IMPORTANTE

Diferencia entre WHILE y DO-WHILE

He realizado 2 clases Ppal: Ppal_WhileTienda.java y Ppal_DoWhileTienda.java

Ambas tienen el mismo código, a excepción del **ejercicio nº 4**, de ahí esta explicación.
Las dos clases anteriores llaman a métodos implementados en la clase Ejercicios (donde están los métodos que se piden en el examen) y en la clase Introducir (para leer por teclado tipos de datos primitivos: int, double, float, char, y los positivos).

Si os dais cuenta, en métodos de la clase Ejercicios, también hago llamadas a métodos de la clase Introducir, eso produce que la implementación sea ágil en la programación y no se repita tanto el código.

Para el **Ejercicio nº 4**, os he desarrollado 2 métodos, uno es tienda_de_repuestos_While() y el otro: tienda_de_repuestos_DoWhile(); es decir, uno lo he diseñado con While y el otro con Do-While.

Pero antes, me gustaría que entendiérais el siguiente código, que he aplicado en ambos métodos:

```
//Obligo a que tipoPieza sea 0,1 ó 2
int tipoPieza=Introducir.enteroPositivo("Dime el tipo de la pieza que desee: (1 o 2), con 0 finaliza el pedido");
while(tipoPieza > 2){ //TAMBIÉN se puede poner while (! (tipoPieza <=2)) ó while (tipoPieza!=0 && tipoPieza!=1 && tipoPieza!=2 )
    System.out.println("Error, tipo debe ser 0, 1 ó 2");
    tipoPieza=Introducir.enteroPositivo("Dime el tipo de la pieza que desee: (1 o 2), con 0 finaliza el pedido");
}
```

EXPLICACIÓN

1. Introduzco el valor del tipo de la pieza (uso el método enteroPositivo => el entero será 0 o +)
2. Como puede ser que sea > 2, no valdría, ya que tan solo se dispone del tipo 1 o del tipo 2 o tipo 0 (salida)
   -¿Qué hago? Un bucle while, obligando a que si he introducido un valor erróneo
   a) lo indique con el mensaje: “Error, tipo debe ser 0,1 ó 2”
   b) vuelvo a introducir el valor de tipo de pieza.

¿Y si utilizo do-while, en lugar de while? Pues sería algo así:

```
//Obligo a que tipoPieza sea 0,1 ó 2
do{
      tipoPieza=Introducir.enteroPositivo( msg: "Dime el tipo de la pieza que desee: (1 o 2), con 0 finaliza el pedido");
}while(tipoPieza>2);
```

También funciona, nos obliga a introducir las veces que sea necesario el tipo de pieza hasta que sea 0, 1 ó 2. Pero NO nos indica el mensaje de error… , cosa que sería interesante, ¿NO?

CONCLUSIÓN

- Si utilizamos **while(condicion_permanencia_bucle)** { cuerpo_codigo;}, la variable que está en la condición de permanencia debe tener un valor YA, es decir, su valor debe conocerse. Eso me obliga a que se introduzca por teclado tipoPieza previamente.
- Si el tipoPieza cumple que es > 2 , permanecerá en el bucle, indicando además el por qué está en el bucle, hasta que SE INTRODUZCA un valor de tipo de pieza entre 0, 1 ó 2 (**condición de salida del bucle** => tipoPieza <= 2).
- ¿Qué ocurre si la 1ª vez que se introduce tipoPieza es correcto **(tipoPieza <= 2)**?
  Respuesta: NO entra en el bucle **while(tipoPieza>2){….}**. Se deduce que:
  a) Puede ser que while se ejecute 0 veces (mínimo)
  b) **La condición de permanencia (tipoPieza > 2) es opuesta a la condición de salida: (tipoPieza <=2)**
     1. También se podría poner como condición de permanencia: **(!(tipoPieza <= 2))**.
     2. También como condición permanencia: **(tipoPieza != 0 && tipoPieza !=1 && tipoPieza !=2)**
        Significa: NO es ni 0, ni tampoco 1 ni tampoco 2, por lo que, continúa en el bucle.
     3. También como condición permanencia: **(!(tipoPieza == || tipoPieza ==1 || tipoPieza ==2))**
        Significa: NO es (1 ó 2 ó 3)
  c) Dentro del código del bucle **se vuelve a introducir el valor de tipoPieza** -justo al final del cuerpo de código-.

Considero importante que entendáis cómo funciona while y su diseño ya que tanto en tratamiento de ficheros, stream y base de datos, trabajan con while.

DISEÑO DEL EJERCICIO nº 4 CON WHILE

```
//Ejercicio con while, mientras que el tipo de pieza sea != 0 continúa el pedido. TAMBIÉN SE PUEDE PONER: while (!(tipoPieza == 0))
while (tipoPieza!=0){ //TAMBIÉN SE PUEDE PONER: while (tipoPieza == 1 || tipoPieza == 2)
    String nombrePieza=Introducir.cadena("Dime el nombre de la pieza: ");
    double precioPieza=Introducir.realDoblePositivo("Dime el precio de la pieza: ");

    totalPago += Introducir.calculo(tipoPieza, precioPieza);

    tipoPieza=Introducir.enteroPositivo("Dime el tipo de la pieza que desee: (1 o 2), con 0 finaliza el pedido");
    while(tipoPieza>2){ //Si se utilizara Introducir.enteroPositivo, entonces while(tipo>2)
        System.out.println("Error, tipo debe ser 0, 1 ó 2");
        tipoPieza=Introducir.enteroPositivo("Dime el tipo de la pieza que desee: (1 o 2), con 0 finaliza el pedido");
    }
}
```

EXPLICACIÓN

1. Conozco el valor del tipo de la pieza.
2. Como puede ser que sea != 0, se entiende que la pieza es del tipo 1 o del tipo 2
   -¿Qué hago? Un bucle while, se desarrolla el cuerpo del bucle
   - Antes de cerrar el bucle while, se vuelve a preguntar por el valor de tipoPieza.

CONCLUSIÓN

- Si utilizamos **while(condicion_permanencia_bucle)** { cuerpo_codigo;}, la variable que está en la condición de permanencia debe tener un valor YA, es decir, su valor debe conocerse. Eso me obliga a que se introduzca por teclado tipoPieza previamente.
- Si el tipoPieza cumple que es != 0, permanecerá en el bucle.
- ¿Qué ocurre si la 1ª vez que se introduce tipoPieza es 0 **(tipoPieza == 0)**?
  Respuesta: NO entra en el bucle **while(tipoPieza != 0){….}**. Se deduce que:
  a) Puede ser que while se ejecute 0 veces (mínimo). El cliente se va sin hacer pedido alguno.
  b) **La condición de permanencia (tipoPieza != 0) es opuesta a la condición de salida: (tipoPieza == 0)**
     1. También se podría poner como condición de permanencia: **(!(tipoPieza == 0))**.
     2. También como condición permanencia: **(tipoPieza == 1 || tipoPieza ==2)**
  c) Dentro del código del bucle **se vuelve a introducir el valor de tipoPieza** -justo al final del cuerpo de código-.

DISEÑO DEL EJERCICIO nº 4 CON DO-WHILE

```
int tipoPieza;
//Ejercicio con do-while, realizar código mientras que el tipo de pieza no sea 0, si el tipo es 0 sale del bucle
do{
   //Obligo a que tipoPieza sea 0,1 ó 2
   tipoPieza=Introducir.enteroPositivo("Dime el tipo de la pieza que desee: (1 o 2), con 0 finaliza el pedido");
   while(tipoPieza > 2){
       System.out.println("Error, tipo debe ser 0, 1 ó 2");
       tipoPieza=Introducir.enteroPositivo("Dime el tipo de la pieza que desee: (1 o 2), con 0 finaliza el pedido");
    }

    if (tipoPieza == 0) break;

    String nombrePieza=Introducir.cadena("Dime el nombre de la pieza: ");
    double precioPieza=Introducir.realDoblePositivo("Dime el precio de la pieza: ");

    totalPago += Introducir.calculo(tipoPieza, precioPieza);

}while(tipoPieza != 0);
```

EXPLICACIÓN

1. No conozco el valor del tipo de la pieza, por lo que, se ejecuta el bucle.
2. Como puede ser que sea == 0, sale del bucle. **Condición de salida (tipoPieza == 0)**
   -Si (tipoPieza != 0) continúa en el bucle y se desarrolla el cuerpo del bucle

CONCLUSIÓN

- Es más fácil la comprensión y diseño del ejercicio.
- Con **do**{ cuerpo_codigo;}**while(condicion_permanencia_bucle);**, la variable que está en la condición de permanencia tendrá valor que se habrá introducido por teclado previamente.
- Si el tipoPieza cumple que es != 0, permanecerá en el bucle.
- ¿Qué ocurre si la 1ª vez que se introduce tipoPieza es 0 **(tipoPieza == 0)**?
  Respuesta:
    a) Puede ser que do-while se ejecute 1 veces (mínimo). El cliente se va sin hacer pedido alguno.
    b) **La condición de salida es (tipoPieza == 0) es opuesta a la condición de permanencia: (tipoPieza != 0)**
        1. También se podría poner como condición de salida: **(!(tipoPieza == 1 || tipoPieza ==2))**.
        2. También como condición de salida: **(tipoPieza != 1 && tipoPieza !=2))**