Tecnológico nacional de México

Instituto tecnológico de Tijuana

Subdirección Académica

Departamento de sistemas  y computación

SEMESTRE:

Enero-Junio 2019

CARRERA:

**Ingeniería en Sistemas Computacionales**

MATERIA:

Programación Orientada a Objetos

Nombre del trabajo:

**Introducción al paradigma de la programación orientado a objetos**

UNIDAD A EVALUAR:

Unidad I

NOMBRE Y NÚMERO DE CONTROL DEL ALUMNO:

Carmona Hernandez Oscar Jafet 18212156

**Paradigma**

Un paradigma es un punto de vista acerca de un caso en particular, tanto de qué maneras se puede visualizar como resolver en caso de ser un problema.

Ahora pasando a lo que es un paradigma de programación, no es tan alejado a la definición original, ver algo de maneras diferentes, pero aqui es basandose en eso y llevándolo junto a la construcción de un programa con esas bases, y respecto a cómo se visualizan se conseguirá diferente resultado. Con respecto a estas maneras de seguir una secuencia específica de instrucciones basadas para organizar un programa, dependiendo el paradigma usado se podrá conseguir distintas variaciones de salida de datos.

**Abstracción**

Es la maneraque se extraen características principales de un objeto de la vida real y de esta manera poder transmitirlo en una forma de código al momento de realizar la programación orientada a objetos, con el fin de lograr un resultado congruente en cada uno de los atributos con los que conlleva la estructura del objeto.

* **Ejemplos**
* Ejemplo 1: ¿Qué características podemos abstraer de los automóviles? o ¿Qué características semejantes tienen todos los automóviles?

Características: Marca, Modelo, Número de chasis, Peso llantas o cauchos, Puertas, Ventanas... Comportamiento: Acelerar, Frenar, Retroceder...

* Ejemplo 2: La gerencia de un taller mecánico necesita un sistema para controlar los vehículos que ingresan a sus instalaciones. En este caso, las características esenciales de la clase vehículo son: Marca, Modelo, Color, Falla detectada, Nombre del Propietario, Dirección del Propietario, Teléfono del Propietario.

**¿Por qué decimos que la *Clase* es el mecanismo de abstracción de los lenguajes OO?**

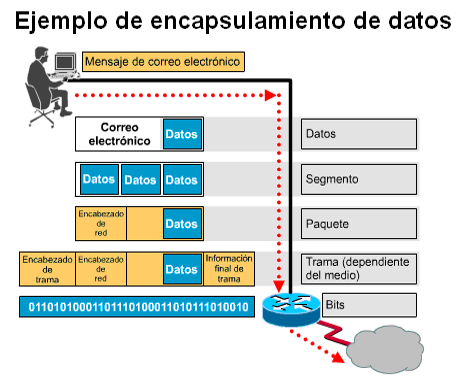
Esto se dice por qué en una clase se encuentran unos elementos llamados atributos, en los cuales se depositaran las características encontradas gracias a la abstracción que se realizó en algún objeto, y gracias a estas relaciones se logra el mecanismo de abstracción en los lenguajes orientados a objetos.

**Encapsulamiento**

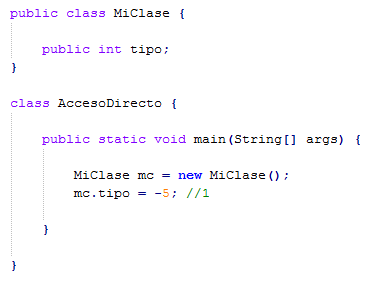
Se le dice encapsulamiento a la acción de ocultar o esconder los datos con los cuales cuenta un objeto, es decir, que solo se puedan acceder mediante los métodos necesarios establecidos por quien lo diseño y no por otros como podría ser por ejemplo, los usuarios.

* Ejemplos:

Con



Sin



**La importancia del encapsulamiento**

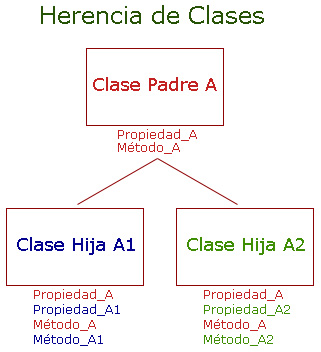
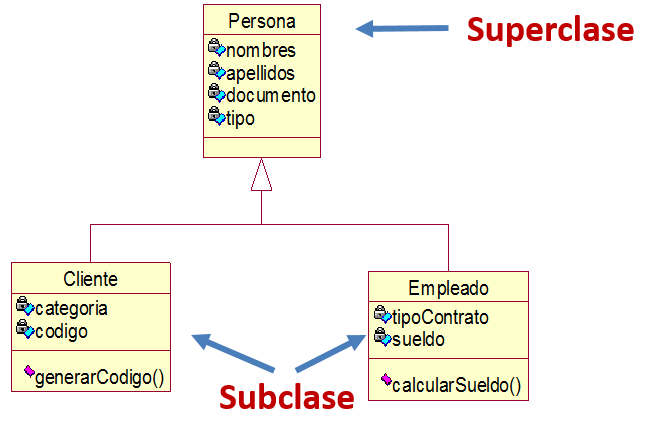
Este método es importante por qué nos permite poner los atributos de nuestras clases en diferentes niveles de acceso, las cuales al momento de programar se pueden ir dando para usarlas, cada una tiene su importancia y momento de colocarlas. Sus niveles son tres, el publico, asi como su nombre lo dice se puede usar en todos los lugares sin problema por qué es visible; el protegido, a este es casi igual que el público pero solo con derivaciones que sean de la misma clase, como por ejemplo los atributos de una clase auto se van a la clase autobús, esto es correcto ya que se utilizan los mismos tipos de datos. Y por último está la privada, esta no muestra sus atributos a nadie y se ocupan de diferentes métodos para poder utilizarse en diferentes partes del código.

**Problemas que se pueden evitar con el uso de encapsulamiento**

Gracias a este método de encapsulamiento se evita el problema de que el código quede desordenado ya qué se utiliza un camino especificado al momento de declarar se utiliza algún tipo de acceso (Público, Protegido, Privado) y con esto se tiene desde un principio la forma establecida de clasificar los atributos del objeto.

**Concepto de Herencia**

La herencia es la manera de crear una clase y reutilizando las bases de otra creando subclases, esto llega a hacer muy práctico por qué nos evitamos rediseñar todo el bloque de una clase y con eso simplemente se puede volver a usar los atributos, un ejemplo de esto es de una clase carro (llantas, motor, ventanas) pueden usarse para crear una clase que se llama autobús, ya que se utilizan los mismo atributos para crear la clase.



**UML: Diagrama de clases**

**Historia de UML**

Después de que la *Rational Software Corporation* contratara a James Rumbaugh de General Electric, en [1994](https://es.wikipedia.org/wiki/1994), la compañía se convirtió en la fuente de los dos esquemas de modelado orientado a objetos más populares de la época: *Object-Modeling Technique* (OMT) de Rumbaugh, que era mejor para análisis orientado a objetos, y el Método Booch (de Grady Booch) que era mejor para el diseño orientado a objetos. Poco después se les unió [Ivar Jacobson](https://es.wikipedia.org/wiki/Ivar_Jacobson), el creador del método de ingeniería de software orientado a objetos. Jacobson se unió a Rational, en [1995](https://es.wikipedia.org/wiki/1995), después de que su compañía Objectory AB fuera comprada por Rational. Los tres metodologistas eran conocidos como los *Tres Amigos*, porque se sabía de sus constantes discusiones sobre las prácticas metodológicas.

En 1996 Rational concluyó que la abundancia de lenguajes de modelado estaba alentando la adopción de la tecnología de objetos, y para orientarse hacia un método unificado, encargaron a los *Tres Amigos* que desarrollaran un "lenguaje unificado de modelado" abierto. Eligieron "cajas" para representar clases en lugar de la notación de Booch que utilizaba símbolos de "nubes".

Bajo la dirección técnica de los *Tres Amigos* (Rumbaugh, Jacobson y Booch) fue organizado un consorcio internacional llamado *UML Partners* en 1996 para completar las especificaciones del UML, y para proponerlo como una respuesta al OMG RFP. El resultado de este trabajo, el UML 1.1, fue presentado ante la OMG en agosto de 1997 y adoptado por la OMG en noviembre de 1997.

**Usos de los diagramas de clases**

* El principal uso que se le da a los diagramas de clases, es simplemente el modelar de manera estática las vistas de diseño de cualquier proyecto OO (orientado a objetos).
* Por demás está decir, que también son utilizados para tomar decisiones acerca de las abstracciones del sistema y de la finalidad del programa.

**Herramientas para el modelado en UML**

**LUCIDCHART**

Basado en HTML5. Con soporte para UML. Permite también la colaboración online en tiempo real. Puede importar ficheros Visio con lo que es una buena alternativa para aquellos equipos que estén buscando una alternativa más ágil a Visio y que sea basada en web. Además de UML, también incluye plantillas para crear modelos ER, procesos de negocio, diagramas de red y muchos otros tipos de modelos.

**GENMYMODEL**

GenMyModel empezó como una herramienta sólo para UML pero desde entonces se ha ido expandiendo hasta cubrir áreas complementarias como sería el modelado de procesos de negocio con BPMN y el modelado de la arquitectura empresarial con Archimate. Ofrece un repositorio centralizado para los equipos (con posibilidad de definir políticas de restricción de acceso) para el modelado colaborativo simultáneo.

A diferencia de otras herramientas en la lista, GenMyModel es una herramienta de modelado pura (con las ventajas e inconvenientes que esto conlleva, todo depende para qué se quieran usar los modelos). Por lo tanto, GenMyModel tiene algunas características únicas como el soporte para exportar modelos como XMI (el “estándar” para el intercambio de modelos) y la generación de código a varios lenguajes.

**GLIFFY**

Con un fuerte énfasis en los aspectos colaborativos y de control de versiones, Gliffy dice ser la “herramienta más popular para la creación de diagramas online” (aunque me pregunto en base a qué datos lo saben). Gliffy permite modelar todos los diagramas UML así como una variedad de otros tipos de diagramas, por ejemplo, soporta muy bien los modelos de proceso BPMN.

**DRAW.IO**

Viene con las formas básicas para los elementos UML, ER y BPMN.

Como punto negativo, esta es claramente una herramienta que no entiende nada de lo que estás modelando. Puedes hacer lo que quieras sin ningún tipo de control. Muy peligroso si no tienes ya cierta experiencia. Tampoco permite la colaboración a la hora de dibujar pero a cambio se integra bien con Google Drive, dropbox, OneDrive lo que permite guardar los modelos en tu herramienta favorita.

Draw.io es open source y se ha implementado con la librería mxGraph.

**CREATELY**

Muy buena colaboración en tiempo real. Con más de 50 tipos de diagramas y miles de ejemplos para empezar a modelar lo más rápidamente posible. También puedes trabajar en modo offline y sincronizar tus modelos después. El único PERO que le veo es que, a día de hoy, necesita Flash para funcionar. Espero que esto lo cambien pronto ya que Adobe ha decidido acabar con Flash. Tienen tiempo pero ¿quién quiere usar una herramienta que utiliza una tecnología con fecha de caducidad?.

Podéis leer más acerca de la herramienta y sus puntos más importantes aquí.

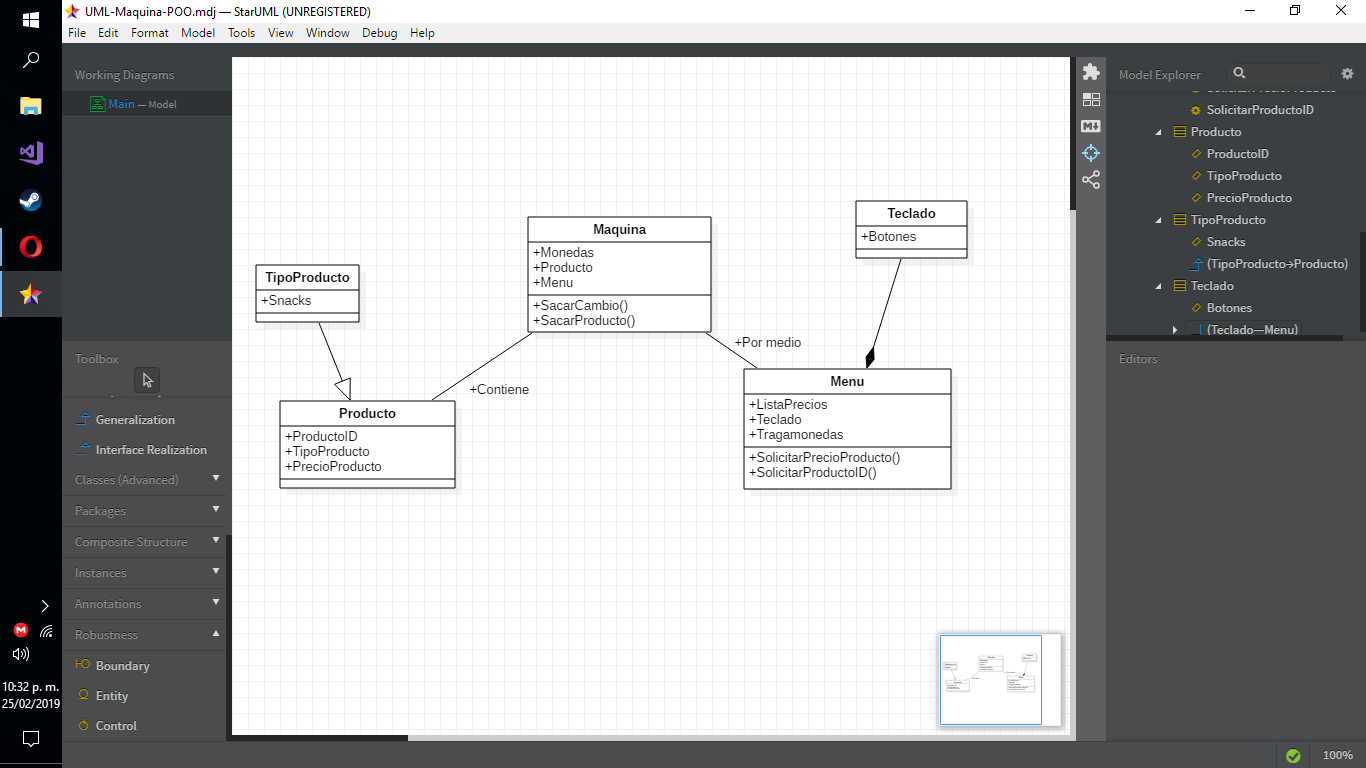
**CACOO**

Herramienta de dibujo con algunas funcionalidades interesantes como el poder guardar múltiples versiones del mismo modelo así como un control de cambios y la posibilidad de que varias personas añadan comentarios al modelo para discutirlo asíncronamente online. Cacoo soporta los diagramas de casos de uso, secuencia, clase, actividades y estados.

**Empresa local con el uso de UML**

Como jamás he utilizado esta herramienta hasta la fecha de realizar esta tarea, no puedo decir con seguridad si alguna empresa lo utiliza, pero por su forma visual de organizarse tampoco lo he visto en algún lugar.

**Propuesta de una máquina de venta de productos utilizando diagrama de clases**

****