Using Mutable Objects inside user defined Immutable Class

|  |  |
| --- | --- |
| We may aware of creating user defined Immutable Class in Java. Basic steps to follow to create immutable class are | [Using Mutable Objects inside user defined Immutable Class](https://3.bp.blogspot.com/-5j8CyC4sOes/U4c2F7eyGAI/AAAAAAAAAoY/qc4LAQMs8MI/s1600/Using+Mutable+Objects+inside+user+defined+Immutable+Class.PNG) |

* Maintain all member variables and class has defined with final keyword so that variables value can't be modifies and class can't be overridden.
* Variables values must to set through constructor or through factory pattern.
* Provide only getter methods for member variable.

In below example lets see simple code which uses mutable object as member variable inside our Immutable class. By returning same original Object in getter which gives loop for user to alter the Objects internal values.

**public** **class** **Employee** {

**private** **int** id;

**private** String name;

**private** **int** age;

**public** **Employee**(**int** id, String name, **int** age) {

**this**.age = age;

**this**.id = id;

**this**.name = name;

}

**public** **int** **getId**() {

**return** id;

}

**public** **void** **setId**(**int** id) {

**this**.id = id;

}

**public** String **getName**() {

**return** name;

}

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

**this**.name = name;

}

**public** **int** **getAge**() {

**return** age;

}

**public** **void** **setAge**(**int** age) {

**this**.age = age;

}

}

**public** **final** **class** **MyImmutableClass** {

**private** **final** **int** count;

**private** **final** Employee obj;

**public** **MyImmutableClass**(**int** count, Employee obj) {

**this**.count = count;

**this**.obj = obj;

}

**public** **int** **getCount**() {

**return** count;

}

**public** Employee **getObj**() {

**return** obj;

}

}

**public** **class** **MyTestClass** {

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

MyImmutableClass obj1 = **new** MyImmutableClass(**1**, **new** Employee(**100**, "Steve", **51**));

// Here we have created Immutable object for "MyImmutableClass" class

// Lets see how to change values of mutable object inside Immutable object

obj1.getObj().setName("James");

System.out.println("Count : "+obj1.getCount());

System.out.println("Emp ID : "+obj1.getObj().getId());

System.out.println("Emp Name : "+obj1.getObj().getName());

System.out.println("Emp Age : "+obj1.getObj().getAge());

}

}

**OUTPUT:**

Count : **1**

Emp ID : **100**

Emp Name : James

Emp Age : **51**

In above example we can see mutable object inside immutable class getting changed. We have created Immutable object with employee name as "**Steve**" but later same immutable objects value getting changed from "**Steve**" to "**James**".  
Next we will see how to avoid changing mutable object in Immutable class. For this we need to change 2 things in our above classes 

* In Employee class implements Cloneable and override clone method.
* Next in MyImmutableClass class we need to return clone object instead of original Employee Object.

Below are the class code changed. 

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

**private** **int** id;

**private** String name;

**private** **int** age;

**public** **Employee**(**int** id, String name, **int** age) {

**this**.age = age;

**this**.id = id;

**this**.name = name;

}

**public** **int** **getId**() {

**return** id;

}

**public** **void** **setId**(**int** id) {

**this**.id = id;

}

**public** String **getName**() {

**return** name;

}

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

**this**.name = name;

}

**public** **int** **getAge**() {

**return** age;

}

**public** **void** **setAge**(**int** age) {

**this**.age = age;

}

**@Override**

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

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

}

}

**public** **final** **class** **MyImmutableClass** {

**private** **final** **int** count;

**private** **final** Employee obj;

**public** **MyImmutableClass**(**int** count, Employee obj) {

**this**.count = count;

**this**.obj = obj;

}

**public** **int** **getCount**() {

**return** count;

}

**public** Employee **getObj**() {

**try** {

**return** (Employee)obj.clone();

} **catch** (CloneNotSupportedException e) {

e.printStackTrace();

}

**return** **null**;

}

}

Next run same code MyTestClass class.

**OUTPUT:**

Count : **1**

Emp ID : **100**

Emp Name : Steve

Emp Age : **51**

Here we can see even we have changed Employee name from "**Steve**" to "**James**" actual object not getting disturbed.

Ref- <https://codepumpkin.com/immutable-class-with-mutable-member-fields-in-java/>

<https://javadiscover.blogspot.com/2014/05/using-mutable-objects-inside-user.html>