

Apellidos y nombre: _____

ID: _____

IMPORTANTE

- Poner cada ejercicio en un paquete separado y llamarlos ejer1 y ejer2.
- Comprimir las dos carpetas en un único archivo .zip y entregar el archivo comprimido en Moodle Centros.

Calificación del examen

- C0. La calificación máxima del examen es de 7 puntos, y en cada ejercicio se parte de la puntuación (P) asociada al mismo.
- C1. Por cada test no superado: se restan $(2P)/(3T)$ puntos, donde T = cantidad de tests del ejercicio.
- C2. Por cada test superado por un algoritmo *ad-hoc*¹: se restan P/T puntos.
- C3. Por cada algoritmo incorrecto utilizado: se restan $P/2$ puntos.
- C4. Por cada detalle algorítmico innecesariamente complicado o ineficiente: se restan $P/4$ puntos.

1. Barras ($P = 3$ puntos)

Programar en Java un método estático llamado `barras` en una clase llamada `Barras`, que reciba como argumentos una lista (de tipo estático `List<Integer>`) de números enteros y un carácter, y que devuelva una cadena formada por barras horizontales separadas por un salto de línea (`\n`). Las barras se crearán con el carácter indicado como segundo argumento. Cada barra deberá tener tantos caracteres como indique el número correspondiente en la lista.

Por ejemplo:

```
Barras.barras(Arrays.asList(1, 3, 4), '*') => "*\n***\n****"
*
***
****

Barras.barras(Arrays.asList(6, 2, 15, 3), '=') => "=====\n==\n=====
=====
====="
```

```
Barras.barras(Arrays.asList(1, 10), '+') => "+\n+++++++"
+
+++++++
```

Indicación: El método `Arrays.asList` devuelve una lista con los elementos que recibe como argumentos. Es simplemente una forma rápida de crear una lista inicializada.


Tests:

```
Barras.barras(Arrays.asList(1, 3, 4), '*').equals("*\n**\n****") == true
Barras.barras(Arrays.asList(6, 2, 15, 3), '=').equals("=====\n==\n=====\\n====") == true
Barras.barras(Arrays.asList(1, 10), '+').equals("+\n+++++++") == true
```

2. Carrito de la compra ($P = 4$ puntos)

Programar en Java un modelo orientado a objetos del funcionamiento del carrito de la compra de una tienda online.

- Cada usuario de la tienda online tiene su propio carrito (lo crea el propio usuario cuando se crea éste).
- El usuario puede:
 - Añadir productos al carrito mientras consulta el catálogo de la tienda: cada vez que añade un producto al carrito, se incrementa en 1 la cantidad de ese producto en el carrito (si el producto no estaba inicialmente en el carrito, se empieza con una cantidad de 1).
 - Reducir en 1 las unidades de un producto que ya esté en el carrito: si sólo había una unidad de ese producto en el carrito, se retira el producto del carrito.
 - Vaciar el carrito.
- Por tanto, cada usuario lleva asociado un carrito donde se irán almacenando los productos añadidos y la cantidad de cada uno de ellos. Para ello, las parejas «producto – cantidad» que forman las entradas del carrito se pueden guardar en un Map (implementado con un HashMap, por ejemplo).
 - El carrito debe disponer de un método `getEntradas` que devuelva el mapa anterior.
- Los productos tienen un código, una denominación y un precio. Además, pertenecen a una categoría.
- Los usuarios pueden consultar todos los productos que pertenecen a una categoría en concreto (para ello, las categorías disponen del método `getProductos`, que devuelve una lista de tipo `List<Producto>` con todos los productos de esa categoría).

 	EXAMEN PRÁCTICO TERCERA EVALUACIÓN (2020/21)	PROGRAMACIÓN – 1.º DAW	
		24-may-2021	Pág. 3 de 3

Tests: Partiendo de:

```

Usuario usuario = new Usuario();
Carrito carrito = usuario.getCarrito();
Categoria tornilleria = new Categoria("Tornillería");
Categoria ferreteria = new Categoria("Ferretería");
Producto tuerca = new Producto(25, "Tuerca", 12.50, tornilleria);
Producto grifo = new Producto(58, "Grifo", 6.30, ferreteria);
carrito.anyadirProducto(tuerca);
carrito.anyadirProducto(tuerca);
Map<Producto,Integer> entradas = carrito.getEntradas();

```

los tests son:

```

entradas.get(tuerca) == 2
entradas.get(grifo) == null
ferreteria.getProductos().contains(tuerca) == false
ferreteria.getProductos().contains(grifo) == true

```

Nota

¹Un algoritmo *ad-hoc* es aquel que está diseñado maliciosamente para superar un test, devolviendo directamente la salida esperada por el test ante los valores de entrada adecuados. Por ejemplo, una función que debería calcular la suma de dos números pero que lo único que hace es devolver directamente 7 ante los argumentos 4 y 3 para superar el test `suma(4, 3) == 7`.