**Core Java & Java frameworks**

**DURATION: 12 Weeks**

**COURSE CONTENTS:**

**First Week:**

* What is java, Features of java, Difference between C++ and java, Internal details of any java program (What happen at compile and runtime), Difference between JDK, JRE, JVM, Internal details of JVM, Why java uses Unicode system.

**Second week:**

* Installation of java, what is Class, what is object, OOPS concept (Polymorphism, Abstraction, Encapsulation, Inheritance, Aggregation, Composition), Variables, Data Types, Methods, Constants, Operators, Literals, Packages.

**Third Week:**

* **Constructors**, Loops, Decision making (if-else, switch) Arrays, Exception handling, Modifiers.

**Fourth Week:**

* Interfaces, Abstract class, Collections, JDBC, Wrapper classes, Casting, Recursion.

**Fifth Week:**

* Java keywords (final, finally, super, this, native, transient, throw, throws), String, String buffer, String builder.
  + Difference between final, finally, finalize();

**Sixth Week:**

* Introduction to SQL, Joins, insert, update, delete, drop, aggregate function, Introduction to functions, triggers, procedures, indexes.

**Seventh week:**

* Introduction to Hibernate, CRUD operations in hibernate, HQL, Hibernate criteria language,

Basic Hibernate questions (get and load difference, save and persist etc.)

**Eight Week:**

* Introduction to web server (Apache Tomcat) and application server (Jboss), Introduction to maven, create a maven project and run it on Apache tomcat and jboss.
* Introduction to java web technologies (JSp & Servlets, Struts 2, Spring), brief introduction of spring mvc and spring security.

**Ninth, Tenth, Eleventh and Twelfth week:**

* Assignments will be given based on course.

Polymorphism, Abstraction, Encapsulation, Inheritance,

Constructors, Loops, Decision making (if-else, switch) Arrays, Exception handling, Modifiers.

Interfaces, Abstract class, Collections, JDBC, Wrapper classes, Casting, Recursion.

Java keywords (final, finally, super, this, native, transient, throw, throws), String, String buffer, String builder.

Difference between final, finally, finalize();

Query:

Public in other class is displaying issue

Java Arraylist

Java Linkedlist

ArrayList Vs LinkedList

JavaHashSet

Java LinkHashSet

Java TreeSet

Java Map interface

Java HashMap

Java TreeHashMap

Java TreeMap

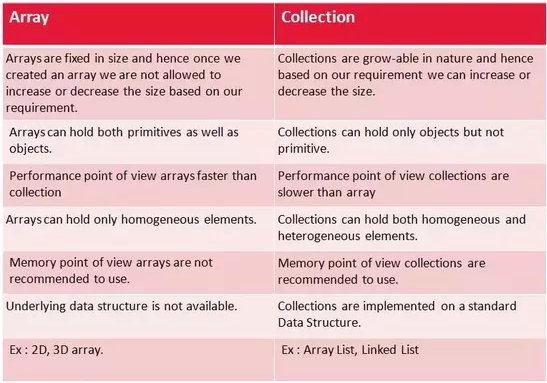
Java HashTable

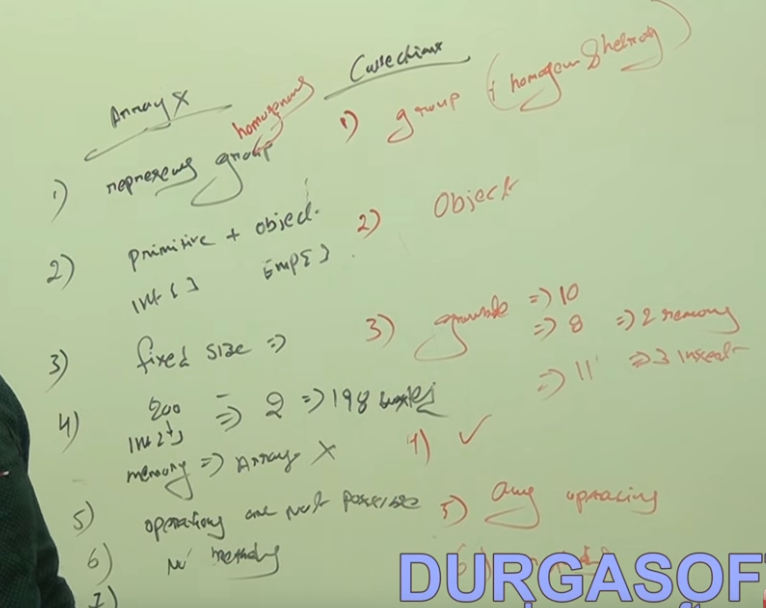
Java HashTable

HashMap Vs HashTable

ArracyList Vs Vactor

///////////////////////////////////////////





https://www.youtube.com/watch?v=BMWrqRMgrEk

**Java Collections**

**18. What are the basic interfaces of Java Collections Framework ?**

[Java Collections Framework](http://docs.oracle.com/javase/7/docs/technotes/guides/collections/overview.html) provides a well designed set of interfaces and classes that support operations on a collections of objects. The most basic interfaces that reside in the Java Collections Framework are:

* [Collection](http://docs.oracle.com/javase/7/docs/api/java/util/Collection.html), which represents a group of objects known as its elements.
* [Set](http://docs.oracle.com/javase/7/docs/api/java/util/Set.html), which is a collection that cannot contain duplicate elements.
* [List](http://docs.oracle.com/javase/7/docs/api/java/util/List.html), which is an ordered collection and can contain duplicate elements.
* [Map](http://docs.oracle.com/javase/7/docs/api/java/util/Map.html), which is an object that maps keys to values and cannot contain duplicate keys.

**19. Why Collection doesn’t extend Cloneable and Serializable interfaces ?**

The [Collection](http://docs.oracle.com/javase/7/docs/api/java/util/Collection.html) interface specifies groups of objects known as elements. Each concrete implementation of a [Collection](http://docs.oracle.com/javase/7/docs/api/java/util/Collection.html) can choose its own way of how to maintain and order its elements. Some collections allow duplicate keys, while some other collections don’t. The semantics and the implications of either cloning or serialization come into play when dealing with actual implementations. Thus, the concrete implementations of collections should decide how they can be cloned or serialized.

**20. What is an Iterator ?**

The [Iterator](http://docs.oracle.com/javase/7/docs/api/java/util/Iterator.html) interface provides a number of methods that are able to iterate over any [Collection](http://docs.oracle.com/javase/7/docs/api/java/util/Collection.html). Each Java [Collection](http://docs.oracle.com/javase/7/docs/api/java/util/Collection.html) contains the [Iterator](http://docs.oracle.com/javase/7/docs/api/java/util/Iterator.html) method that returns an [Iterator](http://docs.oracle.com/javase/7/docs/api/java/util/Iterator.html) instance. Iterators are capable of removing elements from the underlying collection during the iteration.

**21. What differences exist between Iterator and ListIterator ?**

The differences of these elements are listed below:

* An [Iterator](http://docs.oracle.com/javase/7/docs/api/java/util/Iterator.html) can be used to traverse the [Set](http://docs.oracle.com/javase/7/docs/api/java/util/Set.html) and [List](http://docs.oracle.com/javase/7/docs/api/java/util/List.html) collections, while the [ListIterator](http://docs.oracle.com/javase/7/docs/api/java/util/ListIterator.html) can be used to iterate only over [List](http://docs.oracle.com/javase/7/docs/api/java/util/List.html) .
* The [Iterator](http://docs.oracle.com/javase/7/docs/api/java/util/Iterator.html) can traverse a collection only in forward direction, while the [ListIterator](http://docs.oracle.com/javase/7/docs/api/java/util/ListIterator.html) can traverse a [List](http://docs.oracle.com/javase/7/docs/api/java/util/List.html) in both directions.
* The [ListIterator](http://docs.oracle.com/javase/7/docs/api/java/util/ListIterator.html) implements the [Iterator](http://docs.oracle.com/javase/7/docs/api/java/util/Iterator.html) interface and contains extra functionality, such as adding an element, replacing an element, getting the index position for previous and next elements, etc.

**22. What is difference between fail-fast and fail-safe ?**

The [Iterator's](http://docs.oracle.com/javase/7/docs/api/java/util/Iterator.html) fail-safe property works with the clone of the underlying collection and thus, it is not affected by any modification in the collection. All the collection classes in java.util package are fail-fast, while the collection classes in java.util.concurrent are fail-safe. Fail-fast iterators throw a [ConcurrentModificationException](http://examples.javacodegeeks.com/java-basics/exceptions/java-util-concurrentmodificationexception-how-to-handle-concurrent-modification-exception/), while fail-safe iterator never throws such an exception.

**23. How HashMap works in Java ?**

[A HashMap in Java stores key-value pairs](http://www.javacodegeeks.com/2014/03/how-hashmap-works-in-java.html). The [HashMap](http://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html) requires a hash function and uses hashCode and equals methods, in order to put and retrieve elements to and from the collection respectively. When the put method is invoked, the [HashMap](http://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html) calculates the hash value of the key and stores the pair in the appropriate index inside the collection. If the key exists, its value is updated with the new value. Some important characteristics of a [HashMap](http://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html) are its capacity, its load factor and the threshold resizing.

**24. What is the importance of hashCode() and equals() methods ?**

A [HashMap](http://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html) in Java uses the hashCode and equals methods to determine the index of the key-value pair. These methods are also used when we request the value of a specific key. If these methods are not implemented correctly, two different keys might produce the same hash value and thus, will be considered as equal by the collection. Furthermore, these methods are also used to detect duplicates. Thus, the implementation of both methods is crucial to the accuracy and correctness of the [HashMap](http://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html).

**25. What differences exist between HashMap and Hashtable ?**

Both the [HashMap](http://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html) and [Hashtable](http://docs.oracle.com/javase/7/docs/api/java/util/Hashtable.html) classes implement the Map interface and thus, have very similar characteristics. However, they differ in the following features:

* A [HashMap](http://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html) allows the existence of null keys and values, while a [Hashtable](http://docs.oracle.com/javase/7/docs/api/java/util/Hashtable.html) doesn’t allow neither null keys, nor null values.
* A [Hashtable](http://docs.oracle.com/javase/7/docs/api/java/util/Hashtable.html) is synchronized, while a [HashMap](http://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html) is not. Thus, [HashMap](http://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html) is preferred in single-threaded environments, while a [Hashtable](http://docs.oracle.com/javase/7/docs/api/java/util/Hashtable.html) is suitable for multi-threaded environments.
* A [HashMap](http://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html) provides its set of keys and a Java application can iterate over them. Thus, a [HashMap](http://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html) is fail-fast. On the other hand, a [Hashtable](http://docs.oracle.com/javase/7/docs/api/java/util/Hashtable.html) provides an [Enumeration](http://docs.oracle.com/javase/7/docs/api/java/util/Enumeration.html) of its keys.
* The [Hashtable](http://docs.oracle.com/javase/7/docs/api/java/util/Hashtable.html) class is considered to be a legacy class.

**26. What is difference between Array and ArrayList ? When will you use Array over ArrayList ?**

The Array and ArrayList classes differ on the following features:

* [Arrays](http://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html) can contain primitive or objects, while an [ArrayList](http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html) can contain only objects.
* [Arrays](http://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html) have fixed size, while an [ArrayList](http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html) is dynamic.
* An [ArrayList](http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html) provides more methods and features, such as addAll, removeAll, iterator, etc.
* For a list of primitive data types, the collections use autoboxing to reduce the coding effort. However, this approach makes them slower when working on fixed size primitive data types.

**27. What is difference between ArrayList and LinkedList ?**

Both the [ArrayList](http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html) and [LinkedList](http://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html) classes implement the List interface, but they differ on the following features:

* An [ArrayList](http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html) is an index based data structure backed by an [Array](http://docs.oracle.com/javase/7/docs/api/java/lang/reflect/Array.html). It provides random access to its elements with a performance equal to O(1). On the other hand, a [LinkedList](http://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html) stores its data as list of elements and every element is linked to its previous and next element. In this case, the search operation for an element has execution time equal to O(n).
* The Insertion, addition and removal operations of an element are faster in a [LinkedList](http://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html) compared to an [ArrayList](http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html), because there is no need of resizing an array or updating the index when an element is added in some arbitrary position inside the collection.
* A [LinkedList](http://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html) consumes more memory than an [ArrayList](http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html), because every node in a [LinkedList](http://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html) stores two references, one for its previous element and one for its next element. Check also our article [ArrayList](http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html) vs. [LinkedList](http://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html).

**28. What is Comparable and Comparator interface ?**

List their differences. Java provides the [Comparable](http://docs.oracle.com/javase/7/docs/api/java/lang/Comparable.html) interface, which contains only one method, called [compareTo](http://docs.oracle.com/javase/7/docs/api/java/lang/Comparable.html#compareTo(T)). This method compares two objects, in order to impose an order between them. Specifically, it returns a negative integer, zero, or a positive integer to indicate that the input object is less than, equal or greater than the existing object. Java provides the [Comparator](http://docs.oracle.com/javase/7/docs/api/java/util/Comparator.html) interface, which contains two methods, called [compare](http://docs.oracle.com/javase/7/docs/api/java/util/Comparator.html#compare(T,%20T)) and [equals](http://docs.oracle.com/javase/7/docs/api/java/util/Comparator.html#equals(java.lang.Object)). The first method compares its two input arguments and imposes an order between them. It returns a negative integer, zero, or a positive integer to indicate that the first argument is less than, equal to, or greater than the second. The second method requires an object as a parameter and aims to decide whether the input object is equal to the comparator. The method returns true, only if the specified object is also a comparator and it imposes the same ordering as the comparator.

**29. What is Java Priority Queue ?**

The [PriorityQueue](http://docs.oracle.com/javase/7/docs/api/java/util/PriorityQueue.html) is an unbounded queue, based on a priority heap and its elements are ordered in their natural order. At the time of its creation, we can provide a Comparator that is responsible for ordering the elements of the [PriorityQueue](http://docs.oracle.com/javase/7/docs/api/java/util/PriorityQueue.html). A [PriorityQueue](http://docs.oracle.com/javase/7/docs/api/java/util/PriorityQueue.html) doesn’t allow null values, those objects that doesn’t provide natural ordering, or those objects that don’t have any comparator associated with them. Finally, the Java [PriorityQueue](http://docs.oracle.com/javase/7/docs/api/java/util/PriorityQueue.html) is not thread-safe and it requires O(log(n)) time for its enqueing and dequeing operations.

**30. What do you know about the big-O notation and can you give some examples with respect to different data structures ?**

The Big-O notation simply describes how well an algorithm scales or performs in the worst case scenario as the number of elements in a data structure increases. The Big-O notation can also be used to describe other behavior such as memory consumption. Since the collection classes are actually data structures, we usually use the Big-O notation to chose the best implementation to use, based on time, memory and performance. Big-O notation can give a good indication about performance for large amounts of data.

**31. What is the tradeoff between using an unordered array versus an ordered array ?**

The major advantage of an ordered array is that the search times have time complexity of O(log n), compared to that of an unordered array, which is O (n). The disadvantage of an ordered array is that the insertion operation has a time complexity of O(n), because the elements with higher values must be moved to make room for the new element. Instead, the insertion operation for an unordered array takes constant time of O(1).

**32. What are some of the best practices relating to the Java Collection framework ?**

* Choosing the right type of the collection to use, based on the application’s needs, is very crucial for its performance. For example if the size of the elements is fixed and know a priori, we shall use an [Array](http://docs.oracle.com/javase/7/docs/api/java/lang/reflect/Array.html), instead of an [ArrayList](http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html).
* Some collection classes allow us to specify their initial capacity. Thus, if we have an estimation on the number of elements that will be stored, we can use it to avoid rehashing or resizing.
* Always use Generics for type-safety, readability, and robustness. Also, by using Generics you avoid the [ClassCastException](http://docs.oracle.com/javase/7/docs/api/java/lang/ClassCastException.html) during runtime.
* Use immutable classes provided by the Java Development Kit (JDK) as a key in a Map, in order to avoid the implementation of the [hashCode](http://docs.oracle.com/javase/7/docs/api/java/lang/Object.html#hashCode%28%29) and equals methods for our custom class.
* Program in terms of interface not implementation.
* Return zero-length collections or arrays as opposed to returning a null in case the underlying collection is actually empty.

**33. What’s the difference between Enumeration and Iterator interfaces ?**

[Enumeration](http://docs.oracle.com/javase/7/docs/api/java/util/Enumeration.html) is twice as fast as compared to an Iterator and uses very less memory. However, the [Iterator](http://docs.oracle.com/javase/7/docs/api/java/util/Iterator.html) is much safer compared to [Enumeration](http://docs.oracle.com/javase/7/docs/api/java/util/Enumeration.html), because other threads are not able to modify the collection object that is currently traversed by the iterator. Also, [Iterators](http://docs.oracle.com/javase/7/docs/api/java/util/Iterator.html) allow the caller to remove elements from the underlying collection, something which is not possible with [Enumeration](http://docs.oracle.com/javase/7/docs/api/java/util/Enumeration.html).

**34. What is the difference between HashSet and TreeSet ?**

The [HashSet](http://docs.oracle.com/javase/7/docs/api/java/util/HashSet.html) is Implemented using a hash table and thus, its elements are not ordered. The add, remove, and contains methods of a [HashSet](http://docs.oracle.com/javase/7/docs/api/java/util/HashSet.html) have constant time complexity O(1). On the other hand, a [TreeSet](http://docs.oracle.com/javase/7/docs/api/java/util/TreeSet.html) is implemented using a tree structure. The elements in a [TreeSet](http://docs.oracle.com/javase/7/docs/api/java/util/TreeSet.html) are sorted, and thus, the add, remove, and contains methods have time complexity of O(logn).

Jenkins

Github

Anu batta

001301598987

Icic0000013