Hito individual 1er trimestre

Allam E. Miranda Carrasco

Fase 1:

Definimos un algoritmo como el conjunto de instrucciones definidas para llegar a un producto o programa final, en este caso comenzaremos a detallar los distintos pasos para llegar al programa final de gestión de pedidos.

El algoritmo principal del programa se basará en la interfaz principal, las clases hijas y padres, sobreescritura, switch case y uso de módulos.

La aplicación comenzará con la pantalla de gestión de pedidos que presenta al usuario, pedimos sus datos y comenzamos el programa. En función de la dirección del cliente llevaremos una facturación u otra y podrán realizar una serie de operaciones.

El registro del cliente ya dicho que lo haremos con un constructor y con POO para llevar de alguna forma una especie de base de datos que se reprograma al reiniciar el programa.

El siguiente paso sería la selección de productos o la creación de una lista de deseos ya sea cualquiera de las dos las haremos con listas distintas dependiendo de la opción que elija el usuario.

Por último la compra y pago de los productos realizados se hará mediante un print que solamente nos devolverá un mensaje, el seguimiento del envío lo mismo.

El mayor problema que vamos a tener al realizar el programa es entender cómo vamos a llevar esa lista de clientes y sus productos.

Ya definido en funcionamiento de nuestra aplicación podemos comenzar a realizar el “ Caso de uso “ y el “ Diagrama de flujo “ de nuestro programa.

Caso de uso:

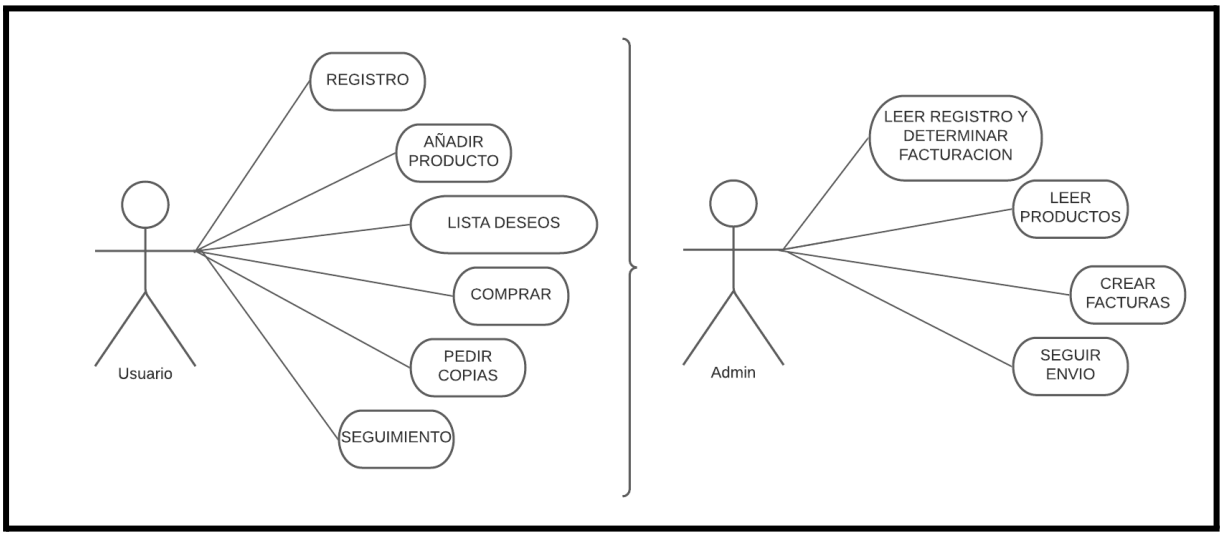
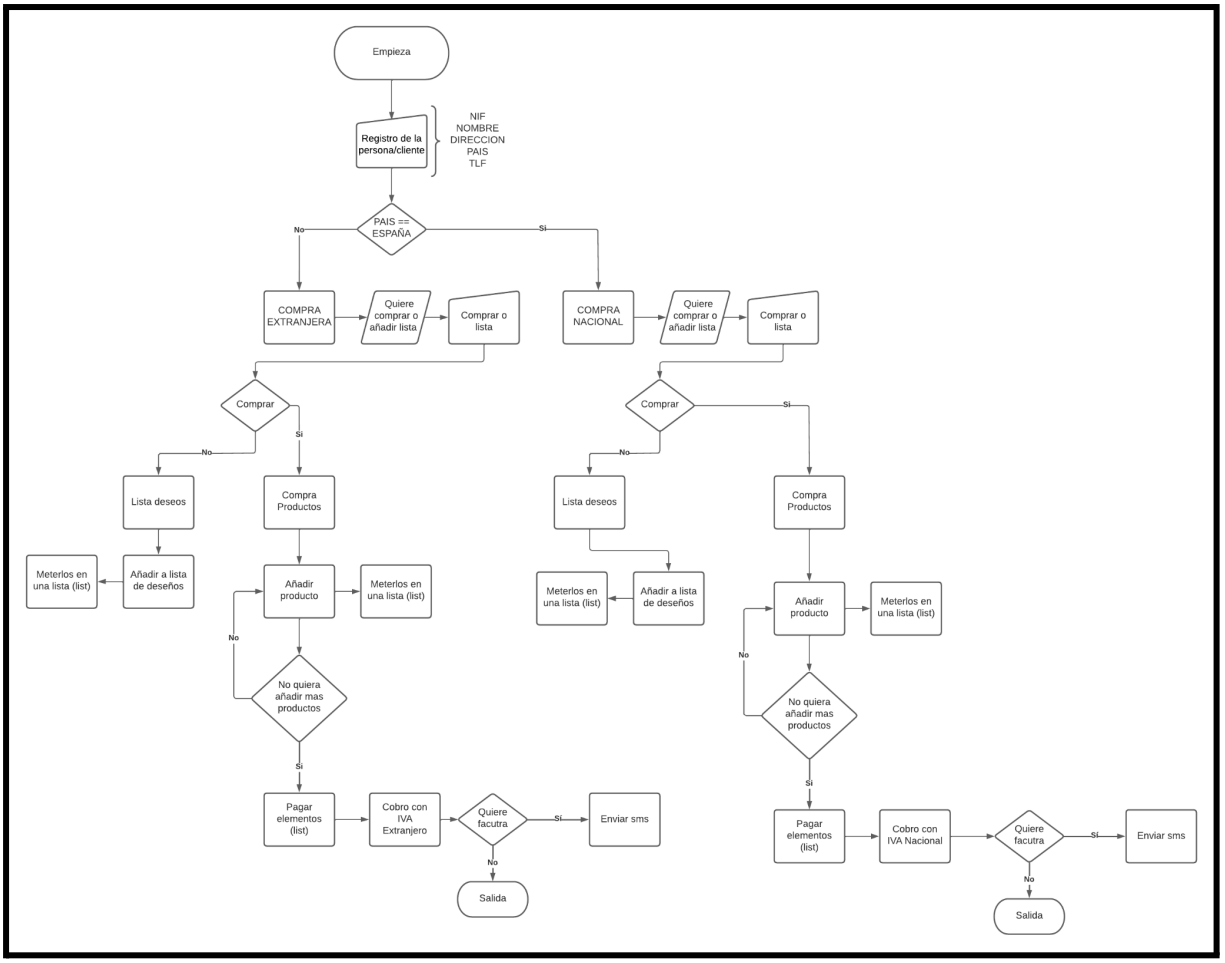


Diagrama de flujo:



Fase 2:

Adjunto documento zip con el proyecto final y sus archivos .py

Fase 3:

El algoritmo principal ha sido realizado mediante una interfaz y con POO.

He intentado crear la aplicación de tal forma que pueda ser fácilmente sustituible o que se pueda añadir código fácilmente, mediante la interfaz lo que queremos es que puedas acceder de distintas formas a la aplicación ya sea en este caso como Admin, Trabajador o Cliente.

El uso completo de la aplicación mediante switch case es debido a que es más fácil de utilizar que los if y contienen dentro suya un bucle propio para cualquier error, el código entero usa switch case.

```
productos = {'3060':399,'3070':499,'3080':699,'i5':200,'i7':250,'monitor':150,'raton':50,'teclado':70,'cascos':20,'altavoces':30,
'portatil':999,'rx580':450,'rx6670':890}
lista_deseosa={}
carrito_compra={}
```

Para las listas de la compra y lista de deseos he usado diccionarios, tenemos la tupla, sets y listas pero el diccionario es el único que nos permite asignar un valor a cada producto y poder sacarlo y usarlo en cualquier momento de manera eficaz.

Algunas de las partes del programa se pueden hacer con programación funcional como la función filter para usar funciones prescritas con ciertas variables. No he usado programación funcional porque prefería la declarativa en este caso.

```
def muestradatos(self): #sobreescritura
```

Muchos apartados de la aplicación están hechos con sobreescritura ya que al ser una interfaz es necesaria al ser las funciones distintas para cada clase.

```python
def inicio(): #creamos una funcion inicial para determinar a que clase pertenece la persona que usa la aplicacion
    modulos_textos.inicioopc()
    j=input('INTRODUZCA EL NUMERO DE LA ACCION A REALIZAR: ')
    print('-------------------------------------')

    match j:
        case '1':
            persona=Cliente()
            persona.pantalla_principal()
        case '2':
            persona=Trabajador()
            print('AUN NO HAY FUNCIONES PARA USTED')
            print('-------------------------------------')
        case '3':
            persona=Admin()
            print('AUN NO HAY FUNCIONES PARA USTED')
            print('-------------------------------------')
        case _:
            print('INTRODUZCA UN NUMERO VALIDO')
            print('-------------------------------------')
            inicio()
```

El registro se hace completamente dentro de la clase cliente en su propio constructor, es una especie de memoria a corto plazo que nos sirve a lo largo del programa para guardar todas las acciones del usuario, en caso de querer guardar listas, diccionarios y otro tipo de variables tras cerrar el programa habría que crear distintos métodos de guardado de archivo que no he usado para no complicarme.

Conceptos basicos de POO usados:

Interfaz: mediante la herencia podemos crear una interfaz usable para distintas clases hijas, estas mismas instancian los objetos automáticamente dentro de su propio constructor.

Sobreescritura: las funciones determinadas en la interfaz son como un botón de un mando en caso de saber dónde a qué clase nos referimos la función cambiará a otra distinta y se reemplazará por la que queremos.

```python
class Persona():
        def __init__(self):
            pass

        def muestradatos(self):
            pass

        def pantalla_principal(self):
            pass
```

```
def muestradatos(self): #sobreescri

    #esto es una sobreescritura la
    distintos

    print(f'BUENAS {self.nombre.upp
    {self.tlf}')
    print('----------------------
    self.pantalla_principal()

def pantalla_principal(self): #la p
    modulos_textos.texto_principal(
    final=input('INTRODUZCA EL NUME
    print('----------------------
```

La misma función en la clase cliente es distinta a la de la Interfaz.

Comentario sobre el código

Me gustaría analizar mi código de arriba a abajo comentando las partes más importantes y el por que de cada una.

```
import modulos_textos

def inicio(): #creamos una funcion inicial para determinar a que clase
pertenece la persona que usa la aplicacion
    modulos_textos.inicioopc()
    j=input('INTRODUZCA EL NUMERO DE LA ACCION A REALIZAR: ')
    print('-----------------------------------')

    match j:
        case '1':
            persona=Cliente()
            persona.pantalla_principal()
        case '2':
            persona=Trabajador()
            print('AUN NO HAY FUNCIONES PARA USTED')
            print('-----------------------------------')
        case '3':
            persona=Admin()
```

```
            print('AUN NO HAY FUNCIONES PARA USTED')
            print('-------------------------------------')
        case _:
            print('INTRODUZCA UN NUMERO VALIDO')
            print('-------------------------------------')
            inicio()
```

El inicio del trabajo es una función simple lo he decidido hacer así para no instanciar uno por uno y dar a elegir al usuario la opción de cómo quiere entrar al programa.

Importamos un módulo propio donde tenemos las funciones de texto y mediante un switch case elegimos en qué clase instanciar al usuario.

```
#Los productos los añadirmos a un dict para poder asignarles un valor
todo ello dentro de una lista
productos =
{'3060':399,'3070':499,'3080':699,'i5':200,'i7':250,'monitor':150,'rato
n':50,'teclado':70,'cascos':20,'altavoces':30,'portatil':999,'rx580':45
0,'rx6670':890}
lista_deseosa={}
carrito_compra={}
```

Aquí he creado el diccionario de los productos y de la lista y carrito antes no veía necesario crear una clase Productos cuando puedes almacenarlos directamente desde un principio fuera de cualquier clase.

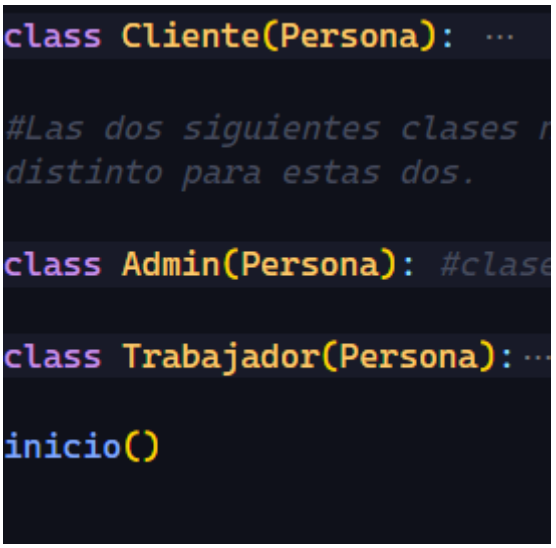
```
#La interface empieza con persona donde tendremos unas funciones base
que cambiaran segun nuestras clases hijas derivadas mediante
soobreescritura
class Persona():
        def __init__(self):
            pass

        def muestradatos(self):
            pass

        def pantalla_principal(self):
            pass
```

He usado para todo el proyecto una interfaz principal llamada persona, esto gracias a la web de https://ellibrodepython.com/abstract-base-class donde explican bien el

tema de las interfaces. La interfaz nos va a permitir hacer una Herencia más simple y unas sobreescrituras más fáciles al estar todo relacionado.



Dentro de la propia interfaz tenemos las tres clases importantes para este hito hemos utilizado solamente la clase Cliente las demás hacen el registro pero no tienen ninguna opción, en caso de que el programa necesite más código se puede añadir.

Ahora vamos a comentar algunos puntos importantes del código.

```
class Persona():
        def __init__(self):
            pass

        def muestradatos(self):
            pass

        def pantalla_principal(self):
            pass
```

Las funciones de la interfaz va a haber que sobre escribirlas en cada clase ya que habrá una muestra de datos distinta para el cliente como para el administrador o el trabajador.

```
class Cliente(Persona):
    def __init__(self):
        super().__init__()
        print('------REGISTRO AUTOMATICO------')
        print('INICIO AUTOMATICO PROCESO REGISTRO....')
```

```
        self.nombre=input('INTRODUZCA SU NOMBRE: ')
        self.direccion=input('INTRODUZCA DIRECCION: ')
        self.pais=input('INTRODUZCA PAIS: ')
        def emailpe():
            self.email=input('INTRODUZCA UN EMAIL VÁLIDO EL EMAIL: ')
            comprobacion="@" in self.email
            if comprobacion == True:
                pass
            else:
                emailpe()
        emailpe()
        def tlecom():
            try:
                self.tlf=int(input('INTRODUZCA UN TELÉFONO VÁLIDO: '))
            except:
                tlecom()
        tlecom()

        print('-------------------------------------')

    def muestradatos(self): #sobreescritura

        #esto es una sobreescritura la debemos hacer por que en caso de
que tengamos admin o trabajador los datos mostrados seran distintos

        print(f'BUENAS {self.nombre.upper()} SU DIRECCION ES:
{self.direccion.upper()} Y VIVE EN {self.pais.upper()} CON NUMERO DE
TLF {self.tlf}')
        print('-------------------------------------')
        self.pantalla_principal()
```

El cliente tiene de inicio el registro automático dentro del constructor esto creará una memoria propia para el mismo. Nos da sus datos en el email y en el teléfono en caso de no ser válidos volverá a pedirlos.

En cuanto a la muestra datos es la sobreescritura de nuestra anterior función.

```
    def pantalla_principal(self): #la pantalla principal es distinta
depediendo de la persona
        modulos_textos.texto_principal()
        final=input('INTRODUZCA EL NUMERO DE LA ACCION A REALIZAR: ')
        print('-------------------------------------')
```

```
        if self.pais.lower() == 'españa': #iva españa
            iva = 1.21
        elif self.pais.lower() == 'francia': #iva francia
            iva = 1.11
        else: #iva fijo resto de paises
            iva = 1.41

        match final:
            case '1':
                self.muestradatos()
            case '2':
                self.carrito_lista()
            case '3':
```

La pantalla principal también está sobreescrita es un switch case para las opciones de todo el programa.

Aquí coloco la variable iva para usarla en el pago.

Al ser el carrito y la lista de deseos iguales voy a explicar solo uno de los dos.

Dentro del carrito tenemos varias opciones como ver el carrito, añadir productos o ver la lista de productos y eliminar los que queramos. Código importante:

```
print('------LISTA PRODUCTOS------')
         for key, value in productos.items():
                 print(key, ' PRECIO: ', value,'€')
print('-------------------------------------')
```

Este es el código para mostrar los productos de un diccionario, mediante un for recorremos el diccionario y asignamos al nombre el valor key y al precio el value con solo un print editado nos mostrará de forma fácil la lista para el usuario.

```
try:

print('-------------------------------------')
     k=input('DIME EL PRODUCTO: ')

carrito_compra[k]=productos[k] #append del dict

print('-------------------------------------')
              print('1. AÑADIR OTRO PRODUCTO')
                  print('2. VOLVER')
```

```
o =input('INTRODUZCA EL NUMERO DE LA ACCION A REALIZAR: ')

print('-------------------------------------')
                    match o:
                                    case '1':
                                              productoc()
                                                     case '2':

agregar_carrito()
                except:

modulos_textos.except_carrilis()
     u = input('INTRODUZCA EL NUMERO DE LA ACCION A REALIZAR: ')

print('-------------------------------------')
                                                          match u:
                                                              case '1':
                                                                  productoc()
                                                              case '2':

agregar_carrito()
                                                              case _:
                                                                  productoc()
```

Para añadir un producto hemos hecho un try except en caso de el producto no existir nos lo dice y en caso de que si hace el append del diccionario.

```
del carrito_compra[eliminarp]
```

Para eliminar un producto con la función del se puede remover del diccionario.