IE-0217 Estructuras de Datos Abstractas y Algoritmos para Ingeniería

II-2022

Aplicaciones requeridas para C/C++ y manejo de reservorios

Tarea 04

Polimorfismo usando Python 3.0

En esta tarea se va a realizar un desarrollo utilizando herencia y polimorfismo según los criterios vistos en clase.

Se va a plantear un esquema de una sucursal de un banco que tiene cuentas de clientes y estas a su vez son clasificadas en categorías.

Agencia Bancaria

Una entidad bancaria requiere una aplicación basada en menús sobre la que puede administrar una lista de usuarios, los cuales están en tres categorías: **UsuariosAhorro**, **UsuarioTasaCero** y **UsuarioPagosServicios**

Planteamiento programático.

Se debe resolver mediante la definición de objetos en Python en el siguiente planteamiento.

Parte I

Clases requeridas:

UsuarioBase:
Se tiene una cuenta base que debe contener los siguientes campos:
Propietario: Nombre del propietario de la cuenta (string).
MontoDisponible: Es el dinero que se tiene en la cuenta (float).
Todos los campos anteriores deben disponer de setter y getters para accesar/modificar los datos. El *MontoDisponible* es mascarado con el decorador con el mismo nombre *MontoDisponible.*
AplicarInversion: Es un método polimórfico.
AsignarOperacion: Es un método polimórfico.
Aplicar: Es un método público que invoca al método *AplicarInversion*
Asignar: Es un método público que invoca al método *AsignarOperacion*

UsuarioAhorro:
Es una clase hija de **UsuarioBase**, además de los campos *Propietario* y *MontoDisponible*, tiene los siguientes elementos:

MontoAhorro: Es el monto que se tiene ahorrado.

Porcentaje: Es un valor entre 1 y 100 (porcentual) que se debe iniciar en por defecto en 1. Y este corresponde al porcentaje del MontoDisponible que debe ahorrar y se coloca en *MontoAhorro*

AplicarInversion: Es el método que corresponde con el declarado en **UsuarioBase** en forma polimórfico. Debe tomar el valor del *MontoDisponible y Porcentaje* para calcular el ahorro y colocarlo en *MontoAhorro* debitándolo *de MontoDisponible.*

Debido a que es polimórfico y para que la aplicación opere correctamente para hacer el ahorro, debe usarse el método *Aplicar* que está en la clase **UsuarioBase**

AsignarOperación: Es el método que corresponde con el declarado en **Usuario Base** en forma polimórfico. No hace ninguna operación en este tipo de Usuario por lo que debe declararse vacío.

Debido a que es polimórfico y para que la aplicación opere correctamente para hacer el ahorro, debe usarse el método *Asignar* que está en la clase **UsuarioBase**

El campo *Porcentaje* debe disponer de setter y getters para accesar/modificar el valor. El campo MontoAhorro solo puede tener getter, ya que el monto se hace calculado y no se puede cambiar directamente, solo mediante la fórmula contenida en el método *AplicarInversion*.

UsuarioTasaCero:
Es una clase hija de **UsuarioBase**, además de los campos *Propietario* y *MontoDisponible*, tiene los siguientes elementos:

MontoConsumoMes: El valor que debe pagar en la modalidad de tasa cero en cada mes.

Plazo (meses): Es el número de meses en que debe pagar.

OperacionVigente: Es un boolean que indica cuando se tiene vigente una operación de tasa cero.

AplicarInversion: Es el método que corresponde con el declarado en **UsuarioBase** en forma polimórfico. Si hay una *OperacionVigente,* entonces debe tomar el valor del *MontoDisponible* y si el *Plazo* es diferente de cero, debe entonces restarle el *MontoConsumeMes* y descontar una unidad del Plazo.

Por ejemplo, si el *MontoDisponible* es 10000 colones, el *MontoConsumoMes* es 2000 colones y el *Plazo* es 3. Entonces el llamado a este método debe dejar el *MontoDisponible* en 8000 colones y el *Plazo* en 2. Cuando el *Plazo* sea CERO debe ponerse *MontoConsumoMes* en cero y debe colocarse *OperacionVigente* en ***False***

Debido a que es polimórfico y para que la aplicación opere correctamente para hacer el uso de tasa cero, debe usarse el método *Aplicar* que está en la clase **UsuarioBase**

AsignarOperacion: Es el método privado que corresponde con el declarado en **Usuario Base** en forma polimórfico y es privado. Se usa para poner una *OperacionVigente* en forma activa (***True***). Usando el *MontoConsumoMes* y el *Plazo* de una nueva adquisición.

Si hay una *OperacionVigente* NO SE PUEDE ASIGNAR una nueva operación y debe indicarse la condición sin detener el programa.

Cuando el Plazo se asigna diferente de CERO y el MontoConsumoMes es diferente de CERO y no hay *OperacionVigente,* entonces se asigna *OperacionVigente* en ***True*** para que la tasa cero sea vigente.

Debido a que es polimórfico y para que la aplicación opere correctamente para hacer el uso de tasa cero, debe usarse el método *Asignar* que está en la clase **UsuarioBase**

El campo *MontoConsumoMes* y *Plazo* deben disponer de getter para accesar/modificar el valor. La asignación se hace solo cuando *OperacionVigente* está en ***False***. En otro caso se debe informar que hay una operación en proceso.

UsuarioPagosServicios:
Es una clase hija de **UsuarioBase**, además de los campos *Propietario* y *MontoDisponible*, tiene los siguientes elementos:

Servicios: Es una lista de servicios públicos (clase **ServicioPublico**), que un usuario coloca para que se aplique pago automático. La clase **ServicioPublico** se declara más adelante.

AplicarInversion: Es el método que corresponde con el declarado en **UsuarioBase** en forma polimórfico. Debe tomar cada elemento de la lista de *Servicios* y debitar de *MontoDisponible* el monto *CargoAutomatico* del servicio. Una vez debitado el monto se debe colocar *CargoAutomático* en CERO.

Debido a que es polimórfico y para que la aplicación opere correctamente para hacer el pago de servicios correctamente, debe usarse el método *Aplicar* que está en la clase **UsuarioBase**

AsignarOperacion: Es el método privado que corresponde con el declarado en **UsuarioBase** en forma polimórfico y es privado. Se usa para asignar un nuevo *CargoAutomático* a cada servicio contenido en la lista. Si existe una *OperacionVigente*

activa (*OperacionVigente es TRUE*) NO SE PUEDE ASIGNAR una nueva operación y debe indicarse la condición sin detener el programa

En caso contrario se de asignar a *OperacionVigente* un valor ***TRUE***

Debido a que es polimórfico y para que la aplicación opere correctamente para hacer el pago de servicios correctamente, debe usarse el método *Asignar* que está en la clase **UsuarioBase**

El campo *Servicios* deben disponer métodos para insertar servicios y visualizar el contenido de la lista.

ServicioPublico:
Es una clase que permite asociar un tipo de servicio público como: agua, electricidad, municipalidad, etc. Tiene los siguientes campos:

EmpresasServicioPublico: Nombre de la empresa de servicio público: AyA, CNFL, etc

CargoAutomatico: Monto que debe debitarse en forma automática.

OperacionVigente: Es un boolean que indica cuando se tiene vigente una operación de tasa cero, es controlada desde el método Aplicar del usuario de tipo **UsuariosPagosServicios**

Aspectos generales.
Debe asignar según el método polimórfico visto en clase los métodos __str__() y Captura la visualización e ingreso de datos.

En TODOS los casos que se debita un monto de *MontoDisponible,* debe verificarse previamente que los fondos alcancen. De otra manera no se aplicará el débito y se enviará un mensaje indicando dicha condición, sin detener el programa.

PARTE II

Procedimiento.

Se debe construir una clase de ***agenciabancaria*** y generar una instancia banco se usa para administrar una lista de usuarios y el despliegue de un menú de operaciones, según el código visto en clase, con el que se pueden realizar las siguientes operaciones:

Los usuarios son contenidos en una lista y esta se administra con estas opciones:

1) Agregar **UsuariosAhorro**: inserta a la lista un usuario destinado al ahorro, y el menú solicita los campos requeridos del usuario.
2) Agregar **UsuarioTasaCero**: inserta a la lista un usuario destinado al pago de tasa cero, y el menú solicita los campos requeridos del usuario.

3) Agregar **UsuariosPagosServicios**: inserta a la lista un usuario destinado al disponer de una sub-lista de servicios públicos registrables, y el menú solicita los campos requeridos del usuario y luego administra los servicios de este usuario.
4) Mostrar lista de usuarios: Muestra todos los datos de cada usuario de la lista.
5) Asignar Fondos: Despliega los usuarios vigentes y permite seleccionar uno al cual se le puede asignar (modificar) el *MontoDisponible.*
6) Asignar Operaciones. Esta opción debe recorrer la lista de usuarios e invocar el método *Asignar* de cada uno con el fin de poder ingresar nuevos datos en cada tipo de transacción asociada a cada tipo de usuario.
7) Procesar Inversiones: Esta opción debe recorrer la lista de usuarios e invocar el método *Aplicar* para hacer las transacciones de ahorro, reducción de tasa cero o pago de todos los servicios públicos según el tipo de usuario que esté asociado.
8) Final: Esta opción finaliza el programa.

La opción (3) una vez asignados los datos del usuario, debe mostrar las opciones de ingresar/mostrar/borrar servicios de este **UsuarioPagoServicios.**

Se sugiere para este caso este menú:
a) Ingresar servicio
b) Mostrar servicios
c) Eliminar servicio.
d) Regresar.

La opción (5) debe mostrar los usuarios y permitir seleccionar (opción desde teclado) el usuario al que se le va a asignar un nuevo valor de *MontoDisponible*
La opción (6) y la opción (7) no requieren otro menú, solo invocan el método correspondiente. El polimorfismo hace el resto.

Parte III

Una vez validada la parte II. Debe desarrollar una aplicación de carga automática desde archivos similar a la vista en clase.

Procedimiento de entrega.

Deben asignar esta tarea dentro del GIT de cada estudiante. La tarea es individual.
Deben mostrar los avances de cada parte. No puede entregarse un solo commit, ya que perderán la nota del uso del GIT.
Deben indicar con un Merge Request que la tarea está disponible para aceptarse.

Nota:

a) Uso correcto del GIT (al menos 4 commits) 30%.
b) Desarrollo del modelo de clases 40%.

c) Desarrollo de aplicación de caso de uso 20%;
d) Desarrollo de aplicación automática 10%

Fecha de entrega: domingo 20 de noviembre 2022 antes de las 12:00 media noche.