Trabajo Práctico 2 - Introducción a la POO

Desde aquí nos adentraremos más en la programación a partir del aprendizaje de un paradigma de programación conocido como Programación Orientada a Objetos (POO).

Esta práctica te ayudará a conocer algunas herramientas que te permitirán escribir programas en Java desde Eclipse que cumplan con lo que propone este paradigma de programación.

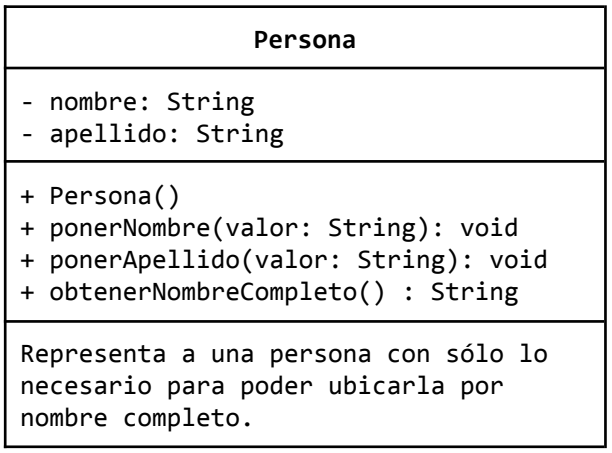
*Resolvé los ejercicios utilizando **Java** desde **Eclipse**. Asegurate de leer al menos dos veces cada ejercicio antes de intentar su resolución.. Los ejercicios están codificados como* **[en clase]** *(para desarrollar con tu docente;* **[en grupo]** *(para desarrollar en clase pero trabajando en grupo) y* **[tarea]** *(para desarrollar por tu cuenta).*

Actividad 1

A partir de la siguiente definición, más que básica, de una persona:

"Una persona tiene nombre y apellido. Su nombre completo se forma uniendo el nombre al apellido en un solo valor".

… observá el *Diagrama de Clase* de **Persona**, cuyo diseño es el siguiente:



Esta clase tiene dos ***atributos*** (el nombre y el apellido, ambos strings) y cuatro ***métodos*** (el constructor, dos *setters* y un método que permite recuperar el nombre completo de la persona) que determinan qué puede hacerse con las personas dentro de este programa.

Creá desde Eclipse un nuevo proyecto llamado **2021C1_THP_TP2**, agregale el paquete **ar.edu.ort.thp.tp2.actividad1.ejercicio0** y allí creá archivo de clase (tal como lo hacías en el trabajo práctico anterior) llamado **Persona**. Pegá en él el código que aparece en la próxima página. Comparalo con lo que ves en el diagrama anterior y tratá de responder las siguientes preguntas discutiéndolas con tus compañeros a partir de lo que ves en el código:

A. ¿Qué está haciendo el *constructor*?
B. ¿Qué hacen los métodos *ponerNombre* y *ponerApellido*?
C. ¿Qué diferencia hay entre estos métodos y el constructor?
D. ¿Qué hace el último método? ¿Qué características lo diferencian de los anteriores?

```
/**
 * Representa a una persona con solo lo necesario para poder ubicarla por nombre
 * completo.
 */
public class Persona {

	private String nombre;
	private String apellido;

	public Persona() {
		nombre = "";
		apellido = "";
	}

	public void ponerNombre(String valor) {
		nombre = valor;
	}

	public void ponerApellido(String valor) {
		apellido = valor;
	}

	public String obtenerNombreCompleto() {
		return nombre + " " + apellido;
	}

}
```

Ejecutá el programa de la clase **Persona**. ¿Qué sucede? ¿Podés hacerlo?

Ahora creá un nuevo de archivo dentro del mismo paquete llamada **Test** y pegá el siguiente código:

```
public class Test {

	public static void main(String[] args) {
		Persona unaPersona = new Persona();

		System.out.println("El nombre de la persona es "
			+ unaPersona.obtenerNombreCompleto());

		unaPersona.ponerNombre("Hortencio");
		unaPersona.ponerApellido("Ortega");
```

```
            System.out.println("El nombre de la persona es "
                  + unaPersona.obtenerNombreCompleto());
      }
}
```

Ejecutá ahora el nuevo archivo y compartí con tus compañeros el resultado de la ejecución. ¿Qué está haciendo de especial **Test** con **Persona**? ¿Por qué no funciona la clase Persona por sí misma? ¿Qué función cumple el método *main()* en un programa? ¿Y qué hace el operador **new** con Persona en este último método?

Y respecto a la clase **Persona**, ¿estaban bien las conclusiones respecto a las preguntas previas? ¿Coincide lo que suponíamos con lo que realmente hace este programa bajo el paradigma de la POO?

Entender en detalle sin experiencia previa el código anterior parece complicado, pero avancemos paso a paso y en poco tiempo estaremos listos para entenderlo a la perfección.

Ejercicios

Desarrollá los siguientes ejercicios armando un *package* (paquete) por cada actividad y dentro por cada ejercicio.

1. **Arrancando despacito**
   Antes de seguir avanzando con más conceptos veamos si podemos diseñar y llevar a código los siguientes ejercicios. No te preocupes por ahora en el desarrollo de cada método (qué hace cada uno de ellos) sino más bien en la correspondencia entre lo que propone el diseño de las clases y el código en Java. Alcanza con escribir lo que va apareciendo en la tabla asociado a cada clase:

Clase	**Atributos**
Fecha [en clase]	- dia (entero) - mes (entero) - anio (entero)
	Métodos
	`+ mostrarComoCadena()` `+ mostrarDiasTranscurridos()`
TelefonoMovil [en grupo]	**Atributos**
	- marca (entero) - encendido (booleano)
	Métodos

	`+ prender(): void`
	`+ apagar(): void`
	`+ llamar(numero: String)`

MaquinaDeCafe [tarea]	**Atributos**
	- nivelAgua (entero) - cantidadVasos (entero)
	Métodos

	`+ prender(): void`
	`+ apagar(): void`
	`+ prepararInfusion(): void`

Actividad 2

Avanzamos sobre el desarrollo de programas viendo cómo las clases se relacionan entre ellas y empezando a *explotar* sus métodos (es decir, a escribir el código dentro de los métodos para que las clases ganen comportamiento).

También veremos un nuevo tipo de dato conocido como **Enumerados** (o como se los suele llamar: ***Enums***).

Ejercicios

2. **Compumundo**: [en clase]
   Escribí las clases que implementen el siguiente escenario:

   "Una persona (de la que sabemos 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 (cuando su temperatura es crítica lo notifica). Si tiene la lectora de DVD, esta también tiene su marca y se sabe además si es capaz de grabar o no."

   Cuando tengas declaradas las clases (aún sin explotar sus métodos) generá el diagrama UML de las mismas a partir del código escrito utilizando la extensión ObjectAid de Eclipse y comparalo con el diagrama de clases generado con UMLetino en Fundamentos de Programación.

3. **Fulano y su Giat** [en grupo]
   Escribí las clases que implementen el siguiente escenario teniendo en cuenta al desarrollarlo la diferencia que hay entre *clase* e *instancia*:

   "Fulano Gómez se compró un vehículo marca Giat Halio de color gris oscuro que tiene una velocidad máxima, de 180 km/h. Así como Fulano posee el Giat Halio, hay otros propietarios que poseen un vehículo al igual que él, los que pueden ser de otras marcas, modelos, colores, etc. Aún Fulano no lo puede probar ya que no sabe conducir."

4. **¡Fulano tiene registro!** [en clase]
   Copiá las clases del ejercicio anterior y escribí y explotá los métodos mencionados en el siguiente escenario extendido:

   "Fulano Gómez finalmente aprendió a conducir y quiere salir a probar su auto. Para ello debe verificar que el vehículo esté encendido (si no lo está, hacerlo), luego acelerar (indicarle al auto qué velocidad incrementar en Km/h), frenar (indicar la cantidad de Km/h a desacelerar) y girar (para lo cual se indicará dirección y cantidad de grados, nunca superior a 90).

   Finalmente podrá apagar el vehículo."

   Actualizá el diagrama de clases y observá las diferencias con el anterior. Escribí un programa que permita a Fulano salir a dar una pequeña vuelta manzana sin superar los 40 km/h para, finalmente, estacionar.

5. **¡Qué complejo el automóvil de Fulano!** [tarea]
   Crear la clase **Automovil** con los siguientes atributos:
   - *marca* (String),
   - *modelo* (String),
   - *patente* (String),
   - *capacidadTanque* (double),
   - *cantidadDeCombustible* (double)
   - *rendimientoPorLitro* (double, indica cuantos kilometros recorre con un litro de combustible).

   Implementar los siguientes métodos y constructores:
   - Constructor parametrizado: recibe marca, modelo, patente y la capacidad del tanque de combustible.
   - Método privado **calcularConsumo()**: recibe la cantidad de kilómetros que se quiere recorrer y devuelve un double indicando los litros de combustible necesarios según los kilómetros requeridos.
   - Método privado **calcularDistancia**(): recibe la cantidad de litros que quiere consumir y devuelve un double indicando los kilómetros que puede recorrer.
   - Método privado **consumirCombustible**(): recibe la cantidad de litros que se quiere consumir y actualiza la cantidad combustible en el tanque. Devuelve un double indicando los litros faltantes si es que no le alcanza, consumirá lo que tiene.
   - Método público **viajar**(): recibe la cantidad de kilómetros que desea recorrer y devuelve la cantidad de kilómetros que efectivamente se realizaron con el combustible existente en el tanque actualizando la cantidad de combustible consumido. Este método deberá valerse de los métodos anteriores.
   - Método público **cargarCombustible**(): recibe por parámetro la cantidad que se quiere cargar, que nunca debe ser menor o igual que cero o mayor al espacio disponible. Si puede, actualiza el atributo correspondiente. Devuelve true o false dependiendo del éxito de la acción.
   - Método privado **espacioDisponible**(), que devuelve la diferencia entre el tamaño del tanque y los litros de combustible almacenados.

- Método público **pocoCombustible**(), que devuelve un valor booleano, indicando verdadero siempre que la cantidad de combustible sea menor al 15% de la capacidad del tanque.
- El método ***toString***() que devolverá todos los valores de estado del objeto incluyendo **espacioDisponible**() y **pocoCombustible**().

En la clase **Test**, crear el objeto a través de su constructor con los parámetros "Ford","Fiesta", "ABCD123", 40. Setear el rendimiento por litro en 10 y llenar el tanque con 20 (veinte) litros de combustible.

Probar todos los métodos recorriendo 180 kilómetros primero e intentando recorrer 50 después, mostrando siempre la cantidad de kilómetros que devuelve viajar().

Actividad 3

Seguimos practicando y conociendo nuevas relaciones entre clases y, ahora, ¡objetos!

Siempre, luego de cada ejercicio, compará el diagrama generado desde Eclipse con el diseñado previamente con UMLetino.

Apoyate en lo visto y desarrollado en Fundamentos de Programación, más lo que ves con tu docente de Taller de Herramientas de Programación para implementar los ejercicios que aparecen a continuación.

Ejercicios

6. **Corralito corralón...** [en clase]

   Una **CuentaBancaria** consta de los siguientes atributos:
   - La clave bancaria uniforme (CBU).
   - El tipo (caja de ahorro o cuenta corriente).
   - El saldo (inicialmente en 0).
   - La fecha de apertura (de tipo Fecha).
   - El titular de la cuenta (de tipo Persona).

   La **Persona** titular de la cuenta tiene
   - DNI
   - nombre
   - apellido
   - fecha de nacimiento
   - domicilio

   Y el **Domicilio** tiene
   - calle
   - altura
   - barrio.

   Cada vez que se crea una nueva cuenta bancaria debe establecerse automáticamente la fecha de apertura y también generarse un nuevo CBU. Para ello, agregá los siguientes métodos estáticos en la clase **CuentaBancaria**, ambos sin parámetros (no es necesario que los implementes):

- **obtenerFechaDeHoy()**: devuelve una Fecha con la fecha de hoy.
- **generarCBU()**: devuelve una nueva clave bancaria uniforme.

Las operaciones que debe poder hacer toda cuenta bancaria son:

- ***verSaldo()***: double. Devuelve el saldo actual de la cuenta.
- ***depositar(double)***: void. Actualiza el saldo de la cuenta, sumando el monto enviado por parámetro (debe controlarse que no sea negativo. Si lo es, ignorarlo.
- ***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 debe realizar la extracción

Implementá el ejercicio basándote en el diagrama UML y explotá cada método (sin olvidarte de los constructores). En el método ***main()*** de la clase **Test** desarrollá lo siguiente:

"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 90123456. Ambos son pareja y viven juntos en Yatay 240, Almagro."

7. **¡Qué robot educado!** [en clase]
   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?". (Reemplazar el "____" por el nombre del robot).
   - ***saludar(Persona)***: void. Similar al anterior pero esta vez diciendo: "Hola ____, mi nombre es ____. ¿En qué puedo ayudarte?", donde el primer "___" debe reemplazarse por el nombre completo de la persona y el segundo "___" por el nombre del robot.

   Desde el método *main()* de la clase de testeo invocar ambas versiones del método saludar para ver si saluda como corresponde.

8. **Dando turnos**. [en clase]
   Desarrollar la clase **Turnera**, la cual sabe dar Turnos.como números enteros consecutivos. La turnera posee un valor entero que representa el último número otorgado y debe implementar los siguientes métodos:
   - ***otorgarProximoNumero()***: entero. Antes de otorgar un turno incrementa el valor que tiene guardado.
   - ***verUltimoNumeroOtorgado()*** Muestra el valor del último turno otorgado pero no lo incrementa.
   - ***resetearContador()*** y ***resetearContador(entero)***, que si no recibe ningún valor asignará cero al contador, o reseteará el atributo con el valor recibido siempre y cuando sea un número mayor o igual a cero.

   Desarrollar una clase de Testeo que permita probar el buen funcionamiento de la Turnera.

9. **Calculín**. [en grupo]
   Desarrollá una calculadora que sabe operar solo con números enteros explotando todos sus métodos (para la división, si el divisor el cero devolver cero). 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`

   Hacer un programa que use la calculadora para probar las cuatro operaciones.

10. **Fracciones, y más fracciones** [tarea]
    Desarrollar la clase **CalculadoraDeFraccion**, la cual sabe operar fracciones. Cada **Fraccion** posee numerador y denominador como atributos y debe responder a los siguientes métodos:
    - ***valorReal():*** double. Devuelve el valor de la fracción expresado como un número real de doble precisión (double).
    - ***sumar(Fraccion, int)***: Fraccion. Devuelve una nueva fracción con el resultado de la suma con el entero recibido como parámetro.
    - ***sumar(Fraccion, Fraccion)***: Fraccion. Devuelve una nueva fracción con el resultado de la suma con la fracción recibida como parámetro.

11. **La Liga de los Programadores** [tarea]
    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 un *setter* reciba un valor fuera de rango, este deberá setear el valor límite correspondiente (si recibe un valor negativo asignar cero y si recibe un valor superior a 100 asignar 100).

    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.

12. **La impresora que impresiona: ¡Qué impresionante!** [tarea]
Desarrollar la clase **ImpresoraMonocromatica**. Sus atributos serán si está o no encendida, la cantidad de hojas actualmente en la bandeja y el nivel de tinta negra. Inicialmente, toda impresora está apagada, sin hojas y con nivel de tinta 100. Debe responder a los siguientes métodos:
    - ***imprimir(Documento)***, el cual recibe un **Documento** compuesto por una **Fecha**, un título y un cuerpo de texto. Imprime en consola los datos del documento, junto a su título y cuerpo, en el siguiente formato:

```
Fecha                    **Titulo**
Cuerpo
```

      Al hacerlo se descuenta 1 punto de nivel de tinta por cada 50 caracteres del cuerpo impresos y se resta 1 hoja de la cantidad en bandeja por cada 20 caracteres del cuerpo impreso. Se debe verificar antes de imprimir que se cuente con el nivel de tinta suficiente y al menos una hoja en la bandeja.
    - ***nivelSegunCantCaracteres(int)***: int Devuelve la cantidad de tinta debería usarse según la cantidad de caracteres recibida por parámetro.
    - ***recargarBandeja(int)***. Suma a la bandeja una cantidad de hojas. El máximo de la bandeja es de 35 hojas. Se debe verificar no excederse de tal valor. Si el parámetro es negativo, la bandeja se deja como está.

13. **Compumundo Reloaded** [tarea]
Retomando el ejercicio de Compumundo, ahora nos piden explotar algunos métodos extra:
    - ***Persona.grabar(...)*** (método ***grabar*** de la clase **Persona**), el cual recibe por parámetro un DVD y devuelve uno de los siguientes valores:
        - SIN_DVD
        - UNIDAD_SIN_GRABADO
        - GRABACION_OK
    - Agregar al DVD los atributos *marca* y *capacidad*, los que deben ser accesibles a través de sus *getters*.
    - ***Persona.mostrarComputadora(...)***, el cual debe mostrar todos los componentes de la computadora incluyendo sus accesorios.

14. **Contador, Acumulador, Promedio** [tarea]
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, sin nombre) y muestre la nota promedio.