Java – OOPS Concepts

Inheritance

We are all humans. Programmers are no different (except for their nerdy characteristics) and they usually operate within the human bounds (whether that's by choice or by compulsion is a topic out of scope for this discussion). And as humans, we have invariably heard about inheritance.

The Merriam Webster dictionary defines inheritance as –

- the act of inheriting property
- the reception of genetic qualities by transmission from parent to offspring
- the acquisition of a possession, condition, or trait from past generations

In conversational language, you inherit your genes/primary behavioral characteristics from your parents.

But you may be thinking, what is inheritance to a programmer? In object oriented programming languages, Inheritance is one of the key concepts – to be learnt, practiced, mastered and inculcated at the heart of your programming style.

The Java Language Specification defines inheritance as a means by which a child class gets all the attributes of its parent class and it can be achieved just by the use of one keyword – "extends".

So long as your Child class extends the Parent class, you get all attributes of the parent class at your service to edit, use or leave alone and find your own attribute. Let's take a look at a primary example –

Parent Class – has an attribute name and its getters and setters

```
public class Parent {

    private String name;

    public String getName() {
        return name;
    }

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

}
```

Child Class – only extends Parent and defines a whoami method to asset identity

```
public class Child extends Parent {

    public void whoami() {
        System.out.println(super.getName());
    }

}
```

Now, let's create an object of the Child class and invoke whoami method on it which prints its name attribute –

```
public class Main {

    public static void main(String[] args) {
        Child child = new Child();
        child.whoami();
    }

}
```

Output of the above program is –

```
<terminated> Main [Java Application] C:\Program Files\Java\jre1.8.0_66\bin\javaw.exe (May 19, 2019 12:57:41 PM)
null
```

Now for the obvious questions and their answers –

Q: Where does this "null" output come from?

A: By virtue of extending Parent, the Child has "inherited" the name attribute. So when the child does a super.getName(), it is invoking the getName() method of the Parent class. Since the Parent class has not initialized the name attribute of type String to a value, the default value of String i.e. null is assigned to it and returned upon invocation of getName().

Q: What if Parent's name attribute was set to some value, will their be a change in the output?

A: Sure thing, let's update the code sample and try it out –

Update Parent to initialize the name attribute to a pre-defined value –

```
public class Parent {

    private String name = "Parent";

    public String getName() {
        return name;
    }

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

}
```

Output –

```
<terminated> Main [Java Application] C:\Program Files\Java\jre1.8.0_66\bin\javaw.exe (May 19, 2019 1:06:17 PM)
Parent
```

Q: What if the Child wants to override the value to assert its identity?

A: Ah, we knew it would come to this – as with every Parent-Child relationship. Let's try this out –

Update Child to override the inherited name attribute to a child-centric value –

```
public class Child extends Parent {

    public void whoami() {
        super.setName("Child");
        System.out.println(super.getName());
    }

}
```

Output –

```
<terminated> Main [Java Application] C:\Program Files\Java\jre1.8.0_66\bin\javaw.exe (May 19, 2019 1:08:11 PM)
Child
```

Q: But wait, can Child only update the inherited value, can it not define its own value? If yes, can the child not have the cake and eat it too – as in can the Child not keep its own value AND the inherited value?

A: A good question. Answer is - Definitely yes. Let's see how that works out –

Updated Child to have its own value –

```
public class Child extends Parent {

    private String name;

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

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

    public void whoami() {
        this.setName("Child");
        System.out.println("Parent Name : " + super.getName());
        System.out.println("Child Name : " + this.getName());
    }

}
```

Output –

```
<terminated> Main [Java Application] C:\Program Files\Java\jre1.8.0_66\bin\javaw.exe (May 19, 2019 1:12:45 PM)
Parent Name : Parent
Child Name : Child
```

Q: Is this the much-talked about “is-a” relationship?

A: Bingo! Yes, every Child is-a Parent – as it extends the parent’s attributes.

================================ END ================================

As you can see, the Child has now asserted its own identity on its own attribute (accessed using the “this” keyword). At the same time, it has also inherited the parent’s attribute (accessed using the “super” keyword).

As you can see, Inheritance is a powerful tool when you wish to do the following –

- Define and group common attributes and use them without having to re-define them
- Override new values when the going gets specific – and still be able to re-use common attributes
- Relate objects using the is-a relationship