Synchronization

What is the synchronization?

* The capability to control the access of multiple threads to any shared resource.
* At a time will be executive one “Thread” and remaining threads is blocks.

Use of Synchronization?

We can use synchronizations is two ways.

1. Methods
2. Block

1.Methods

We can use synchronization key word in method statement is called ‘methods synchronization’.

Methods synchronizations is use both static and non static method.

Static method;-

Examples:-

Public static void add(){

}

Non Static method:-

Examples:-

Public static void add() {

}

2.Block:-

We can use synchronization key word before block is called ‘Block synchronization’.

Block synchronization is use both static and non static blocks.

Static block:-

Synchronization static {

}

Non static block:-

Synchronization { }

Thread Life Cycle:-



1. **New:** Whenever a new thread is created, it is always in the new state.
2. **Active:** When a thread invokes the start() method, it moves from the new state to the active state. The active state contains two states within it: one is **runnable** , and the other is **running**.

\* **Runnable:** A thread,that is ready to run is then moved to the runnable state.

\* **Running:** When the thread gets the CPU, it moves from the runnable to the running state.

1. **Blocked or Waiting:** Whenever a thread is inactive for a span of time (not permanently) then, either the thread is in the blocked state or is in the waiting state.
2. **Timed Waiting:** Sometimes, waiting for leads to starvation. The sleep() method puts the thread in the timed wait state.
3. **Terminated:** A terminated thread means the thread is no more in the system. In other words, the thread is dead, and there is no way one can respawn (active after kill) the dead thread.

**MAPS:**-

* 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.
* It models the mathematical *function* abstraction. The Map interface includes methods for basic operations (such as put, get, remove, contains Key, contains Value, size, and empty), bulk operations (such as put All and clear), and collection views (such as key Set, entry Set, and values).
* The Java platform contains three general purpose Map implementations: [Hash Map](https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html), [Tree Map](https://docs.oracle.com/javase/8/docs/api/java/util/TreeMap.html), and [Linked Hash Map](https://docs.oracle.com/javase/8/docs/api/java/util/LinkedHashMap.html).
* Their behavior and performance are precisely analogous to Hash Set, Tree Set, and Linked Hash Set,

Java Map Hierarchy

1.HashMap :-

> Java **Hash Map** class implements the Map interface which allows us to store key and value pair, where keys should be unique. If you try to insert the duplicate key, it will replace the element of the corresponding key. It is easy to perform opera tions using the key index like updation, deletion, etc. Hash Map class is found in the java.util package.

Hash Map in Java is like the legacy Hash table class, but it is not synchronized. It allows us to store the null elements as well, but there should be only one null key. Since Java 5, it is denoted as Hash Map<K,V>, where K stands for key and V for value. It inherits the Abstract Map class and implements the Map interface.

Points to remember

* Java Hash Map contains values based on the key.
* Java Hash Map contains only unique keys.
* Java Hash Map may have one null key and multiple null values.
* Java Hash Map is non synchronized.
* Java Hash Map maintains no order.
* The initial default capacity of Java Hash Map class is 16 with a load factor of 0.75.

2. Linked Hash Map:-

> Java Linked Hash Map class is Hash table and Linked list implementation of the Map interface, with predictable iteration order. It inherits Hash Map class and implements the Map interface.

Points to remember

* Java Linked Hash Map contains values based on the key.
* Java Linked Hash Map contains unique elements.
* Java Linked Hash Map may have one null key and multiple null values.
* Java Linked Hash Map is non synchronized.
* Java Linked Hash Map maintains insertion order.
* The initial default capacity of Java Hash Map class is 16 with a load factor of 0.75.

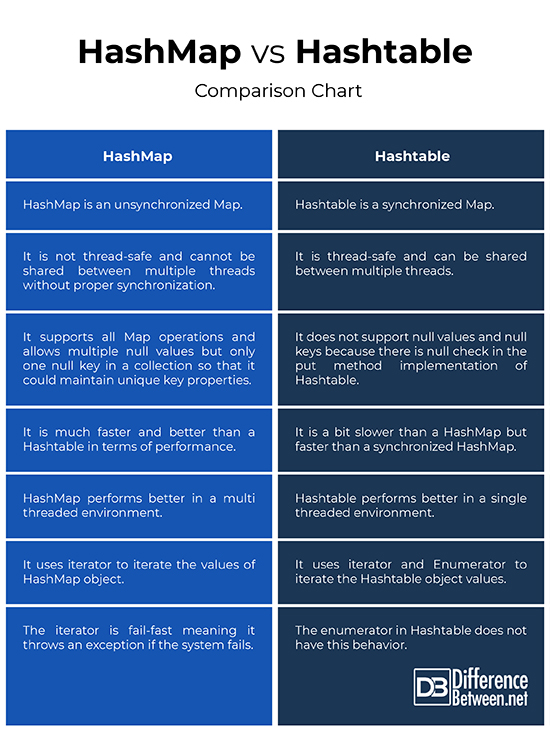
3. Tree Map:-

> Java Tree Map class is a red-black tree based implementation. It provides an efficient means of storing key-value pairs in sorted order.

The important points about Java Tree Map class are:

* Java Tree Map contains values based on the key. It implements the Navigable Map interface and extends Abstract Map class.
* Java Tree Map contains only unique elements.
* Java Tree Map cannot have a null key but can have multiple null values.
* Java Tree Map is non synchronized.
* Java Tree Map maintains ascending order.

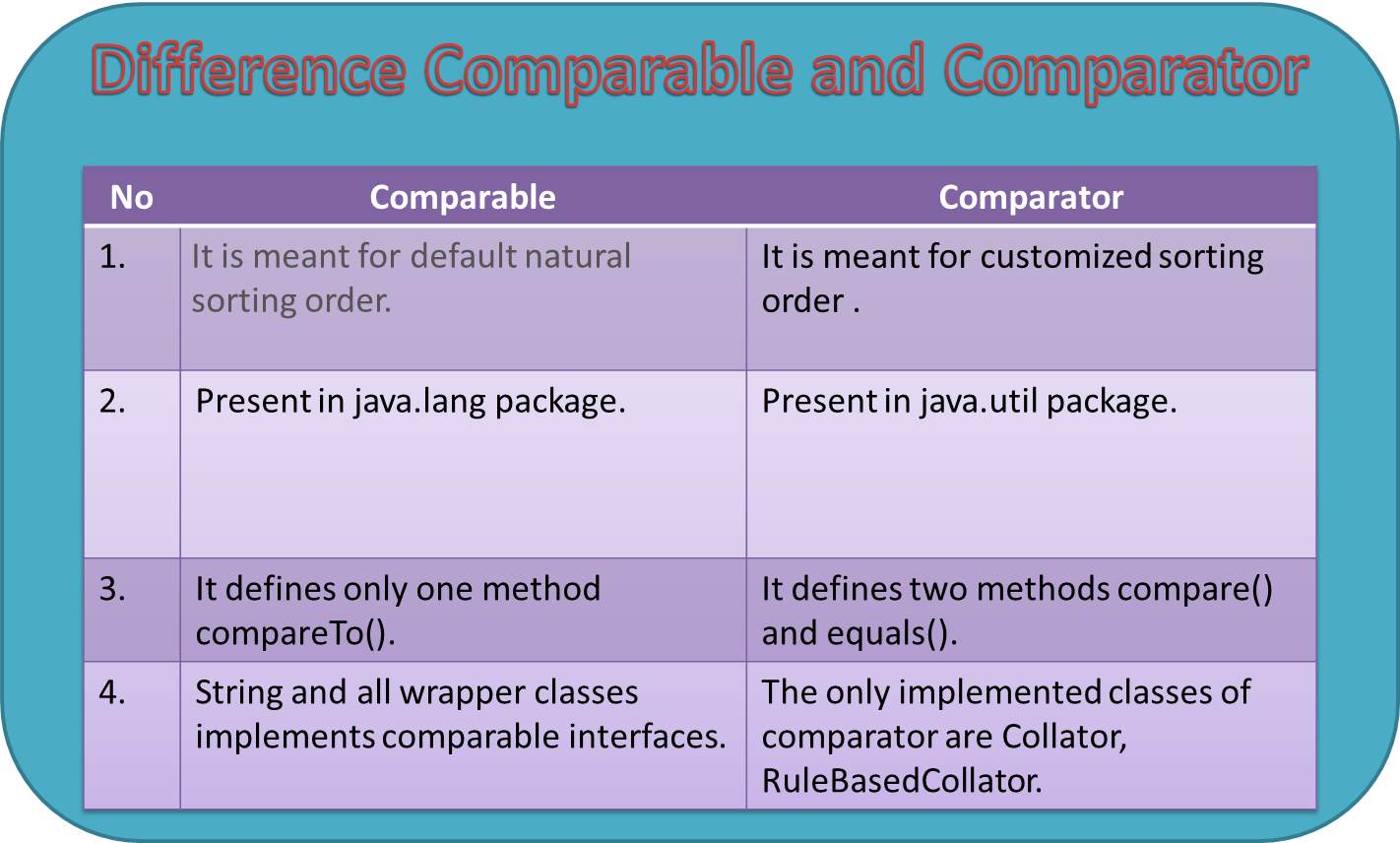
**Difference B/W Hash map And Hash Table:-**



**Comparator:-**

* Java Comparator is an interface for sorting Java [objects](https://www.techtarget.com/searchapparchitecture/definition/object). Invoked by “java.util.comparator,” Java Comparator compares two Java objects in a “compare(Object 01, Object 02)” format.
* Using configurable methods, Java Comparator can compare objects to return an integer based on a positive, equal or negative comparison. Since it is not limited to comparing numbers, this can allow Java Comparator to be set up to order lists alphabetically or numerically. With java.io.Serializable, Java comparator can also be used to successfully order serialized data structures.

**Difference b/w comparator and comparable:-**



**Anonymous:-**

* In Java, a class can contain another class known as nested class. It's possible to create a nested class without giving any name.
* A nested class that doesn't have any name is known as an anonymous class.
* An anonymous class must be defined inside another class. Hence, it is also known as an anonymous inner class.
* Its syntax is:

class outerClass {

// defining anonymous class

object1 = new Type(parameterList) {

// body of the anonymous class

};

}

Anonymous classes usually extend subclasses or implement interfaces.

Here, **Type** can be

1. a superclass that an anonymous class extends
2. an interface that an anonymous class implements

The above code creates an object, object1, of an anonymous class at runtime.

**Note:** Anonymous classes are defined inside an expression. So, the semicolon is used at the end of anonymous classes to indicate the end of the expression.

**Examples:-**

public interface MyInterface {

public void print1to100();

public void print101to200();

}

public class AnonumousClient {

public static void main(String[] args) {

MyInterface mm = new MyInterface() {

public void print1to100() {

for(int i=1;i<101;i++) {

System.out.println(i);

}

}

public void print101to200() {

for(int i=101;i<201;i++) {

System.out.println(i);

}

}

};

mm.print1to100();

mm.print101to200();

}

}