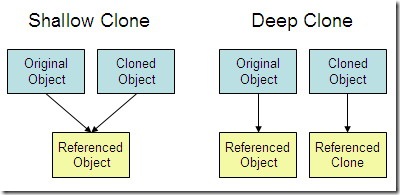
Shallow Copy VS Deep Copy in Java – 2024

**Shallow Copy Vs Deep Copy**



**What is Shallow Copy?**

Shallow copy is a bit-wise copy of an object. A new object is created that has an exact copy of the values in the original object. If any of the fields of the object are references to other objects, just the reference addresses are copied i.e., only the memory address is copied.

Here is an example of Shallow Copy implementation

**public** **class** Employee **implements** Cloneable {

**private** String name;

**private** Address adrs;

**public** Employee(String name, Address adrs) {

**this**.name = name;

**this**.adrs = adrs;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** Address getAdrs() {

**return** adrs;

}

**public** **void** setAdrs(Address adrs) {

**this**.adrs = adrs;

}

**public** Object clone() {

//shallow copy

**try** {

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

} **catch** (CloneNotSupportedException e) {

**return** **null**;

}

}

@Override

**public** String toString() {

**return** "Employee{" +

"name='" + name + '\'' +

", adrs=" + adrs +

'}';

}

}

**public** **class** Address {

**private** String cityName;

**public** Address(String cityName) {

**this**.cityName = cityName;

}

**public** String getCityName() {

**return** cityName;

}

**public** **void** setCityName(String cityName) {

**this**.cityName = cityName;

}

@Override

**public** String toString() {

**return** "Address{" +

"cityName='" + cityName + '\'' +

'}';

}

}

**public class** ShallowCopyTest {  
 Employee actualEmp = **new** Employee("John", **new** Address("New York"));

System.***out***.println("Original Employee: " + actualEmp);//Employee{name='John', adrs=Address{cityName='New York'}}

//Clone Object

Employee clonedEmp = (Employee) actualEmp.clone();;

System.***out***.println("Cloned Employee: " + clonedEmp);//Employee{name='John', adrs=Address{cityName='New York'}}

System.***out***.println("actualEmp == clonedEmp : "+ (actualEmp == clonedEmp)); //false

System.***out***.println("actualEmp.equals(clonedEmp) : "+ (actualEmp.equals(clonedEmp))); //false

// Modify actual emp object

actualEmp.setName("Dan");

actualEmp.getAdrs().setCityName("New Jersey");

**// After modifying reference object from the actual object**

System.***out***.println("Now Original Employee: " + actualEmp); //Employee{name='Dan', adrs=Address{cityName='New Jersey'}}

System.***out***.println("Now Cloned Employee: " + clonedEmp); //Employee{name='Dan', adrs=Address{cityName='New Jersey'}}

// **After modifying reference object from the cloned object**

clonedEmp.setName("Smith Clone");

clonedEmp.getAdrs().setCityName("Texas");

System.***out***.println("Original Employee: " + actualEmp); //Employee{name='Dan', adrs=Address{cityName='Texas'}}

System.***out***.println("Cloned Employee : " + clonedEmp); //Employee{name='Smith Clone', adrs=Address{cityName='Texas'}}

}  
}

It is clear from the above code that since in case of shallow copy, reference object from both actual object and cloned object refer the same memory location. It means, either actual object or cloned object modifies the reference object, the reference object will have the modified data for both.

**What is Deep Copy?**

**In case of deep copy, we create a clone that is independent of the original object and making changes in the cloned object should not affect the original object.**

**How many ways to create Deep Copy**

1. **Using a special implementation of clone method.**
2. **Using Copy constructor**
3. **Using Serialization-Deserialization**

**Now lets see how to implement them in java.**

**Special Implementation of clone method**

**public** Object clone() {

//Deep copy

**try** {

Address address = **new** Address(**this**.adrs.getCityName());

Employee emp = **new** Employee(**this**.name, address);

**return** emp;

} **catch** (Exception e) {

**return** **null**;

}

}

**Copy Constructor**

// Copy Constructor

**public** Employee(Employee emp) {

**this**.name = emp.name;

**this**.adrs = **new** Address(emp.adrs.getCityName());

}

**How to use**

Employee actualEmp = **new** Employee("John", **new** Address("New York"));

System.out.println("Original Employee: " + actualEmp);//Employee{name='John', adrs=Address{cityName='New York'}}

//Clone Object using copy constructor

**Employee clonedEmp = new Employee(actualEmp);**

**Using Serialization-Deserialization**

In case of serialization and deserialization, ensure that object implements Serializable interface.

**public** **static** **byte**[] serializeToByteArray(Employee emp) {

**byte**[] objectBytes = **null**;

ByteArrayOutputStream bos = **null**;

ObjectOutputStream oos = **null**;

**try** {

bos = **new** ByteArrayOutputStream();

oos = **new** ObjectOutputStream(bos);

oos.writeObject(emp);

objectBytes = bos.toByteArray();

**return** objectBytes;

} **catch** (IOException e) {

**throw** **new** RuntimeException(e);

} **finally** {

**try** {

oos.close();

bos.close();

} **catch** (IOException e) {

**throw** **new** RuntimeException(e);

}

}

}

**public** **static** Object deserializeFromByteArray(**byte**[] objectBytes) {

Employee emp = **null**;

ByteArrayInputStream bin = **null**;

ObjectInputStream oin = **null**;

**try** {

bin = **new** ByteArrayInputStream(objectBytes);

oin = **new** ObjectInputStream(bin);

emp = (Employee) oin.readObject();

} **catch** (IOException e) {

**throw** **new** RuntimeException(e);

} **catch** (ClassNotFoundException e) {

**throw** **new** RuntimeException(e);

} **finally** {

**try** {

oin.close();

bin.close();

} **catch** (IOException e) {

**throw** **new** RuntimeException(e);

}

}

**return** emp;

}

**How to use**

Employee actualEmp = **new** Employee("John", **new** Address("New York"));

System.out.println("Original Employee: " + actualEmp);//Employee{name='John', adrs=Address{cityName='New York'}}

// Clone using serialize and deserialize

**byte**[] empBytes = Employee.serializeToByteArray(actualEmp);

Employee clonedEmp = (Employee) Employee.deserializeFromByteArray(empBytes);

Test Program

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

// After Deep Cloning

Employee actualEmp = **new** Employee("John", **new** Address("New York"));

System.***out***.println("Original Employee: " + actualEmp);//Employee{name='John', adrs=Address{cityName='New York'}}

// Clone using serialize and deserialize

**byte**[] empBytes = Employee.*serializeToByteArray*(actualEmp);

Employee clonedEmp = (Employee) Employee.*deserializeFromByteArray*(empBytes);

//Clone Object using copy constructor

// Employee clonedEmp = new Employee(actualEmp);

// Employee clonedEmp = (Employee) actualEmp.clone();

System.***out***.println("Cloned Employee: " + clonedEmp);//Employee{name='John', adrs=Address{cityName='New York'}}

System.***out***.println("actualEmp == clonedEmp : "+ (actualEmp == clonedEmp)); //false

System.***out***.println("actualEmp.equals(clonedEmp) : "+ (actualEmp.equals(clonedEmp))); //false

// Modify actual emp object

actualEmp.setName("Dan");

actualEmp.getAdrs().setCityName("New Jersey");

// After modifying reference object from the actual object

System.***out***.println("Now Original Employee: " + actualEmp); //Employee{name='Dan', adrs=Address{cityName='New Jersey'}}

System.***out***.println("Now Cloned Employee: " + clonedEmp); //Employee{name='John', adrs=Address{cityName='New York'}}

// After modifying reference object from the cloned object

clonedEmp.setName("Smith Clone");

clonedEmp.getAdrs().setCityName("Texas");

System.***out***.println("Original Employee: " + actualEmp); //Employee{name='Dan', adrs=Address{cityName='New Jersey'}}

System.***out***.println("Cloned Employee : " + clonedEmp); //Employee{name='Smith Clone', adrs=Address{cityName='Texas'}}

}

}

Shallow Copy Vs Deep Copy In Java :

Below is the list of differences between shallow copy and deep copy in Java.

|  |  |
| --- | --- |
| **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. |