**ASSIGNMENT 20.3 :**

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

**WRITABLE IN HADOOP :**

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.

Writables are used for creating serialized data types in Hadoop.

**Data Type**

A data type is a set of data with values having predefined characteristics. There are several kinds of data types in Java. For example- **int, short, byte, long, char** etc. These are called as primitive data types. All these primitive data types are bound to classes called as wrapper class.

**Interface in Java**

An interface in Java is a complete abstract class. The methods within an interface are abstract methods which do not accept body and the fields within the interface are public, static and final, which means that the fields cannot be modified.

**Serialization**

Serialization is nothing but converting the raw data into a stream of bytes which can travel along different networks and can reside in different systems. Serialization is not the only concern of Writable interface; it also has to perform compare and sorting operation in Hadoop.

 Hadoop frame work definitely needs Writable type of interface in order 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 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.

Writable variables in Hadoop have the default properties of Comparable. For example:

When we write a key as IntWritable in the Mapper class and send it to the reducer class, there is an intermediate phase between the Mapper and Reducer class i.e., shuffle and sort, where each key has to be compared with many other keys. If the keys are not comparable, then shuffle and sort phase won’t be executed or may be executed with high amount of overhead.

If a key is taken as IntWritable by default, then it has comparable feature because of RawComparator acting on that variable. It will compare the key taken with the other keys in the network. This cannot take place in the absence of Writable.

We can create our custom Writables in a way similar to custom types in Java but with two additional methods, write and readFields. 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.

**EXAMPLE :-**

public class add implements Writable{

public int a;

public int b;

public add(){

this.a=a;

this.b=b;

}

public void write(DataOutput out) throws IOException {

    out.writeInt(a);

    out.writeInt(b);

  }

public void readFields(DataInput in) throws IOException {

    a = in.readInt();

    b = in.readInt();

 }

 public String toString() {

    return Integer.toString(a) + ", " + Integer.toString(b)

}

}

**WRITABLECOMPARABLE IN HADOOP :**

WritableComparable can be defined as a sub interface of Writable, which has the feature of Comparable too. We need to make our custom type, comparable if we want to compare this type with the other. We want to make our custom type as a key, then we should definitely make our key type as WritableComparable rather than simply Writable. This enables the custom type to be compared with other types and it is also sorted accordingly. Otherwise, the keys won’t be compared with each other and they are just passed through the network.

The implementation of WritableComparable is similar to Writable but with an additional ‘CompareTo’ method inside it.

EXAMPLE :-

public class add implements WritableComparable{

public int a;

public int b;

public add(){

this.a=a;

this.b=b;

}

public void write(DataOutput out) throws IOException {

    out.writeint(a);

    out.writeint(b);

  }

public void readFields(DataInput in) throws IOException {

    a = in.readint();

    b = in.readint();

  }

public int CompareTo(add c){

int presentValue=this.value;

int CompareValue=c.value;

return (presentValue < CompareValue ? -1 : (presentValue==CompareValue ? 0 : 1));

}

public int hashCode() {

    return Integer.IntToIntBits(a)^ Integer.IntToIntBits(b);

  }

}