Deep Copy in Java – Explanation with Example

What is Deep Copy?

A **deep copy** in Java means creating a **completely independent** duplicate of an object, including all objects it references. This ensures that modifying the copied object **does not affect** the original object.

Key Properties of Deep Copy in Java

1. **Creates a New Object:** The cloned object is a new instance and does not share memory with the original object.
2. **Clones Referenced Objects:** If an object has fields that are references to other objects, those referenced objects are also cloned.
3. **Ensures Independence:** Changes made to the cloned object do not affect the original object and vice versa.
4. **Implemented in Java using:**
    - Overriding clone() method
    - Using **Serialization and Deserialization**
    - Copy constructors

Example of Deep Copy using clone() Method

Let's create a **Person** class that has an **Address** class as a field. We will implement deep copying so that cloning a Person object also clones its Address object.

Step-by-Step Implementation:

```
// Address class which implements Cloneable for deep copying
class Address implements Cloneable {
    String city;

    Address(String city) {
        this.city = city;
    }

    // Overriding clone() method for deep copy
    @Override
    protected Object clone() throws CloneNotSupportedException {
        return new Address(this.city); // Creating a new Address object
```

```
    }
}

// Person class which contains an Address object
class Person implements Cloneable {
    String name;
    Address address;

    Person(String name, Address address) {
        this.name = name;
        this.address = address;
    }

    // Overriding clone() method for deep copy
    @Override
    protected Object clone() throws CloneNotSupportedException {
        Person clonedPerson = (Person) super.clone(); // Shallow copy of Person
        clonedPerson.address = (Address) address.clone(); // Deep copy of Address
        return clonedPerson;
    }
}

// Main class to test deep copy
public class DeepCopyExample {
    public static void main(String[] args) throws CloneNotSupportedException {
        // Creating an original object
        Address address1 = new Address("Pune");
        Person person1 = new Person("Srushti", address1);

        // Cloning person1 (Deep Copy)
        Person person2 = (Person) person1.clone();
```

```
        // Before modifying person2
        System.out.println("Before Modification:");
        System.out.println("person1 Address: " + person1.address.city);
        System.out.println("person2 Address: " + person2.address.city);

        // Modifying person2's address
        person2.address.city = "Mumbai";

        // After modifying person2
        System.out.println("After Modification:");
        System.out.println("person1 Address: " + person1.address.city);
        System.out.println("person2 Address: " + person2.address.city);
    }
}
```

Output

```
yaml
Before Modification:
person1 Address: Pune
person2 Address: Pune
After Modification:
person1 Address: Pune
person2 Address: Mumbai
```

Explanation

1. **Address Class:**
    - Implements Cloneable
    - Overrides clone() method to return a **new** Address object (deep copy)
2. **Person Class:**
    - Contains a reference to Address
    - Overrides clone() method:
        - Uses super.clone() to create a **shallow copy**

- Calls address.clone() to create a **deep copy** of the Address object

3. **Main Method:**
    - Creates person1 with address "Pune".
    - Clones person1 to create person2.
    - Modifies person2.address.city = "Mumbai".
    - person1.address.city remains "Pune", proving deep copying.

Comparison: Deep Copy vs Shallow Copy

Feature	Shallow Copy	Deep Copy
Object Cloning	Copies only the object, not referenced objects	Copies the object and all referenced objects
Independence	Changes in one object affect the other	Objects remain independent
Performance	Faster (only references are copied)	Slower (new objects are created)
Implementation	Default clone() method	Custom clone() method (with recursive cloning)

Alternative Approach: Deep Copy using Serialization

Another way to implement deep copying is by using **Serialization and Deserialization**.

Example using Serialization

```
import java.io.*;


// Address class implementing Serializable
class Address implements Serializable {
    String city;


    Address(String city) {
        this.city = city;
    }
```

```
}

// Person class implementing Serializable
class Person implements Serializable {
    String name;
    Address address;

    Person(String name, Address address) {
        this.name = name;
        this.address = address;
    }

    // Deep copy using Serialization
    public Person deepCopy() throws IOException, ClassNotFoundException {
        ByteArrayOutputStream bos = new ByteArrayOutputStream();
        ObjectOutputStream oos = new ObjectOutputStream(bos);
        oos.writeObject(this);

        ByteArrayInputStream bis = new ByteArrayInputStream(bos.toByteArray());
        ObjectInputStream ois = new ObjectInputStream(bis);
        return (Person) ois.readObject();
    }
}

public class DeepCopySerialization {
    public static void main(String[] args) throws IOException, ClassNotFoundException {
        Address address1 = new Address("Pune");
        Person person1 = new Person("Srushti", address1);

        // Creating deep copy using serialization
        Person person2 = person1.deepCopy();
```

```
        System.out.println("Before Modification:");
        System.out.println("person1 Address: " + person1.address.city);
        System.out.println("person2 Address: " + person2.address.city);

        // Modifying person2's address
        person2.address.city = "Mumbai";

        System.out.println("After Modification:");
        System.out.println("person1 Address: " + person1.address.city);
        System.out.println("person2 Address: " + person2.address.city);
    }
}
```

Advantages of Serialization-Based Deep Copy

✅ No need to override clone()
✅ Works well for complex objects with multiple nested references
✅ Ensures full independence of cloned objects

Final Thoughts

- **Shallow Copy** only copies object references, leading to shared memory issues.
- **Deep Copy** creates independent objects, preventing unwanted side effects.
- **Best approach depends on the use case:**
    - clone() method is good for simple objects.
    - Serialization is ideal for complex objects.

Would you like further clarifications or variations in implementation? 🚀