19. Programación Orientada a Objetos(V)

19.1 Ámbito.

Al definir los elementos de una clase, se pueden especificar sus ámbitos (scope) de **visibilidad** o accesibilidad con las palabras reservadas public (público), protected (protegido) y private (privado).
En la siguiente tabla se muestra desde dónde es visible/accesible un elemento (atributo o método) según el modificador que lleve.

	En la misma clase	En el mismo paquete	En una subclase	Fuera del paquete
private	✔	✖	✖	✖
protected	✔	✔	✔	✖
public	✔	✔	✔	✔
sin especificar	✔	✔	✖	✖

Por ejemplo, un atributo private solamente es accesible desde la misma clase, sin embargo, a un método protected se podrá acceder desde la misma clase donde esté definido, desde otro fichero dentro del mismo paquete o desde una subclase.

Como regla general, se suelen declarar private los atributos o variables de instancia y public los métodos.

19.2. Tipos enumerados.

Mediante ***enum*** se puede definir un tipo enumerado, de esta forma un atributo solo podrá tener uno de los posibles valores que se dan como opción. Los valores que se especifican en el tipo enumerado se suelen escribir con todas las letras en mayúscula.

```
public class Carta {

    public enum Palo {
        TREBOL, DIAMANTE, CORAZON, PICA
    }

    private int numero;
    private Palo palo;


    public int getNumero() {
        return numero;
    }


    public Palo getPalo() {
        return palo;
    }


    Carta(int numero, Palo palo) {
        this.numero = numero;
        this.palo = palo;
    }


    @Override
    public String toString() {
        return (numero + " - " + palo.toString().toLowerCase());
    }


}
```

Usaremos tipos enumerados cuando los valores no vayan a cambiar en el tiempo, días de la semana, estaciones del año, meses…

Los elementos creados son realmente subclases de la clase Enum:

Enum Javadoc

19.3. Interfaces.

Una interfaz contiene únicamente la cabecera de una serie de métodos (opcionalmente también puede contener constantes). Por tanto se encarga de especificar un comportamiento que luego tendrá que ser implementado. La interfaz no especifica el “cómo” ya que no contiene el cuerpo de los métodos, solo el “qué”.

Una interfaz puede ser útil en determinadas circunstancias. En principio, separa la definición de la implementación o, como decíamos antes, el “qué” del “cómo”.

Tendremos entonces la menos dos ficheros, la interfaz y la clase que implementa esa interfaz. Hay que destacar que cada interfaz puede tener varias implementaciones asociadas.

En Java no existe herencia múltiple, una clase no puede heredar de dos subclases, sin embargo si puede **implementar** múltiples interfaces, de hecho la mayor parte de las clases implementan Clonable (Que se puede clonar) y Serializable (Que se puede retransmitir como una ristra de bits).

```java
public interface Movible {


    public enum Direccion {
        ARRIBA, ABAJO, IZQUIERDA, DERECHA;
    }

    public void mover(Direccion d);

}
```

```java
public class Punto implements Movible{

    int x;
    int y;

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }

    @Override
    public void mover(Movible.Direccion d) {
        switch(d) {
        case ARRIBA: y++;
        case ABAJO: y--;
        case IZQUIERDA: x--;
        case DERECHA: x++;

        }

    }
```

19.1. Se quiere informatizar una biblioteca. Crea las clases Publicacion, Libro y Revista. Las clases deben estar implementadas con la jerarquía correcta. Las características comunes de las revistas y de los libros son el código ISBN, el título, y el año de publicación. Los libros tienen además un atributo prestado.

Cuando se crean los libros, no están prestados. Las revistas tienen un número. La clase Libro debe implementar la interfaz Prestable que tiene los métodos presta, devuelve y estaPrestado.

Programa principal:

```
Libro libro1 = new Libro("123456", "La Ruta Prohibida", 2007);
Libro libro2 = new Libro("112233", "Los Otros", 2016);
Libro libro3 = new Libro("456789", "La rosa del mundo", 1995);
Revista revista1 = new Revista("444555", "Año Cero", 2019, 344);
Revista revista2 = new Revista("002244", "National Geographic", 2003, 255);
System.out.println(libro1);
System.out.println(libro2);
System.out.println(libro3);
System.out.println(revista1);
System.out.println(revista2);
libro2.presta();
if (libro2.estaPrestado()) {
        System.out.println("El libro está prestado");
}
libro2.presta();
libro2.devuelve();
if (libro2.estaPrestado()) {
        System.out.println("El libro está prestado");
}
libro3.presta();
System.out.println(libro2);
System.out.println(libro3);
```

Salida:

```
ISBN: 123456, título: La Ruta Prohibida, año de publicación: 2007 (no prestado)
ISBN: 112233, título: Los Otros, año de publicación: 2016 (no prestado)
ISBN: 456789, título: La rosa del mundo, año de publicación: 1995 (no prestado)
ISBN: 444555, título: Año Cero, año de publicación: 2019
ISBN: 002244, título: National Geographic, año de publicación: 2003
El libro está prestado
Lo siento, ese libro ya está prestado.
ISBN: 112233, título: Los Otros, año de publicación: 2016 (no prestado)
```

ISBN: 456789, título: La rosa del mundo, año de publicación: 1995 (prestado)

19.2 Implementa la clase FichaDomino. Una ficha de dominó tiene dos lados y en cada lado hay un número del 1 al 6 o bien ningún número (blanco). Cuando se crea una ficha, se proporcionan ambos valores. Dos fichas encajan si se pueden colocar una al lado de la otra según el juego del dominó, por ejemplo, las fichas [2 | 5] y [4 | 5] encajan porque se pueden colocar de la forma [2 | 5] [5 | 4]. A continuación se proporciona el contenido del main y el resultado que debe aparecer por pantalla.

Programa principal:

```
FichaDomino f1 = new FichaDomino(6, 1);
FichaDomino f2 = new FichaDomino(0, 4);
FichaDomino f3 = new FichaDomino(3, 3);
FichaDomino f4 = new FichaDomino(0, 1);
System.out.println(f1);
System.out.println(f2);
System.out.println(f3);
System.out.println(f4);
System.out.println(f2.voltea());
System.out.println(f2.encaja(f4));
System.out.println(f1.encaja(f4));
System.out.println(f1.encaja(f3));
System.out.println(f1.encaja(f2));
```

Salida:

```
[6|1]
[ |4]
[3|3]
[ |1]
[4| ]
true
true
false
false
```

19.3 El objetivo es implementar las clases básicas para utilizar en un supermercado.

Necesitaremos por tanto almacenar información de Clientes, Dependientes y Reponedores. De todos ellos necesitamos saber su nombre, apellidos, DNI, dirección y teléfono. De los clientes también nos interesa su código de cliente y el número de compras realizadas, esta última característica es necesaria para obtener el descuento que se le debe aplicar al cliente (cada 100 compras se le aplica un 1% de descuento), debemos por lo tanto implementar un método compra que anote que realizó una compra. De los dependientes y reponedores debemos saber su número de la seguridad social, su salario y el turno al que pertenecen (mañana, tarde o noche). Para obtener el salario se tendrá en cuenta que si trabaja en turno de noche tiene un extra de 150€. De los dependientes almacenaremos su especialidad (carnicería, frutería, caja, etc.) y de los reponedores almacenaremos información de la compañía de la que proceden. Todos los atributos se definirán privados y deberán disponer de métodos de lectura y escritura. Se deberá incluir un método toString que sobrescriba el método toString de Object y que permita imprimir toda la información de cada clase. Emplea clases abstractas para generalizar características comunes entre distintas subclases.

19.4 Vamos a crear una clase llamada Persona. Sus atributos son: nombre, edad y DNI. Tendrá un tipo enumerado para indicar si la persona es Española, Comunitaria o Extracomunitaria. Construye los siguientes métodos para la clase:

- Un constructor, donde los datos pueden estar vacíos.
- Los setters y getters para cada uno de los atributos. Hay que validar las entradas de datos.
- mostrar(): Muestra los datos de la persona.
- esMayorDeEdad(): Devuelve un valor lógico indicando si es mayor de edad.

19.5 Crea una clase llamada Cuenta que tendrá los siguientes atributos: titular (que es una persona) y cantidad (puede tener decimales). El titular será obligatorio y la cantidad es opcional. Implementará una interfaz Mostrable, con el método mostrar(). Construye los siguientes métodos para la clase:

- Un constructor, donde los datos pueden estar vacíos.
- Los setters y getters para cada uno de los atributos. El atributo no se puede modificar directamente, sólo ingresando o retirando dinero.
- mostrar(): Muestra los datos de la cuenta.
- ingresar(cantidad): se ingresa una cantidad a la cuenta, si la cantidad introducida es negativa, no se hará nada.

- retirar(cantidad): se retira una cantidad a la cuenta. La cuenta puede estar en números rojos.

19.6 Vamos a definir ahora una “Cuenta Joven”, para ello vamos a crear una nueva clase CuantaJoven que deriva de la anterior. Cuando se crea esta nueva clase, además del titular y la cantidad se debe guardar una bonificación que estará expresada en tanto por ciento. Construye los siguientes métodos para la clase:

- Un constructor.
- Los setters y getters para el nuevo atributo.
- En esta ocasión los titulares de este tipo de cuenta tienen que ser mayor de edad., por lo tanto hay que crear un método esTitularValido() que devuelve verdadero si el titular es mayor de edad pero menor de 25 años y falso en caso contrario.
- Además la retirada de dinero sólo se podrá hacer si el titular es válido.
- El método mostrar() debe devolver el mensaje de “Cuenta Joven” y la bonificación de la cuenta.
- Piensa los métodos heredados de la clase madre que hay que reescribir.