Programming Mitra (https://programmingmitra.blogspot.in/)

A Friend To Java Programming Language And Its Related Frameworks

Java Cloning - Even Copy Constructors Are Not Sufficient (http://programmingmitra.blogspot.com/2017/01/java-cloning-why-copy-constructors-are-not-sufficient-or-good.html)

posted by Naresh Joshi (https://plus.google.com/108302142446482002391) | on January 14, 2017 | 3 comments (https://programmingmitra.blogspot.in/2017/01/java-cloning-why-copy-constructors-are-not-sufficient-or-good.html#comment-form)

This is my third article on Java Cloning (https://programmingmitra.blogspot.in/search/label/Java%20Cloning?max-results=6) series, In previous articles Java Cloning and Types of Cloning (Shallow and Deep) in Details with Example (https://programmingmitra.blogspot.in/2016/11/Java-Cloning-Types-of-Cloning-Shallow-Deep-in-Details-with-Example.html) and Java Cloning - Copy Constructor versus Cloning (https://programmingmitra.blogspot.in/2017/01/Java-cloning-copy-constructor-

versus-Object-clone-or-cloning.html) I had discussed Java cloning in detail and explained every concept like what is cloning, how does it work, what are the necessary steps we need to follow to implement cloning, how to use Object.clone(), what is Shallow and Deep cloning, how to achieve cloning using Serialization and Copy constructors and advantages copy of copy constructors over Java cloning.

If you have read those articles you can easily understand why it is good to use Copy constructors over cloning or Object.clone(). In this article, I am going to discuss why copy constructor are not sufficient?

```
public Mammal(String type) {
    this.type = type;
}

// Copy Constructor
public Mammal(Mammal original) {
    this.type = original.type;
}

// Defensive Copy Method
public Mammal cloneObject() {
    return new Mammal(this);
}

public static void main(String[] args) {
    Mammal mammal = new Mammal("Human");

    //Use Copy Constructor
    Mammal clonedMammal1 = new Mammal(mammal);
    // Or call defensive copy method which will again call our copy constructor
    Mammal clonedMammal2 = mammal.cloneObject();
}
```

(//2.bp.blogspot.com/-55GvTwONJJQ/WHzFc4zE-8I/AAAAAAAAKY0/C0mGFqV0ohkfItyuPsuUprF8J5hAP3JegCK4B/s1600/Why-Copy-Constructors-Are-Not-Sufficient.png)

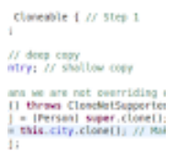
Yes, you are reading it right copy constructors are not sufficient by themselves, copy constructors are not polymorphic because constructors do not get inherited to the child class from the parent class. If we try to refer a child object from parent class reference, we will face

(https://programmingmitra.blogspot.in/2017/02/automatic-spring-data-jpa-auditing-saving-CreatedBy-createddate-lastmodifiedby-lastmodifieddate-automatically.html)

Spring Data JPA Auditing: Saving CreatedBy, CreatedDate, LastModifiedBy, LastModifiedDate automatically (https://programmingmitra.blogspot.in/2017/02/automatic-spring-data-jpa-auditing-saving-CreatedBy-createddate-lastmodifiedby-lastmodifieddate-automatically.html)

(https://programmingmitra.blogspot.in/2016/05/different-ways-to-create-objects-in-java-with-example.html)

5 Different ways to create objects in Java with Example (https://programmingmitra.blogspot.in/2016/05/different-ways-to-create-objects-in-java-with-example.html)



(https://programmingmitra.blogspot.in/2017/01/Java-cloning-copy-constructor-versus-Object-clone-or-cloning.html)

Java Cloning - Copy Constructor versus Cloning (https://programmingmitra.blogspot.in/2017/01/Java-cloning-copy-constructor-versus-Object-clone-or-cloning.html)

problems in cloning it using the copy constructor. To understand it let's take examples of two classes Mammal and Human where Human extends Mammal, Mammal class have one field type and two constructors, one to create object and one copy constructor to create copy of an object

(https://programmingmitra.blogspot.in/2017/02/automatic-jpa-auditing-persisting-audit-logs-automatically-using-entityListeners.html)

JPA Auditing: Persisting Audit Logs Automatically using EntityListeners (https://programmingmitra.blogspot.in/2017/02/automatic-jpa-auditing-persisting-audit-logs-automatically-using-entityListeners.html)

(https://programmingmitra.blogspot.in/2016/05/creating-objects-through-reflection-in-java-with-example.html)

Creating objects through Reflection in Java with Example (https://programmingmitra.blogspot.in/2016/05/creating-objects-through-reflection-in-java-with-example.html)

(https://programmingmitra.blogspot.in/2017/03/AutoWiring-Spring-Beans-Into-Classes-Not-Managed-By-Spring-Like-JPA-Entity-Listeners.html)

AutoWiring Spring Beans Into Classes Not Managed By Spring Like JPA Entity Listeners (https://programmingmitra.blogspot.in/2017/03/AutoWiring-Spring-Beans-Into-Classes-Not-Managed-By-Spring-Like-JPA-Entity-Listeners.html)

(https://programmingmitra.blogspot.in/2016/11/Java-Cloning-Types-of-Cloning-Shallow-Deep-in-Details-with-Example.html)

```
class Mammal {

    protected String type;

    public Mammal(String type) {
        this.type = type;
    }

    public Mammal(Mammal original) {
        this.type = original.type;
    }

    public String getType() {
        return type;
    }

    public void setType(String type) {
        this.type = type;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        Mammal mammal = (Mammal) o;

        if (!type.equals(mammal.type)) return false;

        return true;
    }

    @Override
    public int hashCode() {
        return type.hashCode();
    }

    @Override
    public String toString() {
```

Java Cloning and Types of Cloning (Shallow and Deep) in Details with Example (https://programmingmitra.blogspot.in/2016/11/Java-Cloning-Types-of-Cloning-Shallow-Deep-in-Details-with-Example.html)

(https://programmingmitra.blogspot.in/2017/05/how-does-jvm-handle-method-overriding-internally.html)

How Does JVM Handle Method Overloading and Overriding Internally (https://programmingmitra.blogspot.in/2017/05/how-does-jvm-handle-method-overriding-internally.html)

(https://programmingmitra.blogspot.in/2016/02/embedding-github-code-into-your-blogger.html)

Embedding Github code into your Blogger blog (https://programmingmitra.blogspot.in/2016/02/embedding-github-code-into-your-blogger.html)

(https://programmingmitra.blogspot.in/2016/05/java-build-tools-comparisons-ant-vs.html)

Java Build Tools Comparisons: Ant vs Maven vs Gradle (https://programmingmitra.blogspot.in/2016/05/java-build-tools-comparisons-ant-vs.html)

CATEGORIES

```
        return "Mammal{" + "type='" + type + "'}";
    }
}
```

And Human class which extends Mammal class, have one name field, one normal constructor and one copy constructor to create copy

Ant (https://programmingmitra.blogspot.in/search/label/Ant)

Blog Writing (https://programmingmitra.blogspot.in/search/label/Blog%20Writing)

Build Tools (https://programmingmitra.blogspot.in/search/label/Build%20Tools)

Gist (https://programmingmitra.blogspot.in/search/label/Gist)

Github (https://programmingmitra.blogspot.in/search/label/Github)

Gradle (https://programmingmitra.blogspot.in/search/label/Gradle)

How in Java (https://programmingmitra.blogspot.in/search/label/How%20in%20Java)

Interview Q&A (https://programmingmitra.blogspot.in/search/label/Interview%20Q%26A)

Java (https://programmingmitra.blogspot.in/search/label/Java)

Java 8 (https://programmingmitra.blogspot.in/search/label/Java%208)

Java Cloning (https://programmingmitra.blogspot.in/search/label/Java%20Cloning)

JPA (https://programmingmitra.blogspot.in/search/label/JPA)

```
class Human extends Mammal {

    protected String name;

    public Human(String type, String name) {
        super(type);
        this.name = name;
    }

    public Human(Human original) {
        super(original.type);
        this.name = original.name;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        if (!super.equals(o)) return false;

        Human human = (Human) o;

        if (!type.equals(human.type)) return false;
        if (!name.equals(human.name)) return false;

        return true;
    }

    @Override
    public int hashCode() {
        int result = super.hashCode();
```


Lombok (https://programmingmitra.blogspot.in/search/label/Lombok)

Maven (https://programmingmitra.blogspot.in/search/label/Maven)

Reflection API (https://programmingmitra.blogspot.in/search/label/Reflection%20API)

Solr (https://programmingmitra.blogspot.in/search/label/Solr)

Spring (https://programmingmitra.blogspot.in/search/label/Spring)

Spring Boot (https://programmingmitra.blogspot.in/search/label/Spring%20Boot)

Spring Data (https://programmingmitra.blogspot.in/search/label/Spring%20Data)

What in Java (https://programmingmitra.blogspot.in/search/label/What%20in%20Java)

Why in Java (https://programmingmitra.blogspot.in/search/label/Why%20in%20Java)

```
        result = 31 * result + name.hashCode();
        return result;
    }

    @Override
    public String toString() {
        return "Human{" + "type='" + type + "', name='" + name + "'}";
    }
}
```

> Here in both copy constructors we are doing deep cloning.

Now let's create objects for both classes

```
Mammal mammal = new Mammal("Human");
Human human = new Human("Human", "Naresh");
```

Now if we want to create clone for mammal or human, we can simply do it by calling their respective copy constructor

```
Mammal clonedMammal = new Mammal(mammal);
Human clonedHuman = new Human(human);
```

We will get no error in doing this and both objects will be cloned successfully, as we can see below tests

```
System.out.println(mammal == clonedMammal); // false
System.out.println(mammal.equals(clonedMammal)); // true

System.out.println(human == clonedHuman); // false
System.out.println(human.equals(clonedHuman)); // true
```

But what if we try to refer object of Human from reference of Mammal

```
Mammal mammalHuman = new Human("Human", "Mahesh");
```

In order to clone mammalHuman, we can not use constructor Human, It will give us compilation error because type mammalHuman is Mammal and constructor of Human class accept Human.

```
Mammal clonedMammalHuman = new Human(mammalHuman); // compilation error
```

And if we try clone mammalHuman using copy constructor of Mammal, we will get the object of Mammal instead of Human but mammalHuman holds object of Human

```
Mammal clonedMammalHuman = new Mammal(mammalHuman);
```

So both mammalHuman and clonedMammalHuman are not the same objects as you see in the output below code

```
System.out.println("Object " + mammalHuman + " and copied object " + clonedMammalHuman + " are ==
: " + (mammalHuman == clonedMammalHuman));
System.out.println("Object " + mammalHuman + " and copied object " + clonedMammalHuman + " are eq
ual : " + (mammalHuman.equals(clonedMammalHuman)) + "\n");
```

Output:

```
Object Human{type='Human', name='Mahesh'} and copied object Mammal{type='Human'} are == : false
Object Human{type='Human', name='Mahesh'} and copied object Mammal{type='Human'} are equal : fals
e
```

As we can see copy constructors suffer from inheritance problems and they are not polymorphic as well. So how can we solve this problem, Well there various solutions like creating static Factory methods or creating some generic class which will do this for us and the list will go on.

But there is a very easy solution which will require copy constructors and will be polymorphic as

2017/01/Project-Lombok-The-Boilerplate-Code-Extractor.html)

Java Cloning - Even Copy Constructors Are Not Suff... (https://programmingmitra.blogspot.in/2017/01/java-cloning-why-copy-constructors-are-not-sufficient-or-good.html)

Java Cloning - Copy Constructor versus Cloning (https://programmingmitra.blogspot.in/2017/01/Java-cloning-copy-constructor-versus-Object-clone-or-cloning.html)

► 2016 (https://programmingmitra.blogspot.in/2016/) (16)

SUBMIT A QUERY OR SUGGESTION

Name

Email *

Message *

Send

well.We can solve this problem using defensive copy methods, a method which we are going to include in our classes and call copy constructor from it and again override it the child class and call its copy constructor from it.

Defensive copy methods will also give us advantage of dependency injection, we can inject dependency instead of making our code tightly coupled we can make it loosely coupled, we can even create an interface which will define our defensive copy method and then implement it in our class and override that method.

So in Mammal class we will create a no-argument method cloneObject however, we are free to name this method anything like clone or copy or copyInstance

```
public Mammal cloneObject() {
    return new Mammal(this);
}
```

And we can override same in “Human” class

```
@Override
public Human cloneObject() {
    return new Human(this);
}
```

Now to clone mammalHuman we can simply say

```
Mammal clonedMammalHuman = mammalHuman.clone();
```

And for last two sys out we will get below output which is our expected behavior.

```
Object Human{type='Human', name='Mahesh'} and copied object Human{type='Human', name='Mahesh'} ar
e == : false
Object Human{type='Human', name='Mahesh'} and copied object Human{type='Human', name='Mahesh'} ar
e equal : true
```

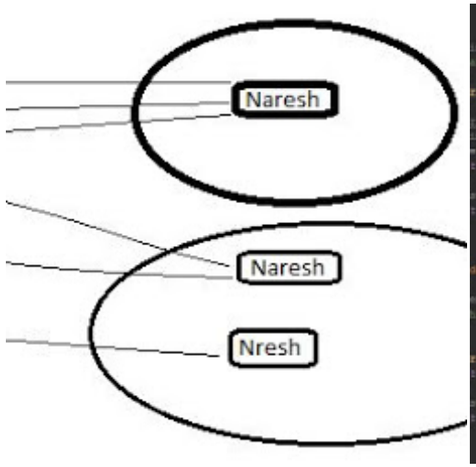
As we can see apart from getting the advantage of polymorphism this option also gives us freedom from passing any argument.

You can found complete code in CopyConstructorExample Java file on Github (https://github.com/njnareshjoshi/exercises/blob/master/src/org/programming/mitra/exercises/CopyConstructorExample.java) and please feel free to give your valuable feedback.

Naresh Joshi

()

RELATED POSTS

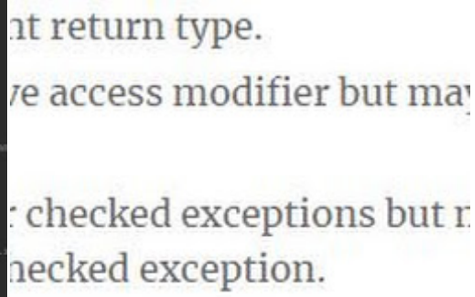


(https://programmingmitra.blogspot.com/2018/02/why-string-is-stored-in-constant-pool.html)

Why String is Stored in String Constant Pool (https://programmingmitra.blogspot.com/2018/02/why-string-is-stored-in-constant-pool.html)

(https://programmingmitra.blogspot.com/2018/02/what-is-variable-shadowing-and-hiding.html)

What is Variable Shadowing and Hiding in Java (https://programmingmitra.blogspot.com/2018/02/what-is-variable-shadowing-and-hiding.html)



(https://programmingmitra.blogspot.com/2017/12/why-we-should-follow-method-overriding-rules.html)

Why We Should Follow Method Overriding Rules (https://programmingmitra.blogspot.com/2017/12/why-we-should-follow-method-overriding-rules.html)

« Newer Post (https://programmingmitra.blogspot.in/2017/01/Project-Lombok-The-Boilerplate-Code-Extractor.html)

Older Post (https://programmingmitra.blogspot.in/2017/01/Java-cloning-copy-constructor-versus-Object-clone-or-cloning.html) »

3 Comments :

1. **Anonymous** 1

FEBRUARY 07, 2017 4:10 PM (HTTPS://PROGRAMMINGMITRA.BLOGSPOT.COM/2017/01/JAVA-CLONING-WHY-COPY-CONSTRUCTORS-ARE-NOT-SUFFICIENT-OR-GOOD.HTML?SHOWCOMMENT=1486464002067#C6237514997136858O5)

Nice article

Reply

Naresh Joshi (Https://Www.Blogger.Com/Profile/01073925481525463593) 1.a

FEBRUARY 07, 2017 4:10 PM (HTTPS://PROGRAMMINGMITRA.BLOGSPOT.COM/2017/01/JAVA-CLONING-WHY-COPY-CONSTRUCTORS-ARE-NOT-SUFFICIENT-OR-GOOD.HTML?SHOWCOMMENT=1486464037851#C3774197726273471420)

Thanks

2. **YouLoseBellyFat (Https://Www.Blogger.Com/Profile/18175607118884749966)** 2

APRIL 09, 2017 6:15 PM (HTTPS://PROGRAMMINGMITRA.BLOGSPOT.COM/2017/01/JAVA-CLONING-WHY-COPY-CONSTRUCTORS-ARE-NOT-SUFFICIENT-OR-GOOD.HTML?SHOWCOMMENT=1491741922701#C4556177478121479965)

java swing (http://java.happycodings.com/swing/)

Reply

(https://www.blogger.com/comment-iframe.g?blogID=3832771837877502009&postID=5040369632151810858&blogspotRpcToken=1974455)