***Java documentation***

***1.Class and Object***

***Class:***

* *A****class****is a user****defined****blueprint or prototype from which objects are created.*
* *It represents the set of properties or methods that are common to all objects of one type generally class can have these components,they are:*

***Modifiers****:*

* *A class can be public or has default access .*

***class keyword:***

* *to create a class by using of keyword class.*

***Class name:***

* *class name should be maintain pascal casing.*

***Superclass :***

* *The name of the class’s parent (superclass), if any, preceded by the keyword extends.*
* *A class can only extend (subclass) one parent.*

***Interfaces :***

* *A comma-separated list of interface* *implemented by the class, if any, preceded by the keyword implements.*
* *A class can implement more than one interface.*

***Body:***

* *The class body surrounded by braces, { }.*

***Example:*** *Class Demo{*

*Int a=23;*

*}*

***Object:*** *Object is an instance of a class.it is a real world entity.*

*Object is created many as per the requirement by using of keyword new.*

***Example:*** *class Sample{*

*Sample s1=new Sample();*

*}*

* *S1 is known as object reference variable.*

***2.Static data and Non static data members***

*Members=methods+variable.*

***Static data members:***

* *they are also known as class data members.*
* *Memory will be allocated only once .whenever the class is loaded in main memory regardles of number of objects are created*
* *Class variables are declared by using keyword static*
* *The variables are accessed with class name.*

***Non static data members:***

* *This are also known as Object level data members.*
* *Memory will be allocated each and every time whenever the object Is created*
* *This variables are declared without using of keyword static.*
* *The variables are accessed with object name.*

***Example:***

*Class Sample{*

*static int a=10; //static variable*

*Public static void run() //static method*

*System.out.println(“run”+a+”km/h”);*

*}*

*}*

*Class Demo*

*{*

*Int b=20; //non static variable*

*Public void walk() //non static method*

*{*

*System.out.println(“walk”+b+”km*  */h”);*

*}*

*}*

*Class main*

*{*

*Public static void main (String[] args)*

*{*

*Sample.run(); //calling by using class name*

*Demo d=new Demo(); //calling by using the creation of object*

*d.walk();*

*}*

*}*

***3.Static methods and Non static methods, private and public methods and data.***

***Static methods:***

* *A****static method****is a method that belongs to a class.*

*But it does not belong to Object of the class*

* *the method can access only static data members and static methods of another class or same class but cannot access non-static methods and variables.*
* ***Static****method uses compile time or early binding*
* *In****static****method, less memory is use for execution because memory allocation happens only once, because the static keyword fixed a particular memory ..*

***Non -static methods:***

* *Every methods in java defaults to non-static method without****static****keyword preceding it.*
* *In****non-static****method, the method can access static data members and static methods as well as non-static members and method of another class or same class*
* ***Non-static****method uses runtime or dynamic binding.*
* *In****non-static****method, much memory is used for execution because here memory allocation happens when the method is invoked and the memory is allocated every time when the method is called.*

***Public:***

1. *It can be specified by using the****public****keyword.*
2. *The variables, classes, and methods declared as public can be accessed from everywhere in the program.*
3. *It does not impose restrictions on the scope of public data members.*
4. *If we declare methods and classes as public, they also break the principle of encapsulation. We can also use it with the top-level classes.*

***Example:***

***Demo1.java***

*Package p1;*

*Class Demo1*

*Public void show(){*

*System.out.println(“show the bike”);*

*}*

*}*

***Demo2.java***

*Package p2;*

*Import p1.\*;*

*Class Demo2*

*Public static void main( String[] args)*

*{*

*Demo1 obj=new Demo1();*

*Obj.show();*

*}*

*}*

***Output:***

*Show the bike.*

***Private:***

1. *It can be specified by using the****private****keyword followed by class name (applied only on nested classes) or method name or data member.*
2. *We cannot use the private access specifier with the top-level classes or interfaces.*
3. *The variables, methods, and classes declared as private can be accessed only in the class in which they are declared or by inheriting the parent classes.*
4. *It is the most restricted access specifier in Java.*
5. *It is the heights form of encapsulation*
6. *We cannot override the methods that are declared as private.*
7. *If we use the private modifier with the constructor, we avoid it from being sub-classed.*
8. *It hides the classes from the other classes within the same package.*

***Example:***

*Class Demo1*

*Private void show(){ //defining method as private*

*System.out.println(“show the bike”);*

*}*

*}*

*Class Demo2*

*Public static void main( String[] args)*

*{*

*Demo1 obj=new Demo1();*

*Obj.show(); //trying to access private method of the class Demo1*

*}*

*}*

*When we execute the above program, it shows the following error:*

*Show() access private in Demo1*

*Obj.show(); //error*

***4.Static Blocks and Non static blocks***

***Static Blocks:***

* *The****Static Block****is a block of statement inside a Java class that will be executed when a class is first loaded into the JVM.*
* *A****Static blocks are helps to initialize static data members****, just like constructors help to initialize instance members.*

***Example:***

*Public class Sample{*

*Static int n;*

*Static int r;*

*Static{*

*n=14;*

*r=18;*

*}*

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

*System.out.println(“value of n is:”+n);*

*System.out.println(“value of r is:”+r);*

*}*

*}*

***Non-static Blocks:***

* *When ever object created non static blocks will be executed before the execution of constructor*
* *Non static blocks are class level block which does not have prototype*
* *To execute any logic whenever object is created irrespective of constructor used in object creation.*
* *Non static blocks are automatically called by JVM for every object creation in java stack area*

***Example:***

*Public class A{*

*System.out.println(“first block”);*

*}*

*{*

*System.out.println(“second block”);*

*}*

*{*

*System.out.println(“third block”);*

*}*

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

*A obj=new A();*

*}*

*}*

***5.Constructors and Constructor types***

***Constructor:***

1. *constructor is a block of codes similar to the method.*
2. *It is called when an instance of the class is created.*
3. *At the time of calling constructor, memory for the object is allocated in the memory.*
4. *It is a special type of method which is used to initialize the object.*
5. *Every time an object is created using the new() keyword, at least one constructor is called.*

*Rules for creating java constructor:*

* *Constructor name must be the same as its class name*
* *A Constructor must have no explicit return type*
* *A Java constructor cannot be abstract, static, final, and synchronized*

***Types of Constructors:***

*There are two types of constructors in Java:*

* *Default constructor (no-arg constructor)*
* *Parameterized constructor*

***Default constructor:***

* *if there is no constructor available in the class.*

*In such case, Java compiler provides a default constructor by default.*

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

***Example:***

*class student{*

*int id;*

*String name;*

*//method to display the value of id and name*

*Void display()*

*{*

*System.out.println(id+” “+name);*

*}*

*Public static void main(String[] args)*

*//creating objects*

*Student s1=new student();*

*Student s2=new student();*

*//displaying values of the object*

*S1.display();*

*S2.display();*

*}*

*}*

***Ouput:***

1. *null*

*0 null*

***Parameterized constructor:***

* *A constructor which has a specific number of parameters is called a parameterized constructor.*
* *The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also*

***Example:***

*class student{*

*int id;*

*String name;*

*//creating aparameterized constructor*

*Student(int i,String n){*

*id=i;*

*name=n;*

*//method to display the value of id and name*

*Void display()*

*{*

*System.out.println(id+” “+name);*

*}*

*Public static void main(String[] args)*

*//creating objects and passing values*

*Student s1=new student(23,”bheem”);*

*Student s2=new student(42,”chutki”);*

*// calling method to displaying values of the object*

*S1.display();*

*S2.display();*

*}*

*}*

***Output:***

*23 bheem*

*42 chutki*

***6.Method Overloading***

* *If a class has multiple methods having same name but different in parameters, it is known as****method overloading****.*
* *If we have to perform only one operation, having same name of the methods.*
* *Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs.*
* *Method overloading*increases the readability of the program.

*There are two ways to overload the method in java*

1. *By* *changing number of arguments*
2. *By changing the data type*

***changing number of arguments***

*class Added{*

*static int add(int a,int b)*

*{*

*Return a+b;*

*}*

*Static int add(int a,int b,int c)*

*{*

*Return a+b+c;*

*}*

*Class TestOverloading{*

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

*System.out.println(Added.add(23,42));*

*System.out.println(Added.add(23,42,55));*

*}*

*}*

***changing the data type***

*class substract{*

*static int sub(int a,int b)*

*{*

*Return a-b;*

*}*

*Static double sub(double a, double b)*

*{*

*Return a-b;*

*}*

*Class TestOverloading{*

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

*System.out.println(**substract.sub(42,23));*

*System.out.println( substract.sub(5.5,2.3));*

*}*

*}*

***7.Types of Classes***

*There are different types of classes.*

* *Top level classes*
* *Nested classes*
* *Member classes*
* *Static member classes.*
* *Non static member classes.*
* *Anonymous classes*
* *Local classes*

***Example:***

*Public class TestCase{*

*Class manager{*

*}*

*Static class Address{*

*}*

*Public void method(){*

*Int a;*

*Class Employee{*

*}*

*Student s=new Student();*

*}*

*}*

***Toplevel class:****Testcase.*

***Non static member class:****manager.*

***Static member class:****Address.*

***Local class:*** *Employee****.***

***Anonymous Class:****Student.*

***8. Association,*** ***Composition and Aggregation  
    ("has a" relation)***

***Association:***

* *Association is relation between two separate classes which establishes through their Objects.*
* *Association can be one-to-one, one-to-many, many-to-one, many-to-many.*
* *In Object-Oriented programming, an Object communicates to other Object to use functionality and services provided by that object.*
* *composition and****Aggregation****are the two forms of association.*

***Example:***

*Class Bank{*

*Private String name;*

*Bank(String name)*

*{*

*This.name=name;*

*}*

*Public String getBankName()*

*{*

*Return this.name;*

*}*

*}*

*Class Employee*

*{*

*Private String name;*

*Employee(String name)*

*{*

*This.name=name;*

*}*

*Public String getEmployeeName()*

*{*

*Return this.name;*

*}*

*}*

*// Association between both the*

*// classes in main method*

*class Association*

*{*

*public static void main (String[] args)*

*{*

*Bank bank = new Bank("Axis");*

*Employee emp = new Employee("Neha");*

*System.out.println(emp.getEmployeeName() +*

*" is employee of " + bank.getBankName());*

*}*

*}*

***Composition:***

* *The composition is a design technique in java to implement a****has-a****relationship.*
* *Java  inheritance is used for code reuse purposes and the same we can do by using composition.*
* *The composition is achieved by using an instance variable that refers to other objects.*
* *If an object contains the other object and the contained object cannot exist without the existence of that object, then it is called composition.*
* *In more specific words composition is a way of describing reference between two or more classes using instance variable and an instance should be created before it is used.*

***Example:***

|  |
| --- |
| *class Book {*  *public String title;*  *public int price;*  *Book(String title,int price)*  *{*  *this.title = title;*  *this.price = price;*  *}*  *}*  *// Libary class contain.*  *// list of books.*  *class Library {*  *//reference to refer to list of books.*  *private final List<Book> books;*  *Library(List<Book> books) {*  *this.books = books;*  *}*    *// Getting total number of books*  *public List<Book> getTotalBooksInLibrary()*  *{*  *return books;*  *}*  *}*  *class Main{*  *public static void main(String[] args)*  *{*  *// Creating the Objects of Book class.*  *Book b1= new Book("ramayana",1000);*  *Book b2  = new Book("mahabaratham",600);*  *Book b3 = new Book("panchathantra”,300);*  *// Creating the list which contains the*  *// no. of books*  *List<Book> book = new ArrayList<Book>();*  *book.add(b1);*  *book.add(b2);*  *book.add(b3);*    *Library library = new Library(book);*  *List<Book> books = library. getTot*  *alBooksInLibrary();*  *for (Book bk : books) {*  *System.out.println(“title”+bk.title+”price:”+bk.price);*  *}*  *}*  *}* |

***Aggregation:***

* *Aggregation is a type of HAS-A relationship.between two objects in which one contain the other’s reference.*
* *Two objects can exist independently. If one is deleted other can still exist.*

***Example:***

*Class Chair{*

*Public void show() {*

*System.out.println(“show chair details.”);*

*}*

*}*

*Public class RoomTest {*

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

*//Chair class object in RoomTest class*

*Chair obj = new Chair();*

*Obj.show();*

*}*

*}*

***9.Packages(creating a package, importing classes from a package, creating a jar file, creating a self executable jar file)***

***Packages:***

* **PACKAGE in Java** is a collection of classes, sub-packages, and interfaces and enums.
* It helps organize your classes into a folder structure and make it easy to locate and use them.
* More importantly, it helps improve code reusability.
* Each package in Java has its unique name and organizes its classes and interfaces into a separate namespace, or name group.
* Although interfaces and classes with the same name cannot appear in the same package, they can appear in different packages. This is possible by assigning a separate namespace to each Java package.

**Syntax:-**

Package nameOfpackage;

**Create a package:**

Creating a package is a simple task as follows

* Choose the name of the package
* Include the package command as the first line of code in your Java Source File.
* The Source file contains the classes, interfaces, etc you want to include in the package
* Compile to create the Java packages

**Import classes in packages:**

1. Right-click on the *Default Package* in the Project Manager pane underneath your project and choose *Import*
2. An Import Wizard window will display. Choose File system and select the Next button
3. You are now prompted to choose a file
4. Simply browse your folder with .java files in it
5. Select desired .java files
6. Click on *Finish* to finish the import wizard

**Creating a jar file:**

* A JAR (Java Archive) file is a ZIP format file that bundles Java classes into a single unit.
* There are two types of JAR file in Java:
* *Library JAR (normal JAR) files*: JARs which are reusable libraries.
* *Executable JAR files*: JARs which can be executed as a standalone Java applications. A main class must be specified when creating this kind of JAR file.

**Create a normal JAR file:**

* Right click on project name in *Project Explorer* view, select **Export…** (or select **Export…** from **File** menu):
* In the dialog *Export*, select **Java > JAR file**:
* Click **Next**. In the screen *JAR File Specification*:
* Under section *Select resources to export*, uncheck the project node, and select *src* directory (we can select other directories if needed, but the *src* directory must be selected).
* Under section *Select export destination*, specify path of the JAR file will be exported.
* Click **Next**. In the next screen, we can select an option to save export information so that we can re-create the JAR file later without redoing the previous steps:
* Check the option “*Save the description of this JAR in the workspace*”.
* Click **Browse**button to specify path of the description file.
* Click **Finish**. All class files of the project are exported to a JAR file and a JAR description file is created, under project’s root directory.
* Note that the file ends with *.jardesc* is the JAR description file which stores the information we have specified in the wizard. And we can use it as a shortcut whenever we want to re-export the JAR file.
* Right click on the *.jardesc* file, select **Create JAR**:

**Create an executable JAR file:**

* Creating an executable JAR file is same as creating a normal JAR file, except it requires to specify a main class. In the screen *JAR Packaging Options*, click **Next** will take us to the screen *JAR Manifest Specification*:
* In this screen we can:
* Specify the manifest: Select the option *Generate the manifest file*.
* Seal content of the JAR file: select the option *Seal JAR*.
* Select the main class that acts as an entry point to the application: Click **Browse** button next to the text field *Main Class*. The dialog *Select Main Class* displays all classes which have main method. Select the desired class, then click **OK**:
* Click **Finish**, the executable JAR file is created.

***10.Wrapper classes, Mutable and*** ***Immutable objects.***

*Wrapper classes provide a way to use primitive data types(int,Boolean,char,etc…) as objects.*

***Primitive Data Type Wrapper Class***

*byte Byte*

*short Short*

*int Integer*

*long Long*

*float Float*

*double Double*

*Boolean Boolean*

*char Character*

***Mutable Objects:***

* *The mutable objects are objects whose value can be changed after initialization.*
* *We can change the object's values, such as field and states, after the object is created.*
* *When we made a change in existing mutable objects, no new object will be created;*
* *instead, it will alter the value of the existing object. These object's classes provide methods to make changes in it.*
* *The Getters and Setters ( get() and set() methods ) are available in mutable objects.*
* *The Mutable object may or may not be thread-safe.*

***Example:***

*Public class Example{*

*Private String s;*

*Example(String s) {*

*this.s=s;*

*}*

*Public String getName() {*

*Return s;*

*}*

*Public void setName(String coursename) {*

*this.s =coursename;*

*}*

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

*Example obj = new Example(“javaTpoint”);*

*System.out.println(obj.getName());*

*// Here,we can update the name using the setName method.*

*Obj.setName(“java Training”);*

*System.out.println(obj.getName());*

*}*

*}*

***Immutable objects:***

* *The immutable objects are objects whose value can not be changed after initialization.*
* *We can not change anything once the object is created.*
* *immutable means unmodified or unchangeable.*
* *Once the immutable objects are created, its object values and state can not be changed.*
* *Only Getters ( get() method) are available not Setters ( set() method) for immutable objects.*

***Example:***

*Public class Example1{*

*Private String s;*

*Example1(String s) {*

*this.s=s;*

*}*

*Public String getName() {*

*Return s;*

*}*

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

*Example obj = new Example(“javaTraining”);*

*System.out.println(obj.getName());*

*}*

*}*

***11.String, StringBuffer, StringBuilder,*** ***StringTokenzer.***

***String:***

* *String is a sequence of characters. But in Java, string is an object that represents a sequence of characters.*
* *The ja*
* *va.lang.String class is used to create a string object.*
* *There are two ways to create String object:*
* *By string literal*
* *By new keyword*

***String literal:***

*Java String literal is created by using double quotes.*

***Example:*** *String s=”welcome”;*

* *Each time you create a string literal, the JVM checks the "string constant pool" first.*
* *If the string already exists in the pool, a reference to the pooled instance is returned.*
* *If the string doesn't exist in the pool, a new string instance is created and placed in the pool.*

***Example:***

*String s1=”welcome”;*

*String s2=”welcome”; //It doesn't create a new instance*

***new keyword:***

*String s=new String(“welcome”); //creates two objects and one reference variable*

* *JVM will create a new string object in normal (non-pool) heap memory, and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in a heap (non-pool).*

***Example:***

*public class StringExample{*

*public static void main(String[] args){*

*String s1=”java”; //creating string by java string literal*

*Char ch[]={‘s’,’t’,’r’,’i’,’n’,’g’,’s’};*

*String s2=new String(ch);//converting char array to string*

*String s3=new String(ch); //creating java string by new keyword*

*System.out.println(s1);*

*System.out.println(s2);*

*System.out.println(s3);*

*}}*

***StringBuilder:***

* *Java StringBuilder class is used to create mutable (modifiable) string.*
* *The Java StringBuilder class is same as StringBuffer class except that it is non-synchronized.*
* *Different type of methods are used in StringBuilder class*

***Examples:***

***1) StringBuilder append() method***

* *The StringBuilder append() method concatenates the given argument with this Strings.*

***Example:***

*class StringBuilderExample{*

*public static void main(String args[]){*

*StringBuilder sb=new StringBuilder("Hello ");*

*sb.append("Java"); //now original string is changed*

*System.out.println(sb); //prints Hello Java*

*}*

*}*

***2) StringBuilder insert() method***

* *The StringBuilder insert() method inserts the given string with this string at the given position.*

***Example:***

*class StringBuilderExample2{*

*public static void main(String args[]){*

*StringBuilder sb=new StringBuilder("Hello ");*

*sb.insert(1,"Java"); //now original string is changed*

*System.out.println(sb);//prints HJavaello*

*}*

*}*

***3) StringBuilder replace() method***

* *The StringBuilder replace() method replaces the given string from the specified beginIndex and endIndex.*

***Example:***

*class StringBuilderExample3{*

*public static void main(String args[]){*

*StringBuilder sb=new StringBuilder("Hello");*

*sb.replace(1,3,"Java");*

*System.out.println(sb);//prints HJavalo*

*}*

*}*

***4) StringBuilder delete() method***

* *The delete() method of StringBuilder class deletes the string from the specified beginIndex to endIndex.*

***Example:***

*class StringBuilderExample4{*

*public static void main(String args[]){*

*StringBuilder sb=new StringBuilder("Hello");*

*sb.delete(1,3);*

*System.out.println(sb); //prints Hlo*

*}*

*}*

***5) StringBuilder reverse() method***

* *The reverse() method of StringBuilder class reverses the current string.*

***Example:***

*class StringBuilderExample5{*

*public static void main(String args[]){*

*StringBuilder sb=new StringBuilder("Hello");*

*sb.reverse();*

*System.out.println(sb); //prints olleH*

*}*

*}*

***6) StringBuilder capacity() method***

* *The capacity() method of StringBuilder class returns the current capacity of the Builder. The default capacity of the Builder is 16. If the number of character increases from its current capacity, it increases the capacity by (oldcapacity\*2)+2. For example if your current capacity is 16, it will be (16\*2)+2=34.*

***Example:***

*class StringBuilderExample6{*

*public static void main(String args[]){*

*StringBuilder sb=new StringBuilder();*

*System.out.println(sb.capacity()); //default 16*

*sb.append("Hello");*

*System.out.println(sb.capacity()); //now 16*

*sb.append("java is my favourite language");*

*System.out.println(sb.capacity()); //now (16\*2)+2=34 i.e (oldcapacity\*2)+2*

*}*

*}*

***7) StringBuilder ensureCapacity() method***

* *The ensureCapacity() method of StringBuilder class ensures that the given capacity is the minimum to the current capacity. If it is greater than the current capacity, it increases the capacity by (oldcapacity\*2)+2. For example if your current capacity is 16, it will be (16\*2)+2=34.*

***Example:***

*class StringBuilderExample7{*

*public static void main(String args[]){*

*StringBuilder sb=new StringBuilder();*

*System.out.println(sb.capacity()); //default 16*

*sb.append("Hello");*

*System.out.println(sb.capacity()); //now 16*

*sb.append("java is my favourite language");*

*System.out.println(sb.capacity()); //now (16\*2)+2=34 i.e (oldcapacity\*2)+2*

*sb.ensureCapacity(10); //now no change*

*System.out.println(sb.capacity()); //now 34*

*sb.ensureCapacity(50); //now (34\*2)+2*

*System.out.println(sb.capacity()); //now 70*

*}*

*}*

***StringBuffer:***

* *StringBuffer is*synchronized*i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously.*
* *StringBuffer is*less efficient*than StringBuilder.*

***StringTokenzer:***

* *The java.util.StringTokenizer class allows you to break a string into tokens. It is simple way to break string.*
* *It doesn't provide the facility to differentiate numbers, quoted strings, identifiers etc. like StreamTokenizer class. We will discuss about the StreamTokenizer class in I/O chapter.*

***Examples:***

*import java.util.StringTokenizer;*

*public class Example{*

*public static void main(String args[]){*

*StringTokenizer st = new StringTokenizer("my name is khan," ") ;*

*while (st.hasMoreTokens()) {*

*System.out.println(st.nextToken());*

*}*

*}*

*}*

***Example of nextToken(String delim) method of StringTokenizer class***

*import java.util.\*;*

*public class Test {*

*public static void main(String[] args) {*

*StringTokenizer st = new StringTokenizer("my,name,is,khan");*

*// printing next token*

*System.out.println("Next token is : " + st.nextToken(","));*

*}*

*}*

***12.Arrays,*** ***Single Dim, Two Dim, Multi Dim Arrays.***

* ***Java array****is an object which contains elements of a similar data type.*
* *Additionally, The elements of an array are stored in a connecting 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.*
* *Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.*
* *In Java, array is an object of a dynamically generated class. Java array inherits the Object class, and implements the Serializable as well as Cloneable interfaces.*
* *We can store primitive values or objects in an array in Java.*

***Single Dimensional array:***

* *It is a collection of variables of same type which is used by a common name.*

***Example:***

*import java.io.\*;*

*class Example {*

*public static void main(String[] args)*

*{*

*int[] a; // valid declaration*

*int b[]; // valid declaration*

*int[] c; // valid declaration*

*}*

*}*

***Two Dimensional Array:***

* *Suppose, you want to create two dimensional array of int type data. So you can declare two dimensional array in many of the following ways:*

***Example:***

*import java.io.\*;*

*class Example{*

*public static void main(String[] args)*

*{*

*int a[][]; // valid*

*int[][] b; // valid*

*int[][] c; // valid*

*int[] d[]; // valid*

*int[][] e; // valid*

*int[] f[]; // valid*

*[][] int g; // invalid*

*[] int[] h; // invalid*

*}*

*}*

***multi Dimensional Array:***

*import java.io.\*;*

*class Example{*

*public static void main(String[] args)*

*{*

*// Here, 'a' is two dimensional array, 'b'*

*// is two dimensional array*

*int[] a[], b[];*

*// Here, 'c' is two dimensional array, 'd'*

*// is two dimensional array*

*int[] c[], d[];*

*// Here, 'e' is two dimensional array, 'f'*

*// is three dimensional array*

*int[][] e, f[];*

*// Here, 'g' is two dimensional array,*

*// 'h' is one dimensional array*

*int[] g[], h;*

*}*

*}*

***13.for each loop, and new changes with switch statements.***

* *the for-each loop is used to iterate through elements of*[*arrays*](https://www.programiz.com/java-programming/arrays)*and collections (like*[*ArrayList*](https://www.programiz.com/java-programming/arraylist)*). It is also known as the enhanced for loop.*

***for-each Loop Sytnax***

*The syntax of the Java for-each loop is:*

*For(datatype item : array) {*

*}*

*Here,*

* *array - an array or a collection*
* *item - each item of array/collection is assigned to this variable*
* *dataType - the data type of the array/collection*

***Example:***

*//calculate the sum of all elements of an array*

*Class Main {*

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

*//an array of numbers*

*Int [ ] numbers={3,4,5,-5,0,12};*

*Int sum=0;*

*// iterating through each element of the array*

*For(int number: numbers) {*

*Sum += number;*

*}*

*System.out.println(“sum =” +sum);*

*}*

***Switch Statements:***

* *The switch statement allows us to execute a block of code among many alternatives.*

***Example:***

*Class Main{*

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

*Int number =49;*

*String size;*

*//switch statement to check size*

*Switch(number) {*

*Case 29:*

*Size=”small” ;*

*Break;*

*Case 42:*

*Size=”medium”;*

*Break;*

*//match the value of week*

*case 44:*

*size=”large” ;*

*break;*

*case 28:*

*size=”Extra Large”;*

*break;*

*default:*

*size=”unknown”;*

*break;*

*}*

*System.out.println(“size: “ +size);*

*}*

*}*

***14.Var arg***

* *The varrags allows the method to accept zero or multiple arguments.*
* *Before varargs either we use overloaded method or take an array as the method parameter but it was not considered good because it leads to the maintenance problem.*
* *If we don't know how many argument we will have to pass in the method, varargs is the better approach.*

***Syntax:***

*return\_type method\_name(data\_type... variableName){}*

***Example:***

*class VarargsExample{*

*static void display(int num, String... values){*

*System.out.println("number is "+num);*

*for(String s:values){*

*System.out.println(s);*

*}*

*}*

*public static void main(String args[]){*

*display(500,"hello"); //one argument*

*display(1000,"my","name","is","varargs"); //four arguments*

*}*

*}*