Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2019-2

22 de agosto de 2019
Actividad Formativa

Actividad 01

Programación Orientada a Objetos

Introducción

El Banco "Deudas, Compras y Créditos", o mejor conocido como Banco DCC, siempre se ha dedicado a entregar un excelente servicio a sus clientes más frecuentes, los alumnos de Programación Avanzada. Lamentablemente, hace poco sufrió un ataque por parte de los mafiosos del Tini Tamburini, escondiendo sus ganancias del lavado de tareas entre los registros de clientes. Tu misión es ayudar al PhD. Pinto, quien tiene un Doctorado en ciberseguridad espacial[1], a encontrar estos clientes fraudulentos y eliminar los dineros sucios del malvado Tini.

Rescatando al Banco DCC

Sabemos que estudiaste arduamente los contenidos durante estas últimas semanas, es por esto que tu misión como alumno de IIC2233 es regresar el estado del banco a como se encontraba justo antes del ataque. Para lo anterior, deberás utilizar todo lo aprendido de POO. A continuación, se describirán las entidades que conforman el sistema y cuáles son las tareas que debes realizar en el código para poder eliminar el sucio dinero de Tini Tamburini. **Para lograr esto deberás completar los archivos de extensión `.py` entregados**.

1. Banco DCC

1.1. Clases

El código que da vida al Banco DCC viene implementado en el archivo `entidades_banco.py`. En este archivo se encuentran las siguientes clases:

- `class Cliente`: Cada instancia tiene tres atributos que recibe como argumentos: `id_cliente` (único para cada usuario), `nombre` y `contrasena`. Además, al crear una instancia se crea un atributo adicional `saldo` que representa la cantidad de dinero del cliente (se asume que todos los clientes al ser creados parten con saldo igual a 0). Además tiene los métodos:
  - `def depositar(self, dinero)`: Suma el dinero al saldo que actualmente tiene el usuario.
  - `def retirar(self, dinero)`: Resta el dinero al saldo que actualmente tiene el usuario.

[1] Esto es real, búsquenlo despues de la actividad

- `class BancoDCC`: Es la clase que se encarga del funcionamiento del banco, posee un único atributo que es `clientes`, que corresponde a una lista de instancias de la clase `Cliente`. Además tiene los métodos:
  - `def cargar_clientes(self)`: Carga los clientes desde un archivo de texto.
  - `def buscar_cliente(self, id_cliente)`: Recibe un identificador de un cliente, busca en su registro a un cliente con dicho `id_cliente` y lo retorna, en caso de no encontrarlo muestra un mensaje por consola y retorna None.

Estas clases **ya vienen implementadas**, por lo que tú **NO** debes implementarlas ni modificar el código que se entrega en el archivo.

1.2. Base de datos

El banco cuenta con un registro de sus clientes, los siguientes archivos contienen información confidencial sobre el banco:

- `transacciones.txt`: Contiene la información de cada movimiento que ha realizado un cliente, tiene tres datos: `id` que es el identificador del cliente, `accion` que puede ser `'retira'` o `'deposita'` y el `monto` que ha sido retirado o depositado según corresponde. Cada línea de este archivo está en el formato `id, accion, monto`.
- `clientes.txt`: Contiene la información de cada cliente, su `id_cliente`, su `nombre`, su `contrasena` y su cantidad de dinero (`saldo`) **actual** (luego de realizar todas las transacciones) . Cada línea de este archivo está en el formato `id, nombre, saldo, contrasena`.

Tras el ataque, los mafiosos han logrado encriptar la base de datos, por lo que en la situación actual es imposible leer la base datos, esto les ha permitido realizar transacciones falsas y modificar a gusto la cantidad de dinero que tienen algunos clientes. Puedes encontrar estos archivos en la carpeta `bd_encriptada`.

2. Recuperando la base de datos

Antes de restaurar el orden en el Banco DCC, debes ser capaz de poder acceder a la información que está en la base de datos. Por suerte, el genio de la ciberseguridad PhD. Pinto ha logrado descubrir como poder recuperar los archivos. Sin embargo, ha dejado algo incompleto su trabajo, por lo que necesitas terminarlo. Para esto, en el archivo `phd_pinto.py` debes rellenar los siguientes *getters* y *setters* de dos *properties* en `DrPintoDesencriptador`:

- `def ruta(self)` : Es un *getter*, simplemente debe retornar el atributo `self._ruta` de la instancia.
- `def ruta(self, nueva_ruta)`: Es un *setter*. Debe comprobar que la ruta ingresada exista[2], de ser así debe actualizar el atributo `self._ruta` por `nueva_ruta`.
- `def texto_desencriptado(self)`: Es un *getter*. Debe retornar las lineas desencriptadas del archivo que señale `self.ruta`.

Nota: La clase `DrPintoDesencriptador` ya tiene el método `desencriptar` implementado. Tú no debes preocuparte de realizar este proceso.

- `def desencriptar(self)`: Se encarga de desencriptar el archivo que señale `self.ruta`. Devuelve una lista donde cada elemento es una línea desencriptada del archivo.

[2]Para esto, puede serte de ayuda el método `exists` de `os.path`.

Una vez completes `DrPintoDesencriptador`, en el archivo `recuperar_bd.py` **tienes ya** implementada la función:

- `def guardar_archivo(ruta, lineas_archivo)`: Recibe una lista de las líneas del archivo ya desencriptadas y las guarda en `ruta`.

Mientras que tú debes implementar la función:

- `def recuperar_archivo(ruta)`: Recibe una ruta de un archivo y deberá aplicar el proceso de desencriptación a cada elemento del archivo. Finalmente devuelve una lista donde cada elemento del archivo es una línea del archivo desencriptada[3].

3. Reimplementación del banco

Una vez ya hayas logrado recuperar los archivos originales de la base de datos y con el objetivo que esto no vuelva a ocurrir deberás completar en el archivo `banco_seguro.py` algunas clases que harán del Banco DCC un mejor banco para todos. Para esto deberás usar herencia, con el objetivo de aprovechar las clases ya implementadas.

- `class ClienteSeguro`: Hereda de `Cliente`. Debes completar los siguentes métodos, además del constructor (`__init__`):
  - `def saldo_actual(self)`: Usando *properties*, debes definir un *getter* y un *setter* para el saldo del cliente. En el caso del *setter*, si el saldo en algún momento es menor a 0, entonces se debe **marcar** a este cliente como fraudulento, para esto utiliza el atributo booleano `self.tiene_fraude`.
  - `def deposito_seguro(self, dinero)`: Deberás usar el método depositar ya implementado en la clase `Cliente`, actualizar `saldo_actual` y por último, guardar inmediatamente la transacción en `banco_seguro/transacciones.txt`.[4]
  - `def retiro_seguro(self, dinero)`: Deberás usar el método retirar ya implementado. Sin embargo antes de realizar el retiro, debes verificar si es un cliente fraudulento, si no lo es, entonces debe realizar el retiro, actualizar `saldo_actual` y registrar la transacción realizada en `banco_seguro/transacciones.txt`.
- `class BancoSeguroDCC`: Hereda de `BancoDCC`. Considerando que se utilizan los valores desencriptados, completa los métodos a continuación, además del constructor (`__init__`):
  - `def cargar_clientes(self, ruta)`: Recibe la dirección del archivo donde se encuentran los clientes. Deberá leerlo y con cada línea crear una instancia de la clase `ClienteSeguro` y guardarlo en en su atributo clientes.
  - `def realizar_transaccion(self, id_cliente, dinero, accion)`: Recibe un identificador de un cliente, deberá buscarlo y realizar un depósito si es que el parámetro acción tiene valor `"depositar"` o bien retirar dinero si es que la acción es `"retirar"`. Utiliza los métodos de `ClienteSeguro`.

  Mencionar que esta clase tiene además los métodos:
  - `def verificar_historial_transacciones(self, historial)`
  - `def validar_monto_cliente(self, ruta)`

[3] Aquí te sería de mucha ayuda usar la clase `DrPintoDesencriptador` que ya implementaste
[4] Abrir el archivo en modo `"a"` te puede ser útil aquí

Estos son métodos que se encargar de ver si el historial de transacciones no ha sido modificado y ver que clientes tienes saldos falsos. Sin embargo de esto se hablará más en detalle en la siguiente sección.

4. Clientes fraudulentos

Como se mencionó en la sección anterior, producto del ataque que sufrió el BancoDCC se insertaron algunos **clientes fraudulentos**. Aquellos se pueden detectar de dos formas: son todos aquellos clientes que tienen al menos una transacción en la que retiraron más dinero que el que tenían disponible (su saldo), o son aquellos tal que su saldo almacenado en el archivo de clientes no corresponde al total que debería según sus transacciones.

Para lograr expulsar a todos los mafiosos que existen en el sistema deberás implementar siguientes métodos:

- `def verificar_historial_transacciones(self, historial)`: Recibe historial, que es una lista donde cada elemento representa una transacción. Cada transacción viene como un texto de la forma `id, accion, monto`. Este método deberá procesar cada transacción utilizando las instancias de sus clientes seguros, de forma que se detecten aquellos movimientos fraudulentos que fueron inyectados al archivo encriptado que se recibe.
- `def validar_monto_clientes(self, ruta)`: Este método deberá tomar el saldo actual del cliente registrado en la instancia del BancoSeguroDCC y compararlo con lo registrado en el archivo que se encuentra en `ruta` (se asume que en dicho archivo se encuentra el saldo final que debería tener el cliente), deberá marcar como fraudulento a todos los clientes cuyo saldo no coincida. Por ejemplo:

./banco_seguro/transacciones.txt:

```
'id, accion, monto'
'1, depositar, 1000'
```

./banco_seguro/clientes.txt:

```
'id, nombre, saldo, contrasena'
'1, Usuario 1, 2000, 123456'
```

El usuario `1` es fraudulento porque tiene una única acción que es cuando depositó `1000`, pero su cuenta actual dice que tiene `2000`, por lo que debe ser marcado también.

Notas

- **NO** puedes modificar ninguno de los códigos que ya se te dan implementados en esta actividad. Solo puedes añadir métodos.
- Se te entregará un archivo `main.py` para que puedas ir probando las funcionalidades que vayas completando de manera más fácil.
- Si no sabes por donde empezar, o como se utilizará cada método te recomendamos mirar el archivo `main.py`, puede que te ayude a comprender mejor el flujo del programa que debes completar.

Requerimientos

- (2.00 pts) Recuperando base de datos.
  - (0.75 pt) Completa la *property* `ruta` correctamente.
  - (0.75 pt) Completa la *property* `texto_desencriptado` correctamente.
  - (0.5 pt) Completa la función `recuperar_archivo` correctamente.
- (3.00 pts) Reimplementación del banco.
  - (0.15 pt) Completa `__init__` de `ClienteSeguro` correctamente.
  - (0.30 pt) Completa la *property* `saldo_actual` correctamente.
  - (0.60 pt) Completa el método `deposito_seguro` correctamente.
  - (0.60 pt) Completa el método `retiro_seguro` correctamente.
  - (0.15 pt) Completa `__init__` de `BancoDCCSeguro` correctamente.
  - (0.60 pt) Completa el método `cargar_clientes` correctamente.
  - (0.60 pt) Completa el método `realizar_transaccion` correctamente.
- (1.00 pt) Clientes fraudulentos.
  - (0.50 pt) Implementar el método `verificar_historial_transacciones` correctamente.
  - (0.50 pt) Implementar el método `validar_monto_clientes` correctamente.

Entrega

- **Lugar:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/AC01/`
- **Hora del *push*:** 16:30

Auto-evaluación

Como esta corresponde a una actividad formativa, te extendemos la instancia de responder, después de terminada la actividad, una auto-evaluación de tu desempeño. Esta se habilitará a las **16:50 de jueves 22 de agosto** y tendrás plazo para responderla hasta las **23:59 del día siguiente**. Puedes acceder al formulario mediante el siguiente enlace:

`https://forms.gle/HLzaUmYf3cFgmxBD6`

El asistir, realizar la actividad y responder la auto-evaluación otorgará como bonificación al alumno 2 décimas para sumar en su peor actividad sumativa del semestre.