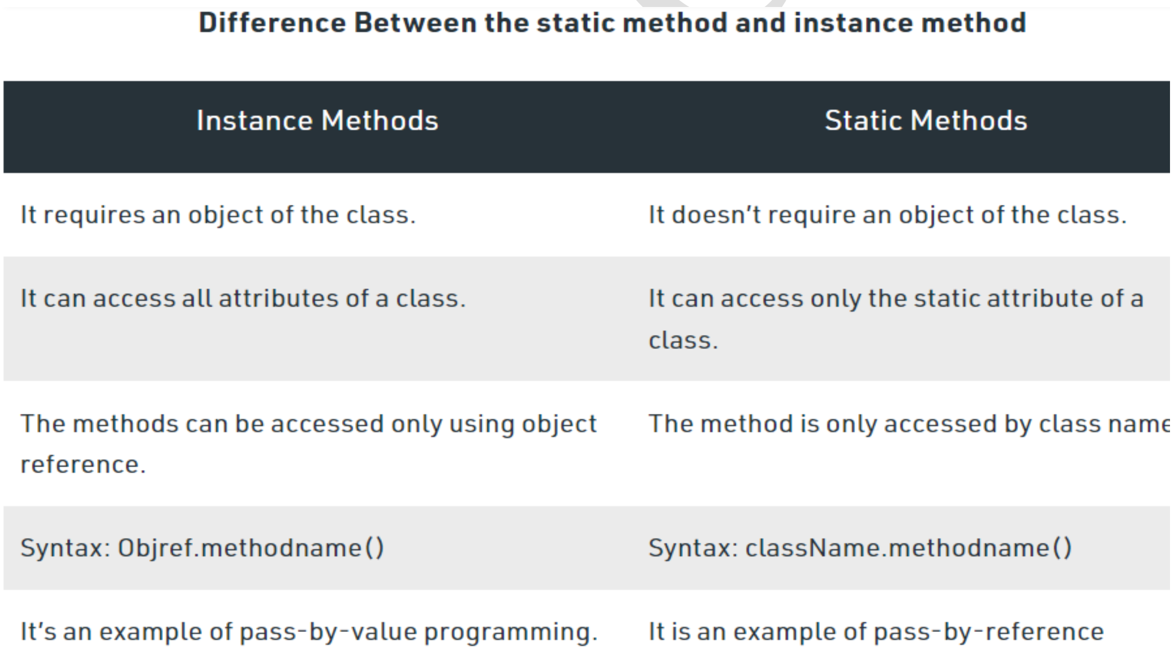
```
class A3{
static{
System.out.println("static block is invoked");
System.exit(0);
}
}
```

Output:

```
static block is invoked
```

Since JDK 1.7 and above, output would be:

```
Error: Main method not found in class A3, please define the main method as:
 public static void main(String[] args)
or a JavaFX application class must extend javafx
```



Difference Between the static method and instance method

Instance Methods	Static Methods
It requires an object of the class.	It doesn't require an object of the class.
It can access all attributes of a class.	It can access only the static attribute of a class.
The methods can be accessed only using object reference.	The method is only accessed by class name
Syntax: Objref.methodname()	Syntax: className.methodname()
It's an example of pass-by-value programming.	It is an example of pass-by-reference

- **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 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 two types of constructors in Java: Default Constructor(no-arg constructor), and parameterized constructor.

- **Rules for defining constructors**

1. Constructors have the same name as that of the class.
2. Constructors are similar to methods in syntax, but constructors do not have a return type; should not even specify void.
3. If you specify return type, then the supposed to be constructor will become a method and will not be considered as a constructor.
4. Constructors can also be overloaded just like methods.
5. Constructors can be invoked only during object creation or from other constructors using this keyword.
6. Only one constructor is invoked based on which overloaded version we specify. This constructor can then call other constructors if required, using this keyword (e.g. this() or this(2) etc.).
7. A default constructor is a constructor that takes no arguments, and mostly does nothing.
8. A default constructor with no parameters is automatically added by Java for a class if there are no user defined constructors.
9. Hence, even if we don't have a constructor for our class, we can instantiate a class using a no-argument constructor.
10. If we provide at least one constructor, the default constructor is no longer added by Java.
11. One constructor can call another constructor using "this" keyword (e.g. this() or this(2)); arguments should match a constructor signature similar to method. You use the method name to specify a method whereas you use the "this" keyword for a constructor. This is called constructor chaining.
12. During inheritance, subclass constructor should call a super class constructor. We use super keyword invoke a super class constructor (e.g. super(), super(1) etc).
13. A Java constructor cannot be abstract, static, final, and synchronized.

- **Default Constructor**

A constructor is called "Default Constructor" when it doesn't have any parameter.

Syntax of default constructor:

```
<class_name>()
{
Block of statements;
}
```

```
 //Java Program to create and call a default constructor
   class Bike1{
   Bike1()
{
System.out.println("Bike is created");
}
 public static void main(String args[]){
   Bike1 b=new Bike1();
   }
   }
```

Output:Bike is created

- **Parameterized Constructor**

A constructor which has a specific number of parameters is called a parameterized constructor.

Why use the parameterized constructor?

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

```
//Java Program to demonstrate the use of the parameterized constructor.
class Student4{
   int id;
   String name;
   //creating a parameterized constructor
   Student4(int i,String n){
   id = i;
   name = n;  }
   void display()
{ System.out.println(id+" "+name);}
   public static void main(String args[]){
   //creating objects and passing values
   Student4 s1 = new Student4(11,"ABC");
   Student4 s2 = new Student4(22,"XYZ");
   //calling method to display the values of object
   s1.display();
   s2.display();
  }
}
Outout:11 ABC
         22 XYZ
```

- **Constructor Overloading in Java**

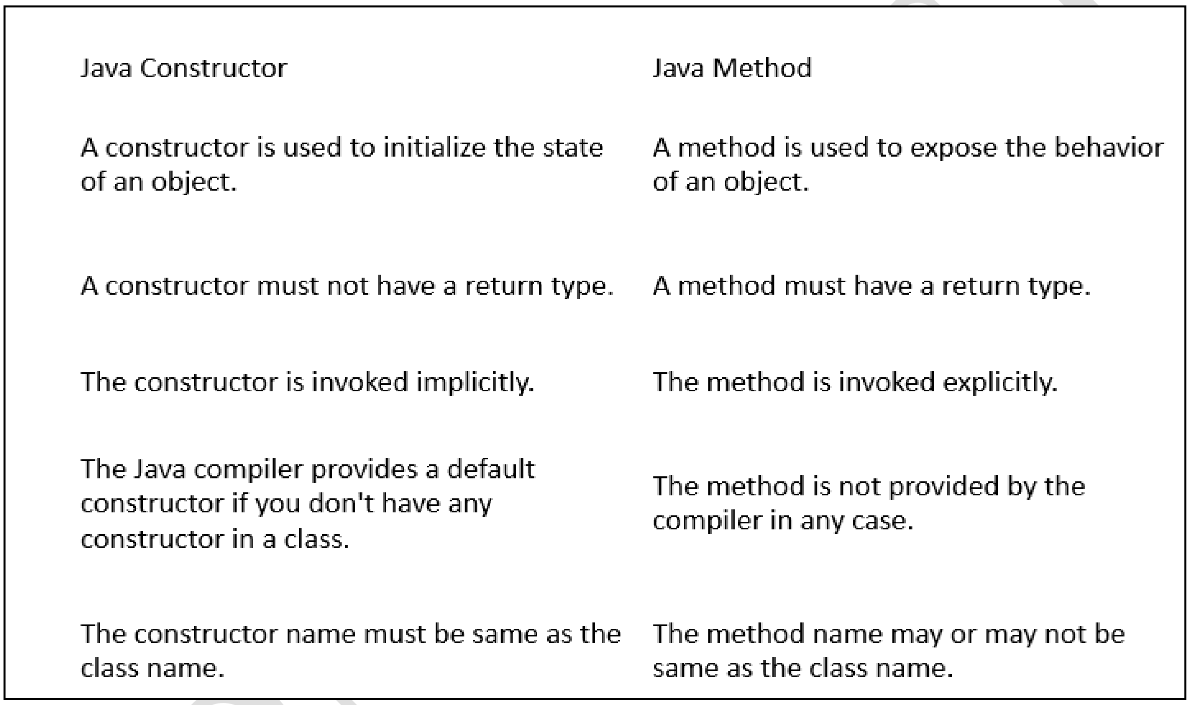
In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods. Constructor overloading in Java is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types.

Example of Constructor Overloading

```
//Java program to overload constructors
class Student5{
   int id;
   String name;
   int age;
   //creating two arg constructor
   Student5(int i,String n){
   id = i;
   name = n;
   }
   //creating three arg constructor
   Student5(int i,String n,int a){
```

```
id = i;
name = n;
age=a;
}
void display(){System.out.println(id+" "+name+" "+age);}
public static void main(String args[]){
Student5 s1 = new Student5(111,"Karan");
Student5 s2 = new Student5(222,"Aryan",25);
s1.display();
s2.display();   }  }
```

- **Difference between constructor and method in Java**



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.

- **Method Overloading**

Method Overloading is a feature that allows a class to have more than one method having the same name, if their argument lists are different. It is similar to constructor overloading in Java, that allows a class to have more than one constructor having different argument lists.

Three ways to overload a method

In order to overload a method, the argument lists of the methods must differ in either of these:

1.Number of parameters:

  add(int, int), add(int, int, int)

2. Data type of parameters.

  add(int, int) , add(int, float)

3. Sequence of Data type of parameters.

  add(int, float) , add(float, int)

- **Different Number of parameters in argument list**

```
class DisplayOverloading{
   public void disp(char c){
        System.out.println(c);
    }
    public void disp(char c, int num)  {
        System.out.println(c + " "+num);
    }}
class Sample{
  public static void main(String args[]){
      DisplayOverloading obj = new DisplayOverloading();
      obj.disp('a');
      obj.disp('a',10);
  }}
```

Output:

```
a
a 10
```

- **Difference in data type of parameters**

```
    class DisplayOverloading2 {
        public void disp(char c){
            System.out.println(c);}
        public void disp(int c) {
          System.out.println(c );
        }}
    class Sample2{
        public static void main(String args[]) {
            DisplayOverloading2 obj = new DisplayOverloading2();
            obj.disp('a');
            obj.disp(5);
       }}
```

- **Sequence of data type of arguments**

```
class DisplayOverloading3{
  public void disp(char c, int num) {
      System.out.println("I'm the first definition of method disp"); }
  public void disp(int num, char c){
      System.out.println("I'm the second definition of method disp" ); }}
class Sample3{
  public static void main(String args[]){
      DisplayOverloading3 obj = new DisplayOverloading3();
      obj.disp('x', 51 );
      obj.disp(52, 'y'); }}
```

- **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

Here is given the 6 usage of java this keyword.

1. this keyword can be used to refer current class instance variable.
2. this() can be used to invoke current class constructor.
3. this keyword can be used to invoke current class method (implicitly)
4. this can be passed as an argument in the method call.
5. this can be passed as argument in the constructor call.
6. this keyword can also be used to return the current class instance.

1) The this keyword can be used to refer current class instance variable.

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 Student10{
    int id;
    String name;
    Student10(int id,String name){
    id = id;
    name = name;}
    void display(){System.out.println(id+" "+name);}
    public static void main(String args[]){
    Student10 s1 = new Student10(111,"Karan");
    Student10 s2 = new Student10(321,"Aryan");
    s1.display();
    s2.display();}}
```

Output:

```
0 null
0 null
```

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

```
//example of this keyword
class Student11{
int id;
String name;
Student11(int id, String name){
this.id = id;
this.name = name;}
void display(){System. out. println (id+" "+name);}
public static void main(String args[]){
Student11 s1 = new Student11(111,"Karan");
Student11 s2 = new Student11(222,"Aryan");
s1.display();
s2.display(); } }
```

Output:

```
111 Karan
222 Aryan
```

If local variables(formal arguments) and instance variables are different, there is no need to use this keyword like in the following program: Program where this keyword is not required

```
class Student12{
int id;
String name;
Student12(int i,String n){
id = i;
name = n;
}
void display(){System.out.println(id+" "+name);}
public static void main(String args[]){
Student12 e1 = new Student12(111,"karan");
Student12 e2 = new Student12(222,"Aryan");
e1.display();
e2.display();
}
}
```

Output:

```
111 Karan
222 Aryan
```

2) this() can be used to invoked current class constructor or class method

The this() constructor call can be used to invoke the current class constructor (constructor chaining). This approach is better if you have many constructors in the class and want to reuse that constructor.

```
class Student13{
int id;
String name;
Student13(){System.out.println("default constructor is invoked");}
Student13(int id,String name){
this ();//it is used to invoked current class constructor.
this.id = id;
this.name = name;}
void display(){System.out.println(id+" "+name);}
public static void main(String args[]){
Student13 e1 = new Student13(111,"karan");
Student13 e2 = new Student13(222,"Aryan");
e1.display();
e2.display();}}
```

Output:

```
default constructor is invoked
default constructor is invoked
111 Karan
222 Aryan
```

3)The this keyword can be used to invoke current class method (implicitly).

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. Let's see the example

```
class S{
void m(){
System.out.println("method is invoked"); }
void n(){
this.m();//no need because compiler does it for you.
}
void p(){
n();//complier will add this to invoke n() method as this.n() }
public static void main(String args[]){
S s1 = new S();
s1.p();
} }
```

Output: method is invoked

- **Java Arrays**

Normally, an array is a collection of similar type of elements which have a 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.

Types of Array in java

There are two types of array.

- Single Dimensional Array
- Multidimensional Array

- **Advantages Of An Array:**

1. Arrays are used to store multiple data items of same type by using only single name.
2. We can access any element randomly by using indexes provided by arrays.
3. Arrays can be used to implement other data structures like linked lists, stacks, queues, trees, graphs etc.
4. Primitive type to wrapper classes object conversion will not happen so it is fast.

-

- **Disadvantages Of An Array:**

1. Fixed Size: We need to mention the size of the array, thus they have fixed size. When array is created, size cannot be changed.
2. Memory Wastage: There is a lot of chance of memory wastage. Suppose we creat an array of length 100 but only 10 elements are inserted, then 90 blocks are empty and thus memory wasted.
3. Strongly Typed: Array stores only similar data type, thus strongly typed.
4. Reduce Performance: The elements of array are stored in consecutive memory locations, thus to delete an element in an array we need to traverse through out the array so this will reduce performance.

- **Single Dimensional Array in Java**

Syntax to Declare an Array in Java

```
dataType[] arr; (or)
dataType []arr; (or)
dataType arr[];
```

- **Example of Java Array**

```
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.print(a[i]);
}}
```

Output:

```
10 20 70 40 50
```

For-each Loop for Java Array

- The Java for-each loop prints the array elements one by one. It holds an array element in a variable, then executes the body of the loop.

The **syntax** of the for-each loop is given below:

```
for(data_type variable:array){
//body of the loop
}
class Testarray1{
public static void main(String args[]){
int arr[]={33,3,4,5};
//printing array using for-each loop
for(int i:arr)
System.out.println(i);
}}
```

- **Multidimensional Array in Java**

In such case, data is stored in row and column based index (also known as matrix form).

Syntax to Declare Multidimensional Array in Java

```
dataType[][] arrayRefVar; (or)
dataType [][]arrayRefVar; (or)
dataType arrayRefVar[][]; (or)
dataType []arrayRefVar[];
```

```
class Array{
public static void main(String args[]){
//declaring and initializing 2D array
int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
//printing 2D array
for(int i=0;i<arr.lenght;i++){
 for(int j=0;j<arr[i].lenght;j++){
  System.out.print(arr[i][j]+" ");
 }
 System.out.println();
}
}}
```

Output:

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

- **Java Garbage Collection**

In java, garbage means unreferenced objects.

Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.

To do so, we use free() function in C language and delete() in C++. But, in java it is performed automatically. So, java provides better memory management.

Advantage of Garbage Collection

It makes java **memory efficient** because garbage collector removes the unreferenced objects from heap memory.

It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extra efforts.

How can an object be unreferenced?

There are many ways:

1. By nulling the reference
2. By assigning a reference to another
3. By anonymous object etc.

1) By nulling a reference:

Employee e=**new** Employee();
e=**null**;

2) By assigning a reference to another:

Employee e1=**new** Employee();
Employee e2=**new** Employee();
e1=e2;

3) By anonymous object:

new Employee();

- **finalize() method**

The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing. This method is defined in Object class as:

- **protected void** finalize(){}

Note: The Garbage collector of JVM collects only those objects that are created by new keyword. So if you have created any object without new, you can use finalize method to perform cleanup processing (destroying remaining objects).

- **gc() method**

The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes.

public static void gc(){}

Note: Garbage collection is performed by a daemon thread called Garbage Collector(GC). This thread calls the finalize() method before object is garbage collected.

Simple Example of garbage collection in java:

```
public class TestGarbage1{
 public void finalize(){System.out.println("object is garbage collected");}
 public static void main(String args[]){
  TestGarbage1 s1=new TestGarbage1();
  TestGarbage1 s2=new TestGarbage1();
  s1=null;
  s2=null;
  System.gc();
 }
}
```

```
object is garbage collected
object is garbage collected
```

- **Command-line argument**

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.

- **Simple example of command-line argument in java**

In this example, we are receiving only one argument and printing it. To run this java program, you must pass at least one argument from the command prompt.

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

compile by > javac CommandLine.java
run by > java CommandLine sipna college amt
Output: Your first argument is: sipna

```
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 sipna college amt 12 13 JPG
Output: sipna
College
Amt
12
13

JPG