Programación

Act10_ArrayList y Excepciones

Francisco José García Cutillas | 1FPGS_DAM

Índice

Ejercicio A01. Perrera

- **Implementa una clase Perro con los atributos código, nombre, fecha de nacimiento y raza. El tipo Raza de Perro debe ser un enum con cinco o seis razas de perro (Indica las que más te gusten).**

```java
package clases;

import excepciones.FechaException;
import excepciones.NombreException;
import excepciones.RazaException;
import java.time.LocalDate;
import java.time.Period;
import java.time.format.DateTimeFormatter;
import java.time.format.DateTimeParseException;

/**
 * Clase Perro. Con ella podemos crear objetos de tipo Perro
 *
 * @author Fran
 */
public class Perro {

    public enum Raza {

        CANICHE, BODEGUERO, PITBULL, BOXER, MASTIN, LABRADOR;

    }

    private int codigo;
    private String nombre;
    private LocalDate fechaNac;
    private Raza razaPerro;
```

  - **La clase debe tener un constructor con todos los parámetros, pero la raza del perro debe ser un String. También deben existir los getters y setters, el método toString y el equals, además de un método imprimir() para mostrar los datos de un perro.**

```java
    /**
     * Constructor de la clase
     *
     * @param _codigo Código del perro
     * @param _nombre Nombre del perro
     * @param _fechaNac Fecha de nacimiento del perro del tipo dd/MM/yyyy
     * @param _razaPerro Raza del perro (Caniche, Bodeguero, Pitbull, Bóxer,
     * Mastin, Labrador)
     * @throws RazaException Si la raza no está entre las permitidas como
     * parámetro
     * @throws FechaException Si la fecha no es de tipo dd/MM/yyyy
     * @throws NombreException Si el nombre tiene menos de 3 caracteres
     */
    public Perro(int _codigo, String _nombre, String _fechaNac, String _razaPerro) throws RazaException, FechaException, NombreException {

        this.codigo = _codigo;
        setNombre(_nombre);
        setFechaNac(_fechaNac);
        setRazaPerro(_razaPerro);

    }
```

```java
    /**
     * Método para obtener el código del perro
     *
     * @return Código del perro de tipo int
     */
    public int getCodigo() {
        return codigo;
    }

    /**
     * Método para establecer un nuevo código al perro
     *
     * @param codigo Nuevo código de tipo int
     */
    public void setCodigo(int codigo) {
        this.codigo = codigo;
    }

    /**
     * Método para obtener el nombre del perro
     *
     * @return Nombre del perro de tipo String
     */
    public String getNombre() {
        return nombre;
    }

```

```java
    /**
     * Método que devuelve la fecha de nacimiento del perro
     *
     * @return Fecha de nacimiento del perro de tipo LocalDate
     */
    public LocalDate getFechaNac() {
        return fechaNac;
    }

    /**
     * Método que devuelve la raza del perro
     *
     * @return Raza del perro de tipo Raza
     */
    public Raza getRazaPerro() {
        return razaPerro;
    }
```

```java
    /**
     * Método toString() de la clase
     *
     * @return Información del perro en forma de cadena
     */
    @Override
    public String toString() {
        return "Código: " + this.codigo + ", Nombre: " + this.nombre + ", Fecha de nacimiento: "
                + fechaNac.getDayOfMonth() + "/" + fechaNac.getMonthValue() + "/" + fechaNac.getYear()
                + ", Raza: " + this.razaPerro;
    }

    /**
     * Método equals de la clase
     *
     * @param obj Objeto a comparar
     * @return True si dos perros tienen el mismo código, False en caso
     * contrario
     */
    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final Perro other = (Perro) obj;
        return this.codigo == other.codigo;
    }
```

```java
    /**
     * Método que imprime por pantalla la información del perro
     */
    public void imprimir() {

        System.out.println(x: "*** Información del perro ***");
        System.out.println("Código: " + this.codigo);
        System.out.println("Nombre: " + this.nombre);
        System.out.println("Fecha de nacimiento: " + fechaNac.getDayOfMonth() + "/" + fechaNac.getMonthValue() + "/" + fechaNac.getYear());
        System.out.println("Raza: " + this.razaPerro);

    }
```

- **El nombre de un perro debe tener al menos 3 caracteres y la fecha de nacimiento no puede ser posterior a la fecha actual (no se mete a mano, si no que se debe calcular la fecha actual). También debemos vigilar que se inserta la raza adecuada. Por ello, tanto en el constructor como en los setters, deberás comprobar que los valores sean válidos y lanzar una Excepción si no lo son. Estas Excepciones deben ser definidas por ti, con unos nombres apropiados.**

```java
    /**
     * Método para establecer un nuevo nombre al perro
     *
     * @param _nombre Nombre de tipo String
     * @throws NombreException Si el nombre tiene menos de 3 caracteres
     */
    public void setNombre(String _nombre) throws NombreException {

        if (_nombre.length() >= 3) {

            this.nombre = _nombre;

        } else {

            throw new NombreException( message: "El nombre debe de tener al menos 3 caracteres");

        }

    }

    /**
     * Método para establecer una nueva fecha de nacimiento al perro
     *
     * @param _fechaNac Nueva fecha de nacimiento en tipo String con formato
     * dd/MM/yyyy
     * @throws FechaException Si la fecha introducida no es del tipo dd/MM/yyyy
     */
    public void setFechaNac(String _fechaNac) throws FechaException {

        try {

            LocalDate fechaRecibida = fechaEsp( fecha: _fechaNac);

            if (fechaRecibida.isBefore( other: LocalDate.now())) {

                this.fechaNac = fechaRecibida;

            } else {

                throw new FechaException( message: "Error. La fecha de nacimiento no puede ser posterior a la actual");

            }

        } catch (DateTimeParseException fechaEx) {

            throw new FechaException( message: "Error. El formato de la fecha debe ser dd/MM/yyyy");

        }

    }
```

```java
    /**
     * Método para establecer una nueva raza al perro
     *
     * @param _razaPerro Nueva raza de tipo String
     * @throws RazaException Si la raza no es correcta (Caniche, Bodeguero,
     * Pitbull, Bóxer, Mastín, Labrador)
     */
    public void setRazaPerro(String _razaPerro) throws RazaException {

        try {

            this.razaPerro = Raza.valueOf( name: convierteCadena( entrada: _razaPerro));

        } catch (IllegalArgumentException ex) {

            throw new RazaException( _raza: _razaPerro);

        }
    }
```

- **Crea un método que devuelva la edad que tiene el perro.**

```java
    /**
     * Método que devuelve la edad del perro en años
     *
     * @return Edad del perro en años de tipo int
     */
    public int devuelveEdad() {

        Period diferencia = Period.between( startDateInclusive: fechaNac,  endDateExclusive: LocalDate.now());
        int edad = diferencia.getYears();
        return edad;

    }
```

- **Métodos auxiliares utilizados en la clase Perro.**

```java
    /**
     * Método privado de la clase que convierte una cadena a Mayúsculas y si
     * lleva tildes se las quita
     *
     * @param entrada String a convertir
     * @return String en mayúsculas sin tildes
     */
    private String convierteCadena(String entrada) {

        String cadena = entrada.toUpperCase();

        String cadenaUtil = cadena.replace( target: "Á",  replacement: "A").replace( target: "É",  replacement: "E").replace( target: "Í",  replacement: "I")
                .replace( target: "Ó",  replacement: "O").replace( target: "Ú",  replacement: "U");

        return cadenaUtil;

    }

    /**
     * Método privado de la clase que convierte una fecha tipo String en una
     * tipo LocalDate
     *
     * @param fecha Fecha de tipo String formato dd/MM/yyyy
     * @return Fecha tipo LocalDate formato yyyy-MM-dd
     */
    private LocalDate fechaEsp(String fecha) {

        DateTimeFormatter fechaCadena = DateTimeFormatter.ofPattern( pattern: "dd/MM/yyyy");

        return LocalDate.parse( text: fecha,  formatter: fechaCadena);

    }
```

- **Clases creadas para las excepciones de la clase Perro.**

```java
package excepciones;

/**
 * Clase exception de fecha. Usada cuando una fecha se introduce de forma incorrecta
 * @author Fran
 */
public class FechaException extends Exception {

    /**
     * Constructor de la clase
     * @param message Mensaje que queremos mostrar cuando se procuzca la excepción
     */
    public FechaException(String message) {
        super(message);
    }

}
```

```java
package excepciones;

/**
 * Clase exception de nombre. Usada cuando un nombre es incorrecto
 * @author Fran
 */
public class NombreException extends Exception {

    /**
     * Constructor de la clase
     * @param message Mensaje que queremos mostrar cuando se procuzca la excepción
     */
    public NombreException(String message) {
        super(message);
    }

}
```

```java
package excepciones;

/**
 * Clase exception de perrera. Usada cuando una perrera tiene un error
 * @author Fran
 */
public class PerreraException extends Exception {

    /**
     * Constructor de la clase
     * @param message Mensaje que queremos mostrar cuando se procuzca la excepción
     */
    public PerreraException(String message) {
        super(message);
    }

}
```

```java
package excepciones;

/**
 * Clase exception de raza. Usada cuando una raza no es correcta
 * @author Fran
 */
public class RazaException extends IllegalArgumentException {

    private String raza;

    /**
     * Constructor de la clase
     * @param _raza raza a la que hacemos referencia en la excepción
     */
    public RazaException(String _raza) {

        this.raza = _raza;

    }

    /**
     * Método que devuelve el mensaje de error
     * @return Mensaje de error cuando una raza no es correcta
     */
    @Override
    public String getMessage() {

        return "La raza " + this.raza + " no es válida";

    }

}
```

- **Luego, haz una clase principal con main para hacer pruebas: instancia varios objetos Perro y utiliza sus métodos, probando distintos valores (algunos válidos y otros incorrectos).**

```java
package com.mycompany.garcia_cutillas_franciscojose_actut10_perrera;

import clases.Perrera;
import clases.Perro;
import excepciones.FechaException;
import excepciones.NombreException;
import excepciones.PerreraException;
import excepciones.RazaException;
import java.util.Collections;
import ordenacion.ComparaPerros;

/**
 *
 * @author Fran
 */
public class Garcia_Cutillas_FranciscoJose_ActUt10_Perrera {

    public static void main(String[] args) {

        Perrera pe1 = new Perrera( _nombre: "El Olvido", _direccion: "Calle Mayor, 3");

        try {

            Perro p1 = new Perro( _codigo: 1, _nombre: "Firulais", _fechaNac: "10/05/2023", _razaPerro: "bóxer");

            Perro p2 = new Perro( _codigo: 2, _nombre: "Boo", _fechaNac: "21/05/2021", _razaPerro: "Pitbull");

            Perro p3 = new Perro( _codigo: 3, _nombre: "Tobi", _fechaNac: "02/09/2019", _razaPerro: "Labrador");

            Perro p4 = new Perro( _codigo: 4, _nombre: "Beethoben", _fechaNac: "30/07/2014", _razaPerro: "Mastín");

            Perro p5 = new Perro( _codigo: 5, _nombre: "Tobi", _fechaNac: "16/02/2022", _razaPerro: "Caniche");

            Perro p6 = new Perro( _codigo: 6, _nombre: "Jackie", _fechaNac: "20/09/2020", _razaPerro: "Bodeguero");

            System.out.println("¿Es Perro p2 igual a p1?: " + p2.equals( obj: p1));
            System.out.println("¿Es Perro p5 igual a p5?: " + p5.equals( obj: p5));

            System.out.println( x: "\nInformación del perro p2");
            p2.imprimir();

            System.out.println("\nEdad del perro p4: " + p4.devuelveEdad() + " Años");
            System.out.println("Edad del perro p6: " + p6.devuelveEdad() + " Años");

        } catch (RazaException rEx) {

            System.out.println( x: rEx.getMessage());

        } catch (FechaException fEx) {

            System.out.println( x: fEx.getMessage());

        } catch (NombreException nEx) {

            System.out.println( x: nEx.getMessage());
```

```
¿Es Perro p2 igual a p1?: false
¿Es Perro p5 igual a p5?: true

Información del perro p2
*** Información del perro ***
Código: 2
Nombre: Boo
Fecha de nacimiento: 21/5/2021
Raza: PITBULL

Edad del perro p4: 8 Años
Edad del perro p6: 2 Años
```

```java
Perro p1 = new Perro( _codigo: 1, _nombre: "Firulais", _fechaNac: "10/05/2023", _razaPerro: "Pastor Alemán");
```

```
--- exec-maven-plugin:3.0.0:exec (default-cli) @ Garcia_Cutilla
La raza Pastor Alemán no es válida
------------------------------------------------------------------------
```

```
Perro p1 = new Perro( _codigo: 1,  _nombre: "Firulais",  _fechaNac: "16/05/2023",  _razaPerro: "bóxer");
```

```
--- exec-maven-plugin:3.0.0:exec (default-cli) @ Garcia_Cutillas_FranciscoJose_ActUt10_Perrera ---
Error. La fecha de nacimiento no puede ser posterior a la actual
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  0.330 s
Finished at: 2023-05-15T19:22:06+02:00
```

```
Perro p2 = new Perro( _codigo: 2,  _nombre: "Bo",  _fechaNac: "21/05/2021",  _razaPerro: "Pitbull");
```

```
--- exec-maven-plugin:3.0.0:exec (default-cli) @ Garcia_Cutillas_Francis
El nombre debe de tener al menos 3 caracteres
------------------------------------------------------------------------
```

- **Implementa una clase Perrera que deberá tener un nombre y una dirección. Una Perrera se considera igual a otro si tiene el mismo nombre y dirección. Además, deberá tener un ArrayList para almacenar los Perros que tiene.**

```java
package clases;

import excepciones.PerreraException;
import java.util.ArrayList;
import java.util.Objects;

/**
 * Clase Perrera. Con ella podemos crear objetos de tipo Perrera
 * @author Fran
 */
public class Perrera {

    private String nombre;
    private String direccion;
    private ArrayList<Perro> perros;
    private final int maxPerrera = 5;

    /**
     * Constructor de la clase
     * @param _nombre Nombre de la perrera de tipo String
     * @param _direccion Dirección de la perrera de tipo String
     */
    public Perrera(String _nombre, String _direccion) {

        this.nombre = _nombre;
        this.direccion = _direccion;
        this.perros = new ArrayList<>();

    }
```

```java
/**
 * Método toString de la clase
 * @return Información de la perrera en una cadena String
 */
@Override
public String toString() {
    return "Nombre de la perrera: " + this.nombre + ", Dirección: " + this.direccion + ", Perros en la perrera: " + this.perros.toString();
}

/**
 * Método que muestra la información por pantalla de la perrera
 */
public void mostrarInfo(){

    System.out.println(x:"*** Información de la perrera ***");
    System.out.println("Nombre: " + this.nombre);
    System.out.println("Dirección: " + this.direccion);
    System.out.println(x:"");
    System.out.println(x:"-Perros de la perrera-");
    for(Perro p: this.perros){

        System.out.println(x:p);

    }


}
```

```java
    /**
     * Método para obtener el nombre de la perrera
     *
     * @return Nombre de la perrera de tipo String
     */
    public String getNombre() {
        return nombre;
    }

    /**
     * Método para establecer un nuevo nombre a la perrera
     *
     * @param nombre Nuevo nombre de la perrera de tipo String
     */
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    /**
     * Método para obtener la dirección de la perrera
     *
     * @return Dirección de la perrera de tipo String
     */
    public String getDireccion() {
        return direccion;
    }

    /**
     * Método para establecer una nueva dirección a la perrera
     *
     * @param direccion Nueva dirección de la perrera de tipo String
     */
    public void setDireccion(String direccion) {
        this.direccion = direccion;
    }

    /**
     * Método para obtener los perros que hay en la perrera
     *
     * @return Perros de la perrera de tipo ArrayList
     */
    public ArrayList getPerros() {
        return perros;
    }

```

```java
    /**
     * Método equals de la clase. Compara dos perreras
     *
     * @param obj Perrera a comparar
     * @return True si dos perreras tienen el mismo nombre y dirección. False en
     * caso contrario
     */
    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final Perrera other = (Perrera) obj;
        if (!Objects.equals( a: this.nombre,  b: other.nombre)) {
            return false;
        }
        return Objects.equals( a: this.direccion,  b: other.direccion);
    }
```

- **Una Perrera se crea siempre sin perros.**

```java
this.perros = new ArrayList<>();
```

- **Vamos a tener un método para insertar un perro en la perrera (siempre y cuando no se encuentre ya en ella) y para eliminarlo de la perrera (si existe).**
- **No es posible tener más de 5 perros en la perrera por lo que el método que inserta perros en la perrera deberá controlar este y en el caso de querer insertar alguno deberá lanzar una Excepción.**

```
131     /**
132      * Método para insertar nuevos perros a la perrera
133      *
134      * @param p Nuevo perro de tipo Perro
135      * @return True si lo ha insertado con éxito, False en caso contrario
136      * @throws PerreraException Si la perrera está llena
137      */
138     public boolean intertaPerro(Perro p) throws PerreraException {
139
140         if (this.perros.size() < maxPerrera) {
141
142             if (!this.perros.contains( o: p)) {
143
144                 this.perros.add( e: p);
145                 return true;
146
147             }
148
149         } else {
150
151             throw new PerreraException( message: "La perrera está llena");
152
153         }
154
155         return false;
156
157     }
158
159     /**
160      * Método para borrar perros de la perrera
161      *
162      * @param p Perro a eliminar
163      * @return True si ha sido posible eliminarlo, False en caso contrario
164      */
165     public boolean delPerro(Perro p) {
166
167         return this.perros.remove( o: p);
168
169     }
170
171 }
172
```

- **Es posible que se quiera ordenar los perros de la perrera con su nombre. A mismo nombre debe desempatar el código.**

```java
package ordenacion;

import clases.Perro;
import java.text.Collator;
import java.util.Comparator;

/**
 * Clase para comparar dos perros. Implementa la interfaz Comparator
 * @author Fran
 */
public class ComparaPerros implements Comparator {

    /**
     * Método compare utilizado para comparar dos perros
     * @param o1 Primer perro a comparar de tipo Perro
     * @param o2 Segundo perro a comparar de tipo Perro
     * @return Perros ordenados por nombre. A igual nombre desempata el código de cada uno
     */
    @Override
    public int compare(Object o1, Object o2) {

        if (o1 == o2) {

            return 0;

        }

        if (o2 == null) {

            return -1;

        }

        if (o1 == null) {

            return 1;

        }

        Perro p1 = (Perro) o1;
        Perro p2 = (Perro) o2;

        Collator col = Collator.getInstance();

        if (col.compare( source:p1.getNombre(), target:p2.getNombre()) == 0) {

            return p1.getCodigo() - p2.getCodigo();

        } else {

            return col.compare( source:p1.getNombre(), target:p2.getNombre());

        }

    }

}
```

- **Clase creada para las excepciones de la clase Perrera.**

```java
package excepciones;

/**
 * Clase exception de perrera. Usada cuando una perrera tiene un error
 * @author Fran
 */
public class PerreraException extends Exception {

    /**
     * Constructor de la clase
     * @param message Mensaje que queremos mostrar cuando se procuzca la excepción
     */
    public PerreraException(String message) {
        super(message);
    }

}
```

- **Pruebas con la clase Perrera**

```
            pel.intertaPerro( p: p3);
            pel.intertaPerro( p: p2);
            pel.intertaPerro( p: p1);
            pel.intertaPerro( p: p5);
            pel.intertaPerro( p: p4);

            // System.out.println(pel.intertaPerro(p6));
            System.out.println( x: pel);

            System.out.println("\nBorra Perro p2, ¿Es posible?: " + pel.delPerro( p: p2));

            System.out.println("Inserta perro p1, ¿Es posible?: " + pel.intertaPerro( p: p1));

            ComparaPerros cp = new ComparaPerros();
            Collections.sort( list: pel.getPerros(),  c: cp);
            System.out.println( x: "");
            pel.mostrarInfo();
```

```
        } catch (PerreraException perEx) {

            System.out.println( x: perEx.getMessage());

        }

    }

}
```

```
Nombre de la perrera: El Olvido, Dirección: Calle Mayor, 3, Perros en la perrera: [Código: 3, Nombre: Tobi, Fecha de nacimiento: 2/9/2019, Raza: LABRADOR, Código: 2,

Borra Perro p2, ¿Es posible?: true
Inserta perro p1, ¿Es posible?: false

*** Información de la perrera ***
Nombre: El Olvido
Dirección: Calle Mayor, 3

-Perros de la perrera-
Código: 4, Nombre: Beethoben, Fecha de nacimiento: 30/7/2014, Raza: MASTIN
Código: 1, Nombre: Firulais, Fecha de nacimiento: 10/5/2023, Raza: BOXER
Código: 3, Nombre: Tobi, Fecha de nacimiento: 2/9/2019, Raza: LABRADOR
Código: 5, Nombre: Tobi, Fecha de nacimiento: 16/2/2022, Raza: CANICHE
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  0.340 s
Finished at: 2023-05-15T19:36:28+02:00
```

- **Intentamos insertar un sexto perro**

```
System.out.println( x: pel.intertaPerro( p: p6));
```

```
La perrera está llena
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  0.349 s
Finished at: 2023-05-15T19:40:23+02:00
```

Ejercicio A02. Tienda

- **Crea una clase Artículo para representar la información general sobre los artículos que puede vender una tienda. De un Artículo vamos a almacenar un código que va a ser de tipo String, su nombre, material, color, precio (de tipo double y que representa los euros que vale en la actualidad ese Artículo) y stock. En esta clase vamos a generar los siguientes métodos (además de los ya indicados por el encapsulamiento básico).**

    - **El constructor debe considerar que precio y stock nunca puede ser inferior a 0. En caso de pasar un número inferior a 0 se lanzarán excepciones (y se almacenará un 0 en dicho valor).**

```java
package clases;

import excepciones.ArticuloException;
import java.util.Objects;

/**
 * Clase Articulo
 *
 * @author Fran
 */
public abstract class Articulo {

    private String codigo;
    private String nombre;
    private String material;
    private String color;
    private double precio;
    private int stock;

    /**
     * Constructor de la Clase
     *
     * @param _codigo Código del artículo de tipo int
     * @param _nombre Nombre del artículo de tipo String
     * @param _material Material del artículo de tipo String
     * @param _color Color del artículo de tipo String
     * @param _precio Precio del artículo de tipo double
     * @param _stock Stock en almacén del artículo de tipo int
     * @throws ArticuloException
     */
    public Articulo(String _codigo, String _nombre, String _material, String _color, double _precio, int _stock) throws ArticuloException {

        this.codigo = _codigo;
        this.nombre = _nombre;
        this.material = _material;
        this.color = _color;
        setPrecio(_precio);
        setStock(_stock);

    }

```

```java
    /**
     * Método para establecer un nuevo precio a un Artículo
     *
     * @param _precio Nuevo precio de tipo double
     * @throws ArticuloException Si el precio que se introduce es negativo
     */
    private void setPrecio(double _precio) throws ArticuloException {

        if (_precio > 0) {

            this.precio = _precio;

        } else {

            this.precio = 0;
            throw new ArticuloException( message: "El precio del artículo no puede ser negativo");

        }

    }

    /**
     * Método para obtener el stock de un Artículo
     *
     * @return Stock de un Artículo de tipo int
     */
    public int getStock() {
        return stock;
    }

    /**
     * Método para establecer un nuevo stock a un Artículo
     *
     * @param _stock Nuevo stock de tipo int
     * @throws ArticuloException Si el stock que se introduce es negativo
     */
    private void setStock(int _stock) throws ArticuloException {

        if (_stock > 0) {

            this.stock = _stock;

        } else {

            this.stock = 0;
            throw new ArticuloException( message: "El stock no puede ser negativo");

        }

    }

}

```

```java
    /**
     * Método para obtener el código de un Artículo
     *
     * @return Código de un artículo de tipo String
     */
    public String getCodigo() {
        return codigo;
    }

    /**
     * Método para establecer un nuevo código a un Artículo
     *
     * @param codigo Nuevo código del Artículo de tipo String
     */
    public void setCodigo(String codigo) {
        this.codigo = codigo;
    }

    /**
     * Método para obtener el nombre de un Artículo
     *
     * @return Nombre del Artículo de tipo String
     */
    public String getNombre() {
        return nombre;
    }

    /**
     * Método para establecer un nuevo nombre a un Artículo
     *
     * @param nombre Nuevo nombre del Artículo de tipo String
     */
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    /**
     * Método para obtener el material de un Artículo
     *
     * @return Material del Artículo de tipo String
     */
    public String getMaterial() {
        return material;
    }

    /**
     * Método para establecer un nuevo material al Artículo
     *
     * @param material Nuevo material de tipo String
     */
    public void setMaterial(String material) {
        this.material = material;
    }

    /**
     * Método para obtener el color de un Artículo
     *
     * @return Color del Artículo de tipo String
     */
    public String getColor() {
        return color;
    }

    /**
     * Método para establecer un nuevo color a un Artículo
     *
     * @param color Nuevo color de tipo String
     */
    public void setColor(String color) {
        this.color = color;
    }

    /**
     * Método para obtener el precio de un Artículo
     *
     * @return Precio de un Artículo de tipo double
     */
    public double getPrecio() {
        return precio;
    }
```

- **No debe ser posible cambiar el valor de precio y stock**
  Por ello declaramos los métodos setPrecio y setStock como privados.

- **El método equals y toString que mostrará toda la Información del artículo.**

```java
    /**
     * Método equals de la clase
     *
     * @param obj Artículo a comparar
     * @return True si dos artículos tienen el mismo código. False en caso
     * contrario
     */
    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null) {
            return false;
        }

        final Articulo other = (Articulo) obj;
        return Objects.equals( a: this.codigo, b: other.codigo);
    }

    /**
     * Método toString de la clase
     *
     * @return Información del artículo en forma de cadena
     */
    @Override
    public String toString() {
        return "Código de artículo: " + this.codigo + ", Nombre del artículo: " + this.nombre + ", Material: " + this.material
                + ", Color: " + this.color + ", Precio del artículo: " + this.precio + "€" + ", Stock: " + stock;
    }

```

- **Crea un método que se llame totalValor que devolverá el total de precio en euros que tenemos para ese artículo. Para calcular el valor debemos tener en cuenta el valor de ese precio y su stock. Si vale 2,25 y tenemos 4 unidades en stock debe devolver 11,25**

```java

    /**
     * Método que devuelve el precio total del stock de un artículo
     *
     * @return Valor total en formato double del stock de un artículo
     */
    public double totalValor() {

        return (this.stock * this.precio);

    }
```

- **Crea un método mostrarInfo() que no va a devolver nada pero que no va a tener código ni va a ser necesario desarrollar y que se quedará sin código para que sea implementado en las subclases.**

```java

    /**
     * Método mostrarInfo que se debe desarrollar en las Clases que hereden de
     * Artículo
     */
    public abstract void mostrarInfo();

```

- **A partir de la clase Artículo vamos a crear la clase Zapatos de la que nos interesa almacenar su número y el tipo y la clase Bolso de la que nos interesa saber su marca y el tipo. El tipo de un bolso que será de tipo String debe almacenar únicamente los valores bandolera, mochila, y fiesta**

    - **Implementa los métodos que necesites para que puedas crear objetos de ambos tipos y aquellos que consideres que pueden ser de tu interés (no debes crear métodos que no pudieran tener ningún tipo de necesidad).**
    - **Genera el método mostrarInfo() de cada clase.**

```java
package clases;

import excepciones.ArticuloException;

/**
 * Clase Zapato
 *
 * @author Fran
 */
public class Zapato extends Articulo {

    private int numero;
    private String tipo;

    /**
     * Constructor de la clase
     *
     * @param _codigo Código del zapato de tipo String
     * @param _nombre Nombre del zapato de tipo String
     * @param _material Material del zapato de tipo String
     * @param _color Color del zapato de tipo String
     * @param _precio Precio del zapato de tipo double
     * @param _stock Stock del zapato de tipo int
     * @param _numero Número del zapato de tipo int
     * @param _tipo Tipo de zapato de tipo String
     * @throws ArticuloException Si el precio introducido o el stock es negativo
     */
    public Zapato(String _codigo, String _nombre, String _material, String _color, double _precio, int _stock, int _numero, String _tipo) throws ArticuloException {

        super(_codigo, _nombre, _material, _color, _precio, _stock);
        this.numero = _numero;
        this.tipo = _tipo;

    }

    /**
     * Método toString de la clase
     *
     * @return Información del zapato en tipo cadena
     */
    @Override
    public String toString() {
        return super.toString() + ", Número: " + this.numero + ", Tipo: " + this.tipo;
    }

```

```java
    /**
     * Método mostrarInfo de la clase. Muestra por pantalla la información del
     * zapato
     */
    @Override
    public void mostrarInfo() {

        System.out.println("*** Información del Zapato ***");
        System.out.println("Código: " + super.getCodigo());
        System.out.println("Nombre: " + super.getNombre());
        System.out.println("Material: " + super.getMaterial());
        System.out.println("Color: " + super.getColor());
        System.out.println("Número: " + this.getNumero());
        System.out.println("Tipo: " + this.getTipo());
        System.out.println("Precio: " + super.getPrecio() + "€");
        System.out.println("Stock: " + super.getStock());

    }

    /**
     * Método para obtener el número del zapato
     *
     * @return Número del zapato de tipo int
     */
    public int getNumero() {
        return numero;
    }

    /**
     * Método para establecer un nuevo número al zapato
     *
     * @param numero Nuevo número de tipo int
     */
    public void setNumero(int numero) {
        this.numero = numero;
    }

    /**
     * Metodo para obtener el tipo de zapato
     *
     * @return Tipo de zapato de tipo String
     */
    public String getTipo() {
        return tipo;
    }

    /**
     * Metodo para establecer un nuevo tipo de zapato
     *
     * @param tipo Nuevo tipo de tipo String
     */
    public void setTipo(String tipo) {
        this.tipo = tipo;
    }

}
```

```java
package clases;

import excepciones.ArticuloException;

/**
 * Clase Bolso
 *
 * @author Fran
 */
public class Bolso extends Articulo {

    /**
     * Enumerado de los tipos de bolso disponibles
     */
    public enum TipoBolso {

        BANDOLERA, MOCHILA, FIESTA;

    }

    private String marca;
    private TipoBolso tipo;

    /**
     * Constructor de la clase
     *
     * @param _codigo Código del bolso de tipo String
     * @param _nombre Nombre del bolso de tipo String
     * @param _material Material del bolso de tipo String
     * @param _color Color del bolso de tipo String
     * @param _precio Precio del bolso de tipo double
     * @param _stock Stock del bolso de tipo int
     * @param _marca Marca del bolso de tipo String
     * @param _tipo Tipo de bolso de tipo TipoBolso (Bandolera, Mochila, Fiesta)
     * @throws ArticuloException Si se introduce un precio o stock negativo, o
     * un tipo de bolso no válido
     */
    public Bolso(String _codigo, String _nombre, String _material, String _color, double _precio, int _stock, String _marca, String _tipo) throws ArticuloException {

        super(_codigo, _nombre, _material, _color, _precio, _stock);
        this.marca = _marca;
        setTipo(_tipo);

    }

    /**
     * Metodo toString de la clase
     *
     * @return Información del bolso en una cadena
     */
    @Override
    public String toString() {
        return super.toString() + ", Marca: " + this.marca + ", Tipo: " + this.tipo;
    }

```

```java
    /**
     * Método mostrarInfo de la clase. Muestra por pantalla la información del
     * bolso
     */
    @Override
    public void mostrarInfo() {

        System.out.println(x: "*** Información del Bolso ***");
        System.out.println("Código: " + super.getCodigo());
        System.out.println("Nombre: " + super.getNombre());
        System.out.println("Material: " + super.getMaterial());
        System.out.println("Color: " + super.getColor());
        System.out.println("Marca: " + this.getMarca());
        System.out.println("Tipo: " + this.getTipo());
        System.out.println("Precio: " + super.getPrecio() + "€");
        System.out.println("Stock: " + super.getStock());

    }

    /**
     * Método para obtener la marca del bolso
     *
     * @return Marca del bolso de tipo String
     */
    public String getMarca() {
        return marca;
    }

    /**
     * Método para establecer una nueva marca al bolso
     *
     * @param marca Nueva marca de tipo String
     */
    public void setMarca(String marca) {
        this.marca = marca;
    }

    /**
     * Método para obtener el tipo de bolso
     *
     * @return Tipo de bolso de tipo String
     */
    public String getTipo() {
        return this.tipo.toString();
    }

    /**
     * Método para establecer un nuevo tipo de bolso
     *
     * @param _tipo Nuevo tipo de tipo String
     * @throws ArticuloException Si el tipo de bolso no se encuentra entre
     * Bandolera, Mochila o Fiesta
     */
    public void setTipo(String _tipo) throws ArticuloException {

        try {

            this.tipo = TipoBolso.valueOf( name: convierteCadena( entrada: _tipo));

        } catch (IllegalArgumentException ex) {

            throw new ArticuloException( message: "El tipo de bolso elegido no es válido");

        }
    }

    /**
     * Método privado de la clase que convierte una cadena en vocal y le quita
     * las tildes que pueda llevar
     *
     * @param entrada Cadena a convertir
     * @return Cadena en mayúscula y sin tildes
     */
    private String convierteCadena(String entrada) {

        String cadena = entrada.toUpperCase();

        String cadenaUtil = cadena.replace( target: "Á",  replacement: "A").replace( target: "É",  replacement: "E").replace( target: "Í",  replacement: "I")
                .replace( target: "Ó",  replacement: "O").replace( target: "Ú",  replacement: "U");

        return cadenaUtil;

    }

}
```

- **Clase utilizada para las excepciones del Artículo.**

```java
package excepciones;

/**
 * Clase utilizada para las excepciones de Artículo
 *
 * @author Fran
 */
public class ArticuloException extends Exception {

    public ArticuloException(String message) {
        super(message);
    }

}
```

- **Crea una clase que se llame Tienda que va a tener un String nombreTienda, que contendrá el nombre de la tienda, un LocalDate (o el tipo Fecha que tú quieras) que guarda la fecha de apertura de dicha tienda y que no podrá ser posterior a 90 días más de la fecha actual y un ArrayList de elementos de tipo Artículo.**

```java
package clases;

import excepciones.FechaException;
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.time.format.DateTimeParseException;
import java.time.temporal.ChronoUnit;
import java.util.ArrayList;

/**
 * Clase Tienda
 *
 * @author Fran
 */
public class Tienda {

    private String nombreTienda;
    private LocalDate fechaApertura;
    private ArrayList<Articulo> articuloTienda;
```

  - **El constructor de esta clase va a recibir el nombre de la tienda, la fecha de creación y un ArrayList con todos los artículos que va a tener esa tienda.**

```java
    /**
     * Constructor de la clase con todos los parámetros
     *
     * @param _nombreTienda Nombre de la tienda de tipo String
     * @param _fechaApertura Fecha de apertura de la tienda de tipo String
     * formato dd/MM/yyyy
     * @param _articuloTienda Artículos de la tienda de tipo ArrayList
     * @throws FechaException Si la fecha de apertura es mayor a 90 días después
     * de la fecha actual
     */
    public Tienda(String _nombreTienda, String _fechaApertura, ArrayList<Articulo> _articuloTienda) throws FechaException {

        creaTienda(_nombreTienda, _fechaApertura, _articuloTienda);

    }
```

- **Es posible tener un constructor en el que no se indique fecha y en tal caso se crea la Tienda con la fecha actual**

```java
    /**
     * Constructor de la clase sin la fecha de apertura. Se establece fecha
     * actual
     *
     * @param _nombreTienda Nombre de la tienda de tipo String
     * @param _articuloTienda Articulos de la tienda de tipo ArrayList
     * @throws FechaException Si la fecha de apertura es mayor a 90 dias después
     * de la fecha actual
     */
    public Tienda(String _nombreTienda, ArrayList<Articulo> _articuloTienda) throws FechaException {

        creaTienda(_nombreTienda, _fechaApertura: fechaAct(), _articuloTienda);

    }
```

- **Crea un método que indique el valor total en productos que tiene esa Tienda. Debemos tener en cuenta que va a tener varios artículos (cada entrada del ArrayList es un artículo) y para cada uno de estos tendrá un número de unidades y cada unidad tiene un valor. Por ejemplo, si tenemos un bolso del que tenemos 4 unidades y vale cada unidad 13,10 y un artículo Zapato para el que tenemos 2 unidades y vale cada de estas unidades 29,25 este método debería devolver à (4 * 13,10) + (2 * 29,25) = 52,4 + 58,5 = 110,90.**

```java
    /**
     * Método que calcula el valor total de género del que dispone
     *
     * @return Valor total del género de tipo double
     */
    public double valorTienda() {

        double total = 0;

        for (Articulo a : articuloTienda) {

            total += a.totalValor();

        }

        return total;

    }
```

- **Crea un método que muestre por pantalla una información resumen de la Tienda indicando los Artículos que contiene el mismo con su tipo correspondiente (si son Bolso o Zapato).**

```java
    /**
     * Método que muestra la información de la tienda por pantalla
     */
    public void muestraInfo() {

        System.out.println( x: "*** Articulos de la tienda ***");

        for (Articulo a : articuloTienda) {

            System.out.println(a + ", Tipo de Objeto: " + a.getClass().getSimpleName());

        }

    }
```

- **Métodos auxiliares de Tienda**

```java
    /**
     * Método auxiliar para crear la tienda en el constructor
     *
     * @param _nombreTienda Nombre de la tienda
     * @param _fechaApertura Fecha de apertura
     * @param _articuloTienda Artículos de la tienda
     * @throws FechaException Si la fecha de apertura es mayor a 90 días después
     * de la fecha actual
     */
    private void creaTienda(String _nombreTienda, String _fechaApertura, ArrayList<Articulo> _articuloTienda) throws FechaException {

        this.nombreTienda = _nombreTienda;
        setFechaApertura(_fechaApertura);
        this.articuloTienda = new ArrayList<>( c: _articuloTienda);

    }
```

```java
    /**
     * Método auxiliar para convertir fechas de tipo dd/MM/yyyy a tipo LocalDate
     *
     * @param fecha Fecha de tipo String dd/MM/yyyy
     * @return Fecha de tipo LocalDate yyyy-MM-dd
     */
    private LocalDate fechaEsp(String fecha) {

        DateTimeFormatter fechaCadena = DateTimeFormatter.ofPattern( pattern: "dd/MM/yyyy");

        return LocalDate.parse( text: fecha,  formatter: fechaCadena);

    }

    /**
     * Método auxiliar para calcular la fecha actual de tipo String dd/MM/yyyy
     *
     * @return Fecha actual en String dd/MM/yyyy
     */
    private String fechaAct() {

        LocalDate fechaLocalDate = LocalDate.now();
        DateTimeFormatter formato = DateTimeFormatter.ofPattern( pattern: "dd/MM/yyyy");
        String fecha = fechaLocalDate.format( formatter: formato);

        return fecha;

    }

    /**
     * Método auxiliar para calcular la fecha entre un día pasado por parámetro
     * y la fecha actual
     *
     * @param fecha Fecha de tipo LocalDate
     * @return Diferencia entre fecha actual y la pasada por parámetro en días
     */
    private static long diferenciaDias(LocalDate fecha) {

        LocalDate fechaAct = LocalDate.now();
        LocalDate fechaIn = fecha;

        long dias = ChronoUnit.DAYS.between( temporal1Inclusive: fechaAct,  temporal2Exclusive: fechaIn);

        return dias;

    }
```

- **Getter y setters**

```java
    /**
     * Método para obtener el nombre de la tienda
     *
     * @return Nombre de la tienda de tipo String
     */
    public String getNombreTienda() {
        return nombreTienda;
    }

    /**
     * Método para establecer un nuevo nombre a la tienda
     *
     * @param nombreTienda Nuevo nombre de la tienda de tipo String
     */
    public void setNombreTienda(String nombreTienda) {
        this.nombreTienda = nombreTienda;
    }

    /**
     * Método para obtener la fecha de apertura de la tienda
     *
     * @return Fecha de apertura de la tienda de tipo LocalDate
     */
    public LocalDate getFechaApertura() {
        return fechaApertura;
    }

    /**
     * Método para establecer una nueva fecha de apertura de la tienda
     *
     * @param _fechaApertura Nueva fecha de apertura de tipo String dd/MM/yyyy
     * @throws FechaException Si la fecha introducida no es del tipo dd/MM/yyyy
     */
    public void setFechaApertura(String _fechaApertura) throws FechaException {

        try {

            LocalDate fechaRecibida = fechaEsp( fecha: _fechaApertura);

            if (diferenciaDias( fecha: fechaEsp( fecha: _fechaApertura)) < 90) {

                this.fechaApertura = fechaRecibida;

            } else {

                throw new FechaException( message: "Error. La fecha de apertura tiene que ser como máximo 90 días posterior a la fecha actual");

            }

        } catch (DateTimeParseException fechaEx) {

            throw new FechaException( message: "Error. El formato de la fecha debe ser dd/MM/yyyy");

        }

    }

    /**
     * Método para obtener los artículos de la tienda
     *
     * @return Artículos de la tienda de tipo ArrayList
     */
    public ArrayList<Articulo> getArticuloTienda() {
        return articuloTienda;
    }

    /**
     * Método para establecer nuevos artículos a la tienda
     *
     * @param articuloTienda Nuevos artículos de la tienda de tipo ArrayList
     */
    public void setArticuloTienda(ArrayList<Articulo> articuloTienda) {
        this.articuloTienda = articuloTienda;
    }

}
```

- **Clase utilizada para las excepciones de la tienda**

```java
package excepciones;

/**
 *
 * @author Fran
 */
public class FechaException extends Exception {

    public FechaException(String message) {
        super(message);
    }

}
```

- **Pruebas de tienda**

```java
package com.mycompany.garcia_cutillas_franciscojose_actut10_perrera;

import clases.Articulo;
import clases.Bolso;
import clases.Tienda;
import clases.Zapato;
import excepciones.ArticuloException;
import excepciones.FechaException;
import java.util.ArrayList;

/**
 *
 * @author Fran
 */
public class Garcia_Cutillas_FranciscoJose_ActUt10_Tienda {

    public static void main(String[] args) {

        try {

            Zapato z1 = new Zapato(_codigo: "1", _nombre: "Zapato boda", _material: "Piel", _color: "Negro", _precio: 84.99, _stock: 5, _numero: 43, _tipo: "Formal");
            Zapato z2 = new Zapato(_codigo: "2", _nombre: "Zapato estándar", _material: "Polipiel", _color: "Marrón", _precio: 39.99, _stock: 50, _numero: 41, _tipo: "Casual");

            z2.mostrarInfo();

            Bolso b1 = new Bolso(_codigo: "23", _nombre: "Bolso beige", _material: "Esparto", _color: "Marrón", _precio: 19.99, _stock: 10, _marca: "Bomba y Loli", _tipo: "Fiesta");
            Bolso b2 = new Bolso(_codigo: "24", _nombre: "Bolso piel", _material: "Polipiel", _color: "Rojo", _precio: 59.99, _stock: 3, _marca: "Adidas", _tipo: "Mochila");

            System.out.println(x: "");
            b1.mostrarInfo();
            System.out.println(x: "");
            System.out.println("Total stock bolso con código " + b1.getCodigo()+ ": " + b1.totalValor() + " Euros");

            ArrayList <Articulo> articulos = new ArrayList <> ();

            articulos.add(e: z1);
            articulos.add(e: z2);
            articulos.add(e: b1);
            articulos.add(e: b2);


            Tienda t1 = new Tienda (_nombreTienda: "Boutique Archena", _fechaApertura: "23/05/2023", _articuloTienda: articulos);
            System.out.println(x: "");
            t1.muestraInfo();
            System.out.println(x: "");
            System.out.println("Valor del género de la tienda: " + t1.valorTienda() + " Euros");


        } catch (ArticuloException aEx) {

            System.out.println(x: aEx.getMessage());

        } catch (FechaException fEx) {

            System.out.println(x: fEx.getMessage());

        }

    }

}
```

```
--- exec-maven-plugin:3.0.0:exec (default-cli) @ Garcia_Cutillas_FranciscoJose_ActUt10_Tienda ---
*** Información del Zapato ***
Código: 2
Nombre: Zapato estándar
Material: Polipiel
Color: Marrón
Número: 41
Tipo: Casual
Precio: 39.99€
Stock: 50

*** Información del Bolso ***
Código: 23
Nombre: Bolso beige
Material: Esparto
Color: Marrón
Marca: Bomba y Loli
Tipo: FIESTA
Precio: 19.99€
Stock: 10

Total stock bolso con código 23: 199.89999999999998 Euros

*** Artículos de la tienda ***
Código de artículo: 1, Nombre del artículo: Zapato boda, Material: Piel, Color: Negro, Precio del artículo: 84.99€, Stock: 5, Número: 43, Tipo: Formal, Tipo de Objeto: Zapato
Código de artículo: 2, Nombre del artículo: Zapato estándar, Material: Polipiel, Color: Marrón, Precio del artículo: 39.99€, Stock: 50, Número: 41, Tipo: Casual, Tipo de Objeto: Zapato
Código de artículo: 23, Nombre del artículo: Bolso beige, Material: Esparto, Color: Marrón, Precio del artículo: 19.99€, Stock: 10, Marca: Bomba y Loli, Tipo: FIESTA, Tipo de Objeto: Bolso
Código de artículo: 24, Nombre del artículo: Bolso piel, Material: Polipiel, Color: Rojo, Precio del artículo: 59.99€, Stock: 3, Marca: Adidas, Tipo: MOCHILA, Tipo de Objeto: Bolso

Valor del género de la tienda: 2804.3199999999997 Euros
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  0.307 s
Finished at: 2023-05-15T21:09:35+02:00
------------------------------------------------------------------------
```