POO: Programación Orientada a Objetos

Entendiendo el Paradigma

Autor: Luz Elena Galindo Gamboa

Fundamentos

Autor: Luz Elena Galindo Gamboa

La Modularización nos permite dividir

La modularización es la **descomposición** de un sistema en distintas partes que en conjunto colaboran entre sí para cumplir un objetivo.

Modularizar un sistema nos permite descomponerlo en distintas piezas de código o módulos.

Autor: Luz Elena Galindo Gamboa

La Abstracción facilita la cohesión

Es un **proceso mental** en el cual tomamos únicamente las características y comportamientos de una Entidad (el qué) e ignoramos el detalle y su complejidad (el cómo).

La cohesión se refiere a la coherencia, integridad y constancia en cada una de las piezas de código.

Con un correcto proceso de abstracción podemos crear **piezas de código con alta cohesión**, es decir, son fáciles de entender, ante las mismas condiciones funcionan igual y hacen justo lo que tienen que hacer (no más, no menos).

Autor: Luz Elena Galindo Gamboa

El Encapsulamiento evita el acoplamiento

El encapsulamiento nos permite **empaquetar** y ocultar los detalles y la complejidad de una abstracción, dejando disponible únicamente aquello que queremos exponer a otras piezas de código.

El acoplamiento se refiere a la dependencia entre las distintas partes de un sistema.

Con un correcto proceso de encapsulamiento podemos lograr tener **piezas de código poco acopladas**, es decir, que no dependen entre ellas y permiten los cambios sin afectarse entre sí.

Autor: Luz Elena Galindo Gamboa

La Jerarquización nos permite organizar

La jerarquización nos permite **estructurar y organizar** un conjunto de elementos o módulos de acuerdo al criterio que más nos convenga, ya sea responsabilidad, dependencia, composición, clasificación, etc.

Composición: Es cuando unos elementos podemos decir que están compuestos de otros, o que unos elementos están presentes en otros.

Clasificación: Este tipo de jerarquización indica que unos elementos son una especialización de otros.

Autor: Luz Elena Galindo Gamboa

Conclusión

La **modularización** y la **jerarquización** nos permite descomponer de forma lógica un sistema en distintas **piezas de código independientes** (poco acopladas) **y que tienen sentido** (alta cohesión).

Autor: Luz Elena Galindo Gamboa

Clases y Objetos

Autor: Luz Elena Galindo Gamboa

Clases

Una clase es una plantilla que **representa una entidad lógica**. Define ciertos características y funcionalidades para una entidad.

Una clase también puede ser vista como un tipo de dato.

Nota: por si sola una clase no hace nada, debe crearse un objeto a partir de una clase para utilizar sus características y funcionalidades.

Miembros que puede contener una clase

- **Constantes**: Valores constantes asociados a la clase.
- **Campos**: Variables de la clase.
- **Métodos**: Cálculos y acciones que pueden realizarse mediante la clase.
- **Propiedades**: Acciones asociadas a la lectura y escritura de propiedades con nombre de la clase.
- **Indizadores**: Acciones asociadas a la indexación de instancias de la clase como una matriz.
- **Eventos**: Notificaciones que puede generar la clase.
- **Operadores**: Conversiones y operadores de expresión admitidos por la clase.
- **Constructores**: Acciones necesarias para inicializar instancias de la clase o la clase propiamente dicha.
- **Finalizadores**: Acciones que deben realizarse antes de que las instancias de la clase se descarten de forma permanente.
- **Tipos**: Tipos anidados declarados por la clase.

Autor: Luz Elena Galindo Gamboa

Objetos

Es la **materialización de una clase**. A la creación de un objeto a partir de una clase se le llama **instancia**. Al instanciar un objeto podremos hacer uso de las características y funcionalidades definidas por la clase.

Ya que las clases pueden ser vistas como un tipo de dato, al instanciar un objeto, estamos creando un objeto de cierto tipo de dato.

Cuando las Propiedades de un Objeto toman valores se le llama **Estado**.

Cuando a un Objeto le decimos que ejecute un Método se le llama **Mensaje**.

Autor: Luz Elena Galindo Gamboa

Pilares de POO

Autor: Luz Elena Galindo Gamboa

Encapsulamiento

El encapsulamiento nos permite **controlar la visibilidad y el acceso** a las propiedades y métodos de un objeto.

Con esto podemos evitar la modificación de las propiedades o la ejecución de los métodos de forma imprevista.

También nos permite **empaquetar u ocultar la lógica** implementada para los atributos y métodos, dejando expuesto sólo aquello que pueda interesarle a otras piezas de código.

Autor: Luz Elena Galindo Gamboa

Herencia

La Herencia nos permite crear **clases basadas en otras clases**.

Una clase derivada hereda las propiedades y métodos de una clase base, además puede extender la funcionalidad de la clase base agregando sus propias propiedades y métodos.

Autor: Luz Elena Galindo Gamboa

Polimorfismo

El polimorfismo nos permite **intercambiar objetos de diferentes clases** (o de diferente tipo) mientras dichos objetos sean derivados de la misma clase base.

Autor: Luz Elena Galindo Gamboa

Abstracción

La abstracción es un **proceso mental** donde identificamos las partes esenciales de una entidad e ignoramos el detalle o el funcionamiento.

La abstracción nos permite **enumerar** las propiedades y métodos esperados de una clase **sin entrar a detalle** de la definición de los mismos.

Autor: Luz Elena Galindo Gamboa

Conceptos aplicados

Autor: Luz Elena Galindo Gamboa

Niveles de acceso

Controla quién puede tener acceso al miembro de una clase.

- **public**: Acceso no limitado.
- **protected**: Acceso limitado a esta clase o a las clases derivadas de esta clase.
- **internal**: Acceso limitado al ensamblado actual (dado por el namespace).
- **protected internal:** Acceso limitado a la clase contenedora, las clases derivadas de la clase contenedora, o las clases dentro del mismo ensamblado.
- **private:** Acceso limitado a esta clase.
- **private protected:** Acceso limitado a la clase contenedora o a las clases derivadas de la clase contenedora dentro del mismo ensamblado.

Autor: Luz Elena Galindo Gamboa

Clase Base y Clase Derivada

Una **clase base** es aquella que ninguno de sus miembros depende de otra clase.

Una **clase derivada** es aquella que hereda a los miembros de su clase base.

La herencia significa que una clase contiene implícitamente todos los miembros de su clase base, excepto la instancia y los constructores estáticos, y los finalizadores de la clase base.

Una clase derivada puede agregar nuevos miembros a aquellos de los que hereda, pero no puede quitar la definición de un miembro heredado.

Autor: Luz Elena Galindo Gamboa

Constructores

Los **constructores** de instancias crean e **inicializan las variables al crear un objeto** de una clase. Los constructores usan el mismo nombre de la clase y pueden recibir parámetros para permitir especificar los valores iniciales.

Los constructores de instancias también se pueden usar para llamar a los constructores de instancias de las clases base, usando ***: base***

Autor: Luz Elena Galindo Gamboa

Métodos Estáticos y de Instancia

Un **miembro estático** de la clase es una propiedad, un método o un campo estático. Los miembros estáticos pueden ser **utilizados sin** necesidad de **crear una instancia** de una clase y sólo pueden tener acceso a otros miembros estáticos.

Un **método estático** se declara con el modificador ***static***.

Un **método de instancia** se declara sin el modificador static. Para ser utilizado se debe instanciar la clase y puede acceder a miembros estáticos y de instancia.

Autor: Luz Elena Galindo Gamboa

Clases Estáticas

Las **clases estáticas** se declaran con el modificar ***static***. Las clases estáticas solo tienen miembros estáticos y **no se pueden instanciar**.

Las clases estáticas son apropiadas para **funcionalidad relativamente invariable**, universal y que no sea fundamental para nuestro software.

Métodos Virtuales y Abstractos

Los **métodos virtuales** se declaran con el modificador ***virtual***. Los métodos virtuales implementan su funcionalidad y permiten (opcionalmente) a las clases derivadas sobreescribir su funcionalidad utilizando el modificador ***override***.

Si no se utiliza el modificador virtual se dice que el método es un **método no virtual**. Un método no virtual también puede ser sobreescrito en una clase derivada utilizando el modificador ***new***.

Los **métodos abstractos** se declaran con el modificador ***abstract***. Los métodos abstractos no tienen implementación de su funcionalidad y obligan a las clases derivadas a sobreescribir su funcionalidad utilizando el modificador ***override***.

Autor: Luz Elena Galindo Gamboa

Clases Abstractas

Las **clases abstractas** se declaran con el modificar ***abstract***. Las clases abstractas tiene definidos métodos abstractos y **no se pueden instanciar**.

Las clases derivadas de una clase abstracta deberán definirse como abstractas si no implementan todos los métodos abstractos de la clase base.

Autor: Luz Elena Galindo Gamboa

Interfaces

Una **interfaz** define un contrato que debe cumplirse por cualquier clase que implemente la interfaz. Una interfaz puede contener métodos, propiedades, eventos e indexadores.

Una interfaz **no proporciona implementaciones** de los miembros que define, simplemente **especifica los miembros que se deben proporcionar mediante clases que implementan la interfaz**.

Clases y Métodos Sellados

Las **clases selladas** se declaran con el modificar ***sealed***. Las clases selladas **no se pueden heredar**.

Se puede utilizar el modificador ***sealed*** con métodos o propiedades **override**, es decir, que sobreescriben métodos o propiedades virtuales de una clase base, para evitar que sean sobreescritos en una clase derivada de la clase actual.

Autor: Luz Elena Galindo Gamboa

Fuentes

Autor: Luz Elena Galindo Gamboa

Fuentes

https://desarrolloweb.com/manuales/teoria-programacion-orientada-objetos.html

https://desarrolloweb.com/articulos/poo-fundamentos-luis-fernandez.html

https://desarrolloweb.com/articulos/beneficios-poo-resumen.html

https://desarrolloweb.com/articulos/499.php

https://desarrolloweb.com/articulos/metodos-atributos-static-poo.html

https://desarrolloweb.com/articulos/herencia-en-programacion-orientada-objetos.html

https://desarrolloweb.com/articulos/polimorfismo-programacion-orientada-objetos-concepto.html

https://desarrolloweb.com/articulos/abstraccion-programacion-orientada-objetos-poo.html

https://gavilanch.wordpress.com/2018/07/05/los-4-pilares-de-la-programacion-orientada-a-objetos/

https://gavilanch.wordpress.com/2019/07/25/introduccion-a-las-clases-estaticas-cuando-utilizarlas/

Autor: Luz Elena Galindo Gamboa

Fuentes

https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/concepts/object-oriented-programming

https://docs.microsoft.com/es-es/dotnet/csharp/tour-of-csharp/classes-and-objects

https://geeks.ms/jorge/2011/11/17/clase-base-interface-y-clase-abstracta-amigas-o-enemigas-lo-que-un-dummy-debe-saber/#:~:text=Una%20clase%20base%20es%20o,clase%20base%20de%20forma%20directa)

Autor: Luz Elena Galindo Gamboa