

---

# Collections in Java

**Collections in java** is a framework that provides an architecture to store and manipulate the group of objects.

All the operations that you perform on a data such as searching, sorting, insertion, manipulation, deletion etc. can be performed by Java Collections.

Java Collection simply means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque etc.) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet etc).

## 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.

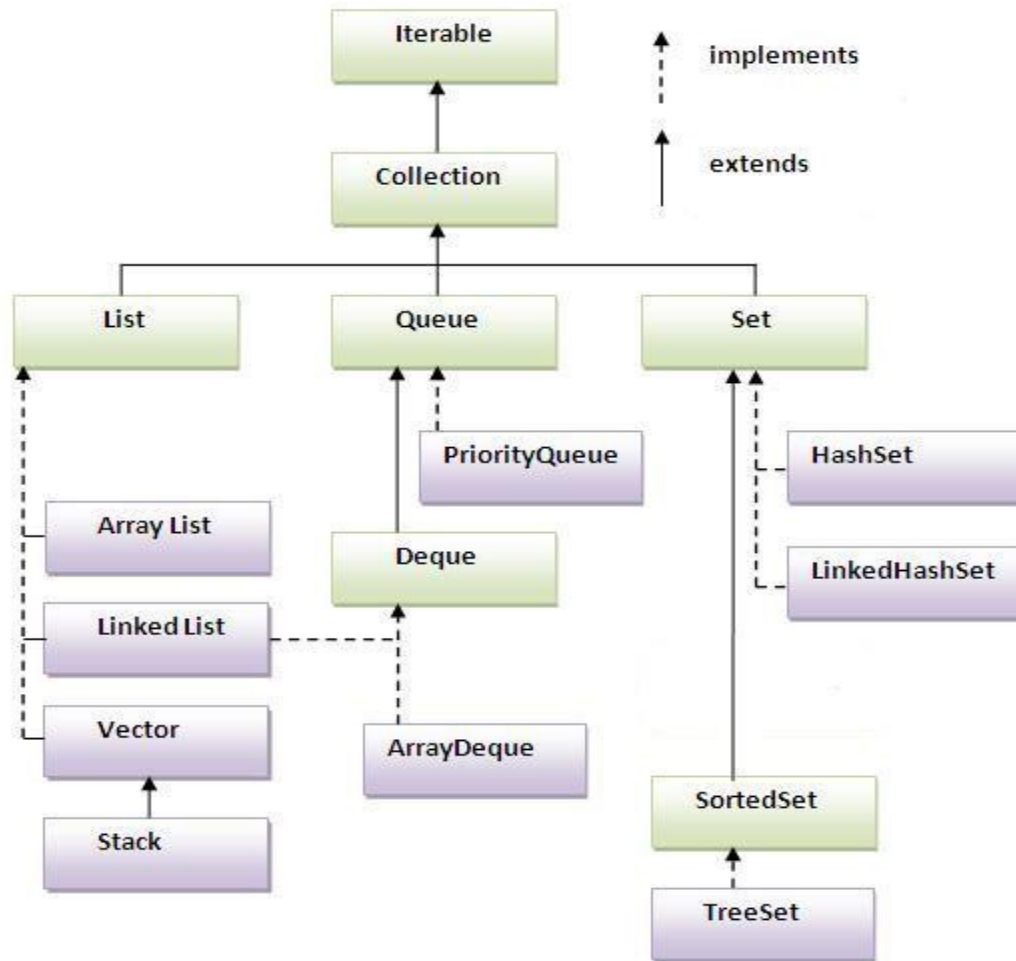
## What is Collection framework

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

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

## Hierarchy of Collection Framework

Let us see the hierarchy of collection framework. The **java.util** package contains all the classes and interfaces for Collection framework.



## Methods of Collection interface

There are many methods declared in the Collection interface. They are as follows:

No.	Method	Description
1	public boolean add(Object element)	is used to insert an element in this collection.
2	public boolean addAll(collection c)	is used to insert the specified collection elements in the invoking collection.
3	public boolean remove(Object element)	is used to delete an element from this collection.
4	public boolean removeAll(Collection c)	is used to delete all the elements of specified collection from the invoking collection.

5	public boolean retainAll(Collection c)	is used to delete all the elements of invoking collection except the specified collection.
6	public int size()	return the total number of elements in the collection.
7	public void clear()	removes the total no of element from the collection.
8	public boolean contains(object element)	is used to search an element.
9	public boolean containsAll(Collection c)	is used to search the specified collection in this collection.
10	public Iterator iterator()	returns an iterator.
11	public Object[] toArray()	converts collection into array.
12	public boolean isEmpty()	checks if collection is empty.
13	public boolean equals(Object element)	matches two collection.
14	public int hashCode()	returns the hashcode number for collection.

## Iterator interface

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

### Methods of Iterator interface

There are only three methods in the Iterator interface. They are:

1. **public boolean hasNext()** it returns true if iterator has more elements.
2. **public Object next()** it returns the element and moves the cursor pointer to the next element.
3. **public void remove()** it removes the last elements returned by the iterator. It is rarely used.

What we are going to learn in Java Collections Framework

1. [ArrayList class](#)
2. [LinkedList class](#)
3. [ListIterator interface](#)
4. [HashSet class](#)

- 
5. [LinkedHashSet class](#)
  6. [TreeSet class](#)
  7. [PriorityQueue class](#)
  8. [Map interface](#)
  9. [HashMap class](#)
  10. [LinkedHashMap class](#)
  11. [TreeMap class](#)
  12. [Hashtable class](#)
  13. [Sorting](#)
  14. [Comparable interface](#)
  15. [Comparator interface](#)
  16. [Properties class in Java](#)
- 

### Java ArrayList class

- Java ArrayList class uses a dynamic array for storing the elements. It extends AbstractList class and implements List interface.
- Java ArrayList class can contain duplicate elements.
- Java ArrayList class maintains insertion order.
- Java ArrayList class is non-synchronized.
- 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 be occurred if any element is removed from the array list.

### Java Non-generic Vs Generic Collection

Java collection framework was non-generic before JDK 1.5. Since 1.5, it is generic.

Java new generic collection allows you to have only one type of object in collection. Now it is type safe so typecasting is not required at run time.

Let's see the old non-generic example of creating java collection.

1. `ArrayList al=new ArrayList();//creating old non-generic arraylist`

Let's see the new generic example of creating java collection.

1. `ArrayList<String> al=new ArrayList<String>();//creating new generic arraylist`

In generic collection, we specify the type in angular braces. Now ArrayList is forced to have only specified type of objects in it. If you try to add another type of object, it gives *compile time error*.

### Example of Java ArrayList class

```

import java.util.*;
class TestCollection1 {
    public static void main(String args[]){

        ArrayList<String> al=new ArrayList<String>();//creating arraylist
        al.add("Ravi");//adding object in arraylist
        al.add("Vijay");
        al.add("Ravi");
        al.add("Ajay");

        Iterator itr=al.iterator();//getting Iterator from arraylist to traverse elements
        while(itr.hasNext()){
            System.out.println(itr.next());
        } } }

```

#### Test it Now

```

Ravi
Vijay
Ravi
Ajay

```

#### Two ways to iterate the elements of collection in java

1. By Iterator interface.
2. By for-each loop.

In the above example, we have seen traversing ArrayList by Iterator. Let's see the example to traverse ArrayList elements using for-each loop.

#### Iterating the elements of Collection by for-each loop

```

import java.util.*;
class TestCollection2 {
    public static void main(String args[]){
        ArrayList<String> al=new ArrayList<String>();
        al.add("Ravi");
        al.add("Vijay");
        al.add("Ravi");
        al.add("Ajay");
        for(String obj:al)
            System.out.println(obj);
    } }

```

### Test it Now

Ravi	Vijay	Ravi	Ajay
------	-------	------	------

### User-defined class objects in Java ArrayList

```
class Student{
    int rollno;
    String name;
    int age;
    Student(int rollno,String name,int age){
        this.rollno=rollno;
        this.name=name;
        this.age=age;
    }
}

import java.util.*;
public class TestCollection3{
    public static void main(String args[]){
        //Creating user-defined class objects
        Student s1=new Student(101,"Sonoo",23);
        Student s2=new Student(102,"Ravi",21);
        Student s3=new Student(103,"Hanumat",25);

        ArrayList<Student> al=new ArrayList<Student>();//creating arraylist
        al.add(s1);//adding Student class object
        al.add(s2);
        al.add(s3);

        Iterator itr=al.iterator();
        //traversing elements of ArrayList object
        while(itr.hasNext()){
            Student st=(Student)itr.next();
            System.out.println(st.rollno+" "+st.name+" "+st.age);
        }
    }
}
```

### Test it Now

101 Sonoo 23
102 Ravi 21
103 Hanumat 25

### Example of addAll(Collection c) method

```
import java.util.*;
```

```

class TestCollection4{
    public static void main(String args[]){

        ArrayList<String> al=new ArrayList<String>();
        al.add("Ravi");
        al.add("Vijay");
        al.add("Ajay");

        ArrayList<String> al2=new ArrayList<String>();
        al2.add("Sonoo");
        al2.add("Hanumat");

        al.addAll(al2);

        Iterator itr=al.iterator();
        while(itr.hasNext()){
            System.out.println(itr.next());
        } } }

```

#### Test it Now

Ravi  
 Vijay  
 Ajay  
 Sonoo  
 Hanumat

#### Example of removeAll() method

```

import java.util.*;
class TestCollection5{
    public static void main(String args[]){
        ArrayList<String> al=new ArrayList<String>();
        al.add("Ravi");
        al.add("Vijay");
        al.add("Ajay");
        ArrayList<String> al2=new ArrayList<String>();
        al2.add("Ravi");
        al2.add("Hanumat");
        al.removeAll(al2);
        System.out.println("iterating the elements after removing the elements of al2...");
        Iterator itr=al.iterator();
        while(itr.hasNext()){
            System.out.println(itr.next());
        } } }

```

#### Test it Now

iterating the elements after removing the elements of al2...

## Difference between ArrayList and LinkedList

ArrayList and LinkedList both implements List interface and maintains insertion order. Both are non synchronized classes.

But there are many differences between ArrayList and LinkedList classes that are given below.

ArrayList	LinkedList
1) ArrayList internally uses <b>dynamic array</b> to store the elements.	LinkedList internally uses <b>doubly linked list</b> to store the elements.
2) Manipulation with ArrayList is <b>slow</b> because it internally uses array. If any element is removed from the array, all the bits are shifted in memory.	Manipulation with LinkedList is <b>faster</b> than ArrayList because it uses doubly linked list so no bit shifting is required in memory.
3) ArrayList class can <b>act as a list</b> only because it implements List only.	LinkedList class can <b>act as a list and queue</b> both because it implements List and Deque interfaces.
4) ArrayList is <b>better for storing and accessing</b> data.	LinkedList is <b>better for manipulating</b> data.

## Example of ArrayList and LinkedList in Java

Let's see a simple example where we are using ArrayList and LinkedList both.

```
import java.util.*;
class TestArrayLinked{
    public static void main(String args[]){

        List<String> al=new ArrayList<String>();//creating arraylist
        al.add("Ravi");//adding object in arraylist
        al.add("Vijay");
        al.add("Ravi");
        al.add("Ajay");

        List<String> al2=new LinkedList<String>();//creating linkedlist
        al2.add("James");//adding object in linkedlist
        al2.add("Serena");
        al2.add("Swati");
        al2.add("Junaid");

        System.out.println("arraylist: "+al);
        System.out.println("linkedlist: "+al2);
    } }
```

Output:

```
arraylist: [Ravi,Vijay,Ravi,Ajay]
linkedlist: [James,Serena,Swati,Junaid]
```



---

## Java Map Interface

A map contains values based on the key i.e. key and value pair. Each pair is known as an entry. Map contains only unique elements.

Commonly used methods of Map interface:

1. **public Object put(object key, Object value):** is used to insert an entry in this map.
2. **public void putAll(Map map):** is used to insert the specified map in this map.
3. **public Object remove(object key):** is used to delete an entry for the specified key.
4. **public Object get(Object key):** is used to return the value for the specified key.
5. **public boolean containsKey(Object key):** is used to search the specified key from this map.
6. **public boolean containsValue(Object value):** is used to search the specified value from this map.
7. **public Set keySet():** returns the Set view containing all the keys.
8. **public Set entrySet():** returns the Set view containing all the keys and values.

### Entry

Entry is the subinterface of Map. So we will access it by Map.Entry name. It provides methods to get key and value.

Methods of Entry interface:

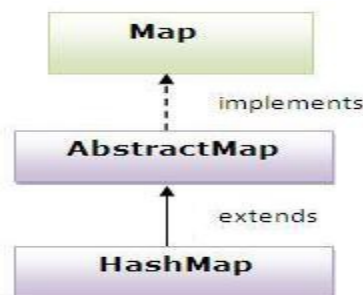
1. **public Object getKey():** is used to obtain key.
2. **public Object getValue():** is used to obtain value.

---

## Java HashMap class

- A HashMap contains values based on the key. It implements the Map interface and extends AbstractMap class.
- It contains only unique elements.
- It may have one null key and multiple null values.
- It maintains no order.

**Hierarchy of HashMap class:**



Example of HashMap class:

```
import java.util.*;
class TestCollection13 {
    public static void main(String args[]){
        HashMap<Integer,String> hm=new HashMap<Integer,String>();
        hm.put(100,"Amit");
        hm.put(101,"Vijay");
        hm.put(102,"Rahul");

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

Output:102 Rahul  
100 Amit  
101 Vijay

What is difference between HashSet and HashMap?

HashSet contains only values whereas HashMap contains entry(key and value).

## Sorting

We can sort the elements of:

1. String objects
2. Wrapper class objects
3. User-defined class objects

**Collections** class provides static methods for sorting the elements of collection.If collection elements are of Set type, we can use TreeSet.But We cannot sort the elements of List.Collections class provides methods for sorting the elements of List type elements.

### Method of Collections class for sorting List elements

**public void sort(List list):** is used to sort the elements of List.List elements must be of Comparable type.

Note: String class and Wrapper classes implements the Comparable interface.So if you store the objects of string or wrapper classes, it will be Comparable.

### Example of Sorting the elements of List that contains string objects

```
import java.util.*;
```

```

class TestSort1 {
public static void main(String args[]){

    ArrayList<String> al=new ArrayList<String>();
    al.add("Viru");
    al.add("Saurav");
    al.add("Mukesh");
    al.add("Tahir");

    Collections.sort(al);
    Iterator itr=al.iterator();
    while(itr.hasNext()){
        System.out.println(itr.next());
    } } }

```

Output:Mukesh

Saurav  
Tahir  
Viru

Example of Sorting the elements of List that contains Wrapper class objects

```

import java.util.*;
class TestSort2 {
public static void main(String args[]){

    ArrayList al=new ArrayList();
    al.add(Integer.valueOf(201));
    al.add(Integer.valueOf(101));
    al.add(230);//internally will be converted into objects as Integer.valueOf(230)

    Collections.sort(al);

    Iterator itr=al.iterator();
    while(itr.hasNext()){
        System.out.println(itr.next());
    } } }

```

Output:101

201  
230

---

# Java Collections Interview Questions Answers

## 1. What is Java Collections Framework? List out some benefits of Collections framework?

Collections are used in every programming language and initial java release contained few classes for collections: **Vector**, **Stack**, **Hashtable**, **Array**. But looking at the larger scope and usage, Java 1.2 came up with Collections Framework that group all the collections interfaces, implementations and algorithms. Java Collections have come through a long way with usage of Generics and Concurrent Collection classes for thread-safe operations. It also includes blocking interfaces and their implementations in java concurrent package. Some of the benefits of collections framework are:

1. Reduced development effort by using core collection classes rather than implementing our own collection classes.
2. Code quality is enhanced with the use of well tested collections framework classes.
3. Reduced effort for code maintenance by using collection classes shipped with JDK.
4. Reusability and Interoperability

## What is the benefit of Generics in Collections Framework?

Java 1.5 came with Generics and all collection interfaces and implementations use it heavily. Generics allow us to provide the type of Object that a collection can contain, so if you try to add any element of other type it throws compile time error. This avoids ClassCastException at Runtime because you will get the error at compilation. Also Generics make code clean since we don't need to use casting and instanceof operator. It also adds up to runtime benefit because the bytecode instructions that do type checking are not generated.

## 2 What are the basic interfaces of Java Collections Framework?

[Collection](#) is the root of the collection hierarchy. A collection represents a group of objects known as its elements. The Java platform doesn't provide any direct implementations of this interface.

[Set](#) is a collection that cannot contain duplicate elements. This interface models the mathematical set abstraction and is used to represent sets, such as the deck of cards.

[List](#) is an ordered collection and can contain duplicate elements. You can access any element from its index. List is more like array with dynamic length.

A [Map](#) is an object that maps keys to values. A map cannot contain duplicate keys: Each key can map to at most one value.

Some other interfaces are [Queue](#), [Deque](#), [Iterator](#), [SortedSet](#), [SortedMap](#) and [ListIterator](#).

## 3 Why Collection doesn't extend Cloneable and Serializable interfaces?

---

Collection interface specifies group of Objects known as elements. How the elements are maintained is left up to the concrete implementations of Collection. For example, some Collection implementations like List allow duplicate elements whereas other implementations like Set don't. A lot of the Collection implementations have a public clone method. However, it doesn't really make sense to include it in all implementations of Collection. This is because Collection is an abstract representation. What matters is the implementation.

The semantics and the implications of either cloning or serializing come into play when dealing with the actual implementation; so concrete implementation should decide how it should be cloned or serialized, or even if it can be cloned or serialized.

So mandating cloning and serialization in all implementations is actually less flexible and more restrictive. The specific implementation should make the decision as to whether it can be cloned or serialized.

#### 4 Why Map interface doesn't extend Collection interface?

Although Map interface and its implementations are part of Collections Framework, Map are not collections and collections are not Map. Hence it doesn't make sense for Map to extend Collection or vice versa.

If Map extends Collection interface, then where are the elements? Map contains key-value pairs and it provides methods to retrieve list of Keys or values as Collection but it doesn't fit into the "group of elements" paradigm.

#### 5 What is an Iterator?

Iterator interface provides methods to iterate over any Collection. We can get iterator instance from a Collection using iterator method. Iterator takes the place of Enumeration in the Java Collections Framework. Iterators allow the caller to remove elements from the underlying collection during the iteration.

#### 6 What is difference between Enumeration and Iterator interface?

Enumeration is twice as fast as Iterator and uses very less memory. Enumeration is very basic and fits to basic needs. But Iterator is much safer as compared to Enumeration because it always denies other threads to modify the collection object which is being iterated by it.

Iterator takes the place of Enumeration in the Java Collections Framework. Iterators allow the caller to remove elements from the underlying collection that is not possible with Enumeration. Iterator method names have been improved to make its functionality clear.

#### 7 Why there is not method like Iterator.add() to add elements to the collection?

The semantics are unclear, given that the contract for Iterator makes no guarantees about the order of iteration. Note, however, that ListIterator does provide an add operation, as it does guarantee the order of the iteration.

#### 8 Why Iterator don't have a method to get next element directly without moving the cursor?

It can be implemented on top of current Iterator interface but since its use will be rare, it doesn't make sense to include it in the interface that everyone has to implement.

---

## 9 What is different between Iterator and ListIterator?

1. We can use Iterator to traverse Set and List collections whereas ListIterator can be used with Lists only.
2. Iterator can traverse in forward direction only whereas ListIterator can be used to traverse in both the directions.
3. ListIterator inherits from Iterator interface and comes with extra functionalities like adding an element, replacing an element, getting index position for previous and next elements.

## 2 What are different ways to iterate over a list?

We can iterate over a list in two different ways – using iterator and using for-each loop.

```
01 List<String> strList = new ArrayList<>();
02 //using for-each loop
03 for(String obj : strList){
04     System.out.println(obj);
05 }
06 //using iterator
07 Iterator<String> it = strList.iterator();
08 while(it.hasNext()){
09     String obj = it.next();
10     System.out.println(obj);
11 }
```

Using iterator is more thread-safe because it makes sure that if underlying list elements are modified, it will throw `ConcurrentModificationException`.

## 3 What do you understand by iterator fail-fast property?

Iterator fail-fast property checks for any modification in the structure of the underlying collection everytime we try to get the next element. If there are any modifications found, it throws `ConcurrentModificationException`. All the implementations of Iterator in Collection classes are fail-fast by design except the concurrent collection classes like `ConcurrentHashMap` and `CopyOnWriteArrayList`.

## 4 What is difference between fail-fast and fail-safe?

Iterator fail-safe property work with the clone of underlying collection, hence it's not affected by any modification in the collection. By design, all the collection classes in `java.util` package are fail-fast whereas collection classes in `java.util.concurrent` are fail-safe. Fail-fast iterators throw `ConcurrentModificationException` whereas fail-safe iterator never throws `ConcurrentModificationException`. Check this post for [CopyOnWriteArrayList Example](#).

## 5 How to avoid ConcurrentModificationException while iterating a collection?

---

We can use concurrent collection classes to avoid `ConcurrentModificationException` while iterating over a collection, for example `CopyOnWriteArrayList` instead of `ArrayList`.  
Check this post for [ConcurrentHashMap Example](#).

## 6 Why there are no concrete implementations of Iterator interface?

Iterator interface declare methods for iterating a collection but it's implementation is responsibility of the Collection implementation classes. Every collection class that returns an iterator for traversing has it's own Iterator implementation nested class.

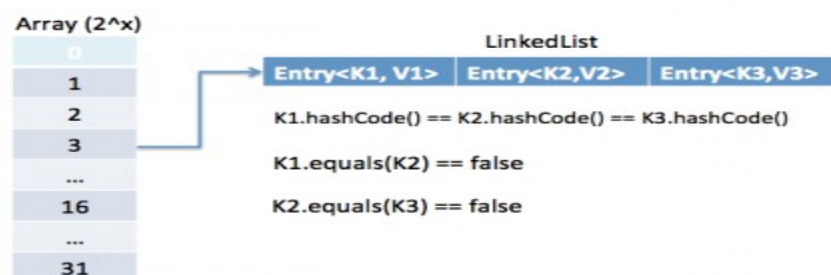
This allows collection classes to chose whether iterator is fail-fast or fail-safe. For example `ArrayList` iterator is fail-fast whereas `CopyOnWriteArrayList` iterator is fail-safe.

## 7 What is UnsupportedOperationException?

`UnsupportedOperationException` is the exception used to indicate that the operation is not supported. It's used extensively in JDK classes, in collections framework `java.util.Collections.UnmodifiableCollection` throws this exception for all `add` and `remove` operations.

## 8 How HashMap works in Java?

HashMap stores key-value pair in `Map.Entry` static nested class implementation. HashMap works on hashing algorithm and uses `hashCode()` and `equals()` method in `put` and `get` methods. When we call `put` method by passing key-value pair, HashMap uses `Key hashCode()` with hashing to find out the index to store the key-value pair. The Entry is stored in the `LinkedList`, so if there are already existing entry, it uses `equals()` method to check if the passed key already exists, if yes it overwrites the value else it creates a new entry and store this key-value Entry. When we call `get` method by passing Key, again it uses the `hashCode()` to find the index in the array and then use `equals()` method to find the correct Entry and return it's value. Below image will explain these detail clearly.



The other important things to know about HashMap are capacity, load factor, threshold resizing. HashMap initial default capacity is 32 and load factor is 0.75. Threshold is capacity multiplied by load factor and whenever we try to add an entry, if map size is greater than threshold, HashMap reshapes the contents of map into a new array with a larger capacity. The capacity is always power of 2, so if you know that you need to store a large number of key-value pairs, for example in caching data from database, it's good idea to initialize the HashMap with correct capacity and load factor.

---

## 9 What is the importance of hashCode() and equals() methods?

HashMap uses Key object hashCode() and equals() method to determine the index to put the key-value pair. These methods are also used when we try to get value from HashMap. If these methods are not implemented correctly, two different Key's might produce same hashCode() and equals() output and in that case rather than storing it at different location, HashMap will consider them same and overwrite them. Similarly all the collection classes that doesn't store duplicate data use hashCode() and equals() to find duplicates, so it's very important to implement them correctly. The implementation of equals() and hashCode() should follow these rules.

1. If `o1.equals(o2)`, then `o1.hashCode() == o2.hashCode()` should always be `true`.
2. If `o1.hashCode() == o2.hashCode()` is true, it doesn't mean that `o1.equals(o2)` will be `true`.
- 2 **Can we use any class as Map key?** We can use any class as Map Key, however following points should be considered before using them.
  1. If the class overrides equals() method, it should also override hashCode() method.
  2. The class should follow the rules associated with equals() and hashCode() for all instances. Please refer earlier question for these rules.
  3. If a class field is not used in equals(), you should not use it in hashCode() method.
  4. Best practice for user defined key class is to make it immutable, so that hashCode() value can be cached for fast performance. Also immutable classes make sure that hashCode() and equals() will not change in future that will solve any issue with mutability.  
For example, let's say I have a class `MyKey` that I am using for HashMap key.

```
01 //MyKey name argument passed is used for equals() and hashCode()
02 MyKey key = new MyKey('Pankaj'); //assume hashCode=1234
03 myHashMap.put(key, 'Value');

05 // Below code will change the key hashCode() and equals()
06 // but it's location is not changed.
07 key.setName('Amit'); //assume new hashCode=7890

09 //below will return null, because HashMap will try to look for key
10 //in the same index as it was stored but since key is mutated,
11 //there will be no match and it will return null.
12 myHashMap.get(new MyKey('Pankaj'));
```
10. This is the reason why String and Integer are mostly used as HashMap keys.

## 2 What are different Collection views provided by Map interface?

Map interface provides three collection views:

1. **Set** `keySet()`: Returns a Set view of the keys contained in this map. The set is backed by the map, so changes to the map are reflected in the set, and vice-versa. If the map is modified while an iteration over the set is in progress (except through the iterator's own remove operation), the results of the iteration are undefined. The set supports element removal, which removes the corresponding mapping from the map, via the `Iterator.remove`, `Set.remove`, `removeAll`, `retainAll`, and `clear` operations. It does not support the `add` or `addAll` operations.



- 
2. **Collection values():** Returns a Collection view of the values contained in this map. The collection is backed by the map, so changes to the map are reflected in the collection, and vice-versa. If the map is modified while an iteration over the collection is in progress (except through the iterator's own remove operation), the results of the iteration are undefined. The collection supports element removal, which removes the corresponding mapping from the map, via the `Iterator.remove`, `Collection.remove`, `removeAll`, `retainAll` and `clear` operations. It does not support the `add` or `addAll` operations.
  3. **Set<Map.Entry<K, V>> entrySet():** Returns a Set view of the mappings contained in this map. The set is backed by the map, so changes to the map are reflected in the set, and vice-versa. If the map is modified while an iteration over the set is in progress (except through the iterator's own remove operation, or through the `setValue` operation on a map entry returned by the iterator) the results of the iteration are undefined. The set supports element removal, which removes the corresponding mapping from the map, via the `Iterator.remove`, `Set.remove`, `removeAll`, `retainAll` and `clear` operations. It does not support the `add` or `addAll` operations.

## 2 What is difference between HashMap and Hashtable?

HashMap and Hashtable both implements Map interface and looks similar, however there are following difference between HashMap and Hashtable.

1. HashMap allows null key and values whereas Hashtable doesn't allow null key and values.
2. Hashtable is synchronized but HashMap is not synchronized. So HashMap is better for single threaded environment, Hashtable is suitable for multi-threaded environment.
3. `LinkedHashMap` was introduced in Java 1.4 as a subclass of HashMap, so incase you want iteration order, you can easily switch from HashMap to LinkedHashMap but that is not the case with Hashtable whose iteration order is unpredictable.
4. HashMap provides Set of keys to iterate and hence it's fail-fast but Hashtable provides Enumeration of keys that doesn't support this feature.
5. Hashtable is considered to be legacy class and if you are looking for modifications of Map while iterating, you should use `ConcurrentHashMap`.

## 2 How to decide between HashMap and TreeMap?

For inserting, deleting, and locating elements in a Map, the HashMap offers the best alternative. If, however, you need to traverse the keys in a sorted order, then TreeMap is your better alternative. Depending upon the size of your collection, it may be faster to add elements to a HashMap, then convert the map to a TreeMap for sorted key traversal.

## 3 What are similarities and difference between ArrayList and Vector?

ArrayList and Vector are similar classes in many ways.

1. Both are index based and backed up by an array internally.
2. Both maintains the order of insertion and we can get the elements in the order of insertion.
3. The iterator implementations of ArrayList and Vector both are fail-fast by design.
4. ArrayList and Vector both allows null values and random access to element using index number.

---

These are the differences between ArrayList and Vector.

1. Vector is synchronized whereas ArrayList is not synchronized. However if you are looking for modification of list while iterating, you should use CopyOnWriteArrayList.
2. ArrayList is faster than Vector because it doesn't have any overhead because of synchronization.
3. ArrayList is more versatile because we can get synchronized list or read-only list from it easily using Collections utility class.

## 2 What is difference between Array and ArrayList? When will you use Array over ArrayList?

Arrays can contain primitive or Objects whereas ArrayList can contain only Objects.

Arrays are fixed size whereas ArrayList size is dynamic.

Arrays doesn't provide a lot of features like ArrayList, such as addAll, removeAll, iterator etc. Although ArrayList is the obvious choice when we work on list, there are few times when array are good to use.

1. If the size of list is fixed and mostly used to store and traverse them.
2. For list of primitive data types, although Collections use autoboxing to reduce the coding effort but still it makes them slow when working on fixed size primitive data types.
3. If you are working on fixed multi-dimensional situation, using `[][]` is far more easier than `List<List<>>`

## 2 What is difference between ArrayList and LinkedList?

ArrayList and LinkedList both implement List interface but there are some differences between them.

1. ArrayList is an index based data structure backed by Array, so it provides random access to it's elements with performance as  $O(1)$  but LinkedList stores data as list of nodes where every node is linked to it's previous and next node. So even though there is a method to get the element using index, internally it traverse from start to reach at the index node and then return the element, so performance is  $O(n)$  that is slower than ArrayList.
2. Insertion, addition or removal of an element is faster in LinkedList compared to ArrayList because there is no concept of resizing array or updating index when element is added in middle.
3. LinkedList consumes more memory than ArrayList because every node in LinkedList stores reference of previous and next elements.

## 2 Which collection classes provide random access of it's elements?

ArrayList, HashMap, TreeMap, Hashtable classes provide random access to it's elements. Download [java collections pdf](#) for more information.

## 3 What is EnumSet?

`java.util.EnumSet` is Set implementation to use with enum types. All of the elements in an enum set must come from a single enum type that is specified, explicitly or implicitly, when the set is created. EnumSet is not synchronized and null

---

elements are not allowed. It also provides some useful methods like `copyOf(Collection c)`, `of(E first, E... rest)` and `complementOf(EnumSet s)`. Check this post for [java enum tutorial](#).

#### 4 Which collection classes are thread-safe?

`Vector`, `Hashtable`, `Properties` and `Stack` are synchronized classes, so they are thread-safe and can be used in multi-threaded environment. Java 1.5 Concurrent API included some collection classes that allow modification of collection while iteration because they work on the clone of the collection, so they are safe to use in multi-threaded environment.

#### 5 What are concurrent Collection Classes?

Java 1.5 Concurrent package (`java.util.concurrent`) contains thread-safe collection classes that allow collections to be modified while iterating. By design iterator is fail-fast and throws `ConcurrentModificationException`. Some of these classes are `CopyOnWriteArrayList`, `ConcurrentHashMap`, `CopyOnWriteArraySet`. Read these posts to learn about them in more detail.

1. [Avoid ConcurrentModificationException](#)
2. [CopyOnWriteArrayList Example](#)
3. [HashMap vs ConcurrentHashMap](#)

#### 2 What is BlockingQueue?

`java.util.concurrent.BlockingQueue` is a Queue that supports operations that wait for the queue to become non-empty when retrieving and removing an element, and wait for space to become available in the queue when adding an element. `BlockingQueue` interface is part of java collections framework and it's primarily used for implementing producer consumer problem. We don't need to worry about waiting for the space to be available for producer or object to be available for consumer in `BlockingQueue` as it's handled by implementation classes of `BlockingQueue`. Java provides several `BlockingQueue` implementations such as `ArrayBlockingQueue`, `LinkedBlockingQueue`, `PriorityBlockingQueue`, `SynchronousQueue` etc. Check this post for use of `BlockingQueue` for [producer-consumer problem](#).

#### 3 What is Queue and Stack, list their differences?

Both Queue and Stack are used to store data before processing them. `java.util.Queue` is an interface whose implementation classes are present in java concurrent package. Queue allows retrieval of element in First-In-First-Out (FIFO) order but it's not always the case. There is also Deque interface that allows elements to be retrieved from both end of the queue.

Stack is similar to queue except that it allows elements to be retrieved in Last-In-First-Out (LIFO) order. Stack is a class that extends `Vector` whereas Queue is an interface.

#### 4 What is Collections Class?

`java.util.Collections` is a utility class consists exclusively of static methods that operate on or return collections. It contains polymorphic algorithms that operate on collections, "wrappers", which return a new collection backed by a

---

specified collection, and a few other odds and ends. This class contains methods for collection framework algorithms, such as binary search, sorting, shuffling, reverse etc.

## 5 What is Comparable and Comparator interface?

Java provides Comparable interface which should be implemented by any custom class if we want to use Arrays or Collections sorting methods. Comparable interface has `compareTo(T obj)` method which is used by sorting methods. We should override this method in such a way that it returns a negative integer, zero, or a positive integer if “this” object is less than, equal to, or greater than the object passed as argument. But, in most real life scenarios, we want sorting based on different parameters. For example, as a CEO, I would like to sort the employees based on Salary, an HR would like to sort them based on the age. This is the situation where we need to use `Comparator` interface because `Comparable.compareTo(Object o)` method implementation can sort based on one field only and we can't choose the field on which we want to sort the Object. `Comparator` interface `compare(Object o1, Object o2)` method needs to be implemented that takes two Object arguments, it should be implemented in such a way that it returns negative int if first argument is less than the second one and returns zero if they are equal and positive int if first argument is greater than second one.

Check this post for use of Comparable and Comparator interface to [sort objects](#).

## 6 What is difference between Comparable and Comparator interface?

Comparable and Comparator interfaces are used to sort collection or array of objects. Comparable interface is used to provide the natural sorting of objects and we can use it to provide sorting based on single logic. Comparator interface is used to provide different algorithms for sorting and we can choose the comparator we want to use to sort the given collection of objects.

## 7 How can we sort a list of Objects?

If we need to sort an array of Objects, we can use `Arrays.sort()`. If we need to sort a list of objects, we can use `Collections.sort()`. Both these classes have overloaded `sort()` methods for natural sorting (using Comparable) or sorting based on criteria (using Comparator). Collections internally uses Arrays sorting method, so both of them have same performance except that Collections take sometime to convert list to array.

## 8 While passing a Collection as argument to a function, how can we make sure the function will not be able to modify it?

We can create a read-only collection using `Collections.unmodifiableCollection(Collection c)` method before passing it as argument, this will make sure that any operation to change the collection will throw `UnsupportedOperationException`.

# RMI (Remote Method Invocation)

The **RMI** (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM.

---

The RMI provides remote communication between the applications using two objects *stub* and *skeleton*.

## Understanding stub and skeleton

RMI uses stub and skeleton object for communication with the remote object.

A **remote object** is an object whose method can be invoked from another JVM. Let's understand the stub and skeleton objects:

### stub

The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

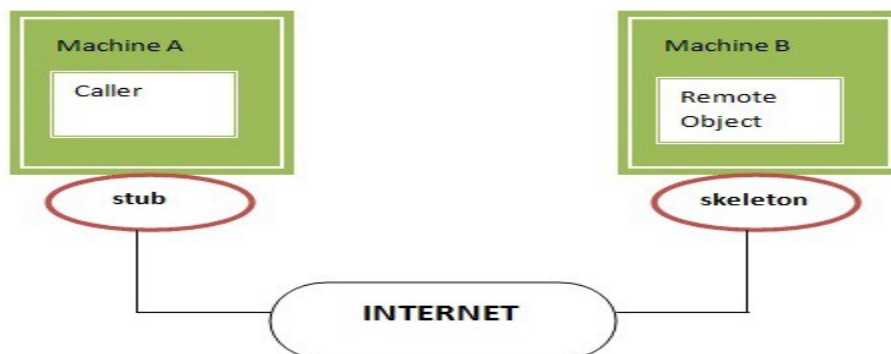
1. It initiates a connection with remote Virtual Machine (JVM),
2. It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),
3. It waits for the result
4. It reads (unmarshals) the return value or exception, and
5. It finally, returns the value to the caller.

### skeleton

The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

1. It reads the parameter for the remote method
2. It invokes the method on the actual remote object, and
3. It writes and transmits (marshals) the result to the caller.

In the Java 2 SDK, an stub protocol was introduced that eliminates the need for skeletons.



## Understanding requirements for the distributed applications

If any application performs these tasks, it can be distributed application.

1. The application need to locate the remote method
2. It need to provide the communication with the remote objects, and
3. The application need to load the class definitions for the objects.

The RMI application have all these features, so it is called the distributed application.

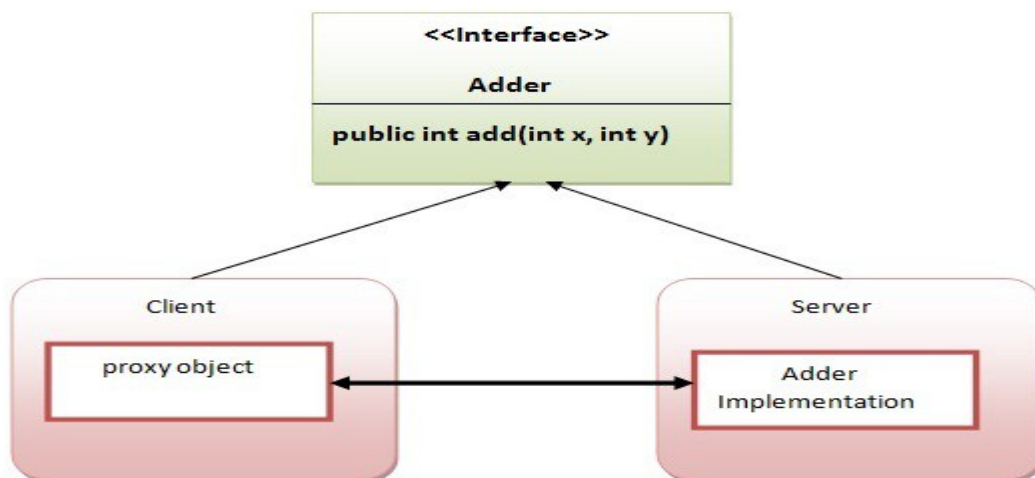
## Steps to write the RMI program

The is given the 6 steps to write the RMI program.

1. Create the remote interface
2. Provide the implementation of the remote interface
3. Compile the implementation class and create the stub and skeleton objects using the rmic tool
4. Start the registry service by rmiregistry tool
5. Create and start the remote application
6. Create and start the client application

## RMI Example

In this example, we have followed all the 6 steps to create and run the rmi application. The client application need only two files, remote interface and client application. In the rmi application, both client and server interacts with the remote interface. The client application invokes methods on the proxy object, RMI sends the request to the remote JVM. The return value is sent back to the proxy object and then to the client application.



### 1) create the remote interface

---

For creating the remote interface, extend the Remote interface and declare the RemoteException with all the methods of the remote interface. Here, we are creating a remote interface that extends the Remote interface. There is only one method named add() and it declares RemoteException.

```
1.      import java.rmi.*;
2.      public interface Adder extends Remote{
3.      public int add(int x,int y)throws RemoteException;
4.      }
```

## 2) Provide the implementation of the remote interface

Now provide the implementation of the remote interface. For providing the implementation of the Remote interface, we need to

- Either extend the UnicastRemoteObject class,
- or use the exportObject() method of the UnicastRemoteObject class

In case, you extend the UnicastRemoteObject class, you must define a constructor that declares RemoteException.

```
1.      import java.rmi.*;
2.      import java.rmi.server.*;
3.      public class AdderRemote extends UnicastRemoteObject implements Adder{
4.      AdderRemote()throws RemoteException{
5.      super();
6.      }
7.      public int add(int x,int y){return x+y;}
8.      }
```

## 3) create the stub and skeleton objects using the rmic tool.

Next step is to create stub and skeleton objects using the rmi compiler. The rmic tool invokes the RMI compiler and creates stub and skeleton objects.

```
1.      rmic AdderRemote
```

## 4) Start the registry service by the rmiregistry tool

Now start the registry service by using the rmiregistry tool. If you don't specify the port number, it uses a default port number. In this example, we are using the port number 5000.

```
1.      rmiregistry 5000
```

## 5) Create and run the server application

---

Now rmi services need to be hosted in a server process. The Naming class provides methods to get and store the remote object. The Naming class provides 5 methods.

1. **public static java.rmi.Remote lookup(java.lang.String) throws java.rmi.NotBoundException, java.net.MalformedURLException, java.rmi.RemoteException;** it returns the reference of the remote object.
2. **public static void bind(java.lang.String, java.rmi.Remote) throws java.rmi.AlreadyBoundException, java.net.MalformedURLException, java.rmi.RemoteException;** it binds the remote object with the given name.
3. **public static void unbind(java.lang.String) throws java.rmi.RemoteException, java.rmi.NotBoundException, java.net.MalformedURLException;** it destroys the remote object which is bound with the given name.
4. **public static void rebind(java.lang.String, java.rmi.Remote) throws java.rmi.RemoteException, java.net.MalformedURLException;** it binds the remote object to the new name.
5. **public static java.lang.String[] list(java.lang.String) throws java.rmi.RemoteException, java.net.MalformedURLException;** it returns an array of the names of the remote objects bound in the registry.

In this example, we are binding the remote object by the name sonoo.

```
1.      import java.rmi.*;
2.      import java.rmi.registry.*;
3.      public class MyServer{
4.      public static void main(String args[]){
5.      try{
6.      Adder stub=new AdderRemote();
7.      Naming.rebind("rmi://localhost:5000/sonoo",stub);
8.      }catch(Exception e){System.out.println(e);}
9.      } }
```

## 6) Create and run the client application

At the client we are getting the stub object by the lookup() method of the Naming class and invoking the method on this object. In this example, we are running the server and client applications, in the same machine so we are using localhost. If you want to access the remote object from another machine, change the localhost to the host name (or IP address) where the remote object is located.

```
1.      import java.rmi.*;
2.      public class MyClient{
3.      public static void main(String args[]){
4.      try{
5.      Adder stub=(Adder)Naming.lookup("rmi://localhost:5000/sonoo");
6.      System.out.println(stub.add(34,4));
7.      }catch(Exception e){}
8.      } }
```

**For running this rmi example,**

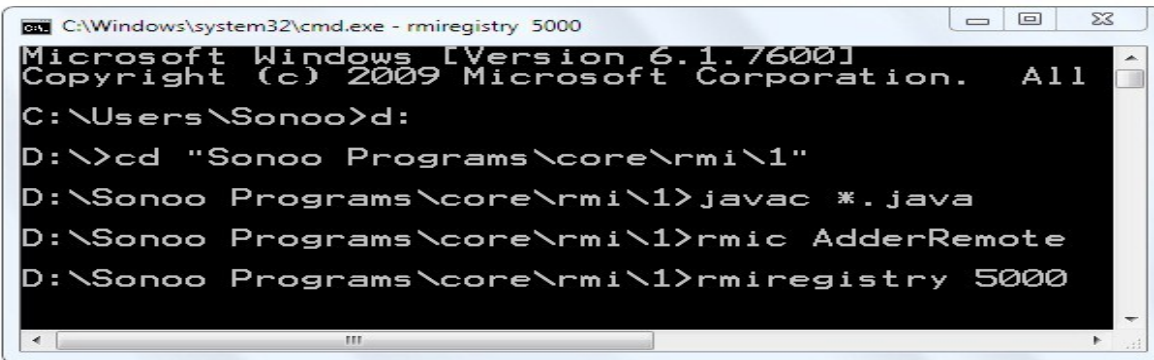
- 1) compile all the java files  
javac \*.java
- 2) create stub and skeleton object by rmic tool  
rmic AdderRemote
- 3) start rmi registry in one command prompt  
rmiregistry 5000
- 4) start the server in another command prompt



---

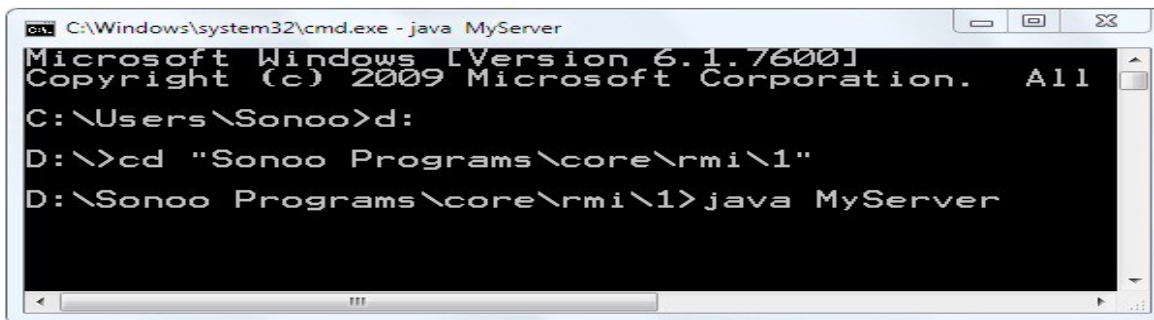
```
java MyServer  
5)start the client application in another command prompt  
java MyClient
```

Output of this RMI example



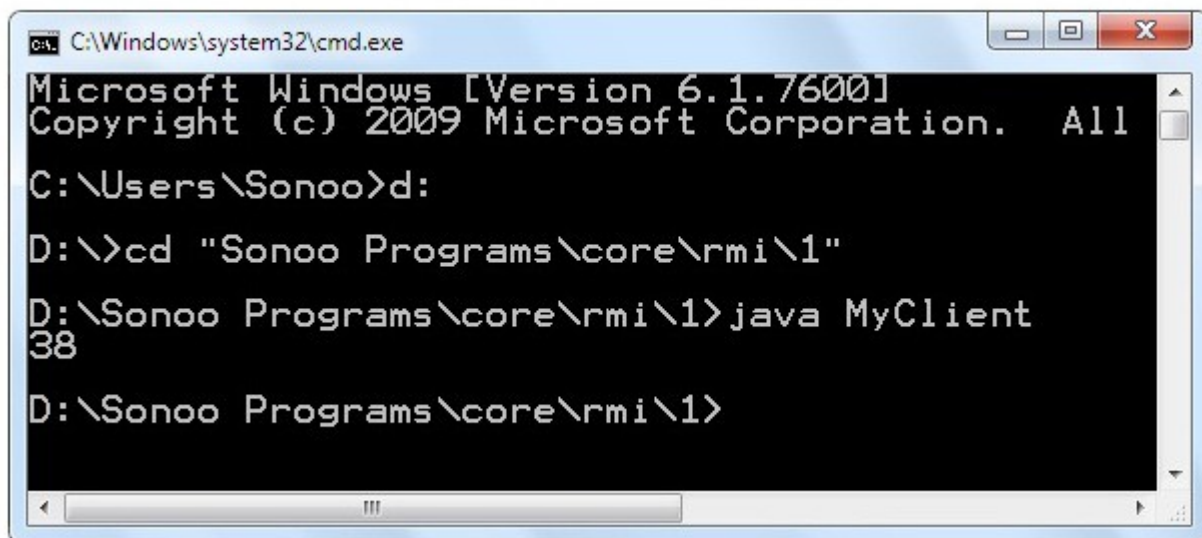
```
C:\Windows\system32\cmd.exe - rmiregistry 5000
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Sonoo>d:
D:\>cd "Sonoo Programs\core\rmi\1"
D:\Sonoo Programs\core\rmi\1>javac *.java
D:\Sonoo Programs\core\rmi\1>rmic AdderRemote
D:\Sonoo Programs\core\rmi\1>rmiregistry 5000
```



```
C:\Windows\system32\cmd.exe - java MyServer
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Sonoo>d:
D:\>cd "Sonoo Programs\core\rmi\1"
D:\Sonoo Programs\core\rmi\1>java MyServer
```



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Sonoo>d:
D:\>cd "Sonoo Programs\core\rmi\1"
D:\Sonoo Programs\core\rmi\1>java MyClient
38
D:\Sonoo Programs\core\rmi\1>
```

## Meaningful example of RMI application with database

Consider a scenario, there are two applications running in different machines. Let's say MachineA and MachineB, machineA is located in United States and MachineB in India. MachineB want to get list of all the customers of MachineA application.

Let's develop the RMI application by following the steps.

## 1) Create the table

First of all, we need to create the table in the database. Here, we are using Oracle10 database.

CUSTOMER400

Table	Data	Indexes	Model	Constraints	Grants	Statistics	UI Defaults	Triggers	Dependencies	SQL
Query	Count Rows	Insert Row								
EDIT	ACC_NO	FIRSTNAME	LASTNAME	EMAIL	AMOUNT					
	67539876	James	Franklin	franklin1james@gmail.com	500000					
	67534876	Ravi	Kumar	ravimalik@gmail.com	98000					
	67579872	Vimal	Jaiswal	jaiswalvimal32@gmail.com	9380000					
						row(s) 1 - 3 of 3				
<a href="#">Download</a>										

## 2) Create Customer class and Remote interface

File: Customer.java

```
1. package com.javatpoint;
2. public class Customer implements java.io.Serializable{
3.     private int acc_no;
4.     private String firstname,lastname,email;
5.     private float amount;
6.     //getters and setters
7. }
```

*Note: Customer class must be Serializable.*

File: Bank.java

```
1. package com.javatpoint;
2. import java.rmi.*;
3. import java.util.*;
4. interface Bank extends Remote{
5.     public List<Customer> getCustomers()throws RemoteException;
6. }
```

## 3) Create the class that provides the implementation of Remote interface

File: BankImpl.java

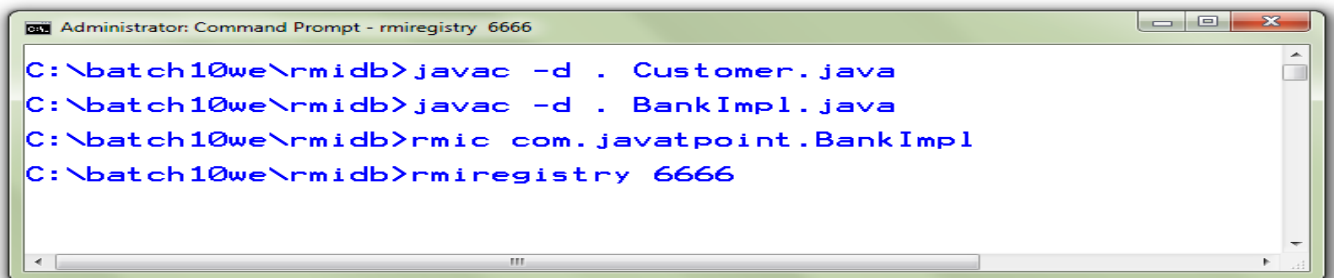
```
1. package com.javatpoint;
2. import java.rmi.*;
3. import java.rmi.server.*;
4. import java.sql.*;
5. import java.util.*;
```

```

6.      class BankImpl extends UnicastRemoteObject implements Bank{
7.      BankImpl()throws RemoteException{}
8.
9.      public List<Customer> getCustomers(){
10.     List<Customer> list=new ArrayList<Customer>();
11.     try{
12.     Class.forName("oracle.jdbc.driver.OracleDriver");
13.     Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
14.     PreparedStatement ps=con.prepareStatement("select * from customer400");
15.     ResultSet rs=ps.executeQuery();
16.
17.     while(rs.next()){
18.     Customer c=new Customer();
19.     c.setAcc_no(rs.getInt(1));
20.     c.setFirstname(rs.getString(2));
21.     c.setLastname(rs.getString(3));
22.     c.setEmail(rs.getString(4));
23.     c.setAmount(rs.getFloat(5));
24.     list.add(c);
25.     }
26.     con.close();
27. }catch(Exception e){System.out.println(e);}
28. return list;
29. }//end of getCustomers() }

```

#### 4) Compile the class rmic tool and start the registry service by rmiregistry tool



```

Administrator: Command Prompt - rmiregistry 6666
C:\batch10we\rmidb>javac -d . Customer.java
C:\batch10we\rmidb>javac -d . BankImpl.java
C:\batch10we\rmidb>rmic com.javatpoint.BankImpl
C:\batch10we\rmidb>rmiregistry 6666

```

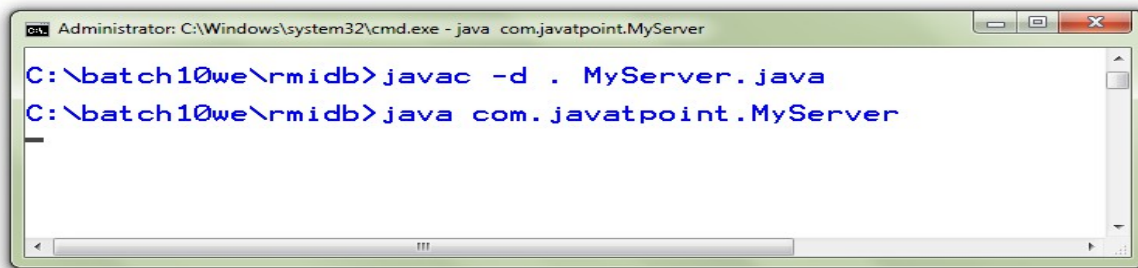
#### 5) Create and run the Server

File: MyServer.java

```

1.      package com.javatpoint;
2.      import java.rmi.*;
3.      public class MyServer{
4.      public static void main(String args[])throws Exception{
5.      Remote r=new BankImpl();
6.      Naming.rebind("rmi://localhost:6666/javatpoint",r);
7.      }}

```

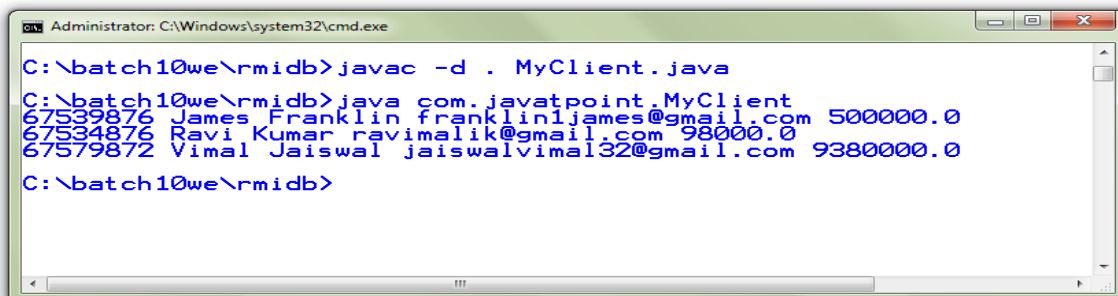


```
Administrator: C:\Windows\system32\cmd.exe - java com.javatpoint.MyServer
C:\batch10we\rmidb>javac -d . MyServer.java
C:\batch10we\rmidb>java com.javatpoint.MyServer
---
```

## 6) Create and run the Client

File: *MyClient.java*

```
1. package com.javatpoint;
2. import java.util.*;
3. import java.rmi.*;
4. public class MyClient{
5.     public static void main(String args[])throws Exception{
6.         Bank b=(Bank)Naming.lookup("rmi://localhost:6666/javatpoint");
7.         List<Customer> list=b.getCustomers();
8.         for(Customer c:list){
9.             System.out.println(c.getAcc_no()+" "+c.getFirstname()+" "+c.getLastname()
10.            +" "+c.getEmail()+" "+c.getAmount());
11.         } }}
```



```
Administrator: C:\Windows\system32\cmd.exe
C:\batch10we\rmidb>javac -d . MyClient.java
C:\batch10we\rmidb>java com.javatpoint.MyClient
67539876 James Franklin franklin1james@gmail.com 500000.0
67534876 Ravi Kumar ravimalik@gmail.com 98000.0
67579872 Vimal Jaiswal jaiswalvimal32@gmail.com 9380000.0
C:\batch10we\rmidb>
```