CENTRE ESPECÍFIC
D´EDUCACIÓ A DISTÀNCIA DE
LA COMUNITAT VALENCIANA

UNIDAD 8
PROGRAMACIÓN ORIENTADA A OBJETOS I
EJERCICIOS

PROGRAMACIÓN
CFGS DAW

Autor:
Lionel Tarazón – lionel.tarazon@ceedcv.es

2019/2020

UD08. PROGRAMACIÓN ORIENTADA A OBJETOS I

Estos ejercicios están pensados para **empezar creando clases muy sencillas** (apartado A) **que luego irás mejorando y ampliando** (apartados B, C…) de modo que practiques y aprendas poco a poco los aspectos fundamentales de la POO. **Lo importante es que entiendas qué está pasando**. Si no lo tienes claro "juega" con el código, haz pruebas "aver qué sucede si…", revisa la teoría, etc. Si aún así no lo entiendes, pregunta en el foro (copia-pega el código si procede).

APARTADO A – CLASES SIMPLES CON ATRIBUTOS

En cada ejercicio debes crear un programa con dos clases: una clase principal (puedes llamarla por ejemplo UD8_ProgramaPunto, según el ejercicio) que solo contendrá la función main, además de **otra clase** (con sus atributos y métodos) que utilizarás desde el main de la clase principal para hcer pruebas sobre su funcionamiento.

En este apartado las clases solo contendrán atributos (variables) y haremos algunas pruebas sencillas con ellas para entender cómo se instancia objetos y se accede a sus atirbutos. Por ahora no utilices ningún modificador en los atributos (public, private, static, final…).

Ejercicio A1 – Punto

Crea un programa con una clase llamada Punto que representará un punto de dos dimensiones en un plano. Solo contendrá dos atributos enteros llamadas **x** e **y** (coordenadas).

En el main de la clase principal instancia 3 objetos Punto con las coordenadas (5,0), (10,10) y (-3, 7). Muestra por pantalla sus coordenadas (utiliza un println para cada punto). Modifica todas las coordenadas (prueba distintos operadores como = + - += *=…) y vuelve a imprimirlas por pantalla.

Ejercicio A2 – Persona

Crea un programa con una clase llamada Persona que representará los datos principales de una persona: **dni**, **nombre**, **apellidos** y **edad**.

En el main de la clase principal instancia dos objetos de la clase Persona. Luego, pide por teclado los datos de ambas personas (guárdalos en los objetos). Por último, imprime dos mensajes por pantalla (uno por objeto) con un mensaje del estilo "Azucena Luján García con DNI … es / no es mayor de edad".

Ejercicio A3 – Rectángulo

Crea un programa con una clase llamada Rectangulo que representará un rectángulo mediante dos coordenadas (x1,y1) y (x2,y2) en un plano, por lo que la clase deberá tener cuatro atributos enteros: **x1**, **y1**, **x2**, **y2**.

En el main de la clase principal instancia 2 objetos Rectangulo en (0,0)(5,5) y (7,9)(2,3). Muestra por pantalla sus coordenadas, perímetros (suma de lados) y áreas (ancho x alto). Modifica todas las coordenadas como consideres y vuelve a imprimir coordenadas, perímetros y áreas.

Ejercicio A4 – Artículo

Crea un programa con una clase llamada Articulo con los siguientes atributos: **nombre**, **precio** (sin IVA), **iva** (siempre será 21) y **cuantosQuedan** (representa cuantos quedan en el almacén).

En el main de la clase principal instancia un objeto de la clase artículo. Asígnale valores a todos sus atributos (los que quieras) y muestra por pantalla un mensaje del estilo "Pijama - Precio:10€ - IVA:21% - PVP:12,1€" (el PVP es el precio de venta al público, es decir, el precio con IVA). Luego, cambia el precio y vuelve a imprimir el mensaje.

APARTADO B – CONSTRUCTORES

El constructor es el método que se ejecuta cuando se instancia un objeto. Si no está definido Java ejecutará un constructor por defecto que creará el objeto e inicializará todas sus variables a cero, pero esta no es una buena práctica. Es mejor definir nosotros un constructor que controle qué debe suceder cuando se cree el objeto.

En este apartado tienes que modificar los programas del apartado anterior (o haz una copia del proyecto si lo prefieres) **y realizar los cambios indicados**.

Ejercicio B1 – Punto

Añade a la clase Punto un constructor con parámetros que copie las coordenadas pasadas como argumento a los atributos del objeto. Así:

```
public Punto(int x, int y){
        this.x = x;
        this.y = y;
}
```

Copiamos los valores pasados como argumento a los atirbutos del objeto. Ten en cuenta que **int x** e **int y** son variables locales del método, NO son los atributos del objeto. Para hacer referencia a los atributos del objeto hay que utilizar **this**.

Fíjate que ya no será posible hacer Punto p = new Punto(). Ahora será obligatorio hacer por ejemplo Punto p = new Punto(2, 7). En el apartado A tenías que recordar asignar valores a x e y tras crear un punto, lo cual no es una buena idea en proyectos grandes con cientos de objetos (es muy fácil equivocarse). Ahora es imposible equivocarse porque Java no te dejará. Hemos asegurado que todos los puntos siempre tendrán coordenadas.

Corrige el main y utiliza el constructor con parámetros para instanciar los objetos, pasándole como argumento los valores deseados.

Ejercicio B2 – Persona

Añade a Persona el constructor de abajo y corrige el main para utilizarlo:

```
public Persona(String dni, String nombre, String apellidos, int edad) {
        this.dni = dni;
        this.nombre = nombre;
        this.apellidos = apellidos;
        this.edad = edad;
}
```

Ten en cuenta que no es obligatorio que los parámetros del constructor se llamen igual que los atributos del objeto (en tal caso no sería necesario utilizar **this**). Podríamos hacerlo así:

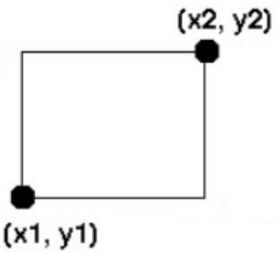
```
public Persona(String id, String nom, String ap, int e) {
        dni = id;
        nombre = nom;
        apellidos = ap;
        edad = e;
}
```

Tampoco es obligatorio pasar al constructor todos los atributos de la clase. Podríamos decidir por ejemplo que en nuestro software todas las personas deben tener nombre, apellidos y edad, pero no es obligatorio el DNI (recién nacidos y niños). Este constructor también sería válido:

```
public Persona(String nom, String ap, int e) {
        nombre = nom;
        apellidos = ap;
        edad = e;
}
```

Una clase puede tener tantos constructores como quieras siempre y cuando tengan distinto número y/o tipo de parámetros (para que no haya ambigüedad en cual utilizar).

Ejercicio B3 – Rectángulo



En nuestro software necesitamos asegurarnos de que la coordenada (x1,y1) represente la esquina inferior izquierda y la (x2,y2) la superior derecha del rectangulo, como en el dibujo.

Añade a Rectangulo un constructor con los 4 parámetros. Incluye un if que compruebe los valores **(*)**. Si son válidos guardará los parámetros en el objeto.

Si no lo son mostrará un mensaje del estilo "ERROR al instanciar Rectangulo..." utilizando System.err.println(...). No podremos evitar que se instancie el objeto pero al menos avisaremos por pantalla.

Corrige el main para utilizar dicho constructor. Debería mostrar un mensaje de error.

(*) Pista: Es suficiente con un if ((condición) && (condición))

Ejercicio B4 – Artículo

Añade un constructor con 4 parámetros que asigne valores a nombre, precio, iva y cuantosQuedan. Dicho constructor deberá mostrar un mensaje de error si alguno de los valores nombre, precio, iva o cuantosQuedan no son válidos. ¿Qué coindiciones crees que podrían determinar si son válidos o no? Razónalo e implementa el código.

Corrige el main y prueba a crear varios artículos. Introduce algunos con valores incorrectos para comprobar si avisa del error.

APARTADO C – GETTERS Y SETTERS

Un pilar fundamental de la programación orientada a objetos (POO) es el encapsulamiento:

> *"Se denomina encapsulamiento al ocultamiento del estado, es decir, de los datos miembro de un objeto de manera que solo se pueda cambiar mediante las operaciones definidas para ese objeto.*
>
> *Cada objeto está aislado del exterior. El aislamiento protege a los datos asociados a un objeto contra su modificación por quien no tenga derecho a acceder a ellos, eliminando efectos secundarios e interacciones. De esta forma el usuario de la clase puede obviar la implementación de los métodos y propiedades para concentrarse solo en cómo usarlos.*
>
> *Por otro lado se evita que el usuario pueda cambiar su estado de maneras imprevistas e incontroladas."* Fuente: Wikipedia

Por ello, una práctica muy habitual en POO consiste en **ocultar todos los atributos** (hacerlos **private**) para que no se puedan modificar directamente desde fuera de la clase. En su lugar añadiremos **métodos getters (get = coger) y setters (set = fijar)** visibles (**public**) que permitan leer y modificar dichos atributos desde fuera de la clase. La clave está en que al tratarse de métodos podremos incluir el código necesario para **controlar el acceso a los atributos y protegerlas de usos incorrectos**.

En este apartado tienes que modificar los programas del apartado anterior (o haz una copia del proyecto si lo prefieres) **y realizar los cambios indicados**.

Ejercicio C1 – Punto

Modifica los atributos de Punto para que sean **private**. Fíjate que desde el main ya no te dejará utilizar ni modificar los atributos x e y de los objetos.

Vamos a añadir los **getteres**: **int getX()** e **int getY()** que devolverán los valores de x e y respectivamente. Es una forma indirecta de leer sus valores.

Añadiremos también los **setters**: **void setX(int x)** y **void setY(int y)** que copiarán el valor pasado como parámetro a los atributos de la clase.

Tanto getters como setters deben ser **public**.

Corrige el main para utilizar los getters y setters. Prueba a instanciar varios objetos, mostrar sus valores por pantalla, modificarlos, etc.

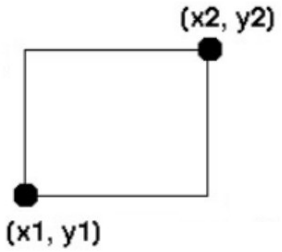
```
public class Punto {

    private int x;
    private int y;

    public Punto(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public int getX(){
        return x;
    }

    public int getY(){
        return y;
    }

    public void setX(int x) {
        this.x = x;
    }

    public void setY(int y) {
        this.x = y;
    }
}
```

Ejercicio C2 – Persona

Aplica el encapsulamiento básico a la clase Persona: Declara todos sus atributos como private y crea todos los getters y setters necesarios (un get y un set por atributo).

Corrige el main para utilizar los getters y setters. Prueba a instanciar varios objetos, mostrar sus valores por pantalla, modificarlos, etc.

Ejercicio C3 – Rectángulo



Aplica el encapsulamiento básico a la clase Rectángulo: Declara todos sus atributos como private y crea todos los getters y setters necesarios (un get y un set por atributo).

¿Recuerdas la condición explicada en B3? Tendrás que programar los setters de modo que comprueben el valor pasado como argumento antes de guardarlo en el objeto. Si no fuera correcto se mostrará un mensaje de error (y NO se guardará el valor).

Corrige el main para utilizar los getters y setters. Prueba a instanciar varios objetos, mostrar sus valores, modificarlos, etc. Prueba varios valores erróneos para comprobar si funciona.

Ejercicio C4 – Artículo

Aplica el encapsulamiento básico a la clase Articulo: Declara todos sus atributos como private y crea todos los getters y setters necesarios (un get y un set por atributo).

Programa los setters para que comprueben los valores y los guarden en el objeto solo si son correctos. En caso contrario muestra un mensaje de error.

APARTADO D – AÑADIENDO MÉTODOS ÚTILES

Una clase bien diseñada debería incluir métodos que realicen operaciones con la información de los objetos. De ese modo la clase dispondrá de funcionalidades útiles tanto para nosotros como para otros programadores. Esto es una práctica habitual y muy recomendable.

Por ejemplo, la clase Scanner tiene métodos como getInt(), getDouble, getLine(), etc. que alguien ha programado y podremos utilizar cuando los necesitemos sin tener que programarlo nosotros ni preocuparnos por cómo funcionan internamente. Lo mismo sucede con la clase String (charAt, substring, toCharArray, etc.), la clase Math (random, min, max, abs, etc.), la clase Arrays, etc.

Todas estas son clases que Java incorpora por defecto (hay miles). Existen muchas otras que permiten hacer todo tipo de cosas como crear interfaces gráficas con botones, trabajar con imágenes, leer y escribir en archivos, reproducir música, enviar o recibir datos a través de la red, etc.

En esta unidad estamos aprendiendo a diseñar y programar nuestras propias clases, los fundamentos de la Programación Orientada a Objetos, necesario en cualquier proyecto software de cierta envergadura.

Recuerda que los métodos pueden ser **private** (solo pueden utilizarse desde dentro de la clase) o **public** (pueden utilizarse desde fuera, forman parte de la interfaz de la clase).

En este apartado tienes que modificar los programas del apartado anterior (o haz una copia del proyecto si lo prefieres) **y añade los métodos que se indican.**

Ejercicio D1 – Punto

Añade a la clase Punto los siguientes métodos públicos:

- **public void imprime()** // Imprime por pantalla las coordenadas. Ejemplo: “(7, -5)”
- **public void setXY(int x, int y)** // Modifica ambas coordenadas. Es como un setter doble.
- **public void desplaza(int dx, int dy)** // Desplaza el punto la cantidad (dx,dy) indicada. Ejemplo: Si el punto (1,1) se desplaza (2,5) entonces estará en (3,6).
- **public int distancia(Punto p)** // Calcula y devuelve la distancia entre el propio objeto (this) y otro objeto (Punto p) que se pasa como parámetro: distancia entre dos coordenadas.

Prueba a utilizar estos métodos desde el main para comprobar su funcionamiento.

Ejercicio D2 – Persona

Añade a la clase Persona los siguientes métodos públicos:

- **public void imprime()** // Imprime la información del objeto: “DNI:… Nombre:… etc.”
- **public boolean esMayorEdad()** // Devuelve true si es mayor de edad (false si no).
- **public boolean esJubilado()** // Devuelve true si tiene 65 años o más (false si no).
- **public int diferenciaEdad(Persona p)** // Devuelve la diferencia de edad entre ‘this’ y p.

Prueba a utilizar estos métodos desde el main para comprobar su funcionamiento.

Ejercicio D3 – Rectángulo

Añade a la clase Rectángulo métodos públicos con las siguientes funcionalidades:

- Método para **imprimir** la información del rectángulo por pantalla.
- Métodos setters dobles y cuadruples: **setX1Y1**, **set X2Y2** y **setAll(…)**.
- Métodos **getPerimetro** y **getArea** que calculen y devuelvan el paerímetro y área del objeto.

Prueba a utilizar estos métodos desde el main para comprobar su funcionamiento.

Ejercicio D4 – Artículo

Añade a la clase Artículo métodos públicos con las siguientes funcionalidades:

- Método para **imprimir** la información del artículo por pantalla.
- Método **getPVP** que devuelva el precio de venta al público (PVP) con iva incluido.
- Método **getPVPDescuento** que devuelva el PVP con un descuento pasado como argumento.
- Método **vender** que actualiza los atributos del objeto tras vender una cantidad ‘x’ (si es posible). Devolverá true si ha sido posible (false en caso contrario).
- Método **almacenar** que actualiza los atributos del objeto tras almacenar una cantidad ‘x’ (si es posible). Devolverá true si ha sido posible (false en caso contrario).

APARTADO E – STATIC Y FINAL

Los modificadores **static** y **final** son opcionales, pueden utilizarse tanto en atributos como en métodos y pueden combinarse ambos:

- **static**: El atributo o método pertenece a la clase (no al objeto). Por ello puede utilizarse sin instanciar ningún objeto, desde el nombre de la clase: NombreClase.atributo o NombreClase.metodo(...). Como el valor se almacena en la clase NO toma valores distintos en cada objeto (como sí sucede con los atirbutos 'normales' no static). Por ejemplo:
  - El atributo salarioMinimo de Empleado (ss común para todos, puede cambiar).
  - Métodos útiles como Arrays.fill(...) o String.valueOf(...) que podemos utilizar directamente desde las clases Arrays y String sin instanciar un objeto.

  Es importante saber que desde un método static no se pueden utilizar atributos ni métodos no static. Al contrario sí es posible.
- **final**: Un atributo final no se puede modificar. Puede tener valores distintos en cada objeto, pero debe fijarse su valor en el constructor. Un método final no se puede redefinir en una sub-clase heredada (veremos herencia en la siguiente unidad). Por ejemplo:
  - El DNI de la clase Persona. No puede cambiar y cada objeto tiene el suyo.
- La combinación de static y final combina ambas características. Por ejemplo:
  - El atributo Math.PI es static y final. Pertenece a la clase y no puede cambiar.

Ejercicio E1 – Punto

Necesitamos un método que nos permita crear un objeto Punto con coordenadas aleatorias. Esta funcionalidad no depende de ningún objeto concreto por lo que será estática. Deberá crear un nuevo Punto (utiliza el constructor) con x e y entre -100 y 100, y luego devolverlo (con return).

- public static Punto creaPuntoAleatorio()

Pruebalo en el main para comprobar que funciona. Crea varios puntos aleatorios con Punto.creaPuntoAleatorio() e imprime su valor por pantalla.

Ejercicio E2 – Persona

El DNI de una persona no puede variar. Añade el modificador final al atributo dni y asegúrate de que se guarde su valor en el constructor. Quita el método setDNI(...) que de todos modos ya no se podrá utilizar porque Java no te dejará modificar el atributo dni.

La mayoría de edad a las 18 años es un valor común a todas las personas y no puede variar. Crea un nuevo atributo llamado mayoriaEdad que sea static y final. Tendrás que inicializarlo a 18 en la declaración. Utilízalo en el método que comprueba si una persona es mayor de edad.

Crea un método **static boolean validarDNI(String dni)** que devuelva true si dni es válido (tiene 8 números y una letra). Si no, devolverá false. Utilízalo en el constructor para comprobar el dni (si no es válido, muestra un mensaje de error y no guardes los valores).

Realiza algunas pruebas en el main para comprobar el funcionamiento de los cambios realizados. También puedes utilizar Persona.validarDNI(...) por ejemplo para comprobar si unos DNI introducidos por teclado son válidos o no (sin necesidad de crear ningún objeto).

Ejercicio E3 – Rectángulo

Necesitamos hacer algunos cambios para que todas las coordenadas estén entre (0,0) y (100,100). Añade a la clase Rectángulo dos atributos llamados min y max. Estos valores son comunes a todos los objetos y no pueden variar. Piensa qué modificados necesitas añadir a min y max.

Utiliza min y max en el constructor y en los setters para comprobar los valores (como de costumbre, si no son correctos muestra un mensaje de error y apliques los cambios).

También necesitamos un método no constructor para crear rectángulos aleatorios. Impleméntalo.

Realiza pruebas en el main para comprobar su funcionamiento.

Ejercicio E4 – Articulo

En España existen tres tipos de IVA según el tipo de producto:

- El IVA general (21%): para la mayoría de productos a la venta.
- El IVA reducido (10%): hostelería, transporte, vivienda, etc.
- El IVA super reducido (4%): alimentos básicos, libros, medicamentos, etc.

Estos tres tipos de IVA no pueden variar y a cada artículo se le aplicará uno de los tres.

Razona qué cambios sería necesario realizar a la clase Articulo e impleméntalos.