Welcome back to the

Java Course

Module 3 - OOP

Inheritance

Now we know what a class is. For example, when we talk about the "Dog" class, we aren't referring to a specific dog, but to the generic and abstract concept with attributes and actions.

Inheritance

However, if we keep using the example of a dog, we have also to consider that a dog is an animal, but not all animals are dogs. A Chihuahua is a dog, but not all dogs are Chihuahuas.

Inheritance

So, how do we consider the relationship between the Dog class and the Animal and Chihuahua classes?

Inheritance

We consider the "Animal" class as a broader and more general class to which the "Dog" class belongs and from which it derives. Similarly, the "Chihuahua" class is a specific type of "Dog," from which it derives.

Inheritance

In our example, the "Animal" class is called parent, and the "Dog" class is called child.

Similarly, the "Chihuahua" is the child class, and the "Dog" is the parent class.

Inheritance

This relationship between classes allows us to use the principle of **Inheritance**, where the child class inherits ALL the attributes and methods of the parent class.

Inheritance

The parent class has no secrets from the child class, which can have its own attributes and methods. Attributes and methods that are not present in the parent class.

Inheritance

Keep in mind that a class can have only one parent, but it can have many children.

For example, Chihuahua derives from the Dog class, but so do Doberman and Husky.

Inheritance

Syntax:

public class *Child* extends *Parent*{

// Attributes and Methods specific to the
//Child class

}

Inheritance

Syntax:

public class *Child* extends *Parent*{

// Attributes and Methods specific to the
//Child class

}

Inheritance

Syntax:

```
public class Child extends Parent{
    // Attributes and Methods specific to the
    //Child class
}
```

Inheritance

Syntax:

public class *Child* extends *Parent*{

// Attributes and Methods specific to the
//Child class

}

Inheritance

Example:

```
public class Husky extends Dog {
    // Attributes and Methods specific to the
    //Husky class
}
```

Inheritance

Similar to "*this*", the keyword "*super*" enables you to directly access methods from the parent class.

Inheritance

Example:

```
public class Husky extends Dog {
        super.bark()
}
```

Let’s practice!

Polymorphism

When we create a function (or method), we know we have to give it an intuitive and unique name.

However, the principle of **Polymorphism** bypasses this uniqueness limit.

Polymorphism

Thanks to Polymorphism, functions with the same name can exist as long as they handle different parameters.

Polymorphism

For example, we know that the print() method can display both numeric values and strings on the screen. This is because the print() method has been implemented using the principle of polymorphism.

Polymorphism

In this case, we are talking specifically about **static polymorphism** (*overloading*), where the different behavior of a function (*method*) is based on the explicit declaration of multiple versions.

Polymorphism

```
public void sum( num1, num2 ) {
    // with 2 passed parameters
}
public void sum( num1, num2, num3) {
    // with 3 passed parameters
}
```

Polymorphism

```
public void sum( num1, num2 ) {
    return num1+ num2 ;
}
public void sum( num1, num2, num3 ) {
    return num1+ num2 + num3 ;
}
```

Let’s practice!

Polymorphism

As we know, the child class inherits all the methods of the parent class.

But what happens if we modify one of these inherited methods within the child class?

Polymorphism

In this case, we are talking specifically about **dynamic polymorphism** (*overriding*), where the different behavior of a function (*method*) is defined in the child class, replacing the inherited one.

Polymorphism

```
public class CardGame {
        private int hand_cards;
        public void draw(){
                this.hand_cards ++;
        }
}
```

Polymorphism

```
public class Poker extend CardGame {
        public void draw(){
            while (super.hand_cards < 5){
                super.hand_cards ++;
            }
        }
}
```

Polymorphism

To inform the compiler that you intend to use method overriding, it is good practice to include *@Override* before the method declaration.

This makes it easier to identify errors in the code.

Polymorphism

Example:

```
@Override
public void draw(){
        while (super.hand_cards < 5){
                super.hand_cards ++;
        }
}
```

Let’s practice!