**Marker Interface –** is a interface which don’t have any method to implement

Ex - Cloneable is a marker interface .

**NOTE –** By looking at a class implementing marker interface JRE gives it a special treatment.

Marker interfaces are available in java upto version 1.5 after that these concept is being called as “***annotations***”.

**LAMBOK jar** - if you use these jar you don’t have to use getter & setter methods each time these jar will take care of that . And these will take care of overriding hashcode() & equals() methods too.

equals() method of String compares content equality

**A Guide to Object Cloning in Java**

So cloning is about creating the copy of original object. Its dictionary meaning is : “[make an identical copy of](https://www.oxforddictionaries.com/definition/english/clone)“.

In java, if a class needs to support cloning it has to do following things:

**A)** You must implement **Cloneable** interface.  
**B)** You must override **clone()** ***method from Object class***. [Its weird. clone() method should have been in Cloneable interface.]

Java docs about clone() method are given below (formatted and extract).

/\*

Creates and returns a copy of this object. The precise meaning of "copy" may depend on the class of the object.

The general intent is that, for any object x, the expression:

1) x.clone() != x will be true

2) x.clone().getClass() == x.getClass() will be true, but these are not absolute requirements.

3) x.clone().equals(x) will be true, this is not an absolute requirement.

\*/

1. First statement **guarantees** that cloned object will have separate memory address assignment.
2. Second statement **suggest** that original and cloned objects should have same class type, but it is not mandatory.
3. Third statement **suggest** that original and cloned objects should have be equal using equals() method, but it is not mandatory.

Lets see a example

package com.ab.cloningConcept;

public class Student implements Cloneable{

private int id; private String name; private Address address;

getters() & setters() for variables

parameterized constructor

overriden toString()

@Override public Student clone() throws CloneNotSupportedException{ return (Student) super.clone(); } }

**Student.java**

**Address.java**

package com.ab.cloningConcept;

public class Address {

private int houseNo; private String streetName; private String cityName;

getter() & setter() methods

parameterized constructors

overriden toString() }

**TestingCloning.java**

package com.ab.cloningConcept;

public class TestingCloning {

public static void main(String[] args) throws CloneNotSupportedException {

Address oldStudentsAddress = new Address (42,"bakers street","new Jersey");

Student oldStudent = new Student (1,"AB",oldStudentsAddress); System.out.println(oldStudent);

Student newStudent = (Student) oldStudent.clone(); System.out.println(newStudent);

newStudent.getAddress().setHouseNo(43);

System.out.println("--- After changing address of newStudent only ---"); System.out.println(newStudent); System.out.println(oldStudent);

Here we will get same houseName=43 for newStudent as well as oldStudent but these should’nt be happening . But if you look carefully it is right . See we have cloned Student only not the address so both students are pointing to same address that’s how changing one students address automatically changes another students address . And these concept is called as **“Shallow cloning”**

}//main

}//TestingCloning

To avoid above shallow cloning effect we have to use **deep cloning .** we will see below how to use deep cloning

package com.ab.cloningConcept;

public class Address {

private int houseNo; private String streetName; private String cityName;

getter() & setter() methods

parameterized constructors

overriden toString() }

@Override public Address clone() throws CloneNotSupportedException{ return (Address) super.clone(); } }

**Address.java**

**Student.java**

package com.ab.cloningConcept;

public class Student implements Cloneable{

private int id; private String name; private Address address;

getters() & setters() for variables

parameterized constructor

overriden toString()

@Override public Student clone() throws CloneNotSupportedException{

Student clonedStudent = (Student) **super**.clone();

clonedStudent.setAddress( getAddress().clone());

**return** clonedStudent; } }

**TestingCloning.java**

Same as above & it will work as we want **because now we have clones address as well**

* You will get CloneNotSupportedException if you used clone() method in some class and not implemented Cloneable interface
* ‘CloneNotSupportedException’ - is a checked exception cause you will get compile time error as soon as you forget too write throws cloneNotSupported

== compares reference equality to do content equality use equals() methos

Cloning can be done using Copy Constructors , DeSerialization , 3rd party libaray Apache Commons