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

**Solution:**

**Writable:**

1. Writable is an interface in Hadoop.

2. Writable in Hadoop acts as a wrapper class to almost all the primitive data type of Java. That is how int of java has become IntWritable in Hadoop and String of Java has become Text in Hadoop.

3. Writables are used for creating serialized data types in Hadoop.

4. 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

5. 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.

6. 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.

7. It has separate read and write fields to read data from network and write data into local disk respectively.

8. Every data inside Hadoop should accept writable and comparable interface properties.

9. Serialization is important in Hadoop because it enables easy transfer of data. If Writable is not present in Hadoop, then it uses the serialization of Java which increases the data over-head in the network.

10. Writable variables in Hadoop have the default properties of Comparable.

11. 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.

12. 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.

**Sample code:**

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:**

1. WritableComparable can be defined as a sub interface of Writable, which has the feature of Comparable too.

2. We need to make our custom type, comparable if we want to compare this type with the other.

3. 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.

4. If we have made our custom type Writable rather than WritableComparable our data won’t be compared with other data types. There is no compulsion that our custom types need to be WritableComparable until unless if it is a key. Because values don’t need to be compared with each other as keys.

5. If our custom type is a key then we should have WritableComparable or else the data won’t be sorted.

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

**Sample code:**

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);

} }