

Programming Mitra

(<https://programmingmitra.blogspot.in/>)

A Friend To Java Programming Language And Its Related Frameworks

[HOME \(HTTPS://PROGRAMMINGMITRA.BLOGSPOT.IN/\)](https://programmingmitra.blogspot.in/)

[JAVA \(/SEARCH/LABEL/JAVA\)](/search/label/java)

[JAVA 8 \(/SEARCH/LABEL/JAVA%208\)](/search/label/java%208)

[INTERVIEW QUE ANS \(/SEARCH/LABEL/INTERVIEW%200%26A\)](/search/label/interview%200%26a)

[JPA \(/SEARCH/LABEL/JPA\)](/search/label/jpa)

[SPRING \(/SEARCH/LABEL/SPRING\)](/search/label/spring)

[SOLR \(/SEARCH/LABEL/SOLR\)](/search/label/solr)

[ABOUT \(/P/ABOUT.HTML\)](/p/about.html)

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

posted by Naresh Joshi (<https://plus.google.com/108302142446482002391>) | on November 20, 2016 | 7 comments (<https://programmingmitra.blogspot.in/2016/11/Java-Cloning-Types-of-Cloning-Shallow-Deep-in-Details-with-Example.html#comment-form>)

In my previous article 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>), I have discussed 5 different ways (new keyword, Class.newInstance() method, Constructor.newInstance() method, clone() method and deserialization) a developer

CONNECT WITH US

f FACEBOOK

([HTTP://WWW.FACEBOOK.COM/PROGRAMMINGMITRA](http://www.facebook.com/PROGRAMMINGMITRA))

g+ GOOGLE

([HTTP://PLUS.GOOGLE.COM/B/103534373840551473813/COMMUNITIES/106845742620040840023](http://plus.google.com/b/103534373840551473813/communities/106845742620040840023))

in LINKEDIN

([HTTPS://WWW.LINKEDIN.COM/IN/NARESH-JOSHI-89493255?TRK=NAV_RESPONSIVE_TAB_PROFILE_PIC](https://www.linkedin.com/in/naresh-joshi-89493255?trk=nav_responsive_tab_profile_pic))

POPULAR POSTS

can use to create objects, if you haven't read it please go ahead.

Cloning is also a way of creating an object but in general, cloning is not just about creating a new object. Cloning means creating a new object from an already present object and copying all data of given object to that new object.

In order to create a clone of an object we generally design our class in such way that

1. Our class should implement Cloneable interface otherwise, JVM will throw CloneNotSupportedException if we will call clone() on our object.

Cloneable interface is a marker interface which JVM uses to analyse whether this object is allowed for cloning or not. According to JVM if yourObject instanceof Cloneable then create a copy of the object otherwise throw CloneNotSupportedException.

2. Our class should have clone method which should handle CloneNotSupportedException.

It is not necessary to define our method by the name of clone, we can give it any name we want e.g. createCopy(). Object.clone() method is protected by its definition so practically child classes of Object outside the package of Object class (java.lang) can only access it through inheritance and within itself. So in order to access clone method on objects of our class we will need to define a clone method inside our class and then call Object.clone() from it. For details on protected access specifier, please read Why an outer Java class can't be private or protected (<https://programmingmitra.blogspot.in/2016/10/why-a-java-class-can-not-be-private-or-protected.html>) .

3. And finally, we need to call the clone() method of the superclass, which will call its super's clone() and this chain will continue until it will reach to clone() method of the Object class. Object.clone() method is the actual worker who creates the clone of your object and another clone() methods just delegates the call to its parent's clone().



(<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>)

All the things also become disadvantage of the cloning strategy to copy the object because all of above steps are necessary to make cloning work. But we can choose to not do all above things follow other strategies e.g. Copy Constructors, To know more read Java Cloning - Copy Constructor versus Cloning (https://programmingmitra.blogspot.in/2017/01/Java-cloning-copy-constructor-versus-Object-clone-or-cloning.html).

```
class Person implements Cloneable { // Step 1
    private String name;
    private int income;
    private City city; // deep copy
    private Country country; // shallow copy

    // No @Override, means we are not overriding clone
    public Person clone() throws CloneNotSupportedException { // Step 2
        Person clonedObj = (Person) super.clone(); // Step 3
        clonedObj.city = this.city.clone(); // Making deep copy of city
        return clonedObj;
    }
}
```

(//4.bp.blogspot.com/-

-63Wf3VoXXo/WDFRVwcVR2I/AAAAAAAAAJ_A/FXk3Lq2pb2wF4lF6kWu9vHe54dTYhNqqwCK4B/s1600/Java%2BCloning.png)

To demonstrate cloning we will create two classes Person and City and override

1. toString() to show the content of the person object,
2. equals() and hashCode() method to compare the objects,
3. clone() to clone the object

However, we are overriding the clone method but it not necessary we can create clone method by any name but if we are naming it as clone then we will need to follow basic rules of method overriding e.g. method should have same name, arguments and return type (after Java 5 you can also use a covariant type as return type). To know more about method overriding read Everything



(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)

About Method Overloading Vs Method Overriding
(<https://programmingmitra.blogspot.in/2017/05/everything-about-method-overloading-vs-method-overriding.html>).

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>)

```

class Person implements Cloneable {
    private String name; // Will holds address of the String object, instead of object itself
    private int income; // Will hold bit representation of int, which is assigned to it
    private City city; // Will holds address of the City object, instead of City object

    public String getName() {
        return name;
    }
    public void setName(String firstName) {
        this.name = firstName;
    }
    public int getIncome() {
        return income;
    }
    public void setIncome(int income) {
        this.income = income;
    }
    public City getCity() {
        return city;
    }
    public void setCity(City city) {
        this.city = city;
    }

    public Person(String firstName, int income, City city) {
        super();
        this.name = firstName;
        this.income = income;
        this.city = city;
    }

    // But we can also create using any other name
    @Override
    public Person clone() throws CloneNotSupportedException {
        return (Person) super.clone();
    }

    // To print the person object
    @Override

```

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>)

```

public String toString() {
    return "Person [name=" + name + ", income=" + income + ", city=" + city + "];"
}

// hashCode(), and equals() to compare person objects
@Override
public int hashCode() {
    final int prime = 31;
    int result = 1;
    result = prime * result + ((city == null) ? 0 : city.hashCode());
    result = prime * result + income;
    result = prime * result + ((name == null) ? 0 : name.hashCode());
    return result;
}

@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    Person other = (Person) obj;
    if (city == null) {
        if (other.city != null)
            return false;
    } else if (!city.equals(other.city))
        return false;
    if (income != other.income)
        return false;
    if (name == null) {
        if (other.name != null)
            return false;
    } else if (!name.equals(other.name))
        return false;
    return true;
}
}

```

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>

NEWSLETTER

Subscribe Our Newsletter

Enter your email address below to subscribe to our newsletter.

Person class has a reference to City class which looks like below

Email address

SUBSCRIBE

BLOG ARCHIVE

- ▶ 2018
(<https://programmingmitra.blogspot.in/2018/>) (4)
- ▶ 2017
(<https://programmingmitra.blogspot.in/2017/>) (11)
- ▼ 2016
(<https://programmingmitra.blogspot.in/2016/>) (16)
 - ▼ November
(<https://programmingmitra.blogspot.in/2016/11/>) (1)
 - Java Cloning and Types of Cloning (Shallow and Dee...
(<https://programmingmitra.blogspot.in/2016/11/Java-Cloning-Types-of-Cloning-Shallow-Deep-in-Details-with-Example.html>)
 - ▶ October
(<https://programmingmitra.blogspot.in/2016/10/>) (3)
 - ▶ June
(<https://programmingmitra.blogspot.in/2016/06/>) (2)
 - ▶ May
(<https://programmingmitra.blogspot.in/2016/05/>) (8)

```
class City implements Cloneable {
    private String name;

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

    public City(String name) {
        super();
        this.name = name;
    }

    @Override
    public City clone() throws CloneNotSupportedException {
        return (City) super.clone();
    }

    @Override
    public String toString() {
        return "City [name=" + name + "]";
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + ((name == null) ? 0 : name.hashCode());
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
```

- February
(<https://programmingmitra.blogspot.in/2016/02/>) (1)
- January
(<https://programmingmitra.blogspot.in/2016/01/>) (1)

SUBMIT A QUERY OR SUGGESTION

Name

Email *

Message *

Send


```
        if (getClass() != obj.getClass())
            return false;
        City other = (City) obj;
        if (name == null) {
            if (other.name != null)
                return false;
        } else if (!name.equals(other.name))
            return false;
        return true;
    }
}
```

And let's test it

```

public class CloningExample {

    public static void main(String[] args) throws CloneNotSupportedException {

        City city = new City("Dehradun");
        Person person1 = new Person("Naresh", 10000, city);
        System.out.println(person1);

        Person person2 = person1.clone();
        System.out.println(person2);

        if (person1 == person2) { // Evaluate false, because person1 and person2 holds different
objects
            System.out.println("Both person1 and person2 holds same object");
        }

        if (person1.equals(person2)) { // Evaluate true, person1 and person2 are equal and have s
ame content
            System.out.println("But both person1 and person2 are equal and have same content");
        }

        if (person1.getCity() == person2.getCity()) {
            System.out.println("Both person1 and person2 have same city object");
        }
    }
}

```

person1.clone() calls super.clone() which means Object.clone() method.

Object.clone() method copy content of the object to other object bit-by-bit, means the values of all instance variables from one object will get copied to instance variables of other object.

So (person1 == person2) will evaluate false because person1 and person2 are the copy of each other but both are different objects and holds different heap memory. While

person1.equals(person2) evaluate true because both have the same content.

But as we know reference variables holds address of the object instead of object itself, which can also be referred from other reference variables and if we change one other will reflect that change.

So while cloning process Object.clone() will copy address which person1.city is holding to person2.city, So now city, person1.city, and person2.city all are holding same city object. That's why (person1.getCity() == person2.getCity()) evaluate true. This behavior of cloning is known as **Shallow Cloning**.

Types of Cloning

This behavior of Object.clone() method classifies cloning into two sections

1. Shallow Cloning

Default cloning strategy provided by Object.clone() which we have seen. The clone() method of object class creates a new instance and copy all fields of the Cloneable object to that new instance (either it is primitive or reference). So in the case of reference types only reference bits gets copied to the new instance, therefore, the reference variable of both objects will point to the same object. The example we have seen above is an example of Shallow Cloning.

2. Deep Cloning

As the name suggest deep cloning means cloning everything from one object to another object. To achieve this we will need to trick our clone() method provide our own cloning strategy. We can do it by implementing Cloneable interface and override clone() method in every reference type we have in our object hierarchy and then call super.clone() and these clone() methods in our object's clone method.

So we can change clone method of Person class in below way

```
public Person clone() throws CloneNotSupportedException {  
    Person clonedObj = (Person) super.clone();  
    clonedObj.city = this.city.clone();  
    return clonedObj;  
}
```

Now (person1.getCity() == person2.getCity()) will evaluate false because in clone() method of Person class we are cloning city object and assigning it to the new cloned person object.

In below example we have deep copied city object by implementing clone() in City class and calling that clone() method of person class, That's why person1.getCity() == person2.getCity() evaluate false because both are separate objects. But we have not done same with Country class and person1.getCountry() == person2.getCountry() evaluate true.

```
public class CloningExample {

    public static void main(String[] args) throws CloneNotSupportedException {

        City city = new City("Dehradun");
        Country country = new Country("India");
        Person person1 = new Person("Naresh", 10000, city, country);
        System.out.println(person1);

        Person person2 = person1.clone();
        System.out.println(person2);

        // Evaluate false, because person1 and person2 holds different objects
        if (person1 == person2) {
            System.out.println("Both person1 and person2 holds same object");
        }

        // Evaluate true, person1 and person2 are equal and have same content
        if (person1.equals(person2)) {
            System.out.println("But both person1 and person2 are equal and have same content");
        }

        // Evaluate false
        if (person1.getCity() == person2.getCity()) {
            System.out.println("Both person1 and person2 have same city object");
        }

        // Evaluate true, because we have not implemented clone in Country class
        if (person1.getCountry() == person2.getCountry()) {
            System.out.println("Both person1 and person2 have same country object");
        }

        // Now lets change city and country object and print person1 and person2
        city.setName("Pune");
        country.setName("IN");

        // person1 will print new Pune city
        System.out.println(person1);
        // while person2 will still print Dehradun city because person2.city holds a separate cit
```

```
y object
    System.out.println(person2);
}

class Person implements Cloneable {
    private String name; // Will holds address of the String object which lives
                        // in SCP, instead of String object itself
    private int income; // Will hold bit representation of int, which is assigned to it
    private City city; // Will holds address of the City object which lives in
                    // heap, instead of City object
    private Country country;

    public String getName() {
        return name;
    }

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

    public int getIncome() {
        return income;
    }

    public void setIncome(int income) {
        this.income = income;
    }

    public City getCity() {
        return city;
    }

    public void setCity(City city) {
        this.city = city;
    }

    public Country getCountry() {
        return country;
    }
}
```

```
}

public void setCountry(Country country) {
    this.country = country;
}

public Person(String name, int income, City city, Country country) {
    super();
    this.name = name;
    this.income = income;
    this.city = city;
    this.country = country;
}

@Override
public Person clone() throws CloneNotSupportedException {
    Person clonedObj = (Person) super.clone();
    clonedObj.city = this.city.clone();
    return clonedObj;
}

@Override
public String toString() {
    return "Person [name=" + name + ", income=" + income + ", city=" + city + ", country=" +
country + "]\n";
}

@Override
public int hashCode() {
    final int prime = 31;
    int result = 1;
    result = prime * result + ((city == null) ? 0 : city.hashCode());
    result = prime * result + ((country == null) ? 0 : country.hashCode());
    result = prime * result + income;
    result = prime * result + ((name == null) ? 0 : name.hashCode());
    return result;
}

@Override
```

```

public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    Person other = (Person) obj;
    if (city == null) {
        if (other.city != null)
            return false;
    } else if (!city.equals(other.city))
        return false;
    if (country == null) {
        if (other.country != null)
            return false;
    } else if (!country.equals(other.country))
        return false;
    if (income != other.income)
        return false;
    if (name == null) {
        if (other.name != null)
            return false;
    } else if (!name.equals(other.name))
        return false;
    return true;
}
}

```

```

class City implements Cloneable {
    private String name;

    public String getName() {
        return name;
    }

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

```



```
public City(String name) {
    super();
    this.name = name;
}

public City clone() throws CloneNotSupportedException {
    return (City) super.clone();
}

@Override
public String toString() {
    return "City [name=\"" + name + "\"]";
}

@Override
public int hashCode() {
    final int prime = 31;
    int result = 1;
    result = prime * result + ((name == null) ? 0 : name.hashCode());
    return result;
}

@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    City other = (City) obj;
    if (name == null) {
        if (other.name != null)
            return false;
    } else if (!name.equals(other.name))
        return false;
    return true;
}
```

```
}

class Country {
    private String name;

    public String getName() {
        return name;
    }

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

    public Country(String name) {
        super();
        this.name = name;
    }

    @Override
    public String toString() {
        return "Country [name=" + name + "]";
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + ((name == null) ? 0 : name.hashCode());
        return result;
    }

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

```
Country other = (Country) obj;
if (name == null) {
    if (other.name != null)
        return false;
} else if (!name.equals(other.name))
    return false;
return true;
}
}
```

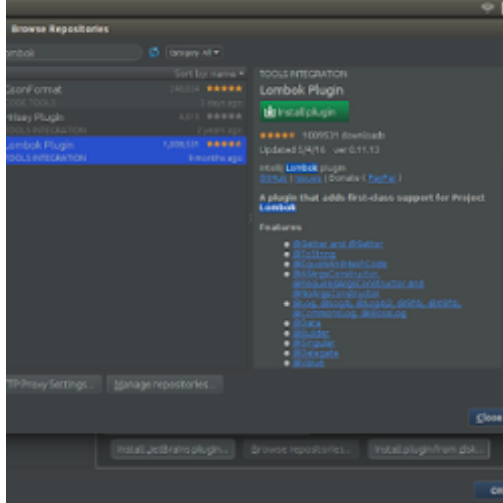
Java cloning is not considered a good way to copy an object and lots of other ways are there to do the same. You can read [Java Cloning - Copy Constructor versus Cloning](https://programmingmitra.blogspot.in/2017/01/Java-cloning-copy-constructor-versus-Object-clone-or-cloning.html) (<https://programmingmitra.blogspot.in/2017/01/Java-cloning-copy-constructor-versus-Object-clone-or-cloning.html>) to get more knowledge on why Java cloning is not a preferred way of cloning and what are other ways to overcome this.



Naresh Joshi

()

RELATED POSTS



(<https://programmingmitra.blogspot.com/2017/01/Project-Lombok-The-Boilerplate-Code-Extractor.html>)

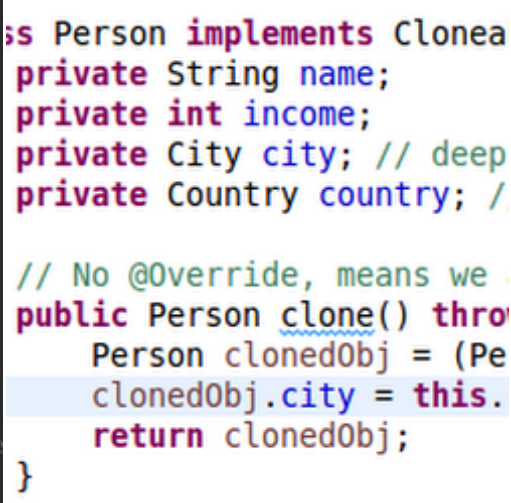
Project Lombok : The Boilerplate Code Extractor

(<https://programmingmitra.blogspot.com/2017/01/Project-Lombok-The-Boilerplate-Code-Extractor.html>)



(<https://programmingmitra.blogspot.com/2017/01/java-cloning-why-copy-constructors-are-not-sufficient-or-good.html>)

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



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

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

Newer Post

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

Older Post

(https://programmingmitra.blogspot.in/2016/10/why-outer-java-class-cant-be-static.html) »

7 Comments :



1. **Aki (Https://Www.Blogger.Com/Profile/00766347974718517117)**

1

NOVEMBER 21, 2016 1:04 PM (HTTPS://PROGRAMMINGMITRA.BLOGSPOT.COM/2016/11/JAVA-CLONING-TYPES-OF-CLONING-SHALLOW-DEEP-IN-DETAILS-WITH-EXAMPLE.HTML?SHOWCOMMENT=1479713659886#C6383169006952734123)

If we want to clone String object by using deep cloning then you need to create another string object by passing existing string object.

```
// No @Override, means we are not overriding clone
public Person clone() throws CloneNotSupportedException {
    Person clonedObj = (Person) super.clone();
    clonedObj.city = this.city.clone();
    clonedObj.name = new String(this.name);
    return clonedObj;
}
```

Reply



Naresh Joshi
(<https://www.Blogger.Com/Profile/01073925481525463593>)

1.a

NOVEMBER 21, 2016 4:03 PM

([https://PROGRAMMINGMITRA.BLOGSPOT.COM/2016/11/JAVA-CLONING-TYPES-OF-CLONING-SHALLOW-DEEP-IN-DETAILS-WITH-EXAMPLE.HTML?](https://PROGRAMMINGMITRA.BLOGSPOT.COM/2016/11/JAVA-CLONING-TYPES-OF-CLONING-SHALLOW-DEEP-IN-DETAILS-WITH-EXAMPLE.HTML?SHOWCOMMENT=1479724392215#C7951587063581398879)

[SHOWCOMMENT=1479724392215#C7951587063581398879](https://PROGRAMMINGMITRA.BLOGSPOT.COM/2016/11/JAVA-CLONING-TYPES-OF-CLONING-SHALLOW-DEEP-IN-DETAILS-WITH-EXAMPLE.HTML?SHOWCOMMENT=1479724392215#C7951587063581398879))

Yes Aki, you are right

2. Anonymous

2

JANUARY 18, 2017 10:49 AM ([https://PROGRAMMINGMITRA.BLOGSPOT.COM/2016/11/JAVA-CLONING-TYPES-OF-CLONING-SHALLOW-DEEP-IN-DETAILS-WITH-EXAMPLE.HTML?](https://PROGRAMMINGMITRA.BLOGSPOT.COM/2016/11/JAVA-CLONING-TYPES-OF-CLONING-SHALLOW-DEEP-IN-DETAILS-WITH-EXAMPLE.HTML?SHOWCOMMENT=1484716779516#C6008378592840496455)

[SHOWCOMMENT=1484716779516#C6008378592840496455](https://PROGRAMMINGMITRA.BLOGSPOT.COM/2016/11/JAVA-CLONING-TYPES-OF-CLONING-SHALLOW-DEEP-IN-DETAILS-WITH-EXAMPLE.HTML?SHOWCOMMENT=1484716779516#C6008378592840496455))

HAVING an @Override annotation is not must, you aren't overriding because the original method returns object, so you cannot put the Override annotation. From your code it implies that because we took the annotation off that's the reason why we aren't overriding.

Reply



Naresh Joshi
(<https://www.Blogger.Com/Profile/01073925481525463593>)

2.a

JANUARY 25, 2017 10:03 AM

([https://PROGRAMMINGMITRA.BLOGSPOT.COM/2016/11/JAVA-CLONING-TYPES-OF-CLONING-SHALLOW-DEEP-IN-DETAILS-WITH-EXAMPLE.HTML?](https://PROGRAMMINGMITRA.BLOGSPOT.COM/2016/11/JAVA-CLONING-TYPES-OF-CLONING-SHALLOW-DEEP-IN-DETAILS-WITH-EXAMPLE.HTML?SHOWCOMMENT=1485318839370#C3644861349357256796)

[SHOWCOMMENT=1485318839370#C3644861349357256796](https://PROGRAMMINGMITRA.BLOGSPOT.COM/2016/11/JAVA-CLONING-TYPES-OF-CLONING-SHALLOW-DEEP-IN-DETAILS-WITH-EXAMPLE.HTML?SHOWCOMMENT=1485318839370#C3644861349357256796))

Yes, it is really confusing statement however what I want to say is because we are not overriding that's why we are not putting @Override on it.

3. Anonymous

3

JANUARY 22, 2017 10:29 PM (HTTPS://PROGRAMMINGMITRA.BLOGSPOT.COM/2016/11/JAVA-CLONING-TYPES-OF-CLONING-SHALLOW-DEEP-IN-DETAILS-WITH-EXAMPLE.HTML?SHOWCOMMENT=1485104343248#C5555992976393842117)

I think we are overriding.. as of JSE 5 covariant returns are possible so the easy clone method is overriding the super class method in this case.

Reply



Naresh Joshi

3.a

(<https://www.blogger.com/profile/01073925481525463593>)

JANUARY 25, 2017 10:06 AM

(HTTPS://PROGRAMMINGMITRA.BLOGSPOT.COM/2016/11/JAVA-CLONING-TYPES-OF-CLONING-SHALLOW-DEEP-IN-DETAILS-WITH-EXAMPLE.HTML?SHOWCOMMENT=1485319004705#C1093974058399262406)

No actually it is not about the return type, it is about the protected access of clone method in Object class, due to which we can't override it. To know about protected access specifier you can read <https://programmingmitra.blogspot.in/2016/10/why-a-java-class-can-not-be-private-or-protected.html>



YouLoseBellyFat

4

(<https://www.blogger.com/profile/18175607118884749966>)


APRIL 08, 2017 6:49 PM (HTTPS://PROGRAMMINGMITRA.BLOGSPOT.COM/2016/11/JAVA-CLONING-TYPES-OF-CLONING-SHALLOW-DEEP-IN-DETAILS-WITH-EXAMPLE.HTML?SHOWCOMMENT=1491657585438#C4173185051664804273)

learning java (http://java.happycodings.com/learning/) to learning java

Reply

(https://www.blogger.com/comment-iframe.g?blogID=3832771837877502009&postID=4953169917953960428&blogspotRpcToken=6129224)

Enter your comment...

 **Comment as:** Amitabh Mandal ▾

Sign out

☐ Notify me

Publish

Preview