<https://www.journaldev.com/129/how-to-create-immutable-class-in-java>

## What is an immutable class in Java?

**Immutable objects** are instances whose state doesn’t change after it has been initialized. For example, [String](https://www.journaldev.com/16928/java-string) is an immutable class and once instantiated its value never changes.

## Benefits of Immutable Class in Java

An immutable class is good for caching purposes because you don’t have to worry about the value changes.

Another benefit of immutable class is that it is inherently [**thread-safe**](https://www.journaldev.com/1061/thread-safety-in-java), so you don’t have to worry about thread safety in case of multi-threaded environment.

Here I am providing a way to create an immutable class via an example for better understanding.

## How to Create an immutable class in Java?

To create an immutable class in Java, you have to do the following steps.

1. Declare the class as final so it can’t be extended.
2. Make all fields private so that direct access is not allowed.
3. Don’t provide setter methods for variables.
4. Make all **mutable fields final** so that its value can be assigned only once.
5. Initialize all the fields via a [constructor](https://www.journaldev.com/18899/constructor-in-java) performing deep copy.
6. Perform [cloning](https://www.journaldev.com/60/java-clone-object-cloning-java) of objects in the getter methods to return a copy rather than returning the actual object reference.

package com.journaldev.java;

import java.util.HashMap;

import java.util.Iterator;

public final class FinalClassExample {

private final int id;

private final String name;

private final HashMap<String,String> testMap;

public int getId() {

return id;

}

public String getName() {

return name;

}

/\*\*

\* Accessor function for mutable objects

\*/

public HashMap<String, String> getTestMap() {

//return testMap;

return (HashMap<String, String>) testMap.clone();

}

/\*\*

\* Constructor performing Deep

\* @param i

\* @param n

\* @param hm

\*/

public FinalClassExample(int i, String n, HashMap<String,String> hm){

System.out.println("Performing Deep for Object initialization");

this.id=i;

this.name=n;

HashMap<String,String> tempMap=new HashMap<String,String>();

String key;

Iterator<String> it = hm.keySet().iterator();

while(it.hasNext()){

key=it.next();

tempMap.put(key, hm.get(key));

}

this.testMap=tempMap;

}

/\*\*

\* Constructor performing Shallow

\* @param i

\* @param n

\* @param hm

\*/

/\*\*

public FinalClassExample(int i, String n, HashMap<String,String> hm){

System.out.println("Performing Shallow for Object initialization");

this.id=i;

this.name=n;

this.testMap=hm;

}

\*/

/\*\*

\* To test the consequences of Shallow and how to avoid it with Deep for creating immutable classes

\* @param args

\*/

public static void main(String[] args) {

HashMap<String, String> h1 = new HashMap<String,String>();

h1.put("1", "first");

h1.put("2", "second");

String s = "original";

int i=10;

FinalClassExample ce = new FinalClassExample(i,s,h1);

//Lets see whether its copy by field or reference

System.out.println(s==ce.getName());

System.out.println(h1 == ce.getTestMap());

//print the ce values

System.out.println("ce id:"+ce.getId());

System.out.println("ce name:"+ce.getName());

System.out.println("ce testMap:"+ce.getTestMap());

//change the local variable values

i=20;

s="modified";

h1.put("3", "third");

//print the values again

System.out.println("ce id after local variable change:"+ce.getId());

System.out.println("ce name after local variable change:"+ce.getName());

System.out.println("ce testMap after local variable change:"+ce.getTestMap());

HashMap<String, String> hmTest = ce.getTestMap();

hmTest.put("4", "new");

System.out.println("ce testMap after changing variable from accessor methods:"+ce.getTestMap());

}

}

**Rules for creating custom immutable java objects**

[(4292) How to create user defined or custom immutable class?|Core Java Interview Question - YouTube](https://www.youtube.com/watch?v=3Q9_stnMb64) by Java Wave

1. Declare class as final
2. All the instance variables should be private and final
3. Don’t provide setter method to instance variables
4. Initialize all the instance variables via public constructor
5. Return copy of mutable instance variable rather that returning original object