## What Is Cloning?

**Cloning** is a process of creating an exact copy of an existing object in the memory.

In java, **clone()** method of **java.lang.Object** class is used for cloning process.

This method creates an exact copy of an object on which it is called through **field-by-field assignment** and returns the reference of that object.

Not all the objects in java are eligible for cloning process. The objects which implement **Cloneable interface** are only eligible for cloning process.

Cloneable interface is a [marker interface](https://javaconceptoftheday.com/marker-interface-java/) which is used to provide the marker to cloning process.

Both shallow copy and deep copy are related to this cloning process.

The default version of clone() method creates the shallow copy of an object.

To create the deep copy of an object, we have to override the clone() method.

# [clone() Method Of java.lang.Object Class](https://javaconceptoftheday.com/clone-method-java-lang-object-class/)

clone() method is a non-static **protected** method of java.lang.Object class.

This method is used to create a clone or copy of the given object. It throws **CloneNotSupportedException** if an object is not clone-able.

Here is the method signature of clone() method.

**protected Object clone() throws CloneNotSupportedException**

Not all the objects in java are clone-able.

In order to make an object clone-able, the class of that object must implement **Cloneable** interface. Cloneable interface is a marker interface.

It does not have any methods or fields in it. It is just used to provide a marker for cloning operation.

CloneNotSupportedException is a checked type of exception.

Therefore, we have to keep calling statement to clone() method in try-catch blocks or specify it using throws clause.

clone() method is a protected method. So, you can’t use it outside the class without overriding it.

//how to create a clone of an object using clone() method.

**package** com.clone.update;

**class** B **implements** Cloneable{

**int** i; j;

**public** B(**int** i, **int** j) {

**this**.i = i;

**this**.j = j;

}

// Overriding clone() method

@Override

**protected** Object clone() **throws** CloneNotSupportedException {

**return** **super**.clone();

}

}

**public** **class** A {

**public** **static** **void** main(String[] args) {

B b1 = **new** B(10, 20);

// Declaring reference variable of Class B and assigning null to it

B b2 = **null**;

// enclosing a1.clone() in try-catch blocks

// as clone() throws CloneNotSupportedException

**try** {

// Creating a clone of a1 and assigning it to a2

b2 = (B) b1.clone();

} **catch** (CloneNotSupportedException e) {

System.***out***.println("Object is not clone-able");

}

System.***out***.println(b2.i); // Output : 10

System.***out***.println(b2.j); // Output : 20

}

}

## Shallow Copy and Deep Copy:

If a cloned object and original objects are not 100% disjoint, then it is called **shallow copy**.

In shallow copy operation, any changes made to clone will be reflected in the original or vice-versa. This happens when an object has reference variables as fields.

For example, in the below program Class B has reference variable ‘a’ of type class A. This will be pointing to an object of type class A.

When we create a clone ‘b2’ of object ‘b1’ of type Class B that clone will also have this reference variable pointing to same object.

Any changes you make to this object through clone will be reflected in the original object.

**package** com.clone.update;

**class** A1 {

**int** i;

**public** A1(**int** i) {

**this**.i = i;

}

}

**class** B1 **implements** Cloneable {

**int** j;

A1 a;

**public** B1(**int** j, A1 a) {

**this**.j = j;

**this**.a = a;

}

@Override

**protected** Object clone() **throws** CloneNotSupportedException {

**return** **super**.clone();

}

}

**public** **class** ShallowCopy {

**public** **static** **void** main(String[] args) {

A1 a = **new** A1(10);

B1 b1 = **new** B1(20, a);

B1 b2 = **null**;

**try** {

// Creating clone of b1 and assigning it to b2

b2 = (B1) b1.clone();

} **catch** (CloneNotSupportedException e) {

System.***out***.println("Onject is not clone-able");

}

// Printing value of b1.a.i

System.***out***.println(b1.a.i); // Output : 10

// Changing the value of b2.a.i to 100

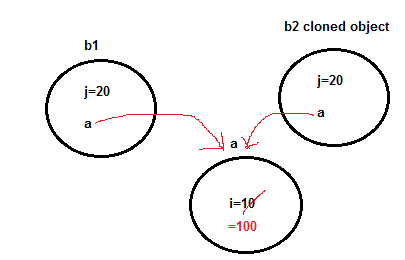
b2.a.i = 100;

// This change will be reflected in original object 'b1'

System.***out***.println(b1.a.i); // Output : 100

}

}



**Deep Copy in Java:**

Deep copy of an object will have exact copy of all the fields of original object just like shallow copy. But in additional, if original object has any references to other objects as fields, then copy of those objects are also created by calling clone() method on them.

That means clone object and original object will be 100% disjoint.

It will be 100% independent of each other. Any changes made to clone object will not be reflected in original object or vice-versa.

To create a deep copy of an object, you have to override the clone() method

**package** com.clone.deepCopy;

**public** **class** Course **implements** Cloneable {

String subject1;

String subject2;

String subject3;

**public** Course(String sub1, String sub2, String sub3) {

**this**.subject1 = sub1;

**this**.subject2 = sub2;

**this**.subject3 = sub3;

}

@Override

**protected** Object clone() **throws** CloneNotSupportedException {

**return** **super**.clone();

}

@Override

**public** String toString() {

**return** "Course [subject1=" + subject1 + ", subject2=" + subject2 + ", subject3=" + subject3 + "]";

}

}

**public** **class** Student **implements** Cloneable {

**int** id;

String name;

Course course;

**public** Student(**int** id, String name, Course course)

{

**this**.id = id;

**this**.name = name;

**this**.course = course;

}

@Override

**protected** Object clone() **throws** CloneNotSupportedException {

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

st.course = (Course)course.clone();

**return** **super**.clone();

}

@Override

**public** String toString() {

**return** "Student [id=" + id + ", name=" + name + ", course=" + course + "]";

}

}

**package** com.clone.deepCopy;

**public** **class** DeepCopyCourseStudent {

**public** **static** **void** main(String[] args) {

Course course = **new** Course("Hindi", "English", "Math");

Student st = **new** Student(1030, "Amit", course);

System.***out***.println("Before deep copy :"+st);

Student stDeepCopy = **null**;

**try** {

stDeepCopy = (Student)st.clone();

System.***out***.println("Before Modification : "+stDeepCopy);

stDeepCopy.course.subject1="Java";

stDeepCopy.course.subject2="Hibernate";

stDeepCopy.course.subject3="spring";

System.***out***.println("After Modification stDeepCopy : "+stDeepCopy);

System.***out***.println("After Modification st : "+st);

} **catch** (CloneNotSupportedException e) {

e.printStackTrace();

}

}

}

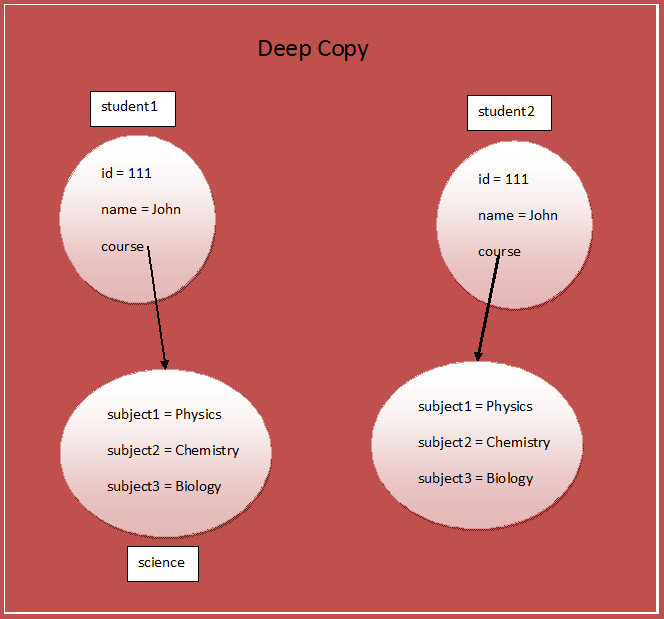
Output:

Before deep copy: Student [id=1030, name=Amit, course=Course [subject1=Hindi, subject2=English, subject3=Math]]

Before Modification: Student [id=1030, name=Amit, course=Course [subject1=Hindi, subject2=English, subject3=Math]]

After Modification stDeepCopy: Student [id=1030, name=Amit, course=Course [subject1=Java, subject2=Hibernate, subject3=spring]]

After Modification st: Student [id=1030, name=Amit, course=Course [subject1=Java, subject2=Hibernate, subject3=spring]]



|  |  |
| --- | --- |
| **Shallow Copy** | **Deep Copy** |
| Cloned Object and original object are not 100% disjoint. | Cloned Object and original object are 100% disjoint. |
| Any changes made to cloned object will be reflected in original object or vice versa. | Any changes made to cloned object will not be reflected in original object or vice versa. |
| Default version of clone method creates the shallow copy of an object. | To create the deep copy of an object, you have to override clone method. |
| Shallow copy is preferred if an object has only primitive fields. | Deep copy is preferred if an object has references to other objects as fields. |
| Shallow copy is fast and also less expensive. | Deep copy is slow and very expensive. |

<https://javaconceptoftheday.com/difference-between-shallow-copy-vs-deep-copy-in-java/>