**JAVA Concepts**

**Collections**

1. Collections are like containers that group multiple items in a single unit. For example; a jar of chocolates, list of names etc
2. Some other important interfaces are [java.util.List](https://www.journaldev.com/11444/java-list), [java.util.Set](https://www.journaldev.com/13242/java-set), java.util.Queue and [java.util.Map](https://www.journaldev.com/11641/java-map).
3. java.util.Collection is the root interface of Collections Framework.
4. Some important collection classes are ArrayList, LinkedList, HashMap, TreeMap, HashSet, TreeSet.
5. It contains some important methods such as size(), iterator(), add(), remove(), clear() that every Collection class must implement.

**List Vs Set Vs Map**

1. **Duplicity:**
2. **-List** allows duplicate elements. Any number of duplicate elements can be inserted into the list without affecting the same existing values and their indexes.  
   **-Set** doesn’t allow duplicates. Set and all of the classes which implements Set interface should have unique elements.  
   **-Map** stored the elements as key & value pair. Map doesn’t allow duplicate keys while it allows duplicate values
3. Null values**:**

**-List** allows any number of null values.  
**-Set** allows single null value at most.  
**-Map** can have single null key at most and any number of null values.

1. **Order:**

**-List** and all of its implementation classes maintains the insertion order.  
**-Set** doesn’t maintain any order; still few of its classes sort the elements in an order such as **LinkedHashSet** maintains the elements in insertion order.  
Similar to Set

**-Map** also doesn’t stores the elements in an order, however few of its classes does the same.

For e.g. **TreeMap** sorts the map in the ascending order of keys and **LinkedHashMap** sorts the elements in the insertion order, the order in which the elements got added to the LinkedHashMap

**Static Block:**

-Any block defined with a static keyword is said to be static block.

-Static block will execute first.

-Static block will execute first, before the execution of main method.

- If we have multiple static block, it will execute in a sequential order.

**Non Static Block:**

-Any block defined without static keyword is said to be Non- static block.

- Non Static block will execute during the Object creation.

**Example**:

**package** com.javaconcepts;

**public** **class** StaticBlock {

StaticBlock ()

{

System.***out***.println("constructor");

}

**static**

{

System.***out***.println("static block");

}

{

System.***out***.println("NON static block");

}

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

StaticBlock obj11=**new** StaticBlock();

StaticBlock obj22=**new** StaticBlock();

}

**static**

{

System.***out***.println("static block 2");

}

}

**OUTPUT:**

static block

static block 2

NON static block

constructor

NON static block

constructor

**Polymorphism**: The process of changing the behavior of an object in multiple forms is called Polymorphism.

 -feature that allows us to perform a single action in different ways.

There are two types of polymorphism in java:  
1) **Static Polymorphism** also known as compile time polymorphism or Static Binding.  
2) **Dynamic Polymorphism** also known as runtime polymorphism

**Static Polymorphism**; The behavior of an object will be decided during compile time itself is called **Static Polymorphism.**

**Method Overloading:**

**- Same Class Name**

**- Same Method Name**

**- Same Implementation**

**-Different arguments/signatures i.e**

**-By No of arguments.**

**-By type of arguments.**

**-By position of arguments.**

**Example:**

class SimpleCalculator

{

int add(int a, int b)

{

return a+b;

}

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

{

return a+b+c;

}

}

public class Demo

{

public static void main(String args[])

{

SimpleCalculator obj = new SimpleCalculator();

System.out.println(obj.add(10, 20));

System.out.println(obj.add(10, 20, 30));

}

}

**Runtime/ Dynamic Polymorphism**: Object behavior changes during execution time of the program is said to be Run time **Polymorphism.**

**Method Overriding:**

**- Different Class Name**

**- Same Method Name**

**- Different Implementation**

**-Different arguments /signatures i.e**

**-By No of arguments.**

**-By type of arguments.**

**-By position of arguments.**

**To Achieve Runtime polymorphism we need to take help:**

1. Inheritance
2. Method Overriding
3. Up casting

Example:

class ABC{

public void myMethod(){

System.out.println("Overridden Method ABC ");

}

}

-----------

public class XYZ extends ABC{

public void myMethod(){

System.out.println("Overriding Method XYZ");

}

------------

public static void main(String args[]){

ABC obj = new XYZ();

obj.myMethod();

}

}

OUTPUT:

Overriding Method XYZ

Example:2

**Runtime Polymorphism example:**  
Animal.java

public class Animal{

public void sound(){

System.out.println("Animal is making a sound");

}

}

Horse.java

class Horse extends Animal{

@Override

public void sound(){

System.out.println("Neigh");

}

public static void main(String args[]){

Animal obj = new Horse();

obj.sound();

}

}

Output:

Neigh

Cat.java

public class Cat extends Animal{

@Override

public void sound(){

System.out.println("Meow");

}

public static void main(String args[]){

Animal obj = new Cat();

obj.sound();

}

}

Output:

Meow