TASK

Introduction to OOP III - Inheritance

Visit our website

Introduction

WELCOME TO THE INTRO TO OOP - INHERITANCE TASK!

In this task, you will learn about one of the core aspects of Object-Oriented Programming, inheritance. Inheritance allows us to reuse code from classes in new classes that will inherit the code from the original class, but can also have additional logic or variations on the logic.

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact an expert code reviewer to get support on any aspect of your course.

The best way to get help is to login to Discord at **https://discord.com/invite/hyperdev** to start a chat with a code reviewer, as well as share your thoughts and problems with peers. You can also schedule a call or get support via email.

Our code reviewers are happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

WHAT IS INHERITANCE?

To explain what Inheritance is, let's begin by observing how inheritance works in the natural world. By taking a look at a child in comparison to their parents, you may notice that the child inherits certain traits from their parents. These traits can include things like eye colour, height, hair colour, etc. They could also inherit certain abilities like athleticism.

With this in mind, let us return to considering coding. Imagine you have created a class with properties and methods, and you would like to define another class with similar properties and methods to the first, along with a few methods that are unique to the new class. Inheritance is the mechanism by which you will were able to produce the second class from the first, without needing to create the second class from scratch or to alter the logic of the first class itself.

Alternatively, suppose you wish to write two related classes that share a significant subset of their respective properties and methods. Instead of creating two separate classes from scratch, we could encapsulate the shared information in a single class and create two more classes - child or sub classes -that will inherit all the information from the first - parent or super - class, saving space and resources as well as improving code quality for the entire project.

In most major object-oriented programming languages, objects of a sub class are also objects of their super class, but not vice-versa. This means that functions that act on the super objects may also act on the sub objects.

Inheritance in action with JavaScript

To demonstrate how inheritance works, we first need to create a super class.
For our first example, let's create an "animal" class :

```
class Animal {

    // Create constructor with the attributes name and age
    constructor(name, age){

    this.name = name;
    this.age = age;
    }
// Method to call any animal by name
      summon(){

            console.log(`You have called ${this.name}!`);
      }

    // Method to display the name and age of any animal
      display(){

            console.log(`This is ${this.name}, they are ${this.age} years
old.`);
      }

    /* Method to check if any animal is growing old - consider to be the case
when the animal hits or passes the age of 8 years - using if statements  */

    ageCheck(){

        if(this.age >= 8){

            console.log(`${this.name} is rather old`);
        }

        else if(this.age < 8){

            console.log(`${this.name} still got some spring in their step`)
        }
    }
}
```

The parent class created above has attributes for the name and age of an animal as well as a few methods to summon, display the name and age, and check whether the animal is growing old.

We can extend this class to a cheetah class (the child or subclass) which will then inherit the attributes and methods from the animal class (the super or parent class), while also having the ability to add additional attributes and methods. Let's have a look at how to do this.

```
class Cheetah extends Animal {

    constructor(name, age, topSpeed){
        super(name, age);

        this.topSpeed = topSpeed;

    }

    speed(){

        console.log(`${this.name} has a record speed of
${this.topSpeed}`);
    }

}
```

There are two main aspects to consider in this example above :

- The **extends** keyword, which will allow us to describe the class we would like to inherit from.

- The **super()** method

    - The super() method allows access to the methods and attributes in the parent class that the child class is inheriting from.

    - The super() method returns a temporary object of the parent class that will allow us to call that parent class's methods. The point of the super() method is to provide a much more abstract and portable solution for initialising classes.

    - Calling the methods created in the parent class with super() saves us from having to rewrite those methods in our child class and enables us to swap out parent classes with minimal code changes.

In the example above, the Cheetah class inherits from the Animal class. Using the super() method, we are then able to use the constructor method of the parent class in the constructor method of the child class, so that the two attributes of the parent class (name and age respectively) are initialised alongside the new attribute of the child class (topSpeed).

When we were to create an object of the child class, we will still have access to the parent class's attributes thanks to the super() method.

With this new knowledge, we can now create an instance of the Cheetah class, and we are able to observe how all the methods and attributes are inherited from the initial Animal class. For instance :

```
const gio = new Cheetah("Gio", 5, 52);

gio.speed();

/* Output : Gio has a record speed of 52 */

gio.display();

/* Output : This is Gio, they are 5 years old */

gio.summon();

/* Output : You have called Gio! */
```

Method Overriding

Method **overriding** is the ability of a class to change the implementation of a method provided by the super class.

Overriding is an essential aspect of OOP; by utilising method overriding we are able to make a duplicate of a class, however at the same time enhance or customise some aspect of its behaviour. Therefore, method overriding is a part of the inheritance mechanism.

Method overriding works by defining within the subclass a method with the same name as the method in the superclass. When we define a method in the subclass with the same name as a method in the superclass, an instance of the subclass will execute the logic within the subclass when said method is called. If this method is not defined in the subclass, then the method within the superclass is executed.

Compulsory Task

- Start by creating a new file called **courses.js**.
- Next, create a class named **Courses** with the attributes **courseName and contactWebsite**.
    - This class should have a method to display the contact website of the selected course.
- Create a subclass named **Subjects** that will inherit from **Courses** with at least two new attributes added to this new class.
    - This subclass should have a method to display what course it belongs to, as well as displaying the other two attributes that you have created.
- Create at least 3 instances of the **Subjects** class.

If you are having any difficulties, please feel free to contact our specialist team **on Discord** for support.

Rate us

Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved, or think we've done a good job?

Click here to share your thoughts anonymously.