Facultad de Ciencias
UNER de la Administración

# TRABAJO PRÁCTICO Nº 3 – ASOCIACIÓN Y DEPENDENCIA

**Unidades 4 y 5 – Asociación y dependencia / Encapsulamiento y abstracción**

**PROGRAMACIÓN 2 - 2024 – 2do cuatrimestre**

**TECNICATURA UNIVERSITARIA EN DESARROLLO WEB**

## EL TRABAJO PRÁCTICO Nº 3 TIENE POR OBJETIVO QUE EL ALUMNO

- Aplique y refuerce los conceptos referidos a Asociación y Dependencia de Clases y Objetos, como así también los mecanismos de Encapsulamiento y Abstracción.
- Sea capaz de interpretar y traducir correctamente los diagramas de Clases en código Python.
- Logre un total entendimiento de la creación y manipulación de los objetos en memoria.

## CONDICIONES DE ENTREGA

- El Trabajo Práctico deberá ser:
    - Realizado en forma **grupal, en equipos de entre 3 y 5 alumnos**.
    - Cargado en la sección del Campus Virtual correspondiente, en un archivo 7z, ZIP o RAR (o cualquier otro tipo de comprimido) con las soluciones a cada ejercicio. Cada solución debe estar contenida en un archivo .py distinto.
    - Deberá indicarse el apellido y nombre de los integrantes del grupo. Todos los integrantes del grupo deben realizar la entrega en el campus y deberá agregarse al comprimido con las soluciones un archivo *integrantes.txt* con la información de los participantes.
    - Entregado antes de la fecha límite informada en el campus.
- El Trabajo Práctico será calificado como Aprobado o Desaprobado.
- Las soluciones del alumno/grupo deben ser de autoría propia. De encontrarse soluciones idénticas entre diferentes grupos, dichos trabajos prácticos serán clasificados como **DESAPROBADO**, lo cual será comunicado en la devolución.

## PROBLEMA

En el desarrollo del Trabajo Práctico Nro. 2 se intentó modelar algunas entidades de la pizzería de Don Pipo (Maestro pizzero, Mozo, Pizza). El camino a emprender para este entregable comprende la expansión de este modelo, modificando las Clases ya existentes y agregando otras nuevas.

La pizzería de Don Pipo cuenta con tres empleados, un maestro pizzero y dos mozos que se encargan de entregar los pedidos a los clientes. Al recibirse un pedido el mozo carga una orden, la cual posee un estado inicial y este va cambiando a medida que el pedido del cliente es procesado. El maestro pizzero puede cocinar todas las pizzas que se necesiten al mismo tiempo, pues las masas ya están estiradas desde temprano, los ingredientes listos en las heladeras y las instalaciones de la cocina así lo permiten. Por su parte, cada mozo puede cargar hasta dos pizzas al mismo tiempo. Cada pizza es de una variedad específica, es decir, no pueden prepararse en mitades y, a su vez, cada variedad cuenta con un costo determinado. Finalmente, cuando todas las pizzas de una orden son entregadas a los mozos, esta se mueve a un estado final, concluyendo el circuito del pedido.

**EJERCICIOS:**

Como punto de inicio, se requiere tomar del Trabajo Práctico Nro. 2 las clases implementadas para los siguientes diagramas:

| MaestroPizzero |
| --- |
| |
| <<Atributos de clase>><br><<Atributos de instancia>><br>nombre: string<br>pizzasPorCocinar: Pizza[]<br>pizzasPorEntregar: Pizza[] |
| <<Constructores>><br>MaestroPizzero(nom: string)<br><<Comandos>><br>establecerNombre(nom: string)<br>tomarPedido(var: string): Pizza<br>*Requiere var no vacío*<br>cocinar()<br>entregar(pizzas: int): Pizza[]<br>*Requiere pizzas <= 2*<br><<Consultas>><br>obtenerNombre(): string<br>obtenerPizzasPorCocinar(): Pizza[]<br>obtenerPizzasPorEntregar(): Pizza[] |

| Pizza |
| --- |
| <<Atributos de clase>><br><<Atributos de instancia>><br>variedad: string |
| <<Constructores>><br>Pizza(var: string)<br><<Comandos>><br>establecerVariedad(var: string)<br><<Consultas>><br>obtenerVariedad(): string |

| Mozo |
| --- |
| <<Atributos de clase>><br><<Atributos de instancia>><br>nombre: string<br>pizzas: Pizza[] |
| <<Constructores>><br>Mozo(nom: string)<br><<Comandos>><br>establecerNombre(nom: string)<br>tomarPizzas(pizzas: Pizza[])<br>*Requiere pizzas ligado*<br>servirPizzas()<br><<Consultas>><br>obtenerNombre(): string<br>obtenerPizzas(): Pizza[]<br>obtenerEstadoLibre(): boolean |


maestropizzero.py


mozo.py


pizza.py

Habiendo tomado las clases presentadas en la página anterior, se requiere que el alumno modifique las clases dadas, e implemente las nuevas, de acuerdo a los diagramas mostrados debajo, cumpliendo con cada uno de los requisitos expuestos en los puntos.

1. Implemente la clase **Orden** de acuerdo al siguiente diagrama:

| **Orden** |
|---|
| **<<Atributos de clase>>**<br>ESTADO_INICIAL = 1<br>ESTADO_PARA_ENTREGAR = 2<br>ESTADO_ENTREGADA = 3<br>**<<Atributos de instancia>>**<br>nroOrden: int<br>pizzas: Pizza[]<br>estadoOrden: int |
| **<<Constructores>>**<br>Orden(nro: int, pizzas: Pizza[])<br>*Requiere pizzas ligado* |
| **<<Comandos>>**<br>establecerNroOrden(nro: int)<br>establecerPizzas(pizzas: Pizza[])<br>*Requiere pizzas ligado*<br>establecerEstadoOrden(est: int)<br>*Requiere est = Orden.ESTADO_INICIAL o Orden.ESTADO_PARA_ENTREGAR o Orden.ESTADO_ENTREGADA*<br>**<<Consultas>>**<br>obtenerNroOrden(): int<br>obtenerPizzas(): Pizza[]<br>obtenerEstadoOrden(): int<br>calcularTotal(): float |

   a. Los atributos de clase *ESTADO_INICIAL*, *ESTADO_PARA_ENTREGAR* y *ESTADO_ENTREGADA* son constantes que identifican el estado en el que una orden se encuentra. Una orden en el estado *ESTADO_INICIAL* puede moverse hacia el estado *ESTADO_PARA_ENTREGAR*, y una en este último estado, a su vez, puede cambiar al estado *ESTADO_ENTREGADA*. Esta transición de estados no puede suceder de otro modo o en distinta dirección.
   b. Inicialmente, el atributo de instancia **estadoOrden** toma el valor *ESTADO_INICIAL*.

c. La consulta **calcularTotal** debe calcular el costo total de la orden. Esto debe hacerse recorriendo los objetos de tipo Pizza ligados a dicha orden y acumulando el costo que tiene cada variedad de pizza.

2. Codifique la clase **PizzaVariedad** siguiendo el siguiente diagrama:

| **PizzaVariedad** |
|---|
| **<<Atributos de clase>>**<br>**<<Atributos de instancia>>**<br>nombreVariedad: string<br>precio: float |
| **<<Constructores>>**<br>PizzaVariedad(nomVar: string, pre: float)<br>*Requiere pre > 0.0* |
| **<<Comandos>>**<br>establecerNombreVariedad(nomVar: string)<br>establecerPrecio(pre: float)<br>*Requiere pre > 0.0*<br>**<<Consultas>>**<br>obtenerNombreVariedad(): string<br>obtenerPrecio(): float |

3. Modifique la clase **Pizza** para que se ajuste al nuevo diagrama:

| **Pizza** |
|---|
| **<<Atributos de clase>>**<br>ESTADO_POR_COCINAR = 1<br>ESTADO_COCINADA = 2<br>ESTADO_ENTREGADA = 3<br>**<<Atributos de instancia>>**<br>variedad: PizzaVariedad<br>estado: int |
| **<<Constructores>>**<br>Pizza(var: PizzaVariedad)<br>*Requiere var ligado* |
| **<<Comandos>>**<br>establecerVariedad(var: PizzaVariedad)<br>*Requiere var ligado*<br>establecerEstado(est: int)<br>**<<Consultas>>**<br>obtenerVariedad(): PizzaVariedad<br>obtenerEstado(): int |

a. Los atributos de clase *ESTADO_POR_COCINAR*, *ESTADO_COCINADA* y *ESTADO_ENTREGADA* son constantes que identifican el estado en el que una pizza se encuentra. Una pizza en el estado *ESTADO_POR*_COCINAR puede moverse hacia el estado *ESTADO_COCINADA*, y una en este último estado, a su vez, puede cambiar al estado *ESTADO_ENTREGADA*. Esta transición de estados no puede suceder de otro modo o en distinta dirección.
b. El atributo de instancia estado toma el valor inicial ESTADO_POR_COCINAR.

4. Modifique la clase **MaestroPizzero** para que se ajuste al siguiente diagrama:

| **MaestroPizzero** |
|---|
| **<<Atributos de clase>>**<br>**<<Atributos de instancia>>**<br>nombre: string<br>ordenes: Orden[] |
| **<<Constructores>>**<br>MaestroPizzero(nom: string) |
| **<<Comandos>>**<br>establecerNombre(nom: string)<br>tomarPedido(orden: Orden)<br>*Requiere orden ligado y orden.estado = Orden.ESTADO_INICIAL*<br>cocinar()<br>entregar(orden: Orden): Pizza[]<br>*Requiere orden ligado y orden.estado = Orden.ESTADO_PARA_ENTREGAR* |
| **<<Consultas>>**<br>obtenerNombre(): string<br>obtenerOrdenes(): Orden[] |

a. El atributo de instancia **ordenes** reemplaza a los atributos de instancia **pizzasPorCocinar** y **pizzasPorEntregar**. Los objetos de la clase Orden mantendrán un estado, y a través de este se diferenciará cuales están listas para ser entregadas y cuales para ser procesadas.
b. El atributo de instancia **ordenes** se inicializa como una lista vacía.
c. El comando ***tomarPedido*** toma una orden y la agrega al final de la lista ligada al atributo **ordenes**.
d. El comando ***cocinar***, por su parte, debe:
    i. Tomar todas las órdenes de la lista ligada al atributo **ordenes** cuyo estado sea igual a *Orden.ESTADO_INICIAL*.

ii. Modificar el estado de dichos objetos a *Orden.ESTADO_PARA_ENTREGAR*.
iii. Recorrer los objetos de la clase Pizza ligados a cada orden, cambiando el estado de ellos a *Pizza.ESTADO_COCINADA*.

e. Finalmente, el comando ***entregar*** tomará un objeto de la clase Orden como parámetro. De dicha orden, se tomará un máximo de dos objetos de la clase Pizza para ser retornados al final del método, cambiando el estado de cada una de ellos a *Pizza.PIZZA_ENTREGADA*.

f. Antes de retornar los elementos en cuestión, se debe verificar si todos los objetos tipo Pizza ligados a la orden cuentan con el estado *Pizza.PIZZA_ENTREGADA*. De ser así, el estado de la orden debe modificarse a Orden.ESTADO_ENTREGADA.

5. Actualice el código de la clase Mozo siguiendo el siguiente diagrama:

| **Mozo** |
|---|
| **<<Atributos de clase>>**<br>**<<Atributos de instancia>>**<br>nombre: string<br>pizzas: Pizza[] |
| **<<Constructores>>**<br>MaestroPizzero(nom: string) |
| **<<Comandos>>**<br>establecerNombre(nom: string)<br>tomarPizzas(pizzas: Pizza[])<br>*Requiere pizzas ligado*<br>servirPIzzas() |
| **<<Consultas>>**<br>obtenerNombre(): string<br>obtenerPizzas(): Pizza[]<br>obtenerEstadoLibre(): boolean |

a. Únicamente se modifica la consulta **obtenerEstadoLibre** para que este retorne el valor True en caso de que el tamaño de la lista ligada al atributo pizzas sea igual a 0.

6. Indique si las siguientes sentencias son Verdaderas o Falsas:
    a. La clase Mozo y la clase MaestroPizzero están asociadas.
    b. Existe una relación de dependencia entre las clases Mozo y MaestroPizzero.
    c. La clase Orden es proveedora de la clase MaestroPizzero.
    d. La clase MaestroPizzero es proveedora de la clase Pizza.
    e. La clase MaestroPizzero es cliente de la clase Pizza.
    f. La clase Pizza y PizzaVariedad están asociadas.
    g. Existe una relación de dependencia entre las clases PizzaVariedad y Pizza.
    h. La clase Mozo y Pizza están asociadas.
    i. Existe una relación de dependencia entre las clases Orden y PizzaVariedad.

7. Construya un programa utilizando las clases presentadas en los puntos 1, 2, 3, 4 y 5 que permita:
    a. Crear objetos de tipo PizzaVariedad, Pizza, Orden, MaestroPizzero y Mozo.
    b. Enviar los mensajes *tomarPedido*, *cocinar* y *entregar* al objeto de la clase MaestroPizzero.
    c. Enviar los mensajes *tomarPizzas* y *servirPizzas* a los objetos de la clase Mozo creados en el punto a.
    d. Mostrar la transición de estados de los objetos de las clases Orden y Pizza.
    e. Calcular y mostrar el costo total de las órdenes creadas.

Para la construcción de dicho programa crear una clase de nombre TesterPizzeria que actúe como cliente de las clases presentadas anteriormente, cuyo único servicio sea de nombre ***main***, que ejecute los puntos descriptos anteriormente. A continuación, un ejemplo de cómo dicho programa puede ser construido:

```
class TesterPizzeria:
      def main(self):
            # Solución de los puntos 7.a., 7.b, 7.c, ...

if __name__ == '__main__':
      testerPizzeria = TesterPizzeria()
      testerPizzeria.main()
```