DEPARTAMENTO DE INFORMÁTICA Y DISEÑO

PROGRAMACIÓN ORIENTADA A OBJETOS

UNIDAD I
PILARES DE LA PROGRAMACIÓN ORIENTADA A OBJETOS

ÍNDICE DE TEMAS

- Introducción
  - Introducción a orientación a objetos.
  - Introducción al UML
- Conceptos base de orientación a objetos
  - Objeto
  - Mensaje
  - Clase
    - Cómo construir una clase en Python
  - Encapsulación

INTRODUCCIÓN

INTRODUCCIÓN A ORIENTACIÓN A OBJETOS

- El proverbio "*El hábito no hace al monje*" se aplica perfectamente a la tecnología de objetos.
    - El hecho de conocer un lenguaje orientado a objetos (ej.: Java) y además tener un acceso a una rica biblioteca es un primer paso necesario pero insuficiente para crear sistemas de objetos.
    - Se requiere además analizar y diseñar un sistema desde la perspectiva de objetos.
- Es necesario "Pensar en objetos"

INTRODUCCIÓN A ORIENTACIÓN A OBJETOS

- Cuando hacemos referencia a la programación orientada a objetos (POO) no estamos hablando de unas cuantas características nuevas añadidas a un lenguaje de programación.
    - Es una nueva forma de pensar acerca del proceso de descomposición de problemas y de desarrollo de soluciones de programación (paradigma).
    - Nos centramos en descomponer en objetos.
        - Esta descomposición nos lleva, ya no a pensar en acciones (verbos) o en lo que hace el problema, sino en cuál es el escenario real del mismo e intentar simular ese escenario en nuestro programa.

INTRODUCCIÓN A ORIENTACIÓN A OBJETOS

- Preguntas cómo:
    - ¿Cómo deberíamos asignar las responsabilidades a las clases de objetos?
    - ¿Cómo deberían interactuar los objetos?
    - ¿Qué clases deberían hacer que?
- Son claves a la hora de hablar del diseño de un sistema.

INTRODUCCIÓN A ORIENTACIÓN A OBJETOS

- Pero, ¿Qué es el análisis y diseño?
    - **Análisis** pone énfasis en una investigación del problema y los requisitos, no en la manera de definir la solución.
        - Ej.: Si se desea un nuevo sistema de información para una biblioteca, ¿Cómo se utilizaría?
    - **Diseño** pone énfasis en una solución conceptual que satisface los requisitos, en vez de colocarlos en la implementación.
        - Ej.: Una descripción del esquema de una base de datos y objetos de software.
- Podemos resumir ambos términos en "*Hacer lo correcto*" (análisis) y "*Hacerlo correcto*" (diseño).

INTRODUCCIÓN A ORIENTACIÓN A OBJETOS

- Observamos que en un problema específico, lo que tenemos son entidades (objetos).
    - Estas entidades poseen un conjunto de *propiedades* o *atributos*, y un conjunto de *métodos* mediante los cuales muestran su *comportamiento*.
    - También podemos descubrir, todo un conjunto de *interrelaciones* entre las entidades, guiadas por el intercambio de *mensajes*.
        - Las entidades de un problema, responden a estos mensajes mediante la ejecución de *acciones*.

INTRODUCCIÓN A ORIENTACIÓN A OBJETOS

- Usando esta manera de descomponer problemas, nos aseguramos la re-usabilidad de nuestro código, es decir, los objetos que hoy escribimos, si están bien escritos, nos servirán para siempre.
    - Si bien de esta manera no se presenta gran innovación con las funciones, lo innovador es que podemos re-usar ciertos comportamientos de un objeto, ocultando aquellos que no nos sirven, o redefinirlos para que el objeto se comporte de acuerdo a las necesidades.
- Ej.: Si tenemos un vehículo y queremos que este sea más rápido, no construimos un vehículo nuevo, simplemente cambiamos el carburador y le añadimos un sistema turbo, pero seguimos usando todas las otras piezas de nuestro vehículo.
    - Desde la perspectiva de OOP, hemos modificado cualidades de nuestro objeto (*métodos*) - el carburador. Y hemos añadido un método nuevo - El sistema turbo.
    - Si vemos la perspectiva de programación tradicional, esto nos hubiese llevado a construir un coche nuevo.

INTRODUCCIÓN A ORIENTACIÓN A OBJETOS

- En términos simples:
    - Si queremos construir un objeto que comparte ciertas cualidades con otro que ya tenemos creado, no tenemos que volver a crearlo desde el principio, simplemente, decimos que queremos usar del antiguo en el nuevo y qué nuevas características tiene nuestro nuevo objeto.
- Si vamos más allá:
    - Con la OOP podemos incorporar objetos que otros han desarrollado e incorporarlos a nuestro proyecto, además de modificar sus comportamientos *sin tener que saber cómo los han construido ellos*.

"EL LENGUAJE UNIFICADO DE MODELADO (UML) ES UN LENGUAJE PARA ESPECIFICAR, VISUALIZAR, CONSTRUIR Y DOCUMENTAR LOS ARTEFACTOS DE LOS SISTEMAS DE SOFTWARE, ASÍ COMO PARA EL MODELO DEL NEGOCIO Y OTROS SISTEMAS NO SOFTWARE"

Object Management Group, 2001

INTRODUCCIÓN A UML

- UML es un lenguaje de modelado, es decir, es un lenguaje cuyo vocabulario y reglas se centran en la representación conceptual y física de un sistema.

- Describe un conjunto de notaciones y diagramas estándar para modelar sistemas mediante orientación a objetos, y describe la semántica esencial de lo que estos diagramas y símbolos significan.

INTRODUCCIÓN A UML

- ¿Qué utilidad tiene?
    - Cada símbolo utilizado en UML tiene una semántica bien definida.
    - Cualquier desarrollador puede interpretar un modelo UML sin ambigüedad.
    - Nos facilita la comunicación al establecer un lenguaje común para entender un proyecto.
    - Nos permite documentar las especificaciones de todas las decisiones de análisis, diseño e implementación.
- Si bien UML no es un lenguaje de programación visual, sus modelos pueden conectarse de forma directa a lenguajes de programación orientados a objetos.
    - Podemos establecer una correspondencia desde un modelo UML y una implementación en un lenguaje de programación - como Java, Python o C++ - mediante generación de código e ingeniería inversa.

CONCEPTOS BASE DE ORIENTACIÓN A OBJETOS

OBJETO

- Es una unidad atómica que encapsulan un estado y un comportamiento.
- Representa una entidad física o abstracta del problema o del espacio de la solución.
    - Ej.: Un objeto de la clase *Coches* puede ser un *Ford Mustang*.
- Un sistema puede verse como un conjunto de objetos autónomos y concurrentes que trabajan de manera coordinada en la consecución u obtención de un fin específico.
- Los objetos poseen las siguientes características fundamentales:
    - Todo objeto posee identidad.
    - Todo objeto posee un estado.
    - Todo objeto posee un comportamiento.

OBJETO

- **Identidad**
  - Cada objeto posee un OID (identificador de objeto) el cual establece la identidad del objeto.
  - Este satisface ciertas características fundamentales:
    - El OID es único para cada objeto del sistema y posee un alcance global dentro del sistema en cuestión. No pueden existir dos objetos con el mismo OID.
    - Un OID es generado en el momento de su creación.
      - Cada vez que se crea un objeto, se establece este OID único, por mas que a simple vista sea igual a otro ya existente.
    - El OID es independiente de las propiedades del objeto, esto quiere decir que es independiente de la estructura y los valores de la misma.
    - El OID persiste durante toda la vida del objeto y si el objeto se elimina el OID del mismo no se vuelve a utilizar.
    - Los OID no se pueden controlar o manipular y la administración de estos es transparente.

OBJETO

- **Estado**
    - El estado se encuentra determinado por los valores que toman los atributos de dicho objeto en un instante dado.
        - Cada atributo toma un valor de un dominio concreto.
            - El dominio corresponde a el conjunto de valores posibles que puede tomar un atributo.
    - El estado de un objeto satisface ciertas características:
        - El estado evoluciona con el paso del tiempo.
        - Algunos atributos pueden ser constantes.
        - El estado de un objeto se ve afectado por el comportamiento de dicho objeto.
        - Las operaciones de un objeto sobre otro también afectan el estado de esto. Las operaciones de un objeto son consecuencia de un estimulo externo representado por medio de un mensaje enviado desde otro objeto.

OBJETO

- **Comportamiento**
  - El comportamiento modela la interacción entre los objetos.
    - La interacción se modela por medio de los mensajes de los objetos.
  - El comportamiento global del sistema queda representado y se basa en la comunicación entre los diferentes objetos que lo componen.

OBJETO

- Un objeto se crea a través de instanciar una clase dentro del código.
- En el ejemplo la variable **persona** es una instancia (objeto) de la clase **Persona.**

```python
class Persona:
    """Una clase persona simple"""
    def __init__(self):
        self.nombre = ""

    def set_nombre(self, nombre):
        self.nombre = nombre

    def get_nombre(self):
        return self.nombre


persona = Persona()
persona.set_nombre("Pedro")

print(persona.get_nombre())
```

MENSAJE

- Corresponde a la unidad de interacción entre objetos.
- Es el soporte de una comunicación que vincula dinámicamente los objetos que fueron separados previamente en el proceso de descomposición del sistema.
- Este desencadenará una acción en el objeto destinatario.
    - Se debe tener dicho método para poder interpretar ese mensaje en el objeto destinatario.
    - Ej.: En el caso de un coche, al apretar el claxon, el objeto claxon envía un mensaje a la bocina para que emita cierto sonido.
- El conjunto de mensajes a los que un objeto puede responder, se denomina protocolo del objeto.

MENSAJE

- La relevancia de este esquema, radica en que el objeto emisor no necesita saber la forma en que el objeto receptor va a realizar la acción. Simplemente este la ejecuta y el emisor se desentiende del como; de hecho ni le importa.
- Para que todo esto sea posible es necesario una buena programación de los eventos y métodos de cada objeto.
    - En el ejemplo de la imagen, el flujo principal le pide al objeto **persona** que le entregue el **nombre**.
    - Es responsabilidad del objeto **persona** responder el mensaje y la implementación detrás de él, no le es relevante al objeto **flujo principal.**
    - De no existir esa implementación, esto fallará indicando que el objeto no es capaz de resolver el mensaje.

```python
class Persona:
    """Una clase persona simple"""
    def __init__(self):
        self.nombre = ""

    def set_nombre(self, nombre):
        self.nombre = nombre

    def get_nombre(self):
        return self.nombre


persona = Persona()
persona.set_nombre("Pedro")

print(persona.get_nombre())
```

CLASE

- Es una descripción de un conjunto de objetos similares.
    - Es una manera de abstracción de la realidad.
- Define el ámbito de definición de un conjunto de objetos.
    - Contiene los atributos y las operaciones sobre esos atributos que hacen que una clase tenga la entidad que se desee.

CLASE

- Cada objeto que existe en el sistema pertenece a una clase específica que dio su origen.
    - Se podría pensar que una clase es la gran fabrica de objetos que poseen las características de esa clase.
    - O la podemos pensar como la matriz para la fabricación de esos objetos.
- Los objetos se crean por la instanciación de una clase.

```python
class Persona:
    """Una clase persona simple"""
    def __init__(self):
        self.nombre = ""

    def set_nombre(self, nombre):
        self.nombre = nombre

    def get_nombre(self):
        return self.nombre


persona = Persona()
persona.set_nombre("Pedro")

print(persona.get_nombre())
```

CLASE

- Una clase tiene:
  - **Atributos** (cosas conocidas del objeto)
    - Ej: radio, centro, color, etc.
  - **Métodos** (cosas que puede hacer el objeto)
    - Ej: mover, cambiar color, etc.

¿CÓMO CONSTRUIR UNA CLASE EN PYTHON?

- Veamos cómo se vería nuestra clase circulo en Python.
- Acá nos vamos a fijar en lo siguiente:
  - **__init__**: corresponde al constructor de la clase.
    - Un constructor define la forma en que se crean los objetos.
    - En Python siempre se llama __init__.
  - **Métodos**
    - Draw y move son dos métodos de la clase circulo.
  - **Atributos**
    - Los atributos de la clase serían center y radio, los cuales están definidos en el constructor (líneas 14 y 15).
    - En este caso ambos atributos son privados a la clase (están encapsulados en el comportamiento de la clase).
      - El concepto de encapsulación lo veremos más adelante.

```
1   """ Clase circulo """
2
3
4   class Circulo:
5
6       """
7           Constructor de la clase
8
9           Este es llamado cuando alguien crea un nuevo círculo.
10          Acá se le asignan parámetros por defecto en el proceso
11          de creación del objeto
12      """
13      def __init__(self, center, radio):
14          self.__center = center
15          self.__radio = radio
16
17      """ Método que usamos para dibujar un círculo """
18      def draw(self, canvas):
19          rad = self.__radio
20          (x1, y1) = (self.__center[0] - rad, self.__center[1] - rad)
21          (x2, y2) = (self.__center[0] + rad, self.__center[1] + rad)
22          canvas.create_oval(x1, y1, x2, y2, fill='green')
23
24      """ Método que usamos para mover un circulo """
25      def move(self, x, y):
26          self.__center = [x, y]
27
```

ENCAPSULACIÓN

- Es una característica fundamental que heredan los objetos.
- Características que provee la encapsulación al modelo orientado a objetos:
    - Se protegen los datos de accesos indebidos o indeseados.
    - El acoplamiento entre las clases disminuye.
        - Acoplamiento tiene relación con la dependencia funcional entre dos clases distintas.
    - Permite aumentar la cohesión, debido a que el acoplamiento disminuye.
        - La cohesión implica que una clase u objeto derivado, se ocupa de una funcionalidad específica.
    - Favorece la modularidad y el mantenimiento.

¿CÓMO VEMOS LA ENCAPSULACIÓN EN PYTHON?

- Python posee encapsulación de los miembros de una clase, sin embargo esta no es tan estricta como otros lenguajes (como Java o C++).

- Para definir un atributo privado, solo basta con anteponer un __ (doble *underscore*) al nombre del método o nombre del atributo y ya tenemos miembro de la clase como *private*.

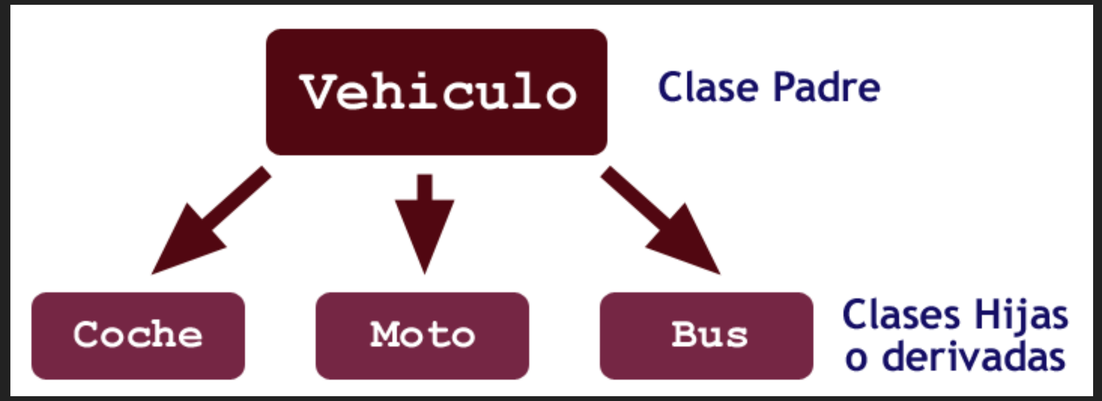
- Para definir un elemento *protected* basta con anteponer un _ (simple *underscore*) al nombre del método o nombre del atributo y ya tenemos el miembro de la clase como *protected*.

- Si no tienen este doble *underscore* se asume que el atributo o método es público.

```python
""" Clase circulo """


class Circulo:

    """
        Constructor de la clase

        Este es llamado cuando alguien crea un nuevo círculo.
        Acá se le asignan parámetros por defecto en el proceso
        de creación del objeto
    """
    def __init__(self, center, radio):
        self.__center = center
        self.__radio = radio

    """ Método que usamos para dibujar un círculo """
    def draw(self, canvas):
        rad = self.__radio
        (x1, y1) = (self.__center[0] - rad, self.__center[1] - rad)
        (x2, y2) = (self.__center[0] + rad, self.__center[1] + rad)
        canvas.create_oval(x1, y1, x2, y2, fill='green')

    """ Método que usamos para mover un circulo """
    def move(self, x, y):
        self.__center = [x, y]
        self.__process(10)

    def __process(self, rad):
        self.__center = [rad, rad]
```

HERENCIA

- A través de este mecanismo podemos crear nuevas clases partiendo de una existente.
    - Se dice entonces, que la nueva clase hereda las características de la clase original, aunque puede añadir más capacidades o modificar las que tiene.
- Podríamos entender cómo que todo el código de la clase *Padre* esta ya escrito dentro de la clase *Hija*.
    - A través de la herencia siempre consigues una especialización de la clase desde donde se hereda.



HERENCIA

- Dos subclases distintas, que derivan de una clase *padre* común, pueden heredar los métodos tal y como estos han sido definidos en esta, o pueden modificarlos todos - o algunos - para adaptarlos a sus necesidades.
    - Ej.: Si tenemos una clase mamífero, de la cual hereda la clase león y la clase humano, tenemos que el método *alimentarse()* - contenida en mamífero - para el león se implementa como *carnívora* vs la clase humano que la implementa como *omnívora.*
    - Ej.: Si enseñamos a la clase padre a imprimirse, todas las clases hijas sabrán automáticamente este método, sin la necesidad de escribir código en ellas.
- A la clase heredada se le llama **subclase o clase hija**, y a la clase desde la que hereda **superclase o clase padre**.
    - Se heredan los atributos y métodos, por lo tanto ambos pueden ser redefinidos en las clases hijas, aunque lo más común es redefinir los métodos y no los atributos.
- Al heredar, la clase heredada toma directamente el comportamiento de su **superclase**, pero puesto a que ésta puede derivar de otra, y esta de otra, etc., una clase toma indirectamente el comportamiento de todas las clases de la rama del árbol de la jerarquía de clases a la que pertenece.
    - Ej.: Una clase humano, hereda todo el comportamiento de la clase primates, y esta a su vez hereda todo el comportamiento de la clase mamíferos, y esta a su vez todo el comportamiento de la clase vertebrados. Así la clase humano puede implementar comportamientos de la clase vertebrados, dado a que se encuentra dentro de su árbol de jerarquía.

HERENCIA

- Solo heredamos aquellos miembros de la clase que tengan visibilidad **public** o **protected**. Los miembros de la clase con visibilidad **private** no son heredados a sus hijas.
- Cuando tenemos un comportamiento en una clase hija que queremos sustituir, no existe mayor dificultad. Simplemente escribimos un método con el mismo nombre de la superclase y lo implementamos según la necesidad de la clase hija.
- Si queremos ampliar el funcionamiento del método existente en la clase padre, entonces lo primero es que tiene que ejecutarse el método de la clase padre y después el de la clase hija.
    - Acá se hace necesario establecer alguna forma de distinguir cuando nos referimos a uno u otro método, dado a que ambos **deben** tener el mismo nombre.
        - Para eso hacemos referencia a dos palabras reservadas para ello: **self** (en algunos lenguajes usan *this*) y **super**.

¿CONSULTAS?