Contents

Collections

The **Collection in Java** is a framework that provides 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 means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, Priority Queue, HashSet, LinkedHashSet, TreeSet).

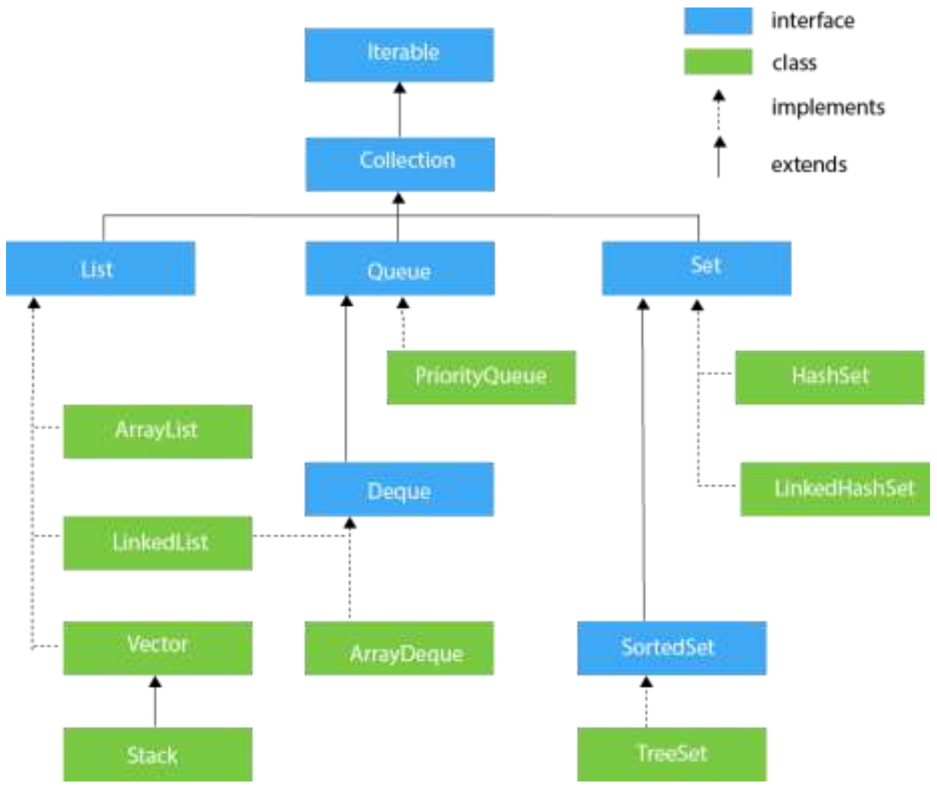
What is Collection framework

The Collection framework represents a unified architecture for storing and manipulating a group of objects. It has:

1. Interfaces and its implementations, i.e., classes
2. Algorithm

Hierarchy of Collection Framework

The **java.util** package contains all the classes and interfaces for the Collection framework.



Iterator interface

Iterator interface provides the facility of iterating the elements in a forward direction only.

Iterable Interface

The Iterable interface is the root interface for all the collection classes. The Collection interface extends the Iterable interface and therefore all the subclasses of Collection interface also implement the Iterable interface.

Collection Interface

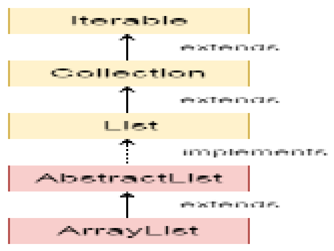
The Collection interface is the interface which is implemented by all the classes in the collection framework. It declares the methods that every collection will have.

List Interface

List interface is the 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.

ArrayList class



Java ArrayList class uses a dynamic array for storing the elements. It inherits AbstractList class and implements List interface.

The important points about Java ArrayList class are:

- Java ArrayList class can contain duplicate elements.
- Java ArrayList class maintains insertion order.
- Java ArrayList allows random access because array works at the index basis.
- In Java ArrayList class, manipulation is slow because a lot of shifting needs to occur if any element is removed from the array list.

```
import java.util.ArrayList;
import java.util.Iterator;

public class ArrayListEx1 {
	public static void main(String args[]){

	ArrayList<String> list=new ArrayList<String>();//Creating arraylist
	list.add("Ravi");//Adding object in arraylist
```

```
        list.add("Vijay");
        list.add("Ravi");
        list.add("Ajay");
        //Traversing list through Iterator
        Iterator itr=list.iterator();
        while(itr.hasNext()){
                System.out.println(itr.next());
        }
        }
}
```

Array vs ArrayList in Java

In Java, following are two different ways to create an array.

1. **Array**: Simple fixed sized arrays that we create in Java, like below

```
        int arr[] = new int[10]
```

2. **ArrayList** : Dynamic sized arrays in Java that implement List interface.

```
3.      ArrayList<Type> arrL = new ArrayList<Type>();
4.      Here Type is the type of elements in ArrayList to
5.      be created
```

An array is basic functionality provided by Java. ArrayList is part of collection framework in Java.

1. Resizable: Array is static in size that is fixed length data structure, One can not change the length after creating the Array object.
ArrayList is dynamic in size . Each ArrayList object has instance variable *capacity* which indicates the size of the ArrayList. As elements are added to an ArrayList its capacity grows automatically.

2. Primitives: ArrayList cannot contains primitive data types (like int , float , double) it can only contains Object while Array can contain both primitive data types as well as objects.

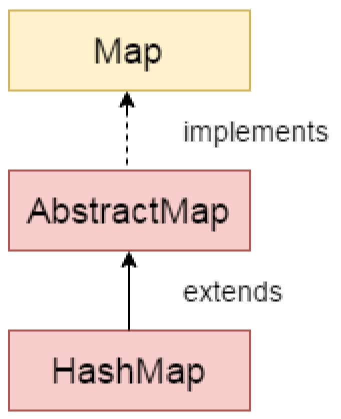
3. Iterating the values: We can use iterator to iterate through ArrayList . The iterators returned by the ArrayList class's iterator and listiterator method are fail-fast. We can use for loop or for each loop to iterate through array

4. Multi-dimensional: Array can be multi dimensional , while ArrayList is always single dimensional.

5. Adding elements : We can insert elements into the arraylist object using the add() method while in array we insert elements using the assignment operator.

6. Length : Length of the ArrayList is provided by the size() method while Each array object has the length variable which returns the length of the array.

Java HashMap class



Java HashMap class implements the map interface by using a hash table. It inherits AbstractMap class and implements Map interface.

Points to remember

- Java HashMap class contains values based on the key.
- Java HashMap class contains only unique keys.
- Java HashMap class may have one null key and multiple null values.
- Java HashMap class maintains no order.

Hierarchy of HashMap class

HashMap class extends AbstractMap class and implements Map interface.

Java HashMap example to add() elements

Here, we see different ways to insert elements.

```
import java.util.*;
class HashMap1{
 public static void main(String args[]){
   HashMap<Integer,String> hm=new HashMap<Integer,String>();
    System.out.println("Initial list of elements: "+hm);
      hm.put(100,"Amit");
      hm.put(101,"Vijay");
      hm.put(102,"Rahul");

      System.out.println("After invoking put() method ");
      for(Map.Entry m:hm.entrySet()){
       System.out.println(m.getKey()+" "+m.getValue());
      }

      hm.putIfAbsent(103, "Gaurav");
      System.out.println("After invoking putIfAbsent() method ");
      for(Map.Entry m:hm.entrySet()){
           System.out.println(m.getKey()+" "+m.getValue());
          }
      HashMap<Integer,String> map=new HashMap<Integer,String>();
      map.put(104,"Ravi");
      map.putAll(hm);
      System.out.println("After invoking putAll() method ");
      for(Map.Entry m:map.entrySet()){
           System.out.println(m.getKey()+" "+m.getValue());
          }
 }
}
```

```
Initial list of elements: {}
After invoking put() method
100 Amit
101 Vijay
102 Rahul
After invoking putIfAbsent() method
100 Amit
101 Vijay
102 Rahul
103 Gaurav
After invoking putAll() method
100 Amit
101 Vijay
102 Rahul
103 Gaurav
104 Ravi
```

Java HashMap example to remove() elements

```
import java.util.*;
public class HashMap2 {
   public static void main(String args[]) {
    HashMap<Integer,String> map=new HashMap<Integer,String>();
      map.put(100,"Amit");
      map.put(101,"Vijay");
      map.put(102,"Rahul");
      map.put(103, "Gaurav");
    System.out.println("Initial list of elements: "+map);
    //key-based removal
    map.remove(100);
    System.out.println("Updated list of elements: "+map);
    //value-based removal
    map.remove(101);
    System.out.println("Updated list of elements: "+map);
    //key-value pair based removal
    map.remove(102, "Rahul");
    System.out.println("Updated list of elements: "+map);
 }
}
```

```
Initial list of elements: {100=Amit, 101=Vijay, 102=Rahul, 103=Gaurav}
Updated list of elements: {101=Vijay, 102=Rahul, 103=Gaurav}
Updated list of elements: {102=Rahul, 103=Gaurav}
Updated list of elements: {103=Gaurav}
```

Properties class in Java

The **properties** object contains key and value pair both as a string. The java.util.Properties class is the subclass of Hashtable.

It can be used to get property value based on the property key. The Properties class provides methods to get data from the properties file and store data into the properties file. Moreover, it can be used to get the properties of a system.

An Advantage of the properties file

Recompilation is not required if the information is changed from a properties file: If any information is changed from the properties file, you don't need to recompile the java class. It is used to store information which is to be changed frequently.

```
FileReader reader=new FileReader("db.properties");

    Properties p=new Properties();
    p.load(reader);
```

db.properties file:

```
user=system

password=oracle
```

```
import java.util.*;
import java.io.*;
public class PropertiesTest {
	public static void main(String[] args)throws Exception{
	    FileReader reader=new FileReader("db.properties");
	    
	    Properties p=new Properties();
	    p.load(reader);
	    
	    System.out.println(p.getProperty("user"));
	    System.out.println(p.getProperty("password"));
	}
}
```

LinkedList

LinkedList implements the Collection interface. It uses a doubly linked list internally to store the elements. It can store the duplicate elements. It maintains the insertion order and is not synchronized. In LinkedList, the manipulation is fast because no shifting is required.

Vector

Vector uses a dynamic array to store the data elements. It is similar to ArrayList. However, It is synchronized and contains many methods that are not the part of Collection framework.

Stack

The stack is the subclass of Vector. It implements the last-in-first-out data structure, i.e., Stack. The stack contains all of the methods of Vector class and also provides its methods like boolean push(), boolean peek(), boolean push(object o), which defines its properties.

Queue Interface

Queue interface maintains the first-in-first-out order. It can be defined as an ordered list that is used to hold the elements which are about to be processed. There are various classes like PriorityQueue, Deque, and ArrayDeque which implements the Queue interface.

PriorityQueue

The PriorityQueue class implements the Queue interface. It holds the elements or objects which are to be processed by their priorities. PriorityQueue doesn't allow null values to be stored in the queue.

Deque Interface

Deque interface extends the Queue interface. In Deque, we can remove and add the elements from both the side. Deque stands for a double-ended queue which enables us to perform the operations at both the ends.

ArrayDeque

ArrayDeque class implements the Deque interface. It facilitates us to use the Deque. Unlike queue, we can add or delete the elements from both the ends.

ArrayDeque is faster than ArrayList and Stack and has no capacity restrictions.

Set Interface

Set Interface in Java is present in java.util package. It extends the Collection interface. It represents the unordered set of elements which doesn't allow us to store the duplicate items. We can store at most one null value in Set. Set is implemented by HashSet, LinkedHashSet, and TreeSet.

HashSet

HashSet class implements Set Interface. It represents the collection that uses a hash table for storage. Hashing is used to store the elements in the HashSet. It contains unique items.

LinkedHashSet

LinkedHashSet class represents the LinkedList implementation of Set Interface. It extends the HashSet class and implements Set interface. Like HashSet, It also contains unique elements. It maintains the insertion order and permits null elements.

SortedSet Interface

SortedSet is the alternate of Set interface that provides a total ordering on its elements. The elements of the SortedSet are arranged in the increasing (ascending) order. The SortedSet provides the additional methods that inhibit the natural ordering of the elements.

TreeSet

Java TreeSet class implements the Set interface that uses a tree for storage. Like HashSet, TreeSet also contains unique elements. However, the access and retrieval time of TreeSet is quite fast. The elements in TreeSet stored in ascending order.

Recursion

Recursion in java is a process in which a method calls itself continuously. A method in java that calls itself is called recursive method.

```
returntype methodname(){
//code to be executed
methodname();//calling same method
}
```

this keyword in java

There can be a lot of usage of **java this keyword**. In java, this is a **reference variable** that refers to the current object.

Usage of java this keyword

Here is given the 6 usage of java this keyword.

1. this can be used to refer current class instance variable.
2. this can be used to invoke current class method (implicitly)
3. this() can be used to invoke current class constructor.
4. this can be passed as an argument in the method call.
5. this can be passed as argument in the constructor call.
6. this can be used to return the current class instance from the method.

1) this: to refer current class instance variable

The this keyword can be used to refer current class instance variable. If there is ambiguity between the instance variables and parameters, this keyword resolves the problem of ambiguity.

Understanding the problem without this keyword

Let's understand the problem if we don't use this keyword by the example given below:

```
class Student{
int rollno;
String name;
float fee;
Student(int rollno,String name,float fee){
rollno=rollno;
name=name;
fee=fee;
}
void display(){System.out.println(rollno+" "+name+" "+fee);}
}
class TestThis1{
public static void main(String args[]){
Student s1=new Student(111,"ankit",5000f);
Student s2=new Student(112,"sumit",6000f);
s1.display();
s2.display();
}}
```

```
0 null 0.0
0 null 0.0
```

Solution of the above problem by this keyword

```java
class Student{
int rollno;
String name;
float fee;
Student(int rollno,String name,float fee){
this.rollno=rollno;
this.name=name;
this.fee=fee;
}
void display(){System.out.println(rollno+" "+name+" "+fee);}
}

class TestThis2{
public static void main(String args[]){
Student s1=new Student(111,"ankit",5000f);
Student s2=new Student(112,"sumit",6000f);
s1.display();
s2.display();
}}
```

```
111 ankit 5000
112 sumit 6000
```

Program where this keyword is not required

```java
class Student{
int rollno;
String name;
float fee;
Student(int r,String n,float f){
rollno=r;
name=n;
fee=f;
}
void display(){System.out.println(rollno+" "+name+" "+fee);}
}

class TestThis3{
public static void main(String args[]){
Student s1=new Student(111,"ankit",5000f);
Student s2=new Student(112,"sumit",6000f);
s1.display();
s2.display();
}}
```

```
111 ankit 5000
112 sumit 6000
```

Super Keyword in Java

The **super** keyword in Java is a reference variable which is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Usage of Java super Keyword

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

1) super is used to refer immediate parent class instance variable.

We can use super keyword to access the data member or field of parent class. It is used if parent class and child class have same fields.

```
class Animal{
String color="white";
}
class Dog extends Animal{
String color="black";
void printColor(){
System.out.println(color);//prints color of Dog class
System.out.println(super.color);//prints color of Animal class
}
}
class TestSuper1{
public static void main(String args[]){
Dog d=new Dog();
      d.printColor();
}}
```

O/P:

```
black
white
```

2) super can be used to invoke parent class method

The super keyword can also be used to invoke parent class method. It should be used if subclass contains the same method as parent class. In other words, it is used if method is overridden.

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void eat(){System.out.println("eating bread...");}
void bark(){System.out.println("barking...");}
void work(){
super.eat();
bark();
}
}
class TestSuper2{
public static void main(String args[]){
Dog d=new Dog();
d.work();
}}
```

Output:

```
eating...
barking...
```

In the above example Animal and Dog both classes have eat() method if we call eat() method from Dog class, it will call the eat() method of Dog class by default because priority is given to local.

To call the parent class method, we need to use super keyword.

3) super is used to invoke parent class constructor.

The super keyword can also be used to invoke the parent class constructor. Let's see a simple example:

```
class Animal{
Animal(){System.out.println("animal is created");}
}
class Dog extends Animal{
Dog(){
super();
System.out.println("dog is created");
}
}
class TestSuper3{
public static void main(String args[]){
Dog d=new Dog();
}}
```

Output:

```
animal is created
dog is created
```

Difference between final, finally and finalize

No.	final	finally	finalize
1)	Final is used to apply restrictions on class, method and variable. Final class can't be inherited, final method can't be overridden and final variable value can't be changed.	Finally is used to place important code, it will be executed whether exception is handled or	Finalize is used to perform clean up processing just before object is garbage

		not.	collected.
2)	Final is a keyword.	Finally is a block.	Finalize is a method.