Ejercicios sobre Programación Orientada a Objetos

Este documento ha sido elaborado con el editor /procesador Markdown laverna.cc

Este es el fichero md (por si le queréis echar un ojo para Lenguaje de Marcas)

Documentación

Vas a tener que tomar múltiples decisiones a la hora de construir un programa bajo el paradigma de la Programación Orientada a Objetos (POO). Este documento Working Classes te ayudará no sólo a tomar estas decisiones de manera rápida, sino también a construir una arquitectura de la aplicación que respete los principios SOLID.

Cómo manejar los errores y la programación por contrato, en el documento Programación Defensiva.

SOLID

Aplicad los principios SOLID que ya conocéis:

- `SRP` (S) o Principio de Única Responsabilidad (Single responsibility principle): una clase sólo debe tener un motivo para cambiar.
- `LSP` (L) o Principio de sustitución de Liskov: los objetos de un programa deberían ser reemplazables por instancias de sus subtipos sin alterar el correcto funcionamiento del programa (herencia y polimorfismo).

Otros ejercicios

Recuerda que puedes resolver también bajo este paradigma los ejercicios propuestos en el documento Ejercicios propuestos matrices y recursividad

Notación

En los ejercicios que se presentan a continuación, se especifican el tipo de las variables utilizadas ya que algunos lenguajes de programación -a diferencia de `Python`- son tipados. No tengáis en cuenta estos detalles cuando consideréis que no son necesarios.

Cuenta Corriente

Construye una clase de nombre `CuentaCorriente` que permita almacenar los datos asociados a la cuenta bancaria de un cliente, e interactuar con ellos. Este es nuestro ADT.

Esta clase tendrá las siguientes propiedades, métodos y constructores:

1. Propiedades privadas (de momento, en Python nos da igual que sean privadas):
    - `nombre, apellidos, dirección, teléfono`: todas de tipo `string`.
    - `NIF`: objeto instancia de la clase `DNI` que resolvimos en clase**. Se trata de una relación “Has-A” o “Tiene-una”.
    - `saldo`: de tipo `double`.
2. Constructores (inicializador en Python):
    - Constructor que por defecto inicializa las propiedades de la clase (programación defensiva).
    - Constructor al que se le pasen como argumentos todas las propiedades que tiene la clase.
3. Métodos públicos:
    - `set()` y `get()` para todas las propiedades de la clase [Abstracción y encapsulamiento].
    - `retirarDinero()`: resta al saldo una cantidad de dinero pasada como argumento.
    - `ingresarDinero()`: añade al saldo una cantidad de dinero.
    - `consultarCuenta()`: visualizará los datos de la cuenta.
    - `saldoNegativo()`: devolverá un valor lógico indicando si la cuenta está o no en números rojos.

** Puedes reutilizar la clase `DNI` que construimos en clase para definir la clase NIF mediante [herencia], si es que fuese necesario alguna especialización o cambio en la clase `DNI`. Evalúa si es posible reutilizarla tal cual.

Clase Hora

Construye una clase de nombre `Hora` que permita almacenar la hora, así como los métodos para manipularla (este es nuestro ADT). Tendrá las siguientes propiedades y métodos:

1. Propiedades (todas ellas privadas):
    - `hora`: de tipo entero (00 - 24)

    - `minutos`: de tipo entero (00 - 59)
    - `segundos`: de tipo entero (00 - 59)
2. Constructor (inicializador en Python):
    - Constructor que, por defecto, inicialice las propiedades de la clase a 0 [programación defensiva].
    - Constructor al que se le pasen como argumentos tres enteros y se los asigne a las propiedades de la clase. Si la cantidad recibida no satisface las restricciones de los valores impuestos a `horas, minutos y segundos`, el valor que se fija es 0 [Manejo de errores]: devolver un valor neutro, aunque en este caso no lo sea.
3. Métodos de la clase (públicos):
    - `setHora()`: recibe como argumentos tres enteros y se los asigna a las propiedades de la clase. Utiliza el mismo nombre en las variables que reciben los argumentos y en las propiedades de la clase. Este método ha de diseñarse mediante programación por contrato, es decir, debe incluir una precondición: si los valores recibidos no satisfacen las restricciones de los valores impuestos a `horas, minutos y segundos`, el valor que se establece es 0 [Manejo de errores: devolver un valor neutro, aunque en este caso no lo sea]. Ya que va a ser utilizado en el cosntructor, este precondición podría implementarse en su propia rutina para ser llamada desde este método y desde el "constructor".
    - `getHora()`: devuelve la hora como una `lista` de la forma `[horas, minutos, segundos]` o como un `string`de la forma `"horas:minutos:segundos"`.
    - `imprmirHora()` que muestra en consola la hora en formato `string`de la forma `"horas:minutos:segundos"`.
    - Métodos `set() y get()` para todas las propiedades [Abstracción y encapsulamiento].

Tarjeta Prepago

Construye una clase de nombre `TarjetaPrepago` que permita interactuar con la información almacenada en una tarjeta de telefonía móvil prepago (este es nuestro ADT).

Esta clase tendrá las siguientes propiedades, métodos y constructores:

1. Propiedades privadas:
    - `numeroTeléfono`: de tipo `string`.

    - `NIF`: objeto instancia de la clase `DNI` que resolvimos en clase**. Se trata de una relación “Has-A” o “Tiene-una”
    - `saldo`: de tipo `double` (en euros).
    - `consumo`: objeto instancia de la clase `Hora`, para almacenar las horas, minutos y segundos consumidos. Se trata de otra relación “Has-A” o “Tiene-una”. Reutiliza la clase `Hora`que has construído en el ejercicio anterior.
2. Constructores:
    - Constructor que inicializa las propiedades de la clase (programación defensiva).
    - Constructor que recibe como argumentos los valores para las propiedades de clase `numeroTelefono`, `NIF` y `saldo`.
3. Métodos públicos:
    - `set()` y `get()` para todas las propiedades de la clase [Abstracción y encapsulamiento].
    - `ingresarSaldo()`: añade al `saldo` una cantidad de dinero.
    - `enviarMensaje()`: recibe como argumento un entero que representa un número de mensajes a enviar, y resta al saldo 9 céntimos por mensaje.
    - `realizarLlamada()`: recibe un entero que representa el número de segundos hablados. Se restará al `saldo` la cantidad correspondiente calculada en base a 15 céntimos por establecimiento de llamada y 1 céntimo por segundo. También se actualizará la propiedad `consumo`.
    - `consultarTarjeta()`: visualizará todos los datos de la tarjeta en consola.
4. Métodos privados.
    - Necesitarás un método que se encargue de la responsabilidad de convertir la hora (hora:minutos:segundos) a segundos para poder sumar la duración de la llamada al total de la duración de las llamadas en la propiedad `consumo`.
    - Haz uso de todos aquellos métodos privados que estimes necesarios.

Clase Fecha

Construye una clase de nombre `Fecha` que represente el tipo de dato abstracto ADT fecha como tres enteros de nombres `dia`, `mes` y `año`. Estas serían las propiedades de la clase.

1. Propiedades (todas ellas privadas):
    - `dia`: de tipo entero (01 - 31)

    - `mes`: de tipo entero (01 - 12)
    - `año`: de tipo entero (1900 - 3000)
2. Constructor (inicializador en Python):
    - Constructor que, por defecto, inicialice las propiedades de la clase a la fecha 01-01-1900 [programación defensiva].
    - Constructor al que se le pasen como argumentos tres enteros y se los asigne a las propiedades de la clase. Si la cantidad recibida no satisface las restricciones de los valores impuestos a `dia`, `mes` y `año`, el valor que se fija es 01 o 1900 [Manejo de errores]: (devolver un valor neutro, aunque en este caso no lo sea).
3. Métodos de la clase (públicos):
    - Método de nombre `setFecha()` que recibe como argumentos tres enteros y se los asigna a las propiedades de la clase (utiliza el mismo nombre en las variables que reciben los argumentos y en las propiedades de la clase). Este método ha de diseñarse mediante programación por contrato, es decir, debe incluir una precondición: si los valores recibidos no satisfacen las restricciones de los valores impuestos a `dia`, `mes` y`año`, el valor que se establece es 01 o 1900 [Manejo de errores: devolver un valor neutro, aunque en este caso no lo sea]. Ya que va a ser utilizado en el cosntructor, este precondición podría implementarse en su propia rutina para ser llamada desde este método y desde el “constructor”.
    - Métodos `set() y get()` para todas las propiedades [Abstracción y encapsulamiento].
    - `incrementarFecha()`: recibe un entero que representa un número de días e incrementa la fecha en dicha cantidad de dias.
    - `imprimirFecha()`: escribe la fecha en el formato `dia-mes-año` en consola. Se mostrará el nombre del mes, no el número.
    - método privado `mesLetra()` que devuelve el mes en letras asociado al mes numérico guardado en una determinada instancia (objeto) de la clase; este método será invocado desde `imprimirFecha()`.
    - método `getFecha()` que devuelve un `string` que contenga la fecha en el formato dia-mes-año.

Un poco de teoría:

- El capítulo 28 del libro de Python que seguimos se desarrolla un ejercicio mediante el que se ilustran los conceptos de POO. Conviene que le echéis un ojo.
- En el capítulo 27 se explican los conceptos básicos de la POO.

.