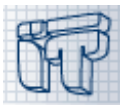


NOMBRE:
APELLIDOS:
NIA:
GRUPO:

Primer parcial

2ª Parte: Problemas (7 puntos sobre 10)

Duración: 70 minutos
Puntuación máxima: 7 puntos
Fecha: 24 de marzo de 2021

Instrucciones para el examen:

- No se permite el uso de libros o apuntes, ni tener teléfonos móviles u otros dispositivos electrónicos encendidos. Incumplir cualquiera de estas normas puede ser motivo de expulsión inmediata del examen.

PROBLEMA 1 (2,5 puntos)

Un taller de ruedas de vehículos necesita gestionar los neumáticos que puede montar en los vehículos de sus clientes. Para ello necesita de un programa en Java que maneje la información que necesita. De los neumáticos interesa conocer la marca (`brand`), la altura (`height`), la anchura (`width`), el índice de carga (`loadIndex`) y la eficiencia del neumático en cuanto al consumo de combustible (`fuelEfficiency`). La marca será una cadena de caracteres. La altura, la anchura y el índice de carga serán números enteros. La eficiencia vendrá dada por un carácter de la "A" a la "E", siendo la "A" la categoría con máxima eficiencia. Se pide:

Apartado 1 (0,5 puntos)

Declare los atributos de la clase neumático (*Tyre*). Para representar la eficiencia de los neumáticos declare una constante que represente a una de las categorías. Es suficiente con que declare una única constante a su elección a modo de ejemplo, no es necesario que declare las 5 constantes.

Apartado 2 (0,75 puntos)

Codifique un constructor con todos los atributos y los métodos get y set únicamente del atributo eficiencia de combustible, teniendo en cuenta que se debe validar que el valor de la categoría es un valor válido y de no serlo se debe lanzar una excepción.

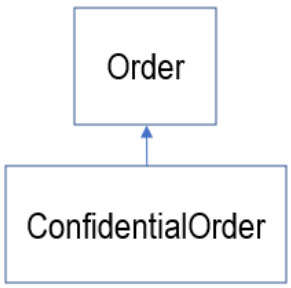
Apartado 3 (0,75 puntos)

Codifique el método *public char maximumEfficiency(Tyre store[])*, que recibirá un array de objetos *Tyre* y devolverá el carácter correspondiente a la categoría de mayor eficiencia existente entre todos los objetos *Tyre* contenidos en el array.

Apartado 4 (0,5 puntos)

Crear un método de testing que compruebe que el método set para el atributo eficiencia de combustible lanza y se captura la excepción.

Problema 2 (3 / 7 puntos)

El proyecto de gestión del almacén en el que ha estado trabajando ha decidido crear un nuevo tipo de pedidos (ConfidentialOrder) para garantizar la confidencialidad de los pedidos de determinados productos. Estos pedidos (ConfidentialOrder) tendrán las propiedades y el comportamiento de cualquier pedido normal (Order), pero necesita además guardar un código que en ningún caso podrá ser consultado por el resto de objetos. Dicho código será una cadena de caracteres. El valor de dicho código se establecerá en el momento de instanciar el objeto, y no podrá ser modificado ni consultado por otros objetos.	

Apartado-1. ConfidentialOrder. Declaración de clase y Atributos (1 punto)
Suponiendo que ya está declarada la clase Order y su constructor public Order(ArrayList<StockableProduct> list, Person client, Person manager) se pide:

- Declare la clase ConfidentialOrder con los atributos necesarios según la especificación anterior y codifique un constructor con todos los atributos, y los métodos get y set que considere necesarios.

Apartado-2. ConfidentialOrder. Método calculateTotalBenefit (1 punto)
Suponiendo que ya está creada la clase Order y su método public double calculateTotalBenefit(). Se pide

- Codificar el método public double calculateTotalBenefit(String code) teniendo en cuenta que para calcular el beneficio total del pedido se deberá aportar un código. Si el código no coincide con el del objeto el beneficio será -1. Si coincide el beneficio se computará de la misma manera que se hace con el resto de pedidos.

Apartado 3. BadCodeException (1 punto)
Modifique el método anterior para que si los códigos no coinciden se lance una excepción. Dicha excepción será una nueva clase llamada BadCodeException que debe declararse y codificar su método constructor.

SOLUCIÓN Y RÚBRICA

Ejercicio 1

Apartado 1 (0,5 puntos)

```
public class Tyre {

        public static final char A = 'A';
        public static final char B = 'B';
        public static final char C = 'C';
        public static final char D = 'D';
        public static final char E = 'E';

        private String brand;
        private int width;
        private int height;
        private int loadIndex;
        private char fuelEfficiency;
}
```

Rúbrica:

- 0 si no tiene sentido o no pone public class Tyre
- 0,15 por tener bien la constante (si falta public o static máximo 0,1)
- 0,35 por tener bien los atributos (si alguno mal máximo 0,1. 0 si alguno no es private).

Apartado 2 (0,75 puntos)

```
public Tyre(String brand, int width, int height, int loadIndex,
                        char fuelEfficiency) {
                try {
                        setBrand(brand);
                        setfuelEfficiency(fuelEfficiency);
                } catch (Exception e) {

                        e.printStackTrace();
                }
                this.width = width;
                this.height = height;
                this.loadIndex = loadIndex;

        }
public char getfuelEfficiency() {
                return fuelEfficiency;
        }

        public void setfuelEfficiency(char fuelEfficiency) throws Exception {

                switch (fuelEfficiency) {
                case A:
                case B:
                case C:
                case D:
                case E:
                        this.fuelEfficiency = fuelEfficiency;
                        break;
                default:
                        throw new Exception("Invalid efficiency category");
                }

        }
```

Rúbrica

- 0 si no tiene sentido.
- 0,1 getter correcto.
- 0,35 constructor correcto (0,2 por capturar la excepción al hacer el setfuelEfficiency o poner throws Exception).
- 0,3 por tener bien el setter (0,2 por lanzar correctamente la excepción).

Apartado 3 (0,75 puntos)

```
public static char maximumEfficiency(Tyre store[]) {

		char max = 'F';
		char current;

		for (int i = 0; i < store.length; i++) {
			current = store[i].getfuelEfficiency();
			if (current < max)
				max = current;
		}

		return max;
	}
```

Rúbrica

- 0 si no tiene sentido.
- 0,3 por recorrer correctamente.
- 0,35 por acceder correctamente a cada posición y comparar con el máximo.
- 0,1 por devolver el valor correcto.

Apartado 4 (0,5 puntos)

```
@Test
public void testSetBrandWithException() {


		try {
			Tyre tyre = new Tyre();
			tyre.setfuelEfficiency("");
			fail("Failed test");
			}catch(Exception e) {}


		//OR JUnit5: assertThrows(Exception.class, ()->{Tyre tyre = new Tyre(); tyre.setfuelEfficiency("");});

	}
```

Rúbrica

- 0 si no tiene sentido.
- 0,1 por poner la anotación.
- 0,1 por crear el objeto Tyre por acceder
- 0,1 Invocar el método setfuelEfficiency con un valor que provoque excepción.
- 0,2 por capturar la Excepción o llamar a assertThrows correctamente.

Ejercicio 2

Apartado 1 (1 punto)

```
public class ConfidentialOrder extends Order {

	private final String code;

	public ConfidentialOrder(ArrayList<StockableProduct> list, Person client,
			Person manager, String code) {
		super(list, client, manager);
		this.code = code;
	}

	private String getCode() {
		return code;
	}
```

Rúbrica

- 0 si no tiene sentido.
- 0,1 por declarar la clase correctamente heredando de Order.
- 0,3 por declarar el atributo correctamente (si no pone final 0,1).
- 0,1 por declarar el constructor correctamente
- 0,2 por llamar a super dentro del constructor.
- 0,1 por asignar el resto de atributos
- 0,2 por crear el método get (y sólo ese. Si pone el set, entonces 0).

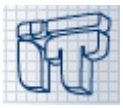
Apartado 2 (1 punto)

```
public double calculateTotalBenefit(String code) {

		if (code.equals(this.getCode()))
			return -1;
		else
			return super.calculatePrice();

	}

}
```



Rúbrica

- 0 si no tiene sentido.
- 0,5 por comparar con el String utilizando equals
- 0,5 por devolver el valor utilizando super()

Apartado 3 (1 punto)

```
public class BadCodeException extends Exception {
   public BadCodeException(String msg) {
      super(msg);
   }
}
```

```
public double calculateTotalBenefit(String code) throws BadCodeException {

                if (code.equals(this.getCode()))
                        throw new BadCodeException("Invalid code");
                else
                        return super.calculateTotalBenefit();

        }

}
```

Rúbrica

- 0 si no tiene sentido.
- 0,25 por declarar correctamente la clase Excepción heredando de Exception
- 0,25 por crear el constructor llamando a super() en la clase BadProductException
- 0,25 por modificar correctamente la firma del método.
- 0,25 por lanzar correctamente la excepción.