**Session 20**

**Assignment 3**

**Problem Statement:**

**Explain in brief Writable and Writable Comparable in Hadoop with an example.**

**Writable:**

* Writable is an interface in Hadoop. Writable in [Hadoop](https://acadgild.com/big-data/big-data-development-training-certification) acts as a wrapper class to almost all the primitive data type of Java.
* That is how int of [java](https://acadgild.com/web-development/java-training-certification) has become IntWritable in Hadoop and String of Java has become Text in Hadoop.
* Writable is used for creating serialized data types in Hadoop.
* Hadoop frame work definitely needs Writable type of interface to perform the following tasks:
* Implement serialization
* Transfer data between clusters and networks
* Store the deserialized data in the local disk of the system

**Implementation:**

* Implementation of writable is similar to implementation of interface in Java. It can be done by simply writing the keyword ‘implements’ and overriding the default writable method.
* Writable is a strong interface in Hadoop which while serializing the data, reduces the data size enormously, so that data can be exchanged easily within the networks.
* It has separate read and write fields to read data from network and write data into local disk respectively.
* Every data inside Hadoop should accept writable and comparable interface properties.

**Why does Hadoop use Writable(s)?**

* Data needs to be transmitted between different nodes in a distributed computing environment.
* This requires serialization and deserialization of data to convert the data that is in structured format to byte stream and vice-versa.
* Hadoop therefore uses simple and efficient serialization protocol to serialize data between maps and reduce phase and these are called Writable(s).
* Some of the examples of writable are IntWritable, LongWritable, BooleanWritable and FloatWritable.
* For implementing Writable, we need few more methods in Hadoop read Fields, reads the data from network and write will write the data into local disk. Both are necessary for transferring data through clusters.
* Thus we can create our custom Writable in a way similar to custom types in Java but with two additional methods, write and read Fields. The custom writable can travel through networks and can reside in other systems.
* This custom type cannot be compared with each other by default, so again we need to make them comparable with each other. This is a disadvantage with the writable.

**Writable Comparable:**

* A [Writable](https://hadoop.apache.org/docs/r2.6.1/api/org/apache/hadoop/io/Writable.html) which is also [comparable](http://download.oracle.com/javase/7/docs/api/java/lang/Comparable.html?is-external=true). Writable Comparable can be compared to each other, typically via Comparators.
* Any type which is to be used as a key in the Hadoop Map-Reduce framework should implement this interface.
* **Writable Comparable** interface is just a sub interface of the Writable and java.lang.Comparable interfaces. For implementing a Writable Comparable we must have compareTo method apart from read Fields and write methods.
* Comparison of types is crucial for Map Reduce, where there is a sorting phase during which keys are compared with one another.
* Implementing a comparator for Writable Comparables like the org.apache.hadoop.io.Raw Comparator interface will definitely help speed up your Map/Reduce (MR) Jobs. As you may recall, a MR Job is composed of receiving and sending key-value pairs.
* We need a custom Writable comparable if our custom data type is going to be used as key rather that the value. The Writable Comparable interface extends from the Writable interface and the Comparable interface.

**Methods in Writable Comparable:**

* Write (Data Output out) – It is used to serialize the fields of the object to ‘out’.
* readFields (Data Input in) – It is used to de serialize the fields of the object from ‘in’.
* compareTo (WritableComparable wc) – It is inherited from Comparable interface and it allows Hadoop to sort the keys in the sort and shuffle phase. Compares this object with the specified object for order. Returns a negative integer, zero, or a positive integer as this object is less than, equal to or greater than the specified object.

If you want to change the way the output should be displayed, you can override tostring () method.