**JAVA**

* Java is platform independent,portable,dynamic
* Object oriented programming- Programmes done using the objects
* **Objects**- An object can be defined as an instance of a class, and there can be multiple instances of a class in a program. An Object is one of the Java OOPs concepts which contains both the data and the function.
* **Class**- objects are created using class. The class is one of the Basic concepts of OOPs which is a group of similar entities. It is only a logical component and not the physical entity.
* Class holds the object
* **Constructor**-create an object
* **Access modifiers**-Define visibility of class
* 4 types-**private,public,protected,default**
* **Data types**- integer,float,char,string,double,byte etc…
* **4 principles of oop**- Encapsulation,Polymorphism,Inheritance,Abstraction
* **Encapsulation**-Hiding data from outside world, Encapsulation is one of the best Java OOPs concepts of wrapping the data under a single code. In this OOPs concept, the variables of a class are always hidden from other classes.
* **Polymorphism**- Polymorphism refers to one of the OOPs concepts in Java which is the ability of a variable, object or function to take on multiple forms. 2 methods-Method overloading &Method over riding
* **Inheritance-** Ability to acquire some or all properties from the parent object.
* **Abstraction-** Process of selecting a data from a larger pool to show only relevant details. Abstraction is one of the OOP Concepts in Java which is an act of representing essential features without including background details.
* **Data types**-primitive and non primitive
* **Variable type**- local variable, instance variable,static variable
* **Local Variables** - are a variable that are declared inside the body of a method
* **Instance Variables** -are defined without the STATIC keyword.
* **Static Variables** -are initialized only once, at the start of the program execution. These variables should be initialized first, before the initialization of any instance variables

**ARRAY**

* Can hold the elements of same data type
* Give the size of the array while declaring
* Array index start from zero
* Int a[]={1,2,3}

**PRE AND POST INCREMENT**

* ++i returns the value after it is incremented, while i++ return the value before it is incremented

**CONSTRUCTOR**

* A constructor in Java is a special method that is used to initialize objects. The constructor is called when an object of a class is created. It can be used to set initial values for object attributes.
* Default constructor- zero argument constructor
* If a programmer is creating a constructor then we won’t have zero argument constructor

Eg:

Main{ java}

Main(string langage){cprogram}

// call constructor with no parameter

Main obj1 = new Main();

// call constructor with a single parameter

Main obj2 = new Main("Python");

Obj1.getname(); --- gives output java

Obj2.getname();----gives output python

**LOOPS**

* For(int i=0;i<5;i++){

s.o.p(a[i]);

}

* Int i=0;

While(condition ex:i<5)

{

I++;

}easo

* Do{

}

While(condition check)

* **Size of data types-**Byte=1byte, short=2bytes,int= 4 bytes, long=8 bytes, float=4 bytes, double= 8 bytes, char= 2 bytes, Boolean-1 bit.
* **SWITCH statements**- It is a multi way branch statement, Default statement is not mandatory in switch statement, one or more switch statement can be added in a single statement.
* **THIS statements**- key word refers to the current object in a method or a constructor, it is a reference variable points to the current objects
* **STATIC**- When a variable is declared as static, and then a single copy of the variable is created and shared among all objects at a class level. Static variables are, essentially, global variables. All instances of the class share the same static variable.
* Variable,classes,blocks,methods can be made static
* **SUPER-** It is a reference variable which can point the super/parent class object, A super class won’t fit to subclass, A sub class fit to super class but shows the features of super class.
* **Method over loading-**Used in polymorphism, consider only the type &number, multiple methods can have the same name with different parameters
* **Method over riding-**used in inheritance, parent class method is also implemented in its child class
* **Abstraction-** Hiding the implementation to the outside world
* Key word-abstract, not applied on variables, only applied to class & method
* **Interfaces-**100%abstract, can hold variables(public static final)
* Class A extends B not B,C- only one class extends is possible-for **inheritance**
* Class A implements B,C-for **Interface**
* **EXCEPTION**
* An exception is a problem that arises during the execution of a program. An exception can occur for many reasons,

1. A user has entered invalid data.
2. A file that needs to be opened cannot be found.
3. A network connection has been lost in the middle of communications or the JVM has run out of memory.

* Three categories of exception,

1. **Checked exceptions**: A checked exception is an exception that is typically a user error or a problem that cannot be foreseen by the programmer. For example, if a file is to be opened, but the file cannot be found, an exception occurs. These exceptions cannot simply be ignored at the time of compilation.
2. **Runtime exceptions**: A runtime exception is an exception that occurs that probably could have been avoided by the programmer. As opposed to checked exceptions, runtime exceptions are ignored at the time of compilation.
3. **Errors**: These are not exceptions at all, but problems that arise beyond the control of the user or the programmer. Errors are typically ignored in your code because you can rarely do anything about an error. For example, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation.

* Java’s built in exceptions are,

ArratIndexOutOfBoundException,FileNotFoundException,ArithmeticException,NumberFormatException,NullPointerException,ClassCastException etc…

* To solve this Try,catch,finally,throws,nested try,multiple catch is used
* **File I/O**
* Create a file, delete a file,can restrict access to file, make a copy of the file
* Stream-flow of data/series of data
* 2 types-byte stream & character stream
* various classes- InputReader,OutputReader, BufferedReader, BufferedWriter
* **BufferedReader-** It uses a reader to read data from the character input stream & creates a default size input buffer
* **InputStreamReader-** Reads only one character from the input stream & the remaining characters still remain in the streams hence there is no buffer in this case.
* **BufferedWriter-** is a sub class of java.io.Write class.BufferedWriter writes text to character output stream,buffering characters so as to provide for the efficient writing of a single character,arrays &strings
* **FileWriter-** Class is used to write character oriented data to a file. It writes directly in to files and should be used only when the number of writes is less.
* **Collections**
* Collecions is a class in java
* It can contain heterogenous elements and can modify the size dynamically
* Collection is an interference
* List –aarayList,LinkedList
* Map-HashMap
* Set-TreeHashSet
* Queue –PriorityQueue
* Int,float,double-data types
* **Wrapper classes**
* Integer ,float, double,long,character,Boolean are used
* The wrapper class implements the technique to convert the primitive into object and object into primitive.
* **Iterator class**
* Iterator bilt in class is for printing the values of any collection type
* In general, to use an iterator to cycle through the contents of a collection, follow these steps:
* Obtain an iterator to the start of the collection by calling the collection's iterator( ) method.

• Set up a loop that makes a call to hasNext( ). Have the loop iterate as long as hasNext( ) returns true.

• Within the loop, obtain each element by calling next( )

* **Hashtable**
* The Hashtable class provides a means of organizing data based on some user-defined key structure.
* Java 2 reengineered Hashtable so that it also implements the Map interface. Thus, Hashtable is now integrated into the collections framework. It is similar to HashMap, but is synchronized.
* Like HashMap, Hashtable stores key/value pairs in a hashtable
* Hashtable(),Hashtable(int size), Hashtable(int size,float fillRatio), Hashtable(Map m)
* **Generics**
* Java Generic methods and generic classes enable programmers to specify, with a single method declaration, a set of related methods or, with a single class declaration, a set of related types, respectively.
* <E>-representation
* **Threads**
* A thread is the path followed when executing a program.
* All java pgrm have at least one thread known as main thread.
* Main thread is created by the java virtual machine(JVM)
* Thread allows a pgrm to operate more efficiently by doing multiple things at the same time.
* **Inner classes**
* Java inner class is a class which is declared inside the class or interface
* It can access all the members of the outer class,including private data members & methods.
* We know a class cannot be associated with the access modifier private, but if we have class as a member of other class, then the inner class can be made private& can also used to access the private members of class.
* 4 types of inner class- Nested inner class,method local inner class, Anonymous inner class, Static nested classes.

Annotations for Junit testing:

**1. @Test:** It is used to specify the test method.

**2. @BeforeClass:** It is used to specify that method will be called only once, before starting all the test cases.

**3. @AfterClass:** It is used to specify that method will be called only once, after finishing all the test cases.

**4. @Before:** It is used to specify that method will be called before each test case.

**5. @After:** It is used to specify that method will be called after each test case.

**6. @Ignore:** It is used to ignore the test case.

In Junit, test suite allows us to aggregate all test cases from multiple classes in one place and run it together.

To run the suite test, you need to annotate a class using below-mentioned annotations:

1. @Runwith(Suite.class)
2. @SuiteClasses(test1.class,test2.class……) or

@Suite.SuiteClasses ({test1.class, test2.class……})

With above annotations, all the test classes in the suite will start