Instituto de Tecnología ORT

Carrera: Analista de Sistemas

Materia: FPR - THP

Versión 2022-04-18

TRABAJO PRÁCTICO Nº 2

Resolvé los ejercicios utilizando los diagramas que correspondan. Asegurate de leer al menos dos veces los enunciados antes de intentar confeccionar las soluciones.
Los ejercicios que dicen **[EC]** son para **desarrollar en clase junto con el Docente**.
También, en algunos ejercicios verán **[FPR]** ó **[THP]**: esas son las partes a desarrollar propias de alguna de las dos materias y no para ambas.

ENUNCIADOS

1) Arrancando despacito... [EC]

a) Desde la clase Ejercicio1 crear el método **sumar(…)** que reciba 2 números y devuelva el valor de la suma entre ambos.
b) Realizar un método de la misma clase llamado **pedirNumero(...)** el cual recibe un texto a mostrar y 2 valores que serán los límites mínimo y máximo (inclusive). Deberá devolver un número comprendido entre los límites recibidos.

2) Modelando una Persona [EC]

Diseñá el UML de la clase Persona incluyendo los atributos y métodos que figuran a continuación. En todos los casos siempre agregá la visibilidad y el tipo de dato a todos los miembros de la clase. Luego, transcribí lo que ves en el diagrama a Java. No es necesario (ni importante ahora) agregar la implementación de los métodos declarados, pero sí asegurar que los atributos sean privados.

Clase	**Atributos**	**Métodos**
Persona	nombre apellido	ponerNombre ponerApellido

Una vez lo hayas hecho (y con ObjectAid instalado) generá el diagrama UML y comparalo con lo que imaginabas. [THP]

3) Preparando a la Persona [EC]

Para terminar de construir una persona agregaremos al código que ya tenemos el ***constructor por defecto***. Este, sin recibir ningún parámetro, debe asegurarse de que cada atributo tenga su valor inicial por defecto (en este caso, cada uno tendrá un string vacío). Además, le agregaremos el nuevo método `obtenerNombreCompleto`, que sin recibir ningún parámetro deberá devolver el nombre completo de la persona a partir de su nombre y su apellido (*concatenando* ambos valores).

Explotá el constructor y los cuatro métodos (todos públicos) ya declarados de Persona:

- `Persona()`: Constructor *por defecto*, que no recibe parámetros y debe crear los atributos en blanco.
- `ponerNombre()`: de tipo void, recibe el valor para el nombre y lo aloja en el atributo correspondiente.
- `ponerApellido()`: Igual que `ponerNombre` pero con el apellido.
- `obtenerNombreCompleto()`: de tipo String, que no recibe parámetros y deberá devolver el nombre y apellido de la persona concatenados.

4) "Instanciando" una Persona [EC]

Ahora por fin vamos a "instanciar" una persona y, de paso, ver cómo funciona. Para esto haremos lo siguiente:

a) En una nueva clase: Test, En el método `main(...)` creá una instancia de la clase **Persona** con el operador ***new***. Esta instancia, para mantenerse viva, debe guardarse en una variable de la clase Persona.
b) Desde la clase **Test** asignale un nombre a la persona utilizando el método `ponerNombre(...)`. El nombre de la persona es "Fulano".
c) Asignar a la misma persona el apellido, que es "Gomez".
d) Mostrá por consola el resultado del método `obtenerNombreCompleto(...)` de la clase **Persona**. ¿Qué se muestra?
e) Creá otra persona con otro nombre y hacé todo el mismo proceso que con la instancia de Fulano Gómez.

5) ¿Dónde vive Fulano? [EC]

Fulano tiene un Domicilio. Este domicilio tiene calle, número y ciudad:

a) Creá la clase **Domicilio** de manera similar a la que empleaste al principio con **Persona** (atributos privados, constructor por defecto que defina los valores por defecto para cada atributo y *setters y getters* de todos los campos del domicilio).
b) Agregá en persona un atributo para el Domicilio, y un setter para poder asignarle el domicilio.
c) Creá un método público que muestre por pantalla el domicilio de la persona. Debe tener en cuenta la posibilidad de que la persona no tenga un domicilio asignado.
d) Desde la clase test creá un domicilio y asignáselo a Fulano.
e) Asignale el mismo domicilio a la otra persona.
f) Modificá un dato del domicilio usando la variable que creaste en el main.
g) Mostrá los datos de ambas personas, incluyendo el domicilio. ¿Qué es lo que se puede observar?

6) Tirando el dado[EC]

Realizá el diagrama de clases UML y luego desarrollalo en Java.

Escribí una clase que simule un dado tradicional de 6 caras, con las propiedades y métodos que consideres más adecuados. Recordá que debe tener un constructor el cual setea el valor inicial del dado, más el método **getValor()** que devuelve.

Escribí la clase Test que utilice la clase dado para lanzar dos dados. Haga lo siguiente:

- Decir si el resultado es el mismo en los dos dados, indicando el valor.
- Si el resultado no es el mismo en los dos dados, mostrar el valor mayor.
- Mostrar por pantalla el promedio de los resultados de 100 lanzamientos (de los dos dados).

Para obtener un número aleatorio en Java debes utilizar el método random colocándolo donde lo consideres necesario.

```
// devuelve un valor entre 0 y 5. El 6 es la cantidad de caras en este caso.
valor = (int)(Math.random() * 6);
```

Para más información podés consultar el siguiente link:

https://www.w3schools.com/java/java_math.asp

7) Compumundo [EC] [FPR]

Realizá un diagrama de clases UML y sus respectivas relaciones de asociación (junto a la cardinalidad) a partir de la abstracción del enunciado:

Una persona (de la que se conoce nombre, apellido y DNI) es propietaria de una computadora. La persona es capaz de trabajar y descansar, mientras que la computadora es capaz de prender, apagar y reiniciar. Una computadora consta de marca, tipo (Desktop, Laptop o All in One) y de un procesador. Algunas también traen una lectora de DVD. El procesador posee su propia marca, modelo y nivel actual de temperatura, además de notificar cuando llega a un nivel de temperatura crítica. De la lectora de DVD se conoce su marca y si es capaz de grabar o no.

8) Que grande esta Tarjeta:

Crea la clase **TarjetaDeCredito** con la siguiente estructura:

- Atributos privados:
  - numero (String),
  - titular (String),
  - limiteDeCompra (double),
  - acumuladoActual (double).

- Método:
  - **Constructor parametrizado y público** que reciba número, titular y límite de compra por parámetros y los asigne al atributo correspondiente. El atributo *acumuladoActual* se inicializará con 0 (cero).
  - Los *getters* de cada uno de sus atributos, públicos, y los *setters*, todos privados, menos el método **setAcumuladoActual()** que no existe.
  - El método público **montoDisponible*()*** que devuelve la diferencia entre el límite de compras y el acumulado actual de gastos, pero si por alguna razón este valor es inferior a cero devuelve cero. Por ejemplo, si gastaste determinado monto y luego cambiaron el límite a un valor menor a éste, el monto disponible debe ser 0 (cero).
  - El método privado **compraPosible()** que según el monto recibido por parámetro devuelve si se puede o no hacer la compra. Para saber si la compra es posible el monto de la misma no debe superar al monto disponible para compras.
  - El método público **actualizarLimite()**, que recibe un nuevo límite de compra.
  - El método privado **acumularGastoActual()**, que recibe el importe de la compra y lo suma al acumulado actual.
  - El método público **realizarCompra()**, el cual dado un monto comprueba si esta se puede realizar (si con la compra no se supera el límite), y si es posible la procesa actualizando los atributos que deba actualizar siempre usando los métodos que corresponda. Este método devuelve un booleano que indica si la compra se pudo realizar o no.
  - El método **toString()** (público), el cual además de los atributos debe incluir el monto disponible para comprar.

En la clase **Test**, se creará un objeto tarjeta con la siguiente información:

- Número 4145-4141-2222-1111
- Titular Juan Perez
- Límite 10000

Luego:

- Hacer una compra de $4000
- Mostrar el estado de la instancia (aprovechando el método *toString()*). Verás que el disponible debería ser de $6000.
- Bajar el límite a $3000.
- Intentar otra compra de $4000 (no debería poder).
- Volver a mostrar el estado de la clase; ahora el disponible debería ser $0.

9) Corralito corralón [EC]

Una cuenta bancaria consta de los siguientes atributos:

- La clave bancaria uniforme (CBU).
- El tipo (caja de ahorro o cuenta corriente).
- El saldo (inicialmente en 0).
- El titular de la cuenta (de tipo **Persona**).

La persona titular de la cuenta tiene DNI, nombre, apellido, y domicilio (calle, altura y barrio).

Cada vez que se crea una nueva cuenta bancaria, debe generarse un nuevo CBU. Para ello, agregá a la clase **CuentaBancaria**, sin parámetros (no es necesario que los implementes en NS+):

- **generarCBU**: devuelve una nueva clave bancaria uniforme.

Las operaciones que debe poder hacer toda cuenta bancaria son:

- **obtenerSaldo(): double**
  Devuelve un valor *double* con el saldo actual de la cuenta.
- **depositar(double): void**
  Actualiza el saldo de la cuenta, sumando el monto enviado por parámetro.
- **extraer(double): boolean**
  Actualiza el saldo de la cuenta, restando el monto enviado por parámetro, siempre y cuando el saldo alcance, de lo contrario, no se realiza la extracción

a) Modelá la solución en UML

b) Realizá la implementación de cada método utilizando diagramas Nassi Schneiderman y luego pasalo a Java. Para el método **generarCBU** alcanza con que devuelvas un String formateado de la siguiente manera (no importa que no sea la forma correcta):

a. Un valor de dos dígitos que se calcula sumando 10 al *valor ordinal* del tipo de cuenta.
b. un guión,
c. 8 dígitos que indican el número de DNI completo,
d. otro guión,
e. el último dígito del DNI.

c) También realizá los constructores de las siguientes clases:

a. **CuentaBancaria**.
b. **Persona.**
c. **Domicilio.**

El método **main** de la clase **Principal** que instancie lo siguiente y luego lo muestre:
Una cuenta bancaria de tipo caja de ahorros le pertenece a Fulano Gómez, cuyo DNI es 12345678 y otra de tipo cuenta corriente le pertenece a Mengana Torres, con DNI 9123456. Ambos son pareja y viven juntos en Yatay 240, Almagro.

10) Que robot educado!! [EC]

Se precisa un robot que permita atender llamadas telefónicas. La compañía puede detectar algunos clientes según su número de teléfono, sin embargo, en otros casos no. Por ello, el robot debe ser capaz de procesar alguno de los siguientes métodos homónimos:

- **saludar(): void**
  Emite por consola un saludo diciendo: **"Hola, mi nombre es ___. ¿En qué puedo ayudarte?"**. En donde el **"_"** debe reemplazarse por el nombre del robot.
- **saludar(Persona): void**
  Emite por consola un saludo diciendo: **"Hola ___, mi nombre es ___. ¿En qué puedo ayudarte?"**. En donde en el primer **" "** debe reemplazarse por el nombre completo de la persona y el segundo **"_"** debe reemplazarse por el nombre del robot.

Modelar la clase **Robot** en UML y realizar la implementación de cada método utilizando diagramas Nassi Schneiderman.

Luego de pasarlo a Java, invocar varias veces el método **saludar** a través del método **main** de la clase **Principal** con diferentes variantes para ver si saluda como corresponde. No olvides instanciar un objeto **Robot** en el **main**.

11) Dando turnos[EC]

Desarrollar la clase **Turnera**, la cual sabe dar números.
La **Turnera** posee un valor entero que representa el último número otorgado.
Esta clase debe implementar los siguientes métodos:

- **otorgarProximoNumero: void**
- **obtenerUltimoNumeroOtorgado: int**
- **resetearContador: void**

Este último método si no recibe ningún valor reseteará el contador en cero, caso contrario resetea el atributo en el valor recibido siempre y cuando sea un número mayor o igual a cero.

12) Calculín

Desarrollar la clase **Calculadora**, la cual sabe operar valores enteros explotando todos sus métodos (en la división, si el divisor es 0, devolver 0).
La **Calculadora** tiene la siguiente definición:

Calculadora	**Métodos**

	`+ sumar(nroA: entero, nroB: entero): entero`
	`+ restar(nroA: entero, nroB: entero): entero`
	`+ multiplicar(nroA: entero, nroB: entero): entero`
	`+ dividir(nroA: entero, nroB: entero): entero`

Para probarlo cargar las variables a y b con los valores 1 y 2, respectivamente, y realizar las siguientes operaciones:

Operación	**Resultado esperado**
a + b	3
a - b	-1
b - a	1

a * b	2
a / b	0
b / a	2
a / 0	0
b / 0	0
0 / a	0
0 / b	0

13) La Liga de los programadores

Crear la clase **SuperHeroe** con los atributos *nombre* (String), *fuerza* (int), *resistencia* (int) y *superpoderes* (int). Todos los atributos numéricos deberán aceptar valores entre 0 y 100; en caso de que el *setter* correspondiente reciba un valor fuera de rango deberá setear el valor límite correspondiente (si recibe un valor negativo asignar cero, si recibe uno superior a cien, asignar cien).
El constructor de la clase recibirá todos los valores de sus atributos por parámetro y usará los setters (todos privados) para validar el rango correcto de los poderes del superhéroe.

También habrá el método **toString()** para devolver el estado completo de la instancia y un método **competir()**, ambos públicos. Este último recibirá otro superhéroe como parámetro y, comparando los poderes de él mismo contra el otro recibido por parámetro, devolverá TRIUNFO, EMPATE o DERROTA, dependiendo del resultado. Para triunfar un superhéroe debe superar al otro en al menos 2 de los 3 ítems.

Escribir la clase Test que contenga el método **main()** para probar el correcto funcionamiento de la clase previamente realizada con el siguiente ejemplo:

superHeroe1: Nombre: “Batman”, Fuerza: 90, Resistencia: 70, Superpoderes: 0

superHeroe2: Nombre: “Superman”, Fuerza: 95, Resistencia: 60, Superpoderes: 70

Hacer jugar al superheroe1 pasándole el objeto superheroe2 y mostrar el resultado por pantalla. Chequear el resultado (debería ser DERROTA).

Hacer jugar al superheroe2 contra el superheroe1 y mostrar el resultado por pantalla. Chequear el resultado (debería ser TRIUNFO).

Crear más superhéroes con distintos valores y jugar.

14) Contador, Acumulador, Promedio

Un **Contador** sabe contar números de uno en uno, empezando en cero (valor inicial). Sólo tiene un método para incrementar su valor (en 1, sin parámetros) y otro para obtenerlo (**obtenerValor()**, devuelve un entero).
Un **Acumulador** es muy parecido, pero para incrementar su valor necesita que le pasen la cantidad a incrementar (un número real). Su método **obtenerValor()** devuelve un número real que puede ser negativo.
Un **Promedio** tiene internamente un contador y un acumulador, los que le servirán para obtener el resultado del promedio. Para cumplir con su funcionalidad tiene un método llamado **incrementar()**, que recibe un número real. Al igual que Contador y Acumulador, tiene un método **obtenerValor()** que en este caso devolverá el valor del promedio (real). Para no tener problemas con la división por cero, que produce un error de ejecución, se decidió que devuelva 0 (cero) cuando se pida el valor promedio antes de haber procesado algún dato.
Escribir un programa que cargue las notas del primer parcial de los integrantes del grupo (sólo las notas entre 0 y 10, sin nombre y mientras la nota no sea -1) y muestre la nota promedio.