**Interface in Java :-**

An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.

The interface in Java is a mechanism to achieve [*abstraction*](https://www.javatpoint.com/abstract-class-in-java). There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple [inheritance in Java](https://www.javatpoint.com/inheritance-in-java).

* It is used to achieve abstraction.
* By interface, we can support the functionality of multiple inheritance.
* It can be used to achieve loose coupling.

**Marker interface in Java:-**

It is an empty interface (no field or methods). Examples of marker interface are Serializable, Clonnable and Remote interface. All these interfaces are empty interfaces.

public interface Serializable

{

// nothing here

}

**Shellow Copy:-**

* Whenever we use default implementation of clone method we get shallow copy of object means it creates new instance and copies all the field of object to that new instance and returns it as object type, we need to explicitly cast it back to our original object. This is shallow copy of the object.

//code illustrating shallow copy

public class Ex {

    private int[] data;

    // makes a shallow copy of values

    public Ex(int[] values) {

        data = values;

    }

    public void showData() {

        System.out.println( Arrays.toString(data) );

    }

}

public class UsesEx{

    public static void main(String[] args) {

        int[] vals = {3, 7, 9};

        Ex e = new Ex(vals);

        e.showData(); // prints out [3, 7, 9]

        vals[0] = 13;

        e.showData(); // prints out [13, 7, 9]

        // Very confusing, because we didn't

        // intentionally change anything about

        // the object e refers to.

    }

}

**Deep Copy:-**

* Whenever we need own copy not to use default implementation we call it as deep copy, whenever we need deep copy of the object we need to implement according to our need.
* So for deep copy we need to ensure all the member class also implement the Cloneable interface and override the clone() method of the object class.

// Code explaining deep copy

public class Ex {

    private int[] data;

    // altered to make a deep copy of values

    public Ex(int[] values) {

        data = new int[values.length];

        for (int i = 0; i < data.length; i++) {

            data[i] = values[i];

        }

    }

    public void showData() {

        System.out.println(Arrays.toString(data));

    }

}

public class UsesEx{

    public static void main(String[] args) {

        int[] vals = {3, 7, 9};

        Ex e = new Ex(vals);

        e.showData(); // prints out [3, 7, 9]

        vals[0] = 13;

        e.showData(); // prints out [3, 7, 9]

       // changes in array values will not be

       // shown in data values.

    }

}

**when to use what**  
There is no hard and fast rule defined for selecting between shallow copy and deep copy but normally we should keep in mind that if an object has only primitive fields, then obviously we should go for shallow copy, but if the object has references to other objects, then based on the requirement, shallow copy or deep copy should be done. If the references are not upda

**Lazy Copy**  
A lazy copy can be defined as a combination of both shallow copy and deep copy. The mechanism follows a simple approach – at the initial state, shallow copy approach is used. A counter is also used to keep a track on how many objects share the data. When the program wants to modify the original object, it checks whether the object is shared or not. If the object is shared, then the deep copy mechanism is initiated.ted then there is no point to initiate a deep copy.

**Clone :-**

**Clone**() method in **Java**. Object **cloning** refers to creation of exact copy of an object. It creates a new instance of the class of current object and initializes all its fields with exactly the contents of the corresponding fields of this object. In **Java**, there is no operator to create copy of an object.

**File Handling :-**

Java FileWriter and FileReader classes are used to write and read data from text files (they are [Character Stream](https://www.geeksforgeeks.org/character-stream-vs-byte-stream-java/) classes). It is recommended **not** to use the FileInputStream and FileOutputStream classes if you have to read and write any textual information as these are Byte stream classes.

# Exception Handling in Java:-

The **Exception Handling in Java** is one of the powerful *mechanism to handle the runtime errors* so that normal flow of the application can be maintained.

## **What is Exception Handling**

Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

1. statement 1;
2. statement 2;
3. statement 3;
4. statement 4;
5. statement 5;//exception occurs
6. statement 6;
7. statement 7;
8. statement 8;
9. statement 9;
10. statement 10;

Suppose there are 10 statements in your program and there occurs an exception at statement 5, the rest of the code will not be executed i.e. statement 6 to 10 will not be executed. If we perform exception handling, the rest of the statement will be executed. That is why we use exception handling in [Java](https://www.javatpoint.com/java-tutorial).

# hierarchy of exception handling