SERVICIO NACIONAL DE APRENDIZAJE – SENA

CENTRO DE COMERCIO REGIONAL ANTIOQUIA

TECNOLOGÍA EN ANÁLISIS Y DESARROLLO DE SOFTWARE - 2675805

Evidencia de conocimiento: GA4-220501095-AA2-EV01 - Taller de conceptos y principios de programación orientada a objetos

DANIEL FELIPE ARIAS CORREDOR

2023

Introducción

Este documento presenta un glosario sobre los términos clave para entender la Programación Orientada a Objetos o POO. Se explica cada término de manera sencilla y se da un ejemplo que ayuda a su comprensión.

Glosario

- **POO**: La programación orientada a objetos o POO es un paradigma en programación, el cual hace hincapié en trasladar la naturaleza de los objetos de la vida real a las entidades que se crean en código, por tanto, los objetos del POO tienen un estado, un comportamiento (que pueden hacer) y unas propiedades. Como ejemplo, un objeto creado en código denominado '*automovil_1*' puede tener atributos similares a uno de la vida real (color, peso, modelo), así como comportamientos (arrancar, frenar) y un estado (detenido, circulando). Los programas se modelan en torno a objetos.
- **Clases**: Es un modelo o plantilla que contiene características comunes a un grupo de objetos. Al crear un objeto se debe llamar a esta clase o instanciarla para que el objeto herede sus atributos. Por ejemplo, se ha creado la clase '*automovil*' con atributos como '*numero_ruedas'*, '*peso'* y *'motor'* y comportamientos como *Arrancar(),* al crear un objeto como automóvil_1 perteneciente a esta clase, el objeto heredará los comportamientos y atributos con un valor predefinido.
- **Objeto**: Es una entidad similar la de la vida real con el que el programa puede interactuar. Un objeto consta de estados, atributos y comportamiento, además de pertenecer a una clase. Ej. *automovil_1* y '*automóvil_2*' son objetos diferentes, aunque compartan una clase y unas propiedades.

- **Atributo**: Son las propiedades que vienen a poseer los objetos dentro de una clase. Los objetos pertenecientes a una clase tienen los mismos atributos, pero sus valores pueden diferir. Ej. *automovil_1* tiene los atributos '*numero_ruedas'*, '*peso'* y *'motor',* al igual que *automovil_2,* aunque para este último el valor de *'motor'* es de 1800 frente a 2100 de *automovil_1.*
- **Método:** Es una función creada para una clase y viene a representar el comportamiento de sus objetos. Determinan las tareas básicas que realizará el objeto. Ej. La clase *automóvil* contiene el método *Arrancar()*, por lo que al ser llamado para un objeto como parámetro: "*>Arrancar(automóvil_1)",* el objeto iniciará este comportamiento.
- **Herencia**: Uno de los pilares del POO. Permite que una clase herede o traiga para sí características provenientes de otras clases. Esta herencia se da en un orden jerárquico, lo que quiere decir que una clase hereda de una clase padre o superclase. Siguiendo el ejemplo, si a la clase *automovil* le es creada una subclase llamada '*camioneta*', esta última puede heredar los atributos de su superclase, por lo que *camioneta* tendrá los atributos '*numero_ruedas'*, '*peso'* y *'motor'* y el comportamiento *Arrancar()*. A la hora de asignar esta herencia se debe apelar a la lógica y considerar la regla que un objeto de la subclase pueda pertenecer a la superclase (una camioneta sí es un automovil).
- **Encapsulamiento:** Otro pilar. Consiste en la posibilidar de aislar ciertos comportamientos o propiedades dentro de una clase para que solo sean visibles y modificados dentro de la misma clase, consiguiendo que no puedan ser accedidos por otras clase o métodos. Ej. La clase automóvil solo tiene como público *Arrancar() y* '*numero_ruedas',* siendo '*peso'* y *'motor'* atributos privados e inalterables.
- **Abstracción:** Tercer pilar. Permite representar características de una clase al exterior, pero ocultado su complejidad interna. No sólo controla el acceso a la información como el encapsulamiento, sino también oculta la complejidad de los métodos. La abstracción junto con el encapsulamiento proporciona modularidad al proyecto de software, permitiendo que otro

módulo o el usuario solo pueda hacer uso de las propiedades o métodos habilitados para un módulo, siendo el resto de la clase una especie de caja negra.

- **Polimorfismo:** Ultimo pilar. Permite que un objeto se pueda comportar de forma diferente dependiendo del contexto. Un objeto de una subclase puede reemplazar a uno de la superclase, por lo que se puede usar de manera indistinta siempre y cuando tenga una base en común. Ej. Se pueden usar un objeto *automóvil_1* y el objeto *motocicleta_1* indistintamente para la superclase *automóvil,* aunque tengan sus peculiaridades internas.