**CORE JAVA**

1. **Creating hello java example**

Let's create the hello java program:

1. **class** Simple{
2. **public** **static** **void** main(String args[]){
3. System.out.println("Hello Java");
4. }
5. }

Understanding first java program

* **class** keyword is used to declare a class in java.
* **static** is a keyword, if we declare any method as static, it is known as static method. The core advantage of static method is that there is no need to create object to invoke the static method. The main method is executed by the JVM, so it doesn't require creating object to invoke the main method. So it saves memory.
* **void** is the return type of the method, it means it doesn't return any value.
* **main** represents startup of the program.
* **String[] args** is used for command line argument. We will learn it later.
* **System.out.println()** is used print statement. We will learn about the internal working of System.out.println statement later.
* **Private:** Limited access to class only
* **Default (no modifier):** Limited access to class and package
* **Protected:** Limited access to class, package and subclasses (both inside and outside package)
* **Public:** Accessible to class, package (all), and subclasses… In short, everywhere.
* **JAVAP:** When we compile Java file it convert to Class file(bite code). JavaP used for convert class file to java file, it gives just structure.

1. **Variable:**

* It is a name of memory location to store values.
* We use Variable to store data
* Local Variables contain garbage values.
* Global variables contain null values/default values.

**Global Variables:**  We declare global variables outside methods including main methods. We can use global variables in any method of that particular Class. But we have to create “Object” for that particular global variable which one we are calling in methods.

**Local Variable:** We declare local variables inside methods only. We can use these variables in only that particular method in which one we created. We cannot use in other or main method also.

**Type Casting:**

When we need to convert a data type variable to other data type variable, we use “Type casting”.

EX: **int** k = (**int**)5.6; //type casting ; automatically it remove .6 and prints 5

**Implicit Conversion:**

When we would like to assign into value to decimal, we use this

**Ex:double** d1 = 5; //implicit conversation; it automatically takes 5.0

1. **Data Type:**

**i) Primitive**

Boolean, char, byte, short, int, long, float, double..

**ii)Non-Primitive**

* These stores only address of an “Object”.

Strings

Array

Classes

Interfaces

1. **Class:**

* It is an entity which binds data member and member methods into single unit.
* Class is a best example of encapsulation
* A class is a blue print or a template for creating different objects which defines its properties and behaviours.

**Wrapper Class:**

* Wehave Wrapper Class for all Primitive data types (Integer, Float, Double, Byte etc..);
  + EX: Integer obj = new Integer(5);
* **Wrapping(Boxing):**
  + Int I = 5;
  + EX: Integer obj = new Integer(i); or Integer obj = I; (Auto Boxing)
* **UnWrapping(Un Boxing):**
  + EX: Integer obj = new Integer(5)
  + Int j = obj.intValue(); // Int j = obj;
* When we work with Frameworks (Ex: Collection Framework) Wrapper Class more useful.

1. **Methods:**

* Method is just a block of java statement that performs a particular task.
* We can reuse the code and avoid duplicity.

Method can be called using (.) dot operator.

Access-specifier static/non static return type method name ()

{

}

1-public void marks ()

{

}

2- Public static void main marks ()

}

1. **Objects:**

* It’s an instance of Class.
* An entity that has and behaviour is known as an object.
* Cloning of Class.
* A single Class can Creates any number of unique objects.

**We should define object by “New”**

**What object knows is Variable**

**What object does is Method**

ClassName objectname= new classname ();

* For non-static method can call through object using dot (.)
* For static method can call using class name using dot (.)

**How to Pass Object in Method as Parameter:**

* In Java everything “Call by Value”

**Anonymous Object:**

* + When we don’t have any name for object
    - A obj = new A() -> obj.show(); Refernce object
    - New A().show() 🡪 Anonymous object
  + When we are not using object again and again, try to use Anonymous object
  + So it will save some memory in Stack, it won’t assign any space.

1. **VarArgs:**

* To avoid methods again and again depends on input arguments.
* Have to give parameters as (int … variablename)
* All parameters should be same data type
* If we have a method with same signature, it will take that first, not VarArgs method.

1. **Static and Non static :**
   * **Static:** static is a non-access modifier in Java which is applicable for the following:
2. blocks
3. variables
4. methods
5. nested classes

* To create a static member (block, variable, method, nested class), precede its declaration with the keyword static.
* When a member is declared static, it can be accessed before any objects of its class are created, and without reference to any object.
* For example, in below java program, we are accessing static method m1() without creating any object of Test class.
* When you mention “static” for method or variable is convers as a static method or variable.
* Static variables are owned by class rather than by its individual instances (objects). Referring static variables outside the class is by **ClassName.myStaticVariable** but inside the class it is similar to other instance variables.
* You can always use static variables in non-static methods but you cannot use non-static variables in static methods reason being when static methods are loaded other non-static instance variables are not created.
* Static Variable never depends on objects, so we can use in all methods of that class.
* So your statement id = ++numberOfBicycles; is perfectly valid and will compile without errors.
* Static block execute first then objects, and executes only 1 for class. And it's values same for all.

Public class addng

{

String name = “BPR”

Static int age = “25”

Public void sum(){}

Public static void mul(){}

Public static void main(){}

}

We can call static method / variable by:

* **Direct calling:**

mul(); // method

s.o.p (age) // variable

* **By class Name:**

Addng . mul(); // method

s.o.p(Adding.age).

1. **String:**

* It is a Class in java not a data type.

String s = new String ();

* String Concatenation is a way to combine two or more strings into a single string by ‘+’ operator.

Ex:

String s1= “iam”;

String s2= “Prathap”

String s3 =s1+s2;

**String Split(CSV):**

String str = “A1,A2, A3”;

String names[] = str.split(“,”);

Syso(names[2])

**String “Immutable”:**

* To improve performance
* We cannot change value once created a string, it over ride and create new object in heap and new address in stock.
* When we use String wrapper class, its use String pool and Object from Heap
* So String is an “Immutable”
* It’s “Flyweight” Concept
* Can make String “Mutable” by using String Buffer and String Builder.
* String Buffer is “Thread Safe” where String Builder not, not Synchronization guarantee.

1. **String Class:**
   * String class has many in-built methods, like comparing, split, concatenate, convert.
   * In java Object class is Parent for all classes, so String class inherits object class.
2. **Constructor:**

* AnyMethod created with class name / It’s an member method
* It never returns anything.
* Constructor executes after object created.
* It allows us to initialize the object.
* It used to allocate memory
* It also allows implicit conversations in parameterized (Ex int to double)

**i) Default Constructor**

**ii) Parameterized Constructor:**

1. **OOPS Concepts:**

*i)* ***Polymorphism*:** an object oriented programming feature that allows us to perform a single action in different ways.

Static polymorphism: Polymorphism that is resolved during compiler time is known as static polymorphism. Method overloading can be considered as static polymorphism example.

Dynamic Polymorphism: it is also known as Dynamic Method Dispatch. Dynamic polymorphism is a process in which a call to an overridden method is resolved at runtime rather, thats why it is called runtime polymorphism.

Child class object can be referred by parent interface reference variable

**a) Over Loading (Compile Binding):**

* When the method name is same with different arguments or input parameters in same class.
* We can Over load main method also
* We cannot create method in side method
* Avoid Duplicate methods
* Public clss prtp()

{

Public void sum(){

s.o.p();

}

Public void sum(int i)

{

s.o.p();

}

Public void sum(int k, int j)

{

S.o.p();

}

Prtp obj = new Prtp();

Sum.obj();

Sum.obj(int i);

Sum.obj(int k, int j);

}

**b) Overriding (Run time poly/Dynamic Method Dispatch):**

When same method presents in parent class as well as in child class, with same name and same number of arguments.

**ii) Inheritance:**

* Child class inherits methods from parent class (super class) by using “Extends”.
* When we call object (Constructor) of child class, it automatically calls parent class object (only Default Constructor).
* But parent class cannot inherit methods from child class.
  + - * **Single Level Inheritance:** One class(ex a) inherited in child class(ex b)
      * **Multi-Level Inheritance:** Inheriting already inherited class,(Ex: a inherited in b, b inherited in c)
      * **Multiple-Inheritance:** We cannot achieve this in java because of **diamond problem (If we have same method name in both parent classes, child class object get confuse which one should take)**, but we can achieve with interfaces.

**iii) Encapsulation:**

Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit. In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class. Therefore, it is also known as **data hiding**.

To achieve encapsulation in Java −

* Declare the variables of a class as private.
* Provide public setter and getter methods to modify and view the variables values.

**iv) Interface:**

* Only Method Declaration, No Implementation
* By using “Implements”, multiple inheritance.
* No method Body, only method proto type
* All methods public and Abstract
* Here we can use variables, variables are static by default
* We cannot change variable values
* No static method
* No main method
* We cannot create an object of an interface, so interface in abstract in nature.

**Types of Interfaces:**

* **Marker Interface**: Without any methods
  + Still we can implements with calss
* **Default Methods**: only one method 🡪 from Java 8, we call functional interface.(Lambda Expressions)
  + Abc obj2 = () -> System.***out***.println("Shown");
* **Regular**

**Java 8 Updates for Interface:**

* **Default Methods**
  + List Interface updated with Stream() API
  + We can define (implements) method in Interface by using **“Default”** keyword for methods.
  + So when we define methods in interfaces, so when we implements multiple interfaces which having same method name and implementation, again we get diamond problem same like inheritance, to avoid that have to implement(override) the method in class which is same in both interfaces.
  + If a class extends another class and implements an interface, child class have more power than interface.
* **Static Methods:**
  + Java 8 supports static methods in Interfaces,
  + no need to create object, simply call with class name
    - EX: interex.show();

Public interface Prtp

{

Int bal = 100;

Public void credit();

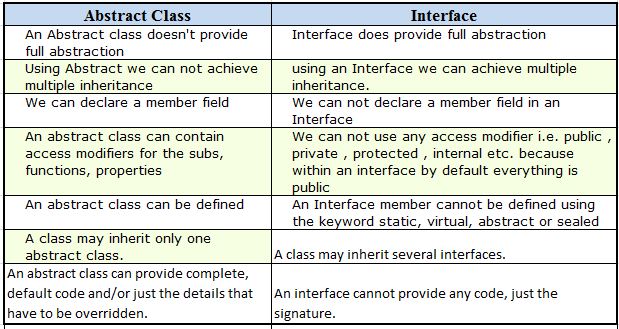
Public void debit();

Public void transfermoney();

}

**v) Abstraction:**

* Have implementation(define) and declaration in abstraction
* The method which we declare it calls abstract method.
* A class which has at least one abstract method is abstract class.
* But Abstract class may or may not contain Abstract methods.
* We cannot create object of abstract class.



1. **LOOPS:**

Which will allow you to run certain piece of run multiple times?

**i) For:**

for (initialize; condition; increment/decrement)

{

Code/logic

}

**ii) While**

* It will execute if condition is true

While (condition)

{

//logic

// incrmnt or decrmnt

}

**iii) Do –While:**

* In this loop code/logic will execute at least once.

Do

{

}while (condition)

**iv)Enhance-For Loop/ For Each**

* When we would like to get all elements from list/array without mentioning index.
* It works with only Arrays and Collections.
  + - * **int** values[] = {5,7,8,9,10};

**for**(**int** i: values) {

System.***out***.println(i);

}

**v) Break:**

* It Break complete loop, if the condition executed where break statement exists.

**vi) Continue:**

* It breaks current iteration depends on condition
* And it start from new Iteration
* It never breaks complete loop
  + EX: **for**(**int** i=1; i<=10; i++) {

**if**(i==7)

{

**continue**;

}

Syso(i)

}

1. **Conditional Statements:**

* It gives power to take decisions when to execute and what to be done of condition true.

**i) IF:**

* It check the condition if condition is true then execute the code

If (condition)

{

}

**ii) If Else:**

* If condition is true then execute if block or execute else block

If()

{

}

Else

{

}

iii) **Nested If else:**

if()

else if()

else

**iv) Switch Statement:**

* It allows you match with multiple cases

Switch(expression)

{

Case value:

//code here

Break; // If we don’t keep break, it execute next cases also

Case value:

//code

Break;

Default:

}

**Points for switch:**

* Data type should be same for comparison
* We can have multiple cases
* We can use only integers string(1.7 later) and Enums
* **Break** is optional but if not provided it will check all cases till the end
* **Default** is option; If none of the cases match it executes default.

**v) Ternary Operator (?:)**

* We use Ternary operator when we have single condition
  + EX:

I=8;

J=0;

If(i>=6)

J=1;

Else

J=2;

* By using Ternary, we can write as (? Works like if, : works like else)
  + J=i>=6?1:2

1. **This:**

* Keyword is used to match the global and local variables. When we have same variable name.
* ‘This’ keyword refers to a reference of the current class.
* By using ‘this’ keyword, it allows you to access methods/attributes of the current class (including its own private methods/attributes).

1. **Inner Class:**

* We can have class in side class;
* Have 3 types of inner classes
  + Member Class
  + Static Class
  + Anonymous Class

a) Member Class:

* + It’s a regular class inside a class
  + To access this class, have to create object depends on outer class
    - Outer.Inner1 ic = oc.**new** Inner1(); (oc-outer class object)

b) Static Class:

* + It’s a staticclass inside a class
  + To access this class, have to create object depends on outer class
    - Outer.Inner2 ic2 = **new** Outer.Inner2(); (Outer –outer class)

1. **Arrays:**
   * 1D Array:
   * Int a[] = new int [5] or int a[] = {1,2,3}
   * 2D Array:

int a[][] = new int[3][3]; or

**int** b[][] = {

{11,12,13},

{18,19,20},

{43,44,45}

* + 1. };
  + 3D Array:
  + Jagged Array:

1. **Super Keyword:**
   * In Java every class extends object class
   * And every child class object always calls parent class default constructor
   * But when we need to call other than Default constructor of parent class, then we use “Super” method in child class and pass arguments.

**public** B(**int** i) {

**super**(i);

System.***out***.println("In Con B Int");

}

* + **Super Variable**: when we have same variable in parent class and child class, but if like to get parent class variable, use “Super Keyword”

**public** **void** show() {

System.***out***.println(**super**.i);

}

1. **Handling exceptions with try/catch/ blocks:**
   * Have a **Throwable** Class in Java for Exceptions and it’s extended by
     + - Exceptions(Which are handle)
         * Checked

IOException

SQLException

* + - * + Unchecked

Runtime Exception

ArrayIndexOutofBound

FileNotFound

Arithmetic Exception

Nullpoint exception

Number format Exception

* + - * Errors(we cannot handle)
  + Handling exceptions is a mechanism to handle Exceptions.
  + We can use Multiple Catch with one Try
  + We can use Try with “Resources” so no need to use Catch or Finally, it automatically decommissioned all resources after try.
    - * **try**(BufferedReader br = **new** BufferedReader(**new** InputStreamReader(System.***in***))){

String str = "";

str = br.readLine();

}

* + **Throws:** 
    - * To supress error
      * It won’t handle the error, just it supress
      * Better to go with Try Catch
      * We can supress multiple exceptions with throws by adding ,
        + EX: **throws** **Exception,IOException,SQLException**
  + **Throw:**
    - * To force an exception By our self
      * By using throw new Exception() (instance of exception class)

**Final/Finally/Finalize:**

**Final: (Key Word)**

* We can use in Variable, Method, Class
* For constant values(variables)
* To prevent overriding(Method)
* To prevent inheritance (Class)

**Finally (Block):**

Code in finally block will be executed even if we have exception or no exception

**Finalize(Method):**

It’s related to garbage collector. Its use to clean up memory processing.

1. **Java Heap:**

It’s nothing but the memory space taken by the **jvm** from **OS.**

**JDK contains Java Compiler + JRE**

**JRE contains JVM + Library Files.**

1. **Garbage Collection:**

It’s provided by JVM. When an object don’t have reference variable, it goes into garbage.

**Reference variable:**

Prtp obj = new prtp();

Prtp **obj;**

Reference Variable for class.

**Multi-Threading:**

**Collection Framework**

* In the form of API which provides architecture to store different data and objects and manipulate data and group of object.
* We can do
  + Searching Data
  + Sorting
  + Insertion Data
  + Delete Data
  + Manipulate Data
* **Interfaces:**
  + Set
  + List
  + Queue
  + D queue
* **Classes**
  + Array List
  + Linked List
  + Vector
  + Tree Set
  + Hash set
  + Priority Queue

What is Collection in java: Collection represents a single unit of objects i.e. a group.

What is framework in java

* provides readymade architecture.
* represents set of classes and interface.
* is optional.

**What is Collection framework**

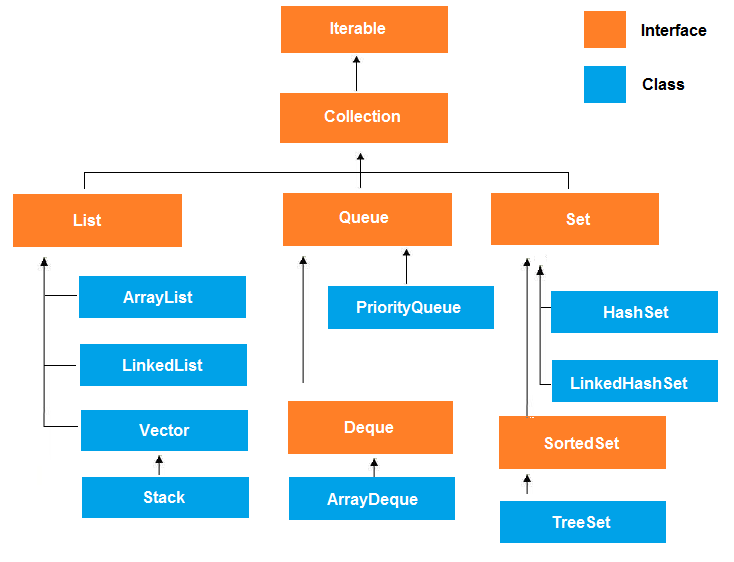
Collection framework represents a unified architecture for storing and manipulating group of objects.

It has:

+Interfaces and its implementations i.e. classes

+Algorithm

These all classes and interfaces is in java.utill package.



**Array List:**

**It is dynamic array**

Properties:

* Contains duplicate values/elements
* Maintain insertion order
* It synchronized
* Allows random access to fetch the elements bcz it stores the values in the basis of index

To print all values from arralist we use “for loop” and “iterator”