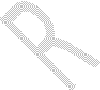
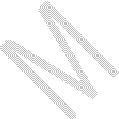
### Q1.Differences Between Arrays And Collections:



|  |  |
| --- | --- |
| **Arrays** | **Collections** |
| **Arrays are Fixed in Size.** | **Collections are Growable in Nature.** |
| **With Respect to Memory Arrays are Not Recommended to Use.** | **With Respect to Memory Collections are Recommended to Use.** |
| **With Respect to Performance Arrays are**  **Recommended to Use.** | **With Respect to Performance Collections are Not**  **Recommended to Use.** |
| **Arrays can Hold Only Homogeneous Data Elements.** | **Collections can Hold Both *Homogeneous* and**  ***Heterogeneous* Elements.** |
| **Arrays can Hold Both Primitives and Objects.** | **Collections can Hold Only Objects but Not Primitives.** |
| **Arrays Concept is Not implemented based on Some Standard Data Structure. Hence Readymade Method Support is Not Available.** | **For every Collection class underlying Data Structure is Available Hence Readymade Method Support is Available for Every Requirement.** |

### Q2.Difference Between Collection (I) and Collections (C):

* **Collection is an Interface which can be used to Represent a Group of Individual Objects as a Single Entity.**
* **Whereas Collections is an Utility Class Present in *java.util* Package to Define Several Utility Methods for Collection Objects.**

**Q3.Differences between *ArrayList* and *Vector*:**

|  |  |
| --- | --- |
| **ArrayList** | **Vector** |
| **Every Method Present Inside ArrayListis Non – Synchronized.** | **Every Method Present in Vector is Synchronized.** |
| **At a Time Multiple Threads are allow to Operate on ArrayList Simultaneously and Hence ArrayList Object is Not**  **Thread Safe.** | **At a Time Only One Thread is allow to Operate on Vector Object and Hence Vector Object is Always Thread Safe.** |
| **Relatively Performance is High because Threads are Not required to Wait.** | **Relatively Performance is Low because Threads are required to Wait.** |
| **Introduced in 1.2 Version and it is Non –**  **Legacy.** | **Introduced in 1.0 Version and it is Legacy.** |

Q4.How to get Synchronized Version of ArrayList Object?

By Default ArrayList Object is Non- Synchronized but we can get Synchronized Version ArrayList Object by using the following Method of Collections Class.

public static List synchronizedList(List l)

Eg:



**Non - Synchronized Version**

**Synchronized Version**

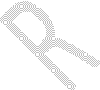
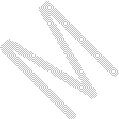
**ArrayListal = new ArrayList ();**

**List l = Collections.synchronizedList(al);**

Similarly we can get Synchronized Version of *Set* and *Map* Objects by using the following Methods of Collection Class.

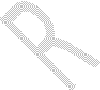
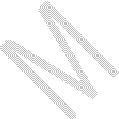
public static Set synchronizedSet(Set s)

public static Map synchronizedMap(Map m)



* **ArrayList is the Best Choice if we want to Perform Retrieval Operation Frequently.**
* **But ArrayList is Worst Choice if Our Frequent Operation is Insertion OR Deletion in the Middle. Because it required Several Shift Operations Internally.**

### Q5.Differences Between Arrays And Collections:



|  |  |
| --- | --- |
| **Arrays** | **Collections** |
| **Arrays are Fixed in Size.** | **Collections are Growable in Nature.** |
| **With Respect to Memory Arrays are Not Recommended to Use.** | **With Respect to Memory Collections are Recommended to Use.** |
| **With Respect to Performance Arrays are**  **Recommended to Use.** | **With Respect to Performance Collections are Not**  **Recommended to Use.** |
| **Arrays can Hold Only Homogeneous Data Elements.** | **Collections can Hold Both *Homogeneous* and**  ***Heterogeneous* Elements.** |
| **Arrays can Hold Both Primitives and Objects.** | **Collections can Hold Only Objects but Not Primitives.** |
| **Arrays Concept is Not implemented based on Some Standard Data Structure. Hence Readymade Method Support is Not Available.** | **For every Collection class underlying Data Structure is Available Hence Readymade Method Support is Available for Every Requirement.** |

Q.6Write a Program to Insert StringBuffer Objects into the TreeSet where Sorting Order is Alphabetical Order:

**importjava.util.\*; classTreeSetDemo {**

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

**TreeSet t = new TreeSet(new MyComparator1()); t.add(new StringBuffer("A"));**

**t.add(new StringBuffer("Z")); t.add(new StringBuffer("K")); t.add(new StringBuffer("L")); System.out.println(t);**

**}**

**}**

**class MyComparator1 implements Comparator { publicint compare(Object obj1, Object obj2) {**

**String s1 = obj1.toString(); String s2 = obj2.toString();**

**return s1.compareTo(s2); //[A, K, L, Z]**

**}**

**}**

:

Write a Program to Insert String and StringBuffer Objects into the TreeSet where Sorting Order is Increasing Length Order. If 2 Objects having Same Length then Consider their Alphabetical Order

**importjava.util.\*; classTreeSetDemo {**

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

**TreeSet t = new TreeSet(new MyComparator()); t.add("A");**

**t.add(new StringBuffer("ABC")); t.add(new StringBuffer("AA")); t.add("XX");**

**t.add("ABCE");**

**t.add("A"); System.out.println(t);**

**}**

**}**

**classMyComparator implements Comparator { publicint compare(Object obj1, Object obj2) {**

**String s1 = obj1.toString(); String s2 = obj2.toString(); int i1 = s1.length();**

**int i2 = s2.length(); if(i1 < i2)return -1; else if(i1 > i2)return 1;**

**elsereturn s1.compareTo(s2); //[A, AA, XX, ABC, ABCE]**

**}**

**}**

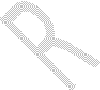
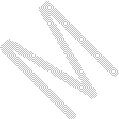
Write a Program to Insert Employee Objects into the TreeSet where DNSO is Based on Ascending Order of EmployeeId and Customized Sorting Order is Based on Alphabetical Order of Names:

importjava.util.\*;

class Employee implements Comparable { String name;

inteid;

Employee(String name, inteid) { this.name = name; this.eid = eid;



}

public String toString() { return name+"-----"+eid;} publicintcompareTo(Object obj) {

int eid1 = this.eid;

Employee e = (Employee)obj; int eid2 = e.eid;

if(eid1 < eid2) return -1;

else if(eid1 > eid2) return 1; else return 0;

}

}

classCompComp {

public static void main(String[] args) {

Employee e1 = new Employee("Nag", 100); Employee e2 = new Employee("Bala", 200); Employee e3 = new Employee("Chiru", 50); Employee e4 = new Employee("Venki", 150); Employee e5 = new Employee("Nag", 100); TreeSet t = new TreeSet();

t.add(e1);

t.add(e2);

t.add(e3);

t.add(e4);

t.add(e5); System.out.println(t);

TreeSet t1 = new TreeSet(new MyComparator()); t1.add(e1);

t1.add(e2);

t1.add(e3);

t1.add(e4);

t1.add(e5); System.out.println(t1);

}

}

### Differences between and HashMap and Hashtable:

|  |  |
| --- | --- |
| **HashMap** | **Hashtable** |
| **No Method Present in HashMap is**  **Synchronized.** | **Every Method Present in Hashtable is**  **Synchronized.** |
| **At a Time Multiple Threads are allowed to Operate on HashMap Object simultaneously and Hence it is Not Thread**  **Safe.** | **At a Time Only One Thread is allowed to Operate on the Hashtable Object and Hence it is Thread Safe.** |
| **Relatively Performance is High.** | **Relatively Performance is Low.** |
| **null is allowed for Both Keys and Values.** | **null is Not allowed for Both Keys and Values. Otherwise we will get NPE.** |
| **Introduced in 1.2 Version and it is**  **Non – Legacy.** | **Introduced in 1.0 Version and it is Legacy.** |

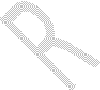
### Differences between and HashMap and LinkedHashtable:

|  |  |
| --- | --- |
| **HashMap** | **LinkedHashMap** |
| **The Underlying Data**  **Structure is Hashtable.** | **The Underlying Data Structure is**  **Combination of Hashtable and LinkedList.** |
| **Insertion is Not Preserved.** | **Insertion Order is Preserved.** |
| **Introduced in 1.2 Version.** | **Introduced in 1.4 Version.** |

## Difference between HashMap and ConcurrentHashMap

|  |  |
| --- | --- |
| **HashMap** | **ConcurrentHashMap** |
| **It is Not Thread Safe.** | **It is Thread Safe.** |
| **Relatively Performance is High because Threads are Not required to wait to Operate**  **on HashMap.** | **Relatively Performance is Low because Some Times Threads are required to wait to Operate**  **on ConcurrentHashMap.** |
| **While One Thread iterating HashMap the Other Threads are Not allowed to Modify Map Objects Otherwise we will get Runtime Exception Saying**  **ConcurrentModificationException.** | **While One Thread iterating ConcurrentHashMap the Other Threads are allowed to Modify Map Objects in Safe Manner and it won’t throw**  **ConcurrentModificationException.** |
| **Iterator of HashMap is Fail-Fast and it throws ConcurrentModificationException.** | **Iterator of ConcurrentHashMap is Fail-Safe**  **and it won’t throwsConcurrentModificationException.** |
| **null is allowed for Both Keys and Values.** | **null is Not allowed for Both Keys and Values. Otherwise we will get NullPointerException.** |
| **Introduced in 1.2 Version.** | **Introduced in 1.5 Version.** |

## Differences between ArrayList and CopyOnWriteArrayList



|  |  |
| --- | --- |
| **ArrayList** | **CopyOnWriteArrayList** |
| **It is Not Thread Safe.** | **It is Not Thread Safe because Every Update Operation will be performed on Separate**  **cloned Coy.** |
| **While One Thread iterating List Object, the Other Threads are Not allowed to Modify List Otherwise we will get**  **ConcurrentModificationException.** | **While One Thread iterating List Object, the Other Threads are allowed to Modify List in Safe Manner and we won’t get**  **ConcurrentModificationException.** |
| **Iterator is Fail-Fsat.** | **Iterator is Fail-Safe.** |
| **Iterator of ArrayList can Perform Remove Operation.** | **Iterator of CopyOnWriteArrayList can’t Perform Remove Operation Otherwise we will get RuntimeException:**  **UnsupportedOperationException.** |
| **Introduced in 1.2 Version.** | **Introduced in 1.5 Version.** |

## Q.Differences between CopyOnWriteArraySet() and synchronizedSet()

|  |  |
| --- | --- |
| **CopyOnWriteArraySet()** | **synchronizedSet()** |
| **It is Thread Safe because Every Update Operation will be performed on Separate Cloned Copy.** | **It is Thread Safe because at a Time Only One Thread can Perform Operation.** |
| **While One Thread iterating Set, the Other Threads are allowed to Modify and we won’t get ConcurrentModificationException.** | **While One Thread iterating, the Other Threads are Not allowed to Modify Seta Otherwise we will get**  **ConcurrentModificationException.** |
| **Iterator is Fail Safe.** | **Iterator is Fail Fast.** |
| **Iterator can Perform Only Read Operation and can’t Perform Remove Operation Otherwise we will get RuntimeException**  **Saying UnsupportedOperationException.** | **Iterator can Perform Both Read and Remove Operations.** |
| **Introduced in 1.5 Version.** | **Introduced in 1.7 Version.** |

**Q. what is difference Enumeration and Iterator?**

|  |  |
| --- | --- |
| **Enumeration** | **Iterator** |
| 1. It is legacy interface and introduced in 1.0 version | 1 It is non-legacy and introduced in 1.2 version |
| 2Applicable only for legacy classes and it is not universal cursor | 2Applicable for any Collection implemented class object. |

|  |  |
| --- | --- |
| 3While iterating the elements we are not allowed to remove the objects just we can perform only read operation | 3While iterating we can perform removal also in addition to read operation. |
| 4By using elements() method we can get  Enumeration object | 4. By using iterator() method we can get  Iterator object |

**What is difference between Iterator and ListIterator?**

o ListIterator is the child interface of the Iterator

o Iterator is the single direction cursor where as ListIterator is bidirectional

cursor.

oWhile iterating the elements by Iterator we can perform only read and remove operations. But by using ListIterator we can perform read,removal, replace

and addition of new objects also.

o Iterator is applicable for every Collecton implemented class object but

ListIterator is applicable only for List implemented class objects.

o Iterator can be get by using iterator() of Collection interface where as

ListIterator can be get by using listIterator() method of List interface

o both are introduced in 1.2 version

**What are differences between List and Set interfaces?**

|  |  |  |
| --- | --- | --- |
| **List** | **Set** | |
| 1Insertion Order is preserved | 1Insertion Order is not preserved | |
| 2Duplicate Objects are allowed | 2 | Duplicate Objects are not allowed |
| 3The implemented classes are | 3 The implemented classes are  HashSet, LinkedHashSet and Tree | |
| ArrayList,LinkedList , Vector and Stack  classes |

**What are differences between List and Set interfaces?**

|  |  |  |
| --- | --- | --- |
| **List** | **Set** | |
| 1Insertion Order is preserved | 1Insertion Order is not preserved | |
| 2Duplicate Objects are allowed | 2 | Duplicate Objects are not allowed |
| 3The implemented classes are | 3 The implemented classes are  HashSet, LinkedHashSet and Tree | |
| ArrayList,LinkedList , Vector and Stack  classes |

What is the difference between Error and Exception?

Error Exception

|  |  |
| --- | --- |
| 1. Error is a problem at runtime, for  which we are unable to provide solutions programmatically. | Exception is a problem, for which, we  are able to provide solution programmatically. |
| Ex: JVM internal Problem StackOverFlowError InSufficientMainMemory | Ex: ArithmeticException  NullPointerException  ArrayIndexOutOfBoundsException |

Q)What is the difference between checked Exception and Unchecked

Exception?

1.Checked Exception is an exception identified by the compiler at compilation time[occurred at runtime, not at compilation time].

**NOTE:while Java program compilations, compiler will recognize same situations to get exceptions at runtime then that exceptions are called as Checked Exceptions.**

Unchecked exceptions are the exceptions recognized by the JVM at runtime, not a compiler at compilation time.

2.Runtime Exceptions and its subclasses, error and its subclasses are treated as Unchecked exceptions and the remaining exception classes are treated as checked exception

There are two types of checked exceptions:

a)pure checked exceptions b)partially checked exceptions.

**Q7. What is the difference between t.start() and t.run() method?**

Ans. In the case of t.start() method, a new thread will be created which is responsible for the execution of run() method.

But in the case of t.run() method no new thread will be created main thread executes run() method just like a normal method call.

**Q13. Explain life cycle of a Thread?**

Ans. Once we create a Thread object then the Thread is said to be in New/Born state once we call t.start() method now the Thread will be entered into ready/Runnable state that is Thread is ready to execute. If Thread Scheduler allocates CPU now the Thread will entered into the Running state and start execution of run() method. After completing run() method the Thread entered into Dead State.

**What is the difference between synchronized method and static synchronized method?**

Ans. If a Thread wants to execute a **synchronized method** first it has to get the **lock of the object**. Once a Thread got the lock then it is allow to execute any **synchronized method on that object**.If a Thread wants to execute **static synchronized method**that Thread has to

get **class level lock** once a Thread got class level lock then only it is allow to execute **static synchronized method**.

Q)What are the differences between “throw” and “throws” keywords?

Ans:

|  |  |
| --- | --- |
| 1.”throw” keyword can be used to rise the exceptions intentionally as  Per the application reqirement. | “throws” keyword will by pass the exceptions from the present method  to the caller method. |
| 2.”throw” keyword will be utilized in  method body. | “throws” keyword will be used in  method declarations or in method prototype (or) in method header part. |
| 3.”throw” keyword will allow only one exception class name. | “throws” keyword will allow more than one exception class name. |

Q)What is the difference between "final","finally" and "finalize" in JAVA?

1."final" is a keyword it can be used to declare constant expressions.

a)final variable:It will not allow modifications over its value. b)final methods:It will not allow method overriding.

c)final class:It will not be extended.

2.finally block:It is a part in try-catch-finally syntax,it will include some instructions,which must be executed by JVM irrespective of getting exception from try block and irrespective of executing catch block.

3.finalize():It is a method in java.lang.Object class,it will be executed before destroyingobjects inorder to give final notification to the user about to destroy objects.