*Java*

* **Object -** . An object is an instance of a class. Object has variable and methods.Example: A dog has variable-color, name, breed method -wagging, barking, eating. An object is an instance of a class.
* **Class -** A class can be defined as a template/blue print that describes that object .
* **Methods –** block of code that performs some actions.
* **Instance Variables -** Each object has its unique set of instance variables. An object's state is created by the values assigned to these instance variables.

Oops concept:

* [Inheritance](https://www.javatpoint.com/inheritance-in-java): child class acquires all the properties and behaviors of a parent class
* [Polymorphism](https://www.javatpoint.com/runtime-polymorphism-in-java) :one task is performed in different ways
* [Abstraction](https://www.javatpoint.com/abstract-class-in-java): Hiding the implementation
* [Encapsulation](https://www.javatpoint.com/encapsulation): wrapping data and method into a single unit

Data type

* Primitive data types: boolean, char, byte, short, int, long, float and double.
* **Non-primitive data types:**[Classes](https://www.javatpoint.com/object-and-class-in-java), [Interfaces](https://www.javatpoint.com/interface-in-java), and [Arrays](https://www.javatpoint.com/array-in-java).

Variables

container which holds the value while the [Java program](https://www.javatpoint.com/simple-program-of-java) is executed

* local variable.
* instance variable.
* static variable.

Type casting

* **Widening Casting** (automatically) - converting a smaller type to a larger type size
* **Narrowing Casting** (manually) - converting a larger type to a smaller size type

Constructor

* 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.
* 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
* Type : Default constructor (no-arg constructor),Parameterized constructor

decision making statements

if Statement:

* An if statement consists of a Boolean expression followed by one or more statements.

Syntax:

if(Boolean\_expression) {

//Statements will execute if the Boolean expression is true

}

if...else Statement:

An if statement can be followed by an optional *else* statement, which executes when the Boolean expression is false.

Syntax:

if(Boolean\_expression){

//Executes when the Boolean expression is true }

else{

//Executes when the Boolean expression is false }

if...else if...else Statement:

An if statement can be followed by an optional *else if...else* statement, which is very useful to test various conditions using single if...else if statement.

Syntax:

The syntax of an if...else is: if(Boolean\_expression1){

//Executes when the Boolean expression 1 is true }

elseif(Boolean\_expression2){

//Executes when the Boolean expression 2 is true

}elseif(Boolean\_expression3){

//Executes when the Boolean expression 3 is true }

else {

//Executes when the none of the above condition is true. }

Nested if...else Statement:

use one if or else if statement inside another if or else if statement.

Syntax:

if(Boolean\_expression1){

//Executes when the Boolean expression 1 is true

if(Boolean\_expression2){

//Executes when the Boolean expression 2 is true }

}

Switch

* Multiple if-else statement.
* It is a keyword and we have conditions in it.
* Value of a case should be a constant.
* There can’t be any duplicate case value.
* We can put break statement in between case, if break statement is not there it continues the execution.
* Condition inside the switch can be of any primitive type/enums/Strings.
* It is not compulsory to have default statement in a switch.

Syntax:

switch(expression){

case value :

//Statements

break;//optional

case value : //Statements

break;//optional

//You can have any number of case statements.

default://Optional

//Statements

}

Looping statements in Java

Used to iterate a part of the program several times

For loop

* iterate a part of the program several times
* if the number of iteration is fixed it is recommended to use for loop
* simple for loop, nested for loop, for-each loop, labelled for loop, infinite for loop.

SYNTAX

for (initialization; condition; increment or decrement)

{

// code to be executed

}

Enhanced for loop in Java:

* This is mainly used for Arrays.

Syntax:

for(declaration : expression) {

//Statements

}

While loop

* it is control flow statement that executes the prgm repeatedly based on given Boolean statement
* used when the number of iteration is not fixed.

SYNTAX

while (condition)

{

// code to be executed

}

do- while loop

* it is control flow statement that executes the prgm atleast once and further execution is based on given Boolean statement.
* Used when the number of iteration is not fixed and we have to execute the program atleast once

SYNTAX

do

{

//code to be executed

}

while(condition);

Block

* Set of code that is within the curly braces{} within any class, method or constructor.

Syntax

{

//block of code

}

break Keyword:

* breakkeyword is used to stop the entire loop.
* The break keyword must be used inside any loop or a switch statement
* The break keyword will stop the execution of the innermost loop and start executing the next line of code after the block.

Syntax:

break;

continue Keyword:

* It causes the loop to immediately jump to the next iteration of the loop.
* In a for loop, the continue keyword causes flow of control to immediately jump to the update statement.
* In a while loop or do/while loop, flow of control immediately jumps to the Boolean expression.

Syntax:

continue;

Array

* Can hold elements of same datatype
* Elements are stored in contiguous memory location.
* Arrays work with respect to Index
* Start with index 0
* Size limit - We can store only fixed size of elements in an array
* Random access- can get any data located at an index position.
* There are 1d as well as multidimensional arrays.

SYNTAX

Datatype[] arrayname =new datatype(size);

Static

* Not an instance variable
* It’s a class variable or global variable
* Used to save the memory
* We can apply static to variables, classes, methods and blocks.

This

* it is a reference variable which points to the current object.
* this can be used on instance variable, passed as an argument, and with the constructor also.
* If local variables and instance variables are same then we can use “this” to distinguish btwn them.
* Invoke current class constructor
* Invoke current class method
* Return the current class object
* Pass an argument in the method call
* Pass an argument in the constructor call

Super

* It is a reference variable which points to the parent class object
* It can be used with methods, constructors and variables

Method Overloading

* Return type doesn’t matter
* If a class has multiple methods having same name but diff parameters or different datatypes then it is known as Method Overloading
* 2 types - By changing the datatype of argument

By changing the arguments

* Not possible by changing the return type of method only.

Method Overriding

* subclass has the same method as declared in the parent class.
* runtime polymorphism
* method must have the same name and same arguments as in the parent class.

Access modifiers

* specifies the accessibility or scope.
* Public : everywhere.
* Private : only within the class.
* Default : only within the package.
* Protected : within the package and outside the package through child class.

Inheritance

* mechanism in which one object acquires all the properties and behaviors of a parent object.
* It is an important part of OOPs (Object Oriented programming system).
* Code Reusability.
* Syntax

class Subclass-name extends Superclass-name

{

//methods and fields

}

* Types of inheritance:
* Single Inheritance: subclasses inherit the features of one superclass.
* Multilevel Inheritance: a derived class will be inheriting a base class and as well as the derived class also act as the base class to other class.
* Hierarchical Inheritance: one class serves as a superclass (base class) for more than one subclass.
* Multiple Inheritance (Through Interfaces): one class can have more than one superclass and inherit features from all parent classes. Please note that Java does not support multiple inheritances with classes. we can achieve multiple inheritances only through Interfaces.
* Hybrid Inheritance(Through Interfaces): It is a mix of two or more of the above types of inheritance. hybrid inheritance is also not possible with classes. we can achieve hybrid inheritance only through Interfaces.

Exception Handling:

* exception is an unwanted or unexpected event, which occurs during the execution of a program .
* i.e run time, that disrupts the normal flow of the program’s instructions.
* Error: An Error indicates serious problem that a reasonable application should not try to catch.
* Exception: Exception indicates conditions that a reasonable application might try to catch.
* Exception Handling in Java is one of the powerful mechanism to handle the runtime errors so that the normal flow of the application can be maintained.
* try-catch block :
* try block is used to enclose the code that might throw an exception. It must be used within the method.
* catch block is used to handle the Exception by declaring the type of exception within the parameter. The declared exception must be the parent class exception ( i.e., Exception) or the generated exception type.
* Syntax:

try{

//code that may throw an exception

}catch(Exception\_class\_Name ref){}

* Multi-catch block:
* try block can be followed by one or more catch blocks. Each catch block must contain a different exception handler.
* if you have to perform different tasks at the occurrence of different exceptions, use java multi-catch block.
* At a time only one exception occurs and at a time only one catch block is executed.
* All catch blocks must be ordered from most specific to most general, i.e. catch for ArithmeticException must come before catch for Exception.
* Nested try block:
* using a try block inside another try block is permitted. It is called as nested try block. Every statement that we enter a statement in try block, context of that exception is pushed onto the stack.
* Syntax:

//main try block

try

{

statement 1;

statement 2;

//try catch block within another try block

try

{

statement 3;

statement 4;

//try catch block within nested try block

}

catch(Exception e1)

{

//exception message

}

}

//catch block of parent (outer) try block

catch(Exception e3)

{

//exception message

}

....

* finally block :
* block used to execute important code such as closing the connection, etc.
* Java finally block is always executed whether an exception is handled or not.
* The finally block follows the try-catch block.
* throw keyword is used to throw an exception explicitly.
* Custom Exception:
* create our own exceptions that are derived classes of the Exception class.
* Creating our own Exception is known as custom exception or user-defined exception.
* Java custom exceptions are used to customize the exception according to user need.
* Syntax:

public class WrongFileNameException extends Exception {

public WrongFileNameException(String errorMessage) {

super(errorMessage);

}

}

Collection:

* framework that provides an architecture to store and manipulate the group of objects.
* Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.
* Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).
* Iterator interface:
* Iterator interface provides the facility of iterating the elements in a forward direction only.

Iterator<T> iterator()

* Collection interface :

which is implemented by all the classes in the collection framework. It declares the methods that every collection will have

* List interface :
* child interface of Collection interface.
* It inhibits a list type data structure in which we can store the ordered collection of objects.
* It can have duplicate values.
* List interface is implemented by the classes ArrayList, LinkedList, Vector, and Stack.
* Syntax:

List <T> al = new ArrayList<> ();

List <T> ll = new LinkedList<> ();

List <T> v = new Vector<> ();

Where T is the type of the object

* ArrayList: ArrayList provides us with dynamic arrays in Java. Though, it may be slower than standard arrays but can be helpful in programs where lots of manipulation in the array is needed. The size of an ArrayList is increased automatically if the collection grows or shrinks if the objects are removed from the collection. Java ArrayList allows us to randomly access the list.
* LinkedList: LinkedList class is an implementation of the LinkedList data structure which is a linear data structure where the elements are not stored in contiguous locations and every element is a separate object with a data part and address part. The elements are linked using pointers and addresses. Each element is known as a node.
* Vector: A vector provides us with dynamic arrays in Java. Though, it may be slower than standard arrays but can be helpful in programs where lots of manipulation in the array is needed. This is identical to ArrayList in terms of implementation. However, the primary difference between a vector and an ArrayList is that a Vector is synchronized and an ArrayList is non-synchronized.
* Stack: Stack class models and implements the Stack data structure. The class is based on the basic principle of last-in-first-out. In addition to the basic push and pop operations, the class provides three more functions of empty, search and peek. The class can also be referred to as the subclass of Vector.
* Priority Queue: A PriorityQueue is used when the objects are supposed to be processed based on the priority. It is known that a queue follows the First-In-First-Out algorithm, but sometimes the elements of the queue are needed to be processed according to the priority and this class is used in these cases.
* . Set Interface: A set is an unordered collection of objects in which duplicate values cannot be stored. This collection is used when we wish to avoid the duplication of the objects and wish to store only the unique objects. This set interface is implemented by various classes like HashSet, TreeSet, LinkedHashSet, etc.

Set<T> hs = new HashSet<> ();

Set<T> lhs = new LinkedHashSet<> ();

Set<T> ts = new TreeSet<> ();

Where T is the type of the object.

* Map Interface: A map is a data structure which supports the key-value pair mapping for the data. This interface doesn’t support duplicate keys because the same key cannot have multiple mappings. A map is useful if there is a data and we wish to perform operations on the basis of the key. This map interface is implemented by various classes like HashMap, TreeMap etc.

File Handling

A File is an abstract data type. A named location used to store related information is known as A File. There are several File Operations like creating a new File, getting information about File, writing into a File, reading from a File and deleting a File.

Stream

A series of data is referred to as a stream. In Java, Stream is classified into two types,

Byte Stream: Is mainly involved with byte data. A file handling process with a byte stream is a process in which an input is provided and executed with the byte data.

Character Stream: Is mainly involved with character data. A file handling process with a character stream is a process in which an input is provided and executed with the character data.

File Operations

We can perform the following operation on a file:

* Create a File: Create a File operation is performed to create a new file. We use the createNewFile() method of file. The createNewFile() method returns true when it successfully creates a new file and returns false when the file already exists.
* Get File Information: The operation is performed to get the file information. We use several methods to get the information about the file like name, absolute path, is readable, is writable and length.
* Write to a File: The next operation which we can perform on a file is "writing into a file". In order to write data into a file, we will use the FileWriter class and its write() method together. We need to close the stream using the close() method to retrieve the allocated resources.
* Read from a File: The next operation which we can perform on a file is "read from a file". In order to write data into a file, we will use the Scanner class. Here, we need to close the stream using the close() method. We will create an instance of the Scanner class and use the hasNextLine() method nextLine() method to get data from the file.
* Delete a File: The next operation which we can perform on a file is "deleting a file". In order to delete a file, we will use the delete() method of the file. We don't need to close the stream using the close() method because for deleting a file, we neither use the FileWriter class nor the Scanner class.

**Generics**

* Generics mean parameterized types. The idea is to allow type (Integer, String, … etc, and user-defined types) to be a parameter to methods, classes, and interfaces. Using Generics, it is possible to create classes that work with different data types.
* An entity such as class, interface, or method that operates on a parameterized type is called a generic entity.
* There are mainly 3 advantages of generics.
  + Type-safety: We can hold only a single type of objects in generics.
  + Type casting is not required: There is no need to typecast the object.
  + Type casting is not required: There is no need to typecast the object.
* Generic class: A class that can refer to any type is known as a generic class. Here, we are using the T type parameter to create the generic class of specific type.

Creating a generic class:

class MyGen<T>

{

T obj;

void add(T obj){this.obj=obj;

}

T get()

{

return obj;

}

}

* Type Parameters:The type parameters naming conventions are important to learn generics thoroughly.
* T - Type
  + E - Element
  + K - Key
  + N - Number
  + V - Value
* Generic Method:We can create a generic method that can accept any type of arguments. Here, the scope of arguments is limited to the method where it is declared. It allows static as well as non-static methods.

Threads

Threads allows a program to operate more efficiently by doing multiple things at the same time. Threads can be used to perform complicated tasks in the background without interrupting the main program. There are two ways to create a thread. It can be created by extending the Thread class and overriding its run() method:

Thread Model

* New (Ready to run): A thread is in New when it gets CPU time.
* Running: A thread is in a Running state when it is under execution.
* Suspended : A thread is in the Suspended state when it is temporarily inactive or under execution.
* Blocked : A thread is in the Blocked state when it is waiting for resources.
* Terminated: A thread comes in this state when at any given time, it halts its execution immediately.

Creating Thread

A thread is created either by "creating or implementing" the Runnable Interface or by extending the Thread class. These are the only two ways through which we can create a thread.

Thread Class

A Thread class has several methods and constructors which allow us to perform various operations on a thread. The Thread class extends the Object class. The Object class implements the Runnable interface. The thread class has the following constructors that are used to perform various operations.

Thread()

Thread(Runnable, String name)

Thread (Runnable target)

Thread (ThreadGroup group, Runnable target, String name)

Thread (ThreadGroup group, Runnable target)

Thread (ThreadGroup group, String name)

Thread (ThreadGroup group, Runnable target, String name, long stackSize)

Runnable Interface(run() method)

The Runnable interface is required to be implemented by that class whose instances are intended to be executed by a thread. The runnable interface gives us the run() method to perform an action for the thread.

start() method

The method is used for starting a thread that we have newly created. It starts a new thread with a new callstack. After executing the start() method, the thread changes the state from New to Runnable. It executes the run() method when the thread gets the correct time to execute it.

Creating thread by implementing the runnable interface

In Java, we can also create a thread by implementing the runnable interface. The runnable interface provides us both the run() method and the start() method.

**Inner class**

Inner class means one class which is a member of another class. There are basically four types of inner classes in java.

* + Nested Inner class: can access any private instance variable of outer class. Like any other instance variable, we can have access modifier private, protected, public and default modifier.
  + Method Local inner classes: Inner class can be declared within a method of an outer class.
  + Anonymous inner classes: Anonymous inner classes are declared without any name at all.
  + Static nested classes: Static nested classes are not technically an inner class. They are like a static member of outer class.

**Garbage Collection**

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