NOMBRE:
APELLIDOS:
NIA:
GRUPO:

Primer parcial

2ª Parte: Problemas (7 puntos sobre 10)

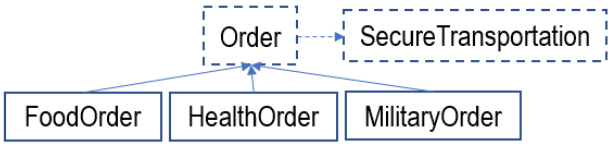
Duración: 70 minutos
Puntuación máxima: 7 puntos
Fecha: 15 de marzo de 2022

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.

Ejercicio 1. Proyecto (3/ 7 puntos)

El programa de gestión de almacenes que has creado para tu proyecto se ha ampliado para permitir que los pedidos (`Order`) puedan ser transportados con diferentes condiciones de seguridad en función de su contenido. Para ello la clase `Order` se ha modificado para implementar la interfaz `SecureTransportation` que permite modelar 4 tipos de seguridad en el transporte de los pedidos.



Se pide:

Interfaz SecureTransportation (0,6 puntos)

- Declara la interfaz `SecureTransportation` que permite modelar mediante constantes de tipo entero 4 tipos de seguridad para los pedidos: `0-NO_EXTRA_SECURITY` (cuando el pedido no requiere ninguna medida extra de seguridad), `1-FOOD_SECURITY` (cuando necesita medidas de seguridad alimentaria), `2-HEALTH_SECURITY` (cuando el pedido necesita medidas de seguridad sanitaria) y `3-MILITARY_SECURITY` (cuando el pedido necesita seguridad militar)
- La interfaz incluye dos métodos `requieresSecurityMeasures()` para indicar si son necesarias o no medidas extra de seguridad en un pedido y `addSecurityMeasures()` que añade las medidas de seguridad necesaria cuando se piden.

Clase Order (1,6 puntos)

Modifica la clase `Order` de la figura teniendo en cuenta lo siguiente:

```
public class Order{
    private static int numOrders;
    private int orderID;
}
```

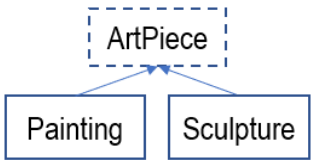
- implementa la interfaz `SecureTransportation` y tiene información suficiente para implementar el método `requiresSecurityMeasures()` pero deja sin implementar el método `addSecurityMeasures`() que será implementado por sus clases hijas.
- Declara los atributos de la clase Order teniendo en cuenta que:
    - contiene un contador `numOrders` que se incrementa automáticamente cada vez que se crea un nuevo pedido.
    - contiene un identificador único llamado `orderID` que se asigna automáticamente y representa el número de pedidos que había en el momento de su creación.
    - contiene un nuevo atributo de tipo entero llamado `securityType`
- Implementa el método `setSecurityType` que permite comprobar si el tipo de seguridad es uno de los valores indicados por las constantes de la interfaz y en caso contrario imprima un mensaje de error.
- Crea un constructor de la clase `Order` que reciba únicamente un entero que representa el tipo de securidad (securityType) y le asigne valor haciendo las comprobaciones necesarias. Al resto de los atributos se les puede asignar valor directamente sin recibir parámetros de entrada.

Clase FoodOrder (0,8 puntos)

- Declara la clase `FoodOrder` teniendo en cuenta que:
- Debe tener un constructor sin parámetros ya que se sabe que el tipo de seguridad correspondiente a los pedidos alimentarios es `FOOD_SECURITY`.
- Debe implementar el método `addSecurityMeasures`(...) que simplemente imprime por pantalla un mensaje indicando las medidas aplicadas con este mensaje "`adding food safety`".

Ejercicio 2. Orientación a objetos (2/ 7 puntos)

Se va a realizar una programa para una galería de arte (`ArtGallery`) que incluye como piezas (`ArtPiece`) tanto pinturas (`Painting`) como esculturas (`Sculpture`) y para realizarlo deberemos modelar estos 4 tipos de objetos.



Se pide:

Clase ArtPiece (1,5)

- Declara la clase `ArtPiece` que contendrá algunos métodos con código y otros que tendrán que ser implementados en sus clases hijas.
- La clase `ArtPiece` permite modelar todos los atributos comunes a las pinturas y esculturas como son: una cadena de caracteres para representar el nombre de la obra (`name`), otra para el autor (`author`), un número decimal para su precio (`price`) y un array (`dimensions`) con tres números decimales que representan respectivamente la altura (`height`), anchura (`width`) y profundidad (`depth`) de la pieza. Todos estos atributos son visibles sólo desde la propia clase.

- Declara un constructor para la clase que reciba los valores necesarios para dar valor a todos sus atributos. Asume que no puedes utilizar ningún método set/get para estos atributos dado que dichos métodos no existen y no se pide que se implementen.
- Añade un método `toString()` que permita indicar las características de la obra en el siguiente formato.
- Añade un método `print()` que permitirá imprimir la obra pero no se puede implementar en esta clase ya que las pinturas se imprimirán en 2D y las esculturas en 3D.

Clase Painting (0,5)

- Declara la clase `Painting` que es una especialización de la clase `ArtPiece`.
- Recibe los mismos atributos que la clase padre.
- Implementa el método `print()` (lo hemos simplificado para que simplemente imprima un mensaje indicando "`printing in 2D`" junto con la información de los atributos en el mismo formato que utiliza el método `toString()`.

Ejercicio 3. Testing (2/ 7 puntos)

Dada la clase `ArtGallery`

```
public class ArtGallery {
    ArrayList<ArtPiece> list;
    public ArtGallery(ArrayList<ArtPiece> list) {
        this.list = list;
    }

    public double totalValue() {
        double totalValue=0;
        for (int i = 0; i < list.size(); i++) {
            totalValue = totalValue + list.get(i).getPrice();
        }
        return totalValue;
    }
}
```

Se pide:

- Implementar un test unitario `ArtGalleryTest` para probar el método `totalValue`. Para hacerlo:
  - Crea un array con las dimensiones de la pintura y otro con las dimensiones de la escultura.
  - Crea un objeto de tipo pintura y otro de tipo escultura.
  - Crea un `ArrayList` que permita añadir ambos objetos.
  - Crea una galería de arte con ese `ArrayList`.
- Crea un test que invoque al método `totalValue` para comprobar el precio de ambos artículos .
- Indica qué cobertura de métodos de la clase `ArtGallery` se consigue al ejecutar el test indicado.

SOLUCIÓN DE REFERENCIA (Varias soluciones son posibles)

Ejercicio 1. Proyecto (3/ 7 puntos)

- Interfaz SecureTransportation (0,6):
  - 0,2 Declaración interfaz
  - 0,2 Declaración constantes
  - 0,2 Declaración método abstracto

```
public interface SecureTransportation {
    public static int NO_EXTRA_SECURITY = 0;
    public static int FOOD_SECURITY = 1;
    public static int HEALTH_SECURITY = 2;
    public static int MILITARY_SECURITY = 3;
    public abstract boolean requiresSecurityMeasures();
    public abstract void addSecurityMeasures();
}
```

- Clase Order (1,6 puntos)
  - 0,4 declaración (abstract + implements)
  - 0,2 atributo static
  - 0,1 atributos no static
  - 0,4 constructor
    - declaración (0 si no usa parámetros correctos)
    - asignación numOrders
    - asignación orderID
    - asignación setSecurityType (0 si no usa set)
  - 0,5 setSecurityType
    - declaración
    - comparación
    - concatenación de condiciones
    - asignación
    - mensaje de error

```
public abstract class Order implements SecureTransportation {
    private static int numOrders;
    private int orderID;
    private int securityType;
    public Order(int securityType) {
        numOrders++;
        orderID = numOrders;
        setSecurityType(securityType);

    }
    public void setSecurityType(int securityType) {
        if (securityType == SecureTransportation.NO_EXTRA_SECURITY ||
            securityType == SecureTransportation.FOOD_SECURITY ||
            securityType == SecureTransportation.HEALTH_SECURITY ||
            securityType == SecureTransportation.MILITARY_SECURITY) {
            this.securityType = securityType;
        } else {
            System.out.println("incorrect securityType");
        }
    }
    public boolean requiresSecurityMeasures() {
         return (securityType!=0); //más breve
    }
}
```

- Clase FoodOrder (0,8 puntos)
  - 0,2 declaración
  - 0,4 constructor
    - declaración
    - llamada a super con el tipo correcto
  - 0,2 metodo addSecurityMeasures

```
public class FoodOrder extends Order{
    public FoodOrder() {
        super(Order.FOOD_SECURITY);
    }

    public void addSecurityMeasures() {
        System.out.println("adding food safety");
    }
}
```

Ejercicio 2. Orientación a objetos (2/ 7 puntos)

- Clase ArtPiece (1,5)
  - Declaración de la clase 0,2 (declaración + abstract)
  - atributos básicos 0,1 (0 si hay alguno mal)
  - atributo tipo array 0,1
  - Constructor : 0,4
    - Declaración: 0,2
    - Asignaciones sin usar set: 0,2 (0 si usa set porque se dice que no existen)

- Método dimensionToString(): 0,2 (declaración + contenido) o funcionalidad equivalente
- Método toString(): 0,3
  - Declaración
  - atributos básicos
  - componentes del atributo dimension
- Método print: 0,2 (declaración + abstract)

```
public abstract class ArtPiece {
    private String name;
    private String author;
    private double[] dimensions;
    private double price;
    public ArtPiece(String name, String author, double[] dimensions,
                    double price) {
        this.name = name;
        this.author = author;
        this.dimensions = dimensions;
        this.price = price;
    }

    private String dimensionToString() {
        return "height="+ dimensions[0] + ", width=" + dimensions[1] +
                ", depth=" + dimensions[2] + "\n";
    }
    public String toString() {
        return "name=" + name + ", author=" + author + "\n" +
                dimensionToString() +
                ", price=" + price + "\n";
    }
    public abstract void print();
    //set y get only for price
}
```

- Clase Painting (0,5)
  - declaración: 0,2
  - constructor: 0,2 (declaración + super)
  - print: 0,1

```
public class Painting extends ArtPiece{
    public Painting(String name, String author, double[] dimensions,
                    double price) {
        super(name, author, dimensions, price);
    }

    public void print() {
        System.out.println("printing in 2D" + toString());
    }
}
```

Ejercicio 3. Testing (2/ 7 puntos)

Test unitario: Clase ArtGalleryTest (1,5):

- @Test: 0,1
- declaración del método: 0,1
- Creación de objetos (0,7):
  - Creación de arrays con las dimensiones: 0,2
  - Creación de objetos Painting y Sculpture: 0,2
  - Creación del ArrayList: 0,2
  - Creación de ArtGallery a partir del ArrayList: 0,1
- Llamada al assertEquals: (0,6)
  - palabra reservada
  - resultado esperado
  - llamada al método

```
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
import java.util.ArrayList;
class ArtGalleryTest {

    @Test
    void testTotalValue() {
        double[] dimensionsP1 = {10, 20, 30};
        double[] dimesionsS1 = {80, 40, 40};
        Painting p1 = new Painting("Bodegon","desconocido",
                                   dimensionsP1, 100);
        Sculpture s1 = new Sculpture("David","Miguel Torres",
                                   dimesionsS1, 500);
        ArrayList<ArtPiece> list = new ArrayList<ArtPiece>();
        list.add(p1);
        list.add(s1);

        ArtGallery myGallery = new ArtGallery(list);
        assertEquals(600, myGallery.totalValue());
    }
}
```

Cobertura de métodos (0,5)

- Se cubre el 100% ya que se llama tanto al constructor como al método totalValue()