Ejemplos iniciales

Para ver en clase

1. **Una persona**. Vamos a modelar una persona incluyendo los atributos nombre, apellido y domicilio. Para cada uno de ellos agregaremos la visibilidad y el tipo de dato. Además agregaremos un método constructor, los setters necesarios y un método que muestre el nombre y el apellido. Luego probaremos nuestra nueva clase desde otra clase llamada PruebaPersona.

2. Para la persona del punto anterior nos solicitan agregar **otro domicilio**. Eso nos hace pensar cuál sería la mejor forma de implementarlo. Decidimos que el Domicilio (que estará compuesto de calle, nro., piso, dpto., etc.) amerita ser una clase independiente. Entonces crearemos la nueva clase y modificaremos la clase Persona para esta nueva situación. Aquí aparece un ejemplo de relación entre clases. En nuestra clase PruebaPersona agregaremos la creación de un Domicilio y lo agregaremos al objeto persona que habíamos creado. Luego crearemos otra Persona y también le agregaremos ese domicilio. Por último modificaremos algún dato del mismo. ¿Qué ocurrirá con las dos personas?

3. Tenemos una **Cafetera**. La misma tiene los siguientes atributos: capacidadMaxima (la cantidad máxima de café que puede contener la cafetera en mililitros) y cantidadActual (la cantidad actual de café que hay en la cafetera en mililitros).

   Nuestra Cafetera es muy moderna y para usarla debemos implementar los siguientes métodos:

   - Constructor predeterminado o por defecto: establece la capacidad máxima en 1000 y la actual en cero (cafetera vacía).
   - Constructor que recibe la capacidad máxima de la cafetera: inicializa la capacidad máxima con lo recibido y la cantidad actual en cero (vacía).
   - Constructor que recibe la capacidad máxima y la cantidad actual. Si la cantidad actual es mayor que la capacidad máxima de la cafetera, la ajustará al máximo.
   - Setters privados y getters públicos. El setter de la capacidad nunca debe permitir un valor menor a 250 (si es menor lo fuerza a 250); la cantidad actual debe controlar que nunca sea menor a cero ni mayor a la capacidad de la cafetera.
   - llenar(): iguala la cantidad actual de la cafetera con la capacidad máxima.
   - servirTaza(int): simula la acción de servir una taza con la capacidad recibida por parámetro. Si la cantidad actual de café no alcanza para llenar la taza, se sirve lo que haya.
   - vaciar(): setea la cantidad de café actual en cero.
   - agregarCafe(int): añade a la cafetera la cantidad de café indicada, en el caso de ser posible. Devuelve la cantidad sobrante.

Realizaremos la clase pruebaCafe para probar el correcto funcionamiento de todos los métodos de la clase previamente realizada. Se debe crear una cafetera por defecto, otra con medio litro de capacidad y una tercera con tres cuartos litros de capacidad y una cantidad inicial de medio litro de café.

Usar un único método testearCafetera(Cafetera) para probar las tres cafeteras por separado, una cada vez.

4. Una **Tarjeta de Crédito** tiene los siguientes atributos: número, titular, limiteDeCompra y acumuladoActual.

   Para probar su funcionamiento debemos implementar los siguientes métodos:

   - 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 existirá.
   - El método toString() (público), el cual además de los atributos debe incluir el monto disponible para comprar.
   - El método público montoDisponible() que devuelve la diferencia entre el límite de compra y el acumulado actual de gastos, pero si por alguna razón este valor es inferior a cero debe retornar 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 recibe un monto y comprueba si la compra se puede realizar (si con la compra no se supera el límite). 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.

Crear la clase pruebaTarjeta para probar el correcto funcionamiento de todos los métodos de la clase previamente realizada. Se debe crear una tarjeta con un límite de $100000. Luego hacer una compra de $40000, mostrar el estado de la instancia (aprovechando el método toString()). El disponible debería ser de $60000. Después bajar el límite a $30000. Intentar otra compra de $40000 (no debería poder realizarla). Volver a mostrar el estado de la clase; ahora el disponible debería ser cero.

5. **Liga de la justicia informática**. Crearemos la clase SuperHeroe con los atributos nombre, fuerza, resistencia y superpoderes. 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 y si recibe un valor superior a cien asignar cien).

   El constructor de la clase recibirá todos los valores de sus atributos y usará los setters (todos privados) para validar el rango correcto de los atributos del superhéroe. Además deberá tener 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 los del otro superhéroe 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.

   Escribiremos la clase PruebaLiga 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.

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

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