Facultad de Ciencias
UNER de la Administración

TRABAJO PRÁCTICO Nº 2 – CLASES Y OBJETOS

Unidades 3 y 4 – Objetos y Clases

PROGRAMACIÓN 2 - 2024 – 2do cuatrimestre

TECNICATURA UNIVERSITARIA EN DESARROLLO WEB

EL TRABAJO PRÁCTICO Nº 2 TIENE POR OBJETIVO QUE EL ALUMNO

- Aplique y refuerce los conceptos fundamentales y relacionados a la Programación Orientada a Objetos.
- Sea capaz de interpretar y traducir correctamente los diagramas de Clases en código Python.

CONDICIONES DE ENTREGA

- El Trabajo Práctico deberá ser:
    - Realizado en forma **grupal, en equipos de entre 3 (tres) y 5 (cinco) 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

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. Cada mozo puede cargar hasta dos pizzas al mismo tiempo. 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. Este último no necesita descansar, aunque solo puede tomar un único pedido a la vez.

EJERCICIOS:

1. Dado el siguiente diagrama de clases:

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

Genere la clase MaestroPizzero, conteniendo los atributos y servicios mencionados en el diagrama de clases anterior.

i. El comando *tomarPedido* debe crear un nuevo objeto de la clase Pizza, de la variedad indicada en el parámetro formal ***var***. Una vez inicializado dicho objeto, debe este agregarse a la lista referenciada por el atributo ***pizzasPorCocinar***.

ii. El comando *cocinar* debe tomar todos los objetos de la clase Pizza de la lista ***pizzasPorCocinar*** y depositarlos en una segunda lista, ***pizzasPorEntregar***. Si no hay pizzas por ser cocinadas, el comando no tiene efecto sobre el estado interno del objeto.

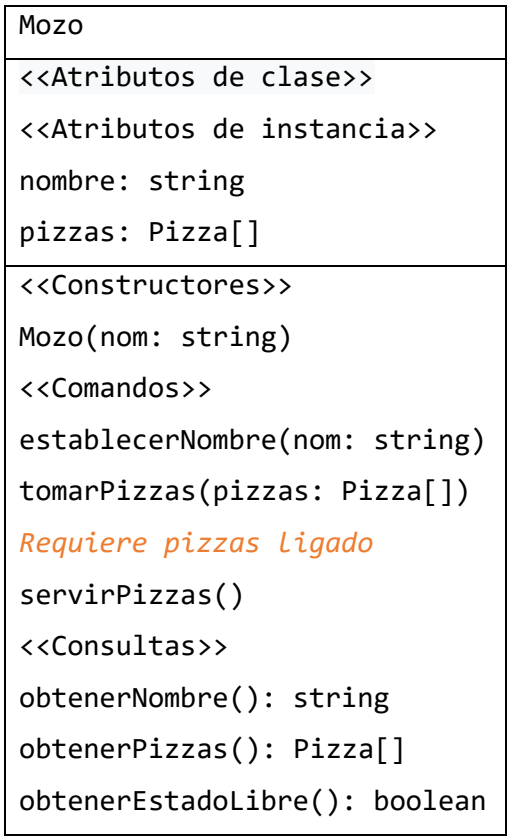
iii. El comando *entregar* retorna hasta un máximo de 2 objetos de la clase Pizza de la lista **pizzasPorEntregar**, removiéndolos de ella. Si no hay pizzas para ser entregadas, se debe retornar una lista vacía.

2. Dado el siguiente diagrama de clases:

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

Genere la clase Pizza con los atributos y servicios mencionados en dicho diagrama.

3. Se agrega el siguiente diagrama, el cual representa a los mozos de la pizzería a modelar:



Habiendo analizado el diagrama, genere la clase Mozo con los atributos y servicios mencionados en dicho diagrama.

a. El atributo ***pizzas*** se inicializa como una lista vacía.
b. El comando *tomarPizzas* agrega los objetos de la clase Pizza referenciados por el parámetro formal ***pizzas***. El mozo puede tomar hasta un máximo de 2 pizzas.
c. *servirPizzas* limpia la lista **pizzas**, haciendo entrega de los pedidos a los clientes.
d. *obtenerEstadoLibre* debe retornar True si es que la lista referenciada por el atributo ***pizzas*** tiene una longitud de entre 0 y 1. Así mismo, debe retornar False si su tamaño es igual a 2.

4. Una vez codificadas en Python las Clases de los puntos anteriores, instancie los objetos tal como sucede en las siguientes instrucciones:

```
mozo1 = Mozo('Alfredo')
mozo2 = Mozo('Alfredo')
```

Luego, responda lo siguiente:

i. ¿Los identificadores *mozo1* y *mozo2* hacen referencia al mismo objeto?
ii. ¿Son objetos equivalentes? Explique que significa que dos objetos lo sean.
iii. ¿Los objetos ligados a *mozo1* y *mozo2* comparten la misma posición de memoria?

5. Construya un programa utilizando las clases proveedoras presentadas en los puntos 1, 2 y 3 que permita:
    a. Crear objetos de tipo MaestroPizzero, Mozo y Pizza.
        i. Para una exacta representación de nuestra pizzería, debe limitarse a un único objeto de tipo MaestroPizzero y dos objetos de la clase 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.

Para la construcción de dicho programa crear una clase de nombre TesterPizzeria que actúe como cliente de las clases proveedoras MaestroPizzero, Mozo y Pizza, 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 5.a., 5.b, 5.c, ...

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