



Construcción de Elementos de Software I

Semana 1

Paradigmas de programación: POO

Media Técnica en Programación de Sistemas de Información
Politécnico Jaime Isaza Cadavid - 2020



Introducción a los Paradigmas de Programación

¿Que es un paradigma de programación?

Un paradigma de programación consiste en un método para llevar a cabo cálculos y la forma en la que deben estructurarse y/o organizarse las tareas para crear un programa, es decir, es una manera determinada de solucionar problemas recurrentes de programación buscando la forma más eficiente y rápida en la que se puede desarrollar y computar el mismo.

Los paradigmas son formas, estilos guías de programación documentados, con buenas prácticas y de gran calidad, que los lenguajes asumen. Algunos pueden ser multi paradigma.

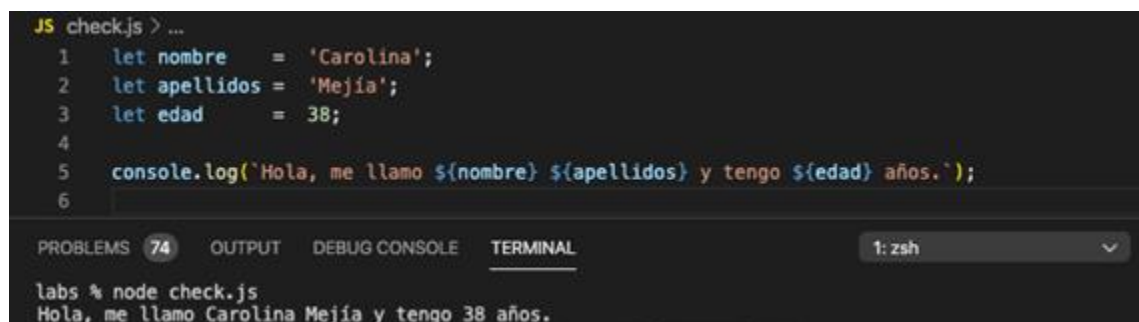
Existen diferentes tipos de paradigmas, algunos de los más conocidos son: Imperativo y la Programación Orientada a Objetos (POO). El paradigma imperativo es usado en lenguajes como C y Python, donde la ejecución del programa se hace línea a línea, mientras que lenguajes como C++, C# o Java tienen implícito un paradigma de POO.

Javascript puede utilizar cualquiera de los paradigmas, ya que no tiene uno definido.

Programación Secuencial o Estructurada

Quizás este es el primer paradigma que aprendemos dependiendo del lenguaje en el que trabajamos, ya que es la manera más simple de programar, en este se ejecuta el código que escribimos línea por línea.

En el siguiente ejemplo, vemos que tenemos definidas tres variables, que posteriormente se muestran en la consola, acá vemos que para poder ejecutar el *console.log*, el código se lee línea por línea guardando las variables en memoria hasta imprimirlas.



```
JS check.js > ...
1 let nombre = 'Carolina';
2 let apellidos = 'Mejía';
3 let edad = 38;
4
5 console.log('Hola, me llamo ${nombre} ${apellidos} y tengo ${edad} años.');
```

PROBLEMS 74 OUTPUT DEBUG CONSOLE TERMINAL 1: zsh

labs % node check.js
Hola, me llamo Carolina Mejía y tengo 38 años.

Imagen 1: Ejemplo de programación secuencial

En el siguiente ejemplo, vemos un comportamiento similar, donde al invocar (instanciar) la ejecución de la función, la multiplicamos por la variable definida previamente.



```
JS check.js > ...
7   let num = 20;
8
9   function multiplicar(valor){
10    return valor*valor;
11  }
12
13  let resultado = multiplicar(5)*num;
14
15  console.log(resultado);
```

PROBLEMS 74 OUTPUT DEBUG CONSOLE TERMINAL 1: zsh

labs % node check.js
Hola, me llamo Carolina Mejía y tengo 38 años.
500

Imagen 2: Ejemplo de programación secuencial referenciando una función

Programación orientada a Objetos (POO)

En este paradigma, se define cada uno de los elementos de la aplicación como objetos, en la que cada objeto tiene sus datos y funcionalidades, llamados **Atributos** y **Métodos** respectivamente.

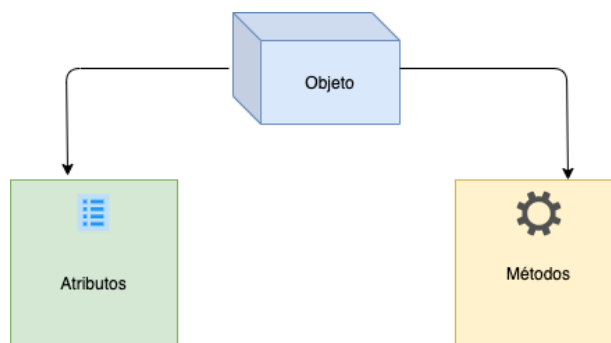


Imagen 3: Diagrama de elementos de un Objeto según POO

La POO nos ayuda a pensar en relaciones y funciones, lo que es conveniente en sistemas grandes y complejos, permitiendo así que sean fáciles y rápidos de escalar.

Un **Objeto** es un conjunto de variables, cada objeto tiene sus **Atributos** y **Métodos**. Pero ¿qué son los Atributos y Métodos? Para entenderlo, debemos primero hablar de un concepto llamado abstracción.



Encapsulamiento

Busca “Encapsular” o restringir que el objeto sólo se pueda modificar mediante sus propios Métodos, por lo que los Atributos internos no son accesibles desde otra clase, y sólo se pueden modificar mediante sus propios Métodos. Para esto, requiere tener permisos que permitan la comunicación entre los mismos.

Abstracción

Es el proceso mediante el cual hacemos la abstracción de un problema e implementamos una solución basada en Atributos y Métodos. En otras palabras es determinar qué Atributos y Métodos debe tener nuestro objeto, y cómo estructurar las clases que tendrá nuestro programa, ya veremos que es una **Clase**.

En la siguiente imagen se ilustra una analogía entre el concepto de un objeto según el paradigma de POO con los elementos de una Pizza:

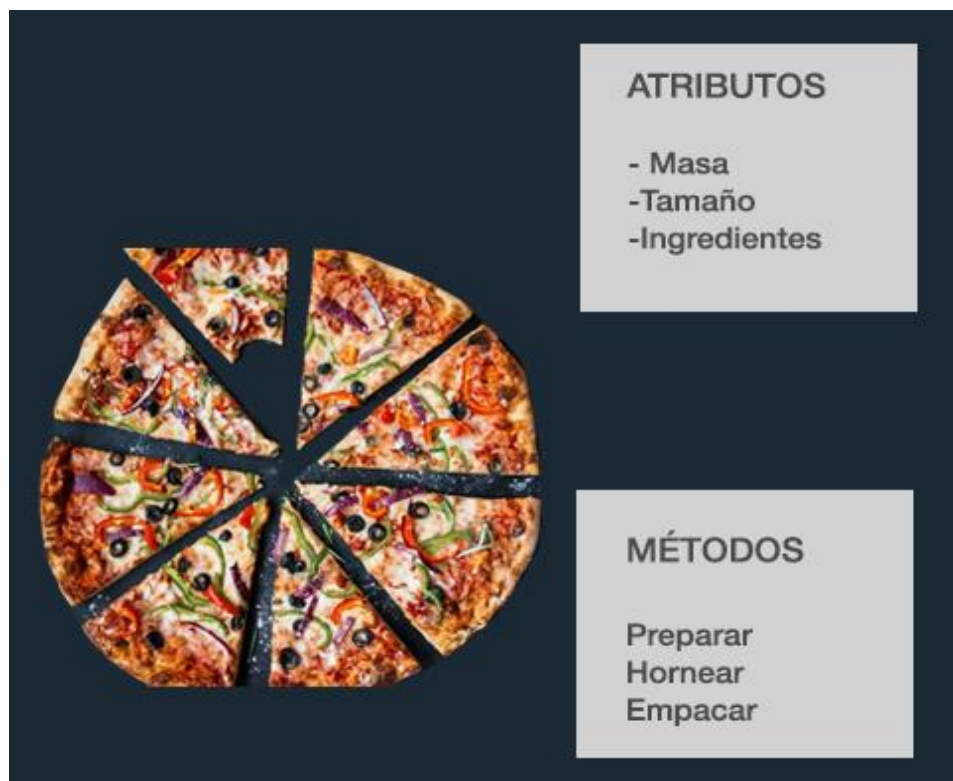


Imagen 4: Analogía de una Pizza con los componentes de un Objeto de POO



Alfredo el pizzero, siempre realiza el mismo proceso al momento de preparar sus pizzas con sus famosas recetas, aunque tiene en su carta una gran variedad de pizzas, todas se preparan de la misma manera, y lo único que cambian son el tipo y cantidad de ingredientes.

Los **Atributos** de una pizza, son:

- Masa: Tipo de masa tal como: Gruesa, Delgada, Gruesa con borde de queso.
- Tamaño: Número de porciones 6 - (Pequeña), 8 (Mediana), 12 (Grande).
- Ingredientes: Según la pizza o los que seleccione el cliente.

Como se aprecia los atributos son las características de nuestra pizza, y también serán los de nuestra **Clase** Pizza.

Los Métodos: son un algoritmo o función que recibe un mensaje y desencadena una acción, es decir, determina lo que el objeto puede hacer, éste puede producir cambios en las propiedades del objeto y retorna una respuesta.

En nuestro ejemplo, Tenemos tres Métodos:

- **Preparar:** Donde se pone la masa, la pasta, y los ingredientes.
- **Hornear:** Se pone la pizza en el horno por determinado tiempo.
- **Empacar:** Se empaca en la caja indicada según su tamaño.

Ahora, vamos a construir la clase Pizza, donde ésta tiene sus **Atributos** y **Métodos**, pero antes debemos saber: ¿Qué es una clase?

Clase

Es una especie de plantilla o molde que guarda los **Atributos** y **Métodos** que posteriormente al ser instanciada nos crea un objeto con los valores en los Atributos y permitiéndonos acceder a los Métodos.

De modo que al instanciar, invocar o “Llamar” esta clase podemos crear los objetos. Y los objetos se crean a partir de **Clases**. Veamos este ejemplo:

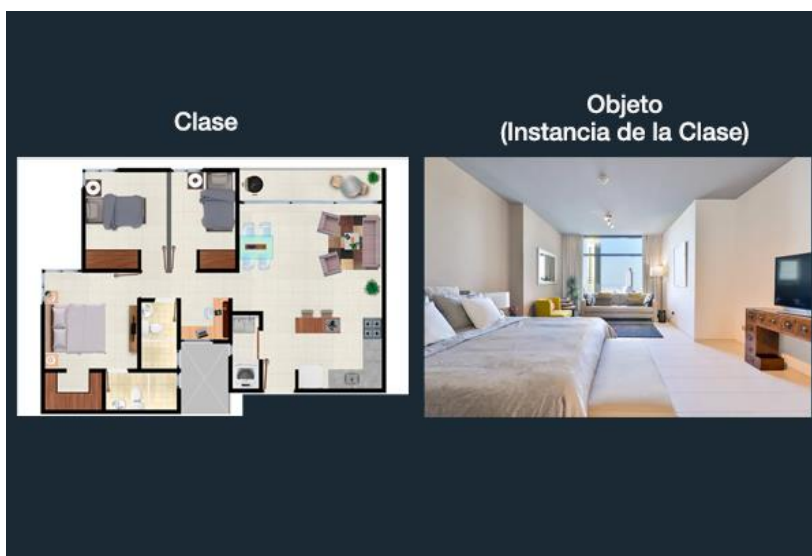


Imagen 5: Diferenciación entre el concepto de Clase y Objeto

La clase es como el plano de una construcción, y a partir de esto podemos crear innumerables objetos, en este caso a partir de un mismo plano, se pueden construir muchos apartamentos.

Creando nuestra clase Pizza:

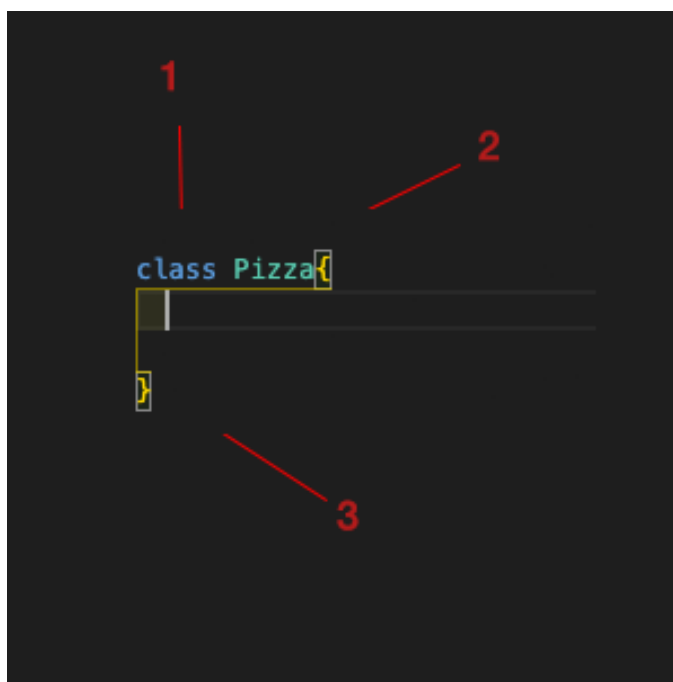


Imagen 6: Sintaxis del código para definir un Objeto o Clase

1. Usamos la palabra reservada **class** para indicar que crearemos una **Clase**.



2. Definimos el nombre de la clase, en este caso nuestra clase se llama: **Pizza**

Es una buena práctica y también muy frecuente, que el nombre de las clases empiece por Mayúscula.

3. Usamos {} para abrir y cerrar nuestra clase.

¡Listo! Ya creamos nuestra clase, ahora vamos a definir los Atributos y Métodos.

Vamos a crear un Método constructor, que se ejecuta una vez creamos un objeto nuevo instanciando nuestra clase, así que, vamos a definir los Atributos que tendrá nuestra clase Pizza: *masa, tamaño e ingredientes*

```
class Pizza{  
    constructor(masa,tamano,ingredientes){  
        this.masa = masa;  
        this.tamano = tamano;  
        this.ingredientes = ingredientes;  
    }  
}
```

Imagen 7: Código del método constructor de la Clase Pizza

Sus Atributos los asignaremos de los parámetros que estamos recibiendo en el constructor, es decir, al crear el Atributo *masa*, le asignamos el parámetro **masa** que está recibiendo el constructor:
this.masa = masa;

```
this.masa = masa;
```

Imagen 8: Código de asignación de un atributo según el parámetro recibido

De esta misma manera asignamos los parámetros que recibe el constructor a los atributos de nuestra clase.

```
class Pizza{  
    constructor(masa,tamano,ingredientes){  
        this.masa = masa;  
        this.tamano = tamano;  
        this.ingredientes = ingredientes;  
    }  
}
```

Imagen 9: Código de la clase Pizza con su método constructor

Hasta este punto, ya tenemos nuestra Clase, ahora vamos a crear los Métodos que tendrá, y que se ejecutarán al crear o “instanciar” un Objeto.

El Método preparar() retornará un mensaje con los atributos de la pizza que tendrá el Objeto. Lo veremos en breve. Por ahora crea un Método como se muestra a continuación:



```
preparar(){  
  console.log(`Preparando pizza ${this.tamano}, ${this.masa}, Ingredientes: ${this.ingredientes} `);  
  return this;  
}
```

Imagen 10: Código del método “preparar” de la clase Pizza
Creamos otro método llamado hornear() que también retornará un mensaje.

```
hornear(){  
  console.log(`La Pizza está en el horno...`);  
  return this;  
}
```

Imagen 11: Código del método “hornear” de la clase Pizza

Por último creamos el Método empacar que nos dirá cuándo la pizza está lista para llevar.

```
empacar(){  
  console.log(`Empacando la Pizza...`);  
  console.log(`Pizza lista para ser enviada! :)`);  
  return this;  
}
```

Imagen 11: Código del método “empacar” de la clase Pizza

Acá tenemos el código completo de nuestra clase Pizza.

```
class Pizza{  
  constructor(masa,tamano,ingredientes){  
    this.masa = masa;  
    this.tamano = tamano;  
    this.ingredientes = ingredientes;  
  }  
  preparar(){  
    console.log(`Preparando pizza ${this.tamano}, ${this.masa},  
    Ingredientes: ${this.ingredientes} `);  
    return this;  
  }  
  hornear(){  
    console.log(`La Pizza está en el horno...`);  
    return this;  
  }  
  empacar(){  
    console.log(`Empacando la Pizza...`);  
    console.log(`Pizza lista para ser enviada! :)`);  
    return this;  
  }  
}
```

Imagen 12: Código de la clase Pizza



Hasta ahora hemos definido nuestra clase, vamos a crear un objeto a partir de la misma, en el siguiente ejemplo, creamos una constante que almacenará el objeto de la instancia, y pasamos los tres parámetros que se definieron en la clase: (**masa, tamaño e ingredientes**)

En este caso **ingredientes** es un arreglo (array) con todos los ingredientes. Estos parámetros deben conservar el mismo orden definido en la clase.

```
const pizzaHawaianaPepperoni = new Pizza("Masa Gruesa","Mediana",["Queso","Pepperoni","Piña"]);  
  
console.log(pizzaHawaianaPepperoni.preparar());  
console.log(pizzaHawaianaPepperoni.hornear());  
console.log(pizzaHawaianaPepperoni.empacar());
```

Imagen 13: Código de creación o instanciación de un objeto de la clase Pizza

Luego, podemos tener acceso a sus **Métodos**, tal como se ilustra en la imagen 13. En este ejemplo, vamos a “Preparar” una Pizza Hawaiana, y creamos el objeto: **pizzaHawaianaPepperoni**

Al hacerlo podemos ver que tenemos acceso a los **Métodos** definidos en la clase, y al ejecutar cada uno de ellos, nos retorna un mensaje del estado en el que se encuentra la Pizza.

```
PROBLEMS  TERMINAL  ...  2: zsh  v  +  []  
  
labs % node poo.js  
Preparando pizza Mediana, Masa Gruesa, Ingredientes:  Queso,Pepperoni,Piña  
Pizza {  
  masa: 'Masa Gruesa',  
  tamaño: 'Mediana',  
  ingredientes: [ 'Queso', 'Pepperoni', 'Piña' ] }  
La Pizza está en el horno...  
Pizza {  
  masa: 'Masa Gruesa',  
  tamaño: 'Mediana',  
  ingredientes: [ 'Queso', 'Pepperoni', 'Piña' ] }  
Empacando la Pizza...  
Pizza lista para ser enviada! :)  
Pizza {  
  masa: 'Masa Gruesa',  
  tamaño: 'Mediana',  
  ingredientes: [ 'Queso', 'Pepperoni', 'Piña' ] }
```

Imagen 14: Mensajes que presentan cuando se instancia un Objeto de la Clase Pizza

Ahora y usando la misma clase, vamos a crear otra instancia, de una pizza diferente:



```
const pizzaPolloChampinones = new Pizza("Masa Delgada","Grande",  
[ "Queso","Pollo","Champiñones" ] );  
  
console.log(pizzaPolloChampinones.preparar());  
console.log(pizzaPolloChampinones.hornear());  
console.log(pizzaPolloChampinones.empacar());
```

Imagen 15: Otro ejemplo de código de creación o instanciación de un objeto de la clase Pizza

Creamos un objeto diferente que tiene los mismos **Métodos** pero con diferentes **Atributos**, ya que esta Pizza es diferente en su masa, tamaño e ingredientes.

```
PROBLEMS  OUTPUT  TERMINAL  ...  2: zsh  +  []  
  
labs % node poo.js  
Preparando pizza Grande, Masa Delgada, Ingredientes: Queso,Pollo,Champiñones  
Pizza {  
  masa: 'Masa Delgada',  
  tamano: 'Grande',  
  ingredientes: [ 'Queso', 'Pollo', 'Champiñones' ] }  
La Pizza está en el horno...  
Pizza {  
  masa: 'Masa Delgada',  
  tamano: 'Grande',  
  ingredientes: [ 'Queso', 'Pollo', 'Champiñones' ] }  
Empacando la Pizza...  
Pizza lista para ser enviada! :)  
Pizza {  
  masa: 'Masa Delgada',  
  tamano: 'Grande',  
  ingredientes: [ 'Queso', 'Pollo', 'Champiñones' ] }
```

Imagen 16: Mensajes que presentan cuando se instancia un Objeto de la Clase Pizza

Polimorfismo

Es la capacidad de procesar de diferentes maneras los objetos, lo que nos permite re utilizar Atributos y Métodos entre las clases, permitiendo enviar mensajes con la misma estructura a objetos de distintos tipos. En nuestro caso, creamos dos objetos diferentes, que utilizaban los mismos tipos de Atributos y Métodos.



Herencia

La Herencia está orientada a la re utilización, es la forma en la que una Clase hereda los **Atributos** y **Métodos** de una “Clase Padre”, para poder ser re utilizadas, sin embargo, la clase que hereda estas características, también tiene sus propios **Atributos** y **Métodos**.

A continuación vemos una nueva clase llamada “Combos”, pero en esta oportunidad la clase “Combos” hereda los **Atributos y Métodos** de la Clase principal (Clase Pizza), por lo que creamos nuestro constructor, (es lo primero que se ejecuta al instanciar la clase), por lo tanto, cuando una clase hereda de otra, usamos la palabra reservada **super()**, que invoca el Constructor de la clase padre.

```
class Combos extends Pizza{  
    constructor(tamano,masa,ingredientes){  
        super(tamano,masa,ingredientes);  
    }  
}
```

Imagen 17: Código ejemplo del método constructor aplicando el concepto de Herencia

Por esta razón, dentro de **Super()**, definimos los atributos de la misma manera que están definidos en el constructor de la clase Pizza, tal como lo ilustra la imagen 17.

Luego, agregamos nuevos **Atributos** a la clase: *combo*, *cantidad*, *bebida* y *postre*



```
class Combos extends Pizza{  
  
    constructor(combo,cantidad,tamano,masa,ingredientes,bebida,postre){  
        super(tamano,masa,ingredientes);  
        this.combo = combo;  
        this.cantidad = cantidad;  
        this.bebida = bebida;  
        this.postre = postre;  
    }  
}
```

Imagen 18: Código completo del método constructor aplicando el concepto de Herencia

Una vez agregamos estos nuevos **Atributos** como los tenemos en la imagen 18, vamos a crear un nuevo **Método**:

```
elegirCombo(){  
    this.preparar();  
    this.hornear();  
    this.empacar();  
  
    console.log(`Pedido: Combo ${this.combo} - ${this.cantidad}  
    Pizza ${this.porciones}, ${this.masa}, ${this.ingredientes}  
    + ${this.bebida} + ${this.postre}`);  
    return this;  
}
```

Imagen 19: Código ejemplo del método “elegirCombo”

A continuación se presenta el código completo de la clase Combos.



```
class Combos extends Pizza{

    constructor(combo,cantidad,tamano,masa,ingredientes,bebida,postre){
        super(tamano,masa,ingredientes);
        this.combo = combo;
        this.cantidad = cantidad;
        this.bebida = bebida;
        this.postre = postre;
    }

    elegirCombo(){
        this.preparar();
        this.hornear();
        this.empacar();

        console.log(`Pedido: Combo ${this.combo} - ${this.cantidad}
        Pizza ${this.porciones}, ${this,this.masa}, ${this.ingredientes}
        + ${this.bebida} + ${this.postre}`);
        return this;
    }
}
```

Imagen 20: Código completo de la Clase “Combos”

Una vez terminado nuestro Método, el cual se encargará de definir el tipo de combo, y lo que éste incluye, también nos traerá los Métodos heredados de la clase Pizza.

Ahora creamos un objeto llamado **Pedido**:

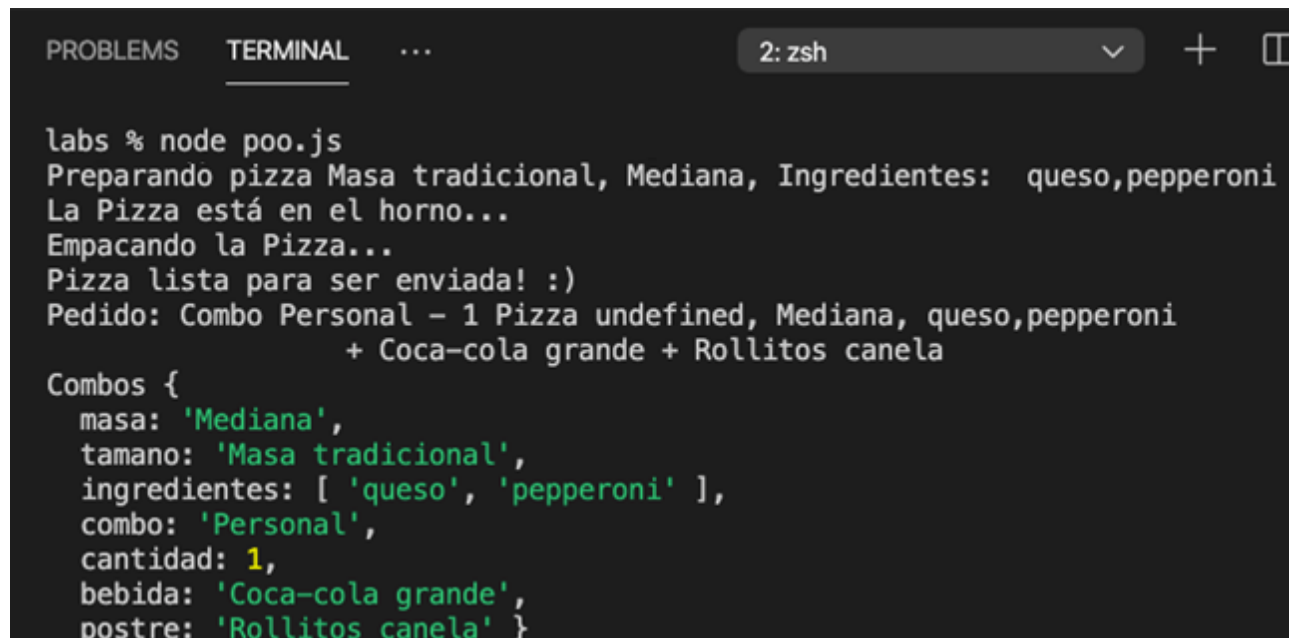
```
const Pedido = new Combos("Personal",1,"Mediana","Masa tradicional",["queso","pepperoni"],
"Coca-cola grande","Rollitos canela",1);

console.log(Pedido.elegirCombo());
```

Imagen 22: Código completo Objeto de la Clase “Combos”



Donde definimos en el primer Parámetro el **tipo de combo, cantidad, tamaño, masa, ingredientes, bebida y postre** que incluye



```
labs % node poo.js
Preparando pizza Masa tradicional, Mediana, Ingredientes: queso,pepperoni
La Pizza está en el horno...
Empacando la Pizza...
Pizza lista para ser enviada! :)
Pedido: Combo Personal - 1 Pizza undefined, Mediana, queso,pepperoni
+ Coca-cola grande + Rollitos canela

Compos {
  masa: 'Mediana',
  tamano: 'Masa tradicional',
  ingredientes: [ 'queso', 'pepperoni' ],
  combo: 'Personal',
  cantidad: 1,
  bebida: 'Coca-cola grande',
  postre: 'Rollitos canela' }
```

Imagen 23: Resultado al instanciar del Objeto Pedidos que instancia la clase “Compos”.

Finalmente, se ejecutan los **Métodos** de **preparar()**, **hornear()** y **empacar()** heredados de la clase Pizza.

Donde se procesa el pedido de un Combo personal, que re utiliza los **Atributos** y **Métodos** propios de la clase Pizza.

Bibliografía

Programming JavaScript Applications: Robust Web Architecture with Node, HTML5, and Modern JS Libraries. 253 Pages · 2014 · English. by Eric Elliott. **node js**

Composing Software. 229 Pages · 2019 · English. by Eric Elliott.