



Programación

Act10_ArrayList y Excepciones

Francisco José García Cutillas | 1FPGS_DAM

Índice

Ejercicio A01. Perrera	3
Ejercicio A02. Tienda	16

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).

```

1  package clases;
2
3  import excepciones.FechaException;
4  import excepciones.NombreException;
5  import excepciones.RazaException;
6  import java.time.LocalDate;
7  import java.time.Period;
8  import java.time.format.DateTimeFormatter;
9  import java.time.format.DateTimeParseException;
10
11 /**
12  * Clase Perro. Con ella podemos crear objetos de tipo Perro
13  *
14  * @author Fran
15  */
16 public class Perro {
17
18     public enum Raza {
19
20         CANICHE, BODEGUERO, PITBULL, BOXER, MASTIN, LABRADOR;
21
22     }
23
24     private int codigo;
25     private String nombre;
26     private LocalDate fechaNac;
27     private Raza razaPerro;
28

```

- 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.

```

29  /**
30   * Constructor de la clase
31   *
32   * @param _codigo Código del perro
33   * @param _nombre Nombre del perro
34   * @param _fechaNac Fecha de nacimiento del perro del tipo dd/MM/yyyy
35   * @param _razaPerro Raza del perro (Caniche, Bodeguero, Pitbull, Bóxer,
36   * Mastin, Labrador)
37   * @throws RazaException Si la raza no está entre las permitidas como
38   * parámetro
39   * @throws FechaException Si la fecha no es de tipo dd/MM/yyyy
40   * @throws NombreException Si el nombre tiene menos de 3 caracteres
41   */
42  public Perro(int _codigo, String _nombre, String _fechaNac, String _razaPerro) throws RazaException, FechaException, NombreException {
43
44      this.codigo = _codigo;
45      setNombre(_nombre);
46      setFechaNac(_fechaNac);
47      setRazaPerro(_razaPerro);
48  }
49

```

```
111 | /**  
112 |  * Método para obtener el código del perro  
113 |  *  
114 |  * @return Código del perro de tipo int  
115 |  */  
116 | public int getCodigo() {  
117 |     return codigo;  
118 | }  
119 |  
120 | /**  
121 |  * Método para establecer un nuevo código al perro  
122 |  *  
123 |  * @param codigo Nuevo código de tipo int  
124 |  */  
125 | public void setCodigo(int codigo) {  
126 |     this.codigo = codigo;  
127 | }  
128 |  
129 | /**  
130 |  * Método para obtener el nombre del perro  
131 |  *  
132 |  * @return Nombre del perro de tipo String  
133 |  */  
134 | public String getNombre() {  
135 |     return nombre;  
136 | }  
137 |  
  
189 | /**  
190 |  * Método que devuelve la fecha de nacimiento del perro  
191 |  *  
192 |  * @return Fecha de nacimiento del perro de tipo LocalDate  
193 |  */  
194 | public LocalDate getFechaNac() {  
195 |     return fechaNac;  
196 | }  
197 |  
198 | /**  
199 |  * Método que devuelve la raza del perro  
200 |  *  
201 |  * @return Raza del perro de tipo Raza  
202 |  */  
203 | public Raza getRazaPerro() {  
204 |     return razaPerro;  
205 | }  
206 |
```

```

51  /**
52   * Método toString() de la clase
53   *
54   * @return Información del perro en forma de cadena
55   */
56  @Override
57  public String toString() {
58      return "Código: " + this.codigo + ", Nombre: " + this.nombre + ", Fecha de nacimiento: "
59          + fechaNac.getDayOfMonth() + "/" + fechaNac.getMonthValue() + "/" + fechaNac.getYear()
60          + ", Raza: " + this.razaPerro;
61  }
62
63  /**
64   * Método equals de la clase
65   *
66   * @param obj Objeto a comparar
67   * @return True si dos perros tienen el mismo código, False en caso
68   * contrario
69   */
70  @Override
71  public boolean equals(Object obj) {
72      if (this == obj) {
73          return true;
74      }
75      if (obj == null) {
76          return false;
77      }
78      if (getClass() != obj.getClass()) {
79          return false;
80      }
81      final Perro other = (Perro) obj;
82      return this.codigo == other.codigo;
83  }
84

```

```

85  /**
86   * Método que imprime por pantalla la información del perro
87   */
88  public void imprimir() {
89
90      System.out.println("**** Información del perro ****");
91      System.out.println("Código: " + this.codigo);
92      System.out.println("Nombre: " + this.nombre);
93      System.out.println("Fecha de nacimiento: " + fechaNac.getDayOfMonth() + "/" + fechaNac.getMonthValue() + "/" + fechaNac.getYear());
94      System.out.println("Raza: " + this.razaPerro);
95
96  }
97

```

- 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.

```

138  /**
139   * Método para establecer un nuevo nombre al perro
140   *
141   * @param _nombre Nombre de tipo String
142   * @throws NombreException Si el nombre tiene menos de 3 caracteres
143   */
144  public void setNombre(String _nombre) throws NombreException {
145
146      if (_nombre.length() >= 3) {
147
148          this.nombre = _nombre;
149
150      } else {
151
152          throw new NombreException(message: "El nombre debe de tener al menos 3 caracteres");
153
154      }
155
156  }
157
158  /**
159   * Método para establecer una nueva fecha de nacimiento al perro
160   *
161   * @param _fechaNac Nueva fecha de nacimiento en tipo String con formato
162   * dd/MM/yyyy
163   * @throws FechaException Si la fecha introducida no es del tipo dd/MM/yyyy
164   */
165  public void setFechaNac(String _fechaNac) throws FechaException {
166
167      try {
168
169          LocalDate fechaRecibida = fechaEsp(fecha: _fechaNac);
170
171          if (fechaRecibida.isBefore(othor: LocalDate.now())) {
172
173              this.fechaNac = fechaRecibida;
174
175          } else {
176
177              throw new FechaException(message: "Error. La fecha de nacimiento no puede ser posterior a la actual");
178
179          }
180
181      } catch (DateTimeParseException fechaEx) {
182
183          throw new FechaException(message: "Error. El formato de la fecha debe ser dd/MM/yyyy");
184
185      }
186
187  }
188

```

```

207  /**
208   * Método para establecer una nueva raza al perro
209   *
210   * @param _razaPerro Nueva raza de tipo String
211   * @throws RazaException Si la raza no es correcta (Caniche, Bodeguero,
212   * Pitbull, Bóxer, Mastín, Labrador)
213   */
214  public void setRazaPerro(String _razaPerro) throws RazaException {
215
216      try {
217
218          this.razaPerro = Raza.valueOf(name: convierteCadena(entrada: _razaPerro));
219
220      } catch (IllegalArgumentException ex) {
221
222          throw new RazaException(_raza: _razaPerro);
223
224      }
225
226  }

```

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

```

98  /**
99  * Método que devuelve la edad del perro en años
100  *
101  * @return Edad del perro en años de tipo int
102  */
103  public int devuelveEdad() {
104
105      Period diferencia = Period.between(startDateInclusive: fechaNac, endDateExclusive: LocalDate.now());
106      int edad = diferencia.getYears();
107      return edad;
108  }
109
110

```

- Métodos auxiliares utilizados en la clase Perro.

```

327  /**
328  * Método privado de la clase que convierte una cadena a Mayúsculas y si
329  * lleva tildes se las quita
330  *
331  * @param entrada String a convertir
332  * @return String en mayúsculas sin tildes
333  */
334  private String convierteCadena(String entrada) {
335
336      String cadena = entrada.toUpperCase();
337
338      String cadenaUtil = cadena.replace(target: "Á", replacement: "A").replace(target: "É", replacement: "E").replace(target: "Í", replacement: "I")
339                          .replace(target: "Ó", replacement: "O").replace(target: "Ú", replacement: "U");
340
341      return cadenaUtil;
342  }
343
344
345  /**
346  * Método privado de la clase que convierte una fecha tipo String en una
347  * tipo LocalDate
348  *
349  * @param fecha Fecha de tipo String formato dd/MM/yyyy
350  * @return Fecha tipo LocalDate formato yyyy-MM-dd
351  */
352  private LocalDate fechaEsp(String fecha) {
353
354      DateTimeFormatter fechaCadena = DateTimeFormatter.ofPattern(pattern: "dd/MM/yyyy");
355
356      return LocalDate.parse(text: fecha, formatter: fechaCadena);
357  }
358
359

```

- Clases creadas para las excepciones de la clase Perro.

```

1  package excepciones;
2
3  /**
4  * Clase exception de fecha. Usada cuando una fecha se introduce de forma incorrecta
5  * @author Fran
6  */
7  public class FechaException extends Exception {
8
9      /**
10     * Constructor de la clase
11     * @param message Mensaje que queremos mostrar cuando se produzca la excepción
12     */
13     public FechaException(String message) {
14         super(message);
15     }
16
17 }
18

```

```

1  package excepciones;
2
3  /**
4   * Clase exception de nombre. Usada cuando un nombre es incorrecto
5   * @author Fran
6   */
7  public class NombreException extends Exception {
8
9      /**
10     * Constructor de la clase
11     * @param message Mensaje que queremos mostrar cuando se produzca la excepción
12     */
13     public NombreException(String message) {
14         super(message);
15     }
16
17 }
18

```

```

1  package excepciones;
2
3  /**
4   * Clase exception de perrera. Usada cuando una perrera tiene un error
5   * @author Fran
6   */
7  public class PerreraException extends Exception {
8
9      /**
10     * Constructor de la clase
11     * @param message Mensaje que queremos mostrar cuando se produzca la excepción
12     */
13     public PerreraException(String message) {
14         super(message);
15     }
16
17 }
18

```

```

1  package excepciones;
2
3  /**
4   * Clase exception de raza. Usada cuando una raza no es correcta
5   * @author Fran
6   */
7  public class RazaException extends IllegalArgumentException {
8
9      private String raza;
10
11     /**
12     * Constructor de la clase
13     * @param _raza raza a la que hacemos referencia en la excepción
14     */
15     public RazaException(String _raza) {
16         this.raza = _raza;
17     }
18
19
20
21     /**
22     * Método que devuelve el mensaje de error
23     * @return Mensaje de error cuando una raza no es correcta
24     */
25     @Override
26     public String getMessage() {
27         return "La raza " + this.raza + " no es válida";
28     }
29
30 }
31
32 }
33

```


- 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).

```

1 package com.mycompany.garcia_cutillas_franciscojose_actutl0_perrera;
2
3 import clases.Perrera;
4 import clases.Perro;
5 import excepciones.FechaException;
6 import excepciones.NombreException;
7 import excepciones.PerreraException;
8 import excepciones.RazaException;
9 import java.util.Collections;
10 import ordenacion.ComparaPerros;
11
12 /**
13  *
14  * @author Fran
15  */
16 public class Garcia_Cutillas_FranciscoJose_ActUtl0_Perrera {
17
18     public static void main(String[] args) {
19
20         Perrera pel = new Perrera(_nombre: "El Olvido", _direccion: "Calle Mayor, 3");
21
22         try {
23
24             Perro p1 = new Perro(_codigo: 1, _nombre: "Firulais", _fechaNac: "10/05/2023", _razaPerro: "bóxer");
25
26             Perro p2 = new Perro(_codigo: 2, _nombre: "Boo", _fechaNac: "21/05/2021", _razaPerro: "Pitbull");
27
28             Perro p3 = new Perro(_codigo: 3, _nombre: "Tobi", _fechaNac: "02/09/2019", _razaPerro: "Labrador");
29
30             Perro p4 = new Perro(_codigo: 4, _nombre: "Beethoven", _fechaNac: "30/07/2014", _razaPerro: "Mastin");
31
32             Perro p5 = new Perro(_codigo: 5, _nombre: "Tobi", _fechaNac: "16/02/2022", _razaPerro: "Caniche");
33
34             Perro p6 = new Perro(_codigo: 6, _nombre: "Jackie", _fechaNac: "20/09/2020", _razaPerro: "Bodeguero");
35
36             System.out.println("¿Es Perro p2 igual a p1?: " + p2.equals(obj: p1));
37             System.out.println("¿Es Perro p5 igual a p5?: " + p5.equals(obj: p5));
38
39             System.out.println(":\nInformación del perro p2");
40             p2.imprimir();
41
42             System.out.println("\nEdad del perro p4: " + p4.devuelveEdad() + " Años");
43             System.out.println("Edad del perro p6: " + p6.devuelveEdad() + " Años");
44
45             ...
46
47         } catch (RazaException rEx) {
48
49             System.out.println(":\nrEx.getMessage());
50
51         } catch (FechaException fEx) {
52
53             System.out.println(":\nfEx.getMessage());
54
55         } catch (NombreException nEx) {
56
57             System.out.println(":\nnEx.getMessage());
58
59         }
60
61     }
62 }

```

```

¿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

```

```

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_ActUtl0_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:"Be", _fechaNac:"21/05/2021", _razaPerro:"Pitbull");

```

```

--- exec-maven-plugin:3.0.0:exec (default-cli) @ Garcia_Cutillas_FranciscoJose_ActUtl0_Perrera ---
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.

```

1 package clases;
2
3 import excepciones.PerreraException;
4 import java.util.ArrayList;
5 import java.util.Objects;
6
7 /**
8  * Clase Perrera. Con ella podemos crear objetos de tipo Perrera
9  * @author Fran
10  */
11 public class Perrera {
12
13     private String nombre;
14     private String direccion;
15     private ArrayList<Perro> perros;
16     private final int maxPerrera = 5;
17
18     /**
19      * Constructor de la clase
20      * @param _nombre Nombre de la perrera de tipo String
21      * @param _direccion Dirección de la perrera de tipo String
22      */
23     public Perrera(String _nombre, String _direccion) {
24
25         this.nombre = _nombre;
26         this.direccion = _direccion;
27         this.perros = new ArrayList<>();
28     }
29
30

```

```

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

```

```

86  [ ] /**
87  [ ] * Método para obtener el nombre de la perrera
88  [ ] *
89  [ ] * @return Nombre de la perrera de tipo String
90  [ ] */
91  [ ] public String getNombre() {
92  [ ]     return nombre;
93  [ ] }
94  [ ]
95  [ ] /**
96  [ ] * Método para establecer un nuevo nombre a la perrera
97  [ ] *
98  [ ] * @param nombre Nuevo nombre de la perrera de tipo String
99  [ ] */
100 [ ] public void setNombre(String nombre) {
101 [ ]     this.nombre = nombre;
102 [ ] }
103 [ ]
104 [ ] /**
105 [ ] * Método para obtener la dirección de la perrera
106 [ ] *
107 [ ] * @return Dirección de la perrera de tipo String
108 [ ] */
109 [ ] public String getDireccion() {
110 [ ]     return direccion;
111 [ ] }
112 [ ]
113 [ ] /**
114 [ ] * Método para establecer una nueva dirección a la perrera
115 [ ] *
116 [ ] * @param direccion Nueva dirección de la perrera de tipo String
117 [ ] */
118 [ ] public void setDireccion(String direccion) {
119 [ ]     this.direccion = direccion;
120 [ ] }
121 [ ]
122 [ ] /**
123 [ ] * Método para obtener los perros que hay en la perrera
124 [ ] *
125 [ ] * @return Perros de la perrera de tipo ArrayList
126 [ ] */
127 [ ] public ArrayList getPerros() {
128 [ ]     return perros;
129 [ ] }
130 [ ]

```

```

32
33
34      /**
35       * Método equals de la clase. Compara dos perreras
36       *
37       * @param obj Perrera a comparar
38       * @return True si dos perreras tienen el mismo nombre y dirección. False en
39       * caso contrario
40       */
41      @Override
42      public boolean equals(Object obj) {
43          if (this == obj) {
44              return true;
45          }
46          if (obj == null) {
47              return false;
48          }
49          if (getClass() != obj.getClass()) {
50              return false;
51          }
52          final Perrera other = (Perrera) obj;
53          if (!Objects.equals(a: this.nombre, b: other.nombre)) {
54              return false;
55          }
56          return Objects.equals(a: this.direccion, b: other.direccion);
57      }

```

- Una Perrera se crea siempre sin perros.

```

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(o: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.

```

1 package ordenacion;
2
3 import clases.Perro;
4 import java.text.Collator;
5 import java.util.Comparator;
6
7 /**
8  * Clase para comparar dos perros. Implementa la interfaz Comparator
9  * @author Fran
10  */
11 public class ComparaPerros implements Comparator {
12
13     /**
14      * Método compare utilizado para comparar dos perros
15      * @param o1 Primer perro a comparar de tipo Perro
16      * @param o2 Segundo perro a comparar de tipo Perro
17      * @return Perros ordenados por nombre. A igual nombre desempata el código de cada uno
18      */
19     @Override
20     public int compare(Object o1, Object o2) {
21
22         if (o1 == o2) {
23             return 0;
24         }
25
26         if (o2 == null) {
27             return -1;
28         }
29
30         if (o1 == null) {
31             return 1;
32         }
33
34         Perro p1 = (Perro) o1;
35         Perro p2 = (Perro) o2;
36
37         Collator col = Collator.getInstance();
38
39         if (col.compare(source:p1.getNombre(), target:p2.getNombre()) == 0) {
40             return p1.getCodigo() - p2.getCodigo();
41         } else {
42             return col.compare(source:p1.getNombre(), target:p2.getNombre());
43         }
44     }
45 }

```

- Clase creada para las excepciones de la clase Perrera.

```

1 package excepciones;
2
3 /**
4  * Clase exception de perrera. Usada cuando una perrera tiene un error
5  * @author Fran
6  */
7 public class PerreraException extends Exception {
8
9     /**
10      * Constructor de la clase
11      * @param message Mensaje que queremos mostrar cuando se produzca la excepción
12      */
13     public PerreraException(String message) {
14         super(message);
15     }
16
17 }
18

```

○ 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: Tobí, 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: Beethoven, Fecha de nacimiento: 30/7/2014, Raza: MASTIN
Código: 1, Nombre: Firulaís, Fecha de nacimiento: 10/5/2023, Raza: BOXER
Código: 3, Nombre: Tobí, Fecha de nacimiento: 2/9/2019, Raza: LABRADOR
Código: 5, Nombre: Tobí, 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));
System.out.println(x: pel);

```

```

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).

```

1 package clases;
2
3 import excepciones.ArticuloException;
4 import java.util.Objects;
5
6 /**
7  * Clase Artículo
8  *
9  * @author Fran
10  */
11 public abstract class Articulo {
12
13     private String codigo;
14     private String nombre;
15     private String material;
16     private String color;
17     private double precio;
18     private int stock;
19
20     /**
21      * Constructor de la Clase
22      *
23      * @param _codigo Código del artículo de tipo int
24      * @param _nombre Nombre del artículo de tipo String
25      * @param _material Material del artículo de tipo String
26      * @param _color Color del artículo de tipo String
27      * @param _precio Precio del artículo de tipo double
28      * @param _stock Stock en almacén del artículo de tipo int
29      * @throws ArticuloException
30      */
31     public Articulo(String _codigo, String _nombre, String _material, String _color, double _precio, int _stock) throws ArticuloException {
32
33         this.codigo = _codigo;
34         this.nombre = _nombre;
35         this.material = _material;
36         this.color = _color;
37         setPrecio(_precio);
38         setStock(_stock);
39     }
40
41 }

```



```
171 /**  
172  * Método para establecer un nuevo precio a un Artículo  
173  *   
174  * @param _precio Nuevo precio de tipo double  
175  * @throws ArtículoException Si el precio que se introduce es negativo  
176  */  
177 private void setPrecio(double _precio) throws ArtículoException {  
178   
179     if ( _precio > 0 ) {  
180         this.precio = _precio;  
181     } else {  
182   
183         this.precio = 0;  
184         throw new ArtículoException(message: "El precio del artículo no puede ser negativo");  
185     }  
186   
187 }  
188   
189 }  
190   
191   
192 /**  
193  * Método para obtener el stock de un Artículo  
194  *   
195  * @return Stock de un Artículo de tipo int  
196  */  
197 public int getStock() {  
198     return stock;  
199 }  
200   
201   
202 /**  
203  * Método para establecer un nuevo stock a un Artículo  
204  *   
205  * @param _stock Nuevo stock de tipo int  
206  * @throws ArtículoException Si el stock que se introduce es negativo  
207  */  
208 private void setStock(int _stock) throws ArtículoException {  
209   
210     if ( _stock > 0 ) {  
211         this.stock = _stock;  
212     } else {  
213   
214         this.stock = 0;  
215         throw new ArtículoException(message: "El stock no puede ser negativo");  
216     }  
217   
218 }  
219   
220 }  
221   
222 }  
223
```

```

90  |  /**
91  |  * Método para obtener el código de un Artículo
92  |  *
93  |  * @return Código de un artículo de tipo String
94  |  */
95  |  public String getCodigo() {
96  |      return codigo;
97  |  }
98  |
99  |  /**
100 |  * Método para establecer un nuevo código a un Artículo
101 |  *
102 |  * @param codigo Nuevo código del Artículo de tipo String
103 |  */
104 |  public void setCodigo(String codigo) {
105 |      this.codigo = codigo;
106 |  }
107 |
108 |  /**
109 |  * Método para obtener el nombre de un Artículo
110 |  *
111 |  * @return Nombre del Artículo de tipo String
112 |  */
113 |  public String getNombre() {
114 |      return nombre;
115 |  }
116 |
117 |  /**
118 |  * Método para establecer un nuevo nombre a un Artículo
119 |  *
120 |  * @param nombre Nuevo nombre del Artículo de tipo String
121 |  */
122 |  public void setNombre(String nombre) {
123 |      this.nombre = nombre;
124 |  }
125 |
126 |  /**
127 |  * Método para obtener el material de un Artículo
128 |  *
129 |  * @return Material del Artículo de tipo String
130 |  */
131 |  public String getMaterial() {
132 |      return material;
133 |  }
134 |
135 |  /**
136 |  * Método para establecer un nuevo material al Artículo
137 |  *
138 |  * @param material Nuevo material de tipo String
139 |  */
140 |  public void setMaterial(String material) {
141 |      this.material = material;
142 |  }
143 |
144 |  /**
145 |  * Método para obtener el color de un Artículo
146 |  *
147 |  * @return Color del Artículo de tipo String
148 |  */
149 |  public String getColor() {
150 |      return color;
151 |  }
152 |
153 |  /**
154 |  * Método para establecer un nuevo color a un Artículo
155 |  *
156 |  * @param color Nuevo color de tipo String
157 |  */
158 |  public void setColor(String color) {
159 |      this.color = color;
160 |  }
161 |
162 |  /**
163 |  * Método para obtener el precio de un Artículo
164 |  *
165 |  * @return Precio de un Artículo de tipo double
166 |  */
167 |  public double getPrecio() {
168 |      return precio;
169 |  }
170 |

```

- **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.**

```

42  /**
43   * Método equals de la clase
44   *
45   * @param obj Artículo a comparar
46   * @return True si dos artículos tienen el mismo código. False en caso
47   * contrario
48   */
49  @Override
50  public boolean equals(Object obj) {
51      if (this == obj) {
52          return true;
53      }
54      if (obj == null) {
55          return false;
56      }
57
58      final Artículo other = (Artículo) obj;
59      return Objects.equals(this.codigo, other.codigo);
60  }
61
62  /**
63   * Método toString de la clase
64   *
65   * @return Información del artículo en forma de cadena
66   */
67  @Override
68  public String toString() {
69      return "Código de artículo: " + this.codigo + ", Nombre del artículo: " + this.nombre + ", Material: " + this.material
70          + ", Color: " + this.color + ", Precio del artículo: " + this.precio + "€" + ", Stock: " + stock;
71  }
72

```

- **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**

```

72
73  /**
74   * Método que devuelve el precio total del stock de un artículo
75   *
76   * @return Valor total en formato double del stock de un artículo
77   */
78  public double totalValor() {
79
80      return (this.stock * this.precio);
81
82  }
83

```

- **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.**

```

83
84  /**
85   * Método mostrarInfo que se debe desarrollar en las Clases que hereden de
86   * Artículo
87   */
88  public abstract void mostrarInfo();
89

```

- 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.

```

1 package clases;
2
3 import excepciones.ArticuloException;
4
5 /**
6  * Clase Zapato
7  *
8  * @author Fran
9  */
10 public class Zapato extends Articulo {
11
12     private int numero;
13     private String tipo;
14
15     /**
16      * Constructor de la clase
17      *
18      * @param _codigo Código del zapato de tipo String
19      * @param _nombre Nombre del zapato de tipo String
20      * @param _material Material del zapato de tipo String
21      * @param _color Color del zapato de tipo String
22      * @param _precio Precio del zapato de tipo double
23      * @param _stock Stock del zapato de tipo int
24      * @param _numero Número del zapato de tipo int
25      * @param _tipo Tipo de zapato de tipo String
26      * @throws ArticuloException Si el precio introducido o el stock es negativo
27      */
28     public Zapato(String _codigo, String _nombre, String _material, String _color, double _precio, int _stock, int _numero, String _tipo) throws ArticuloException {
29         super(_codigo, _nombre, _material, _color, _precio, _stock);
30         this.numero = _numero;
31         this.tipo = _tipo;
32     }
33
34
35
36     /**
37      * Método toString de la clase
38      *
39      * @return Información del zapato en tipo cadena
40      */
41     @Override
42     public String toString() {
43         return super.toString() + ", Número: " + this.numero + ", Tipo: " + this.tipo;
44     }
45

```

```

46  /**
47   * Método mostrarInfo de la clase. Muestra por pantalla la información del
48   * zapato
49   */
50  @Override
51  public void mostrarInfo() {
52
53      System.out.println("==== Información del Zapato ====");
54      System.out.println("Código: " + super.getCodigo());
55      System.out.println("Nombre: " + super.getNombre());
56      System.out.println("Material: " + super.getMaterial());
57      System.out.println("Color: " + super.getColor());
58      System.out.println("Número: " + this.getNumero());
59      System.out.println("Tipo: " + this.getTipo());
60      System.out.println("Precio: " + super.getPrecio() + "€");
61      System.out.println("Stock: " + super.getStock());
62
63  }
64
65  /**
66   * Método para obtener el número del zapato
67   *
68   * @return Número del zapato de tipo int
69   */
70  public int getNumero() {
71      return numero;
72  }
73
74  /**
75   * Método para establecer un nuevo número al zapato
76   *
77   * @param numero Nuevo número de tipo int
78   */
79  public void setNumero(int numero) {
80      this.numero = numero;
81  }
82
83  /**
84   * Metodo para obtener el tipo de zapato
85   *
86   * @return Tipo de zapato de tipo String
87   */
88  public String getTipo() {
89      return tipo;
90  }
91
92  /**
93   * Metodo para establecer un nuevo tipo de zapato
94   *
95   * @param tipo Nuevo tipo de tipo String
96   */
97  public void setTipo(String tipo) {
98      this.tipo = tipo;
99  }
100
101  }

```

```

1  package clases;
2
3  import excepciones.ArticuloException;
4
5  /**
6   * Clase Bolso
7   *
8   * @author Fran
9   */
10 public class Bolso extends Articulo {
11
12     /**
13      * Enumerado de los tipos de bolso disponibles
14      */
15     public enum TipoBolso {
16
17         BANDOLERA, MOCHILA, FIESTA;
18
19     }
20
21     private String marca;
22     private TipoBolso tipo;
23
24     /**
25      * Constructor de la clase
26      *
27      * @param _codigo Código del bolso de tipo String
28      * @param _nombre Nombre del bolso de tipo String
29      * @param _material Material del bolso de tipo String
30      * @param _color Color del bolso de tipo String
31      * @param _precio Precio del bolso de tipo double
32      * @param _stock Stock del bolso de tipo int
33      * @param _marca Marca del bolso de tipo String
34      * @param _tipo Tipo de bolso de tipo TipoBolso (Bandolera, Mochila, Fiesta)
35      * @throws ArticuloException Si se introduce un precio o stock negativo, o
36      * un tipo de bolso no válido
37      */
38     public Bolso(String _codigo, String _nombre, String _material, String _color, double _precio, int _stock, String _marca, String _tipo) throws ArticuloException {
39
40         super(_codigo, _nombre, _material, _color, _precio, _stock);
41         this.marca = _marca;
42         setTipo(_tipo);
43
44     }
45
46     /**
47      * Metodo toString de la clase
48      *
49      * @return Información del bolso en una cadena
50      */
51     @Override
52     public String toString() {
53         return super.toString() + ", Marca: " + this.marca + ", Tipo: " + this.tipo;
54     }
55 }

```

```

56  /**
57   * Método mostrarInfo de la clase. Muestra por pantalla la información del
58   * bolso
59   */
60  @Override
61  public void mostrarInfo() {
62
63      System.out.println("==== Información del Bolso ====");
64      System.out.println("Código: " + super.getCodigo());
65      System.out.println("Nombre: " + super.getNombre());
66      System.out.println("Material: " + super.getMaterial());
67      System.out.println("Color: " + super.getColor());
68      System.out.println("Marca: " + this.getMarca());
69      System.out.println("Tipo: " + this.getTipo());
70      System.out.println("Precio: " + super.getPrecio() + "€");
71      System.out.println("Stock: " + super.getStock());
72  }
73
74  /**
75   * Método para obtener la marca del bolso
76   *
77   * @return Marca del bolso de tipo String
78   */
79  public String getMarca() {
80      return marca;
81  }
82
83  /**
84   * Método para establecer una nueva marca al bolso
85   *
86   * @param marca Nueva marca de tipo String
87   */
88  public void setMarca(String marca) {
89      this.marca = marca;
90  }
91
92  /**
93   * Método para obtener el tipo de bolso
94   *
95   * @return Tipo de bolso de tipo String
96   */
97  public String getTipo() {
98      return this.tipo.toString();
99  }
100
101  /**
102   * Método para establecer un nuevo tipo de bolso
103   *
104   * @param _tipo Nuevo tipo de tipo String
105   * @throws ArtículoException Si el tipo de bolso no se encuentra entre
106   *   Bandolera, Mochila o Fiesta
107   */
108  public void setTipo(String _tipo) throws ArtículoException {
109
110      try {
111
112          this.tipo = TipoBolso.valueOf(name.ConvierteCadena(_tipo));
113
114      } catch (IllegalArgumentException ex) {
115
116          throw new ArtículoException(message:"El tipo de bolso elegido no es válido");
117
118      }
119  }
120
121  /**
122   * Método privado de la clase que convierte una cadena en vocal y le quita
123   * las tildes que pueda llevar
124   *
125   * @param entrada Cadena a convertir
126   * @return Cadena en mayúscula y sin tildes
127   */
128  private String convierteCadena(String entrada) {
129
130      String cadena = entrada.toUpperCase();
131
132      String cadenaUtil = cadena.replace(target:"Á", replacement:"A").replace(target:"È", replacement:"E").replace(target:"Ì", replacement:"I")
133          .replace(target:"Ò", replacement:"O").replace(target:"Ù", replacement:"U");
134
135      return cadenaUtil;
136  }
137
138  }
139
140
141

```

- Clase utilizada para las excepciones del Artículo.

```

1 package excepciones;
2
3 /**
4  * Clase utilizada para las excepciones de Artículo
5  *
6  * @author Fran
7  */
8 public class ArtículoException extends Exception {
9
10     public ArtículoException(String message) {
11         super(message);
12     }
13
14 }

```

- 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.

```

1 package clases;
2
3 import excepciones.FechaException;
4 import java.time.LocalDate;
5 import java.time.format.DateTimeFormatter;
6 import java.time.format.DateTimeParseException;
7 import java.time.temporal.ChronoUnit;
8 import java.util.ArrayList;
9
10 /**
11  * Clase Tienda
12  *
13  * @author Fran
14  */
15 public class Tienda {
16
17     private String nombreTienda;
18     private LocalDate fechaApertura;
19     private ArrayList<Articulo> articuloTienda;
20 }

```

- 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.

```

21 /**
22  * Constructor de la clase con todos los parámetros
23  *
24  * @param _nombreTienda Nombre de la tienda de tipo String
25  * @param _fechaApertura Fecha de apertura de la tienda de tipo String
26  *      formato dd/MM/yyyy
27  * @param _articuloTienda Artículos de la tienda de tipo ArrayList
28  * @throws FechaException Si la fecha de apertura es mayor a 90 días después
29  *      de la fecha actual
30  */
31 public Tienda(String _nombreTienda, String _fechaApertura, ArrayList<Articulo> _articuloTienda) throws FechaException {
32     creaTienda(_nombreTienda, _fechaApertura, _articuloTienda);
33 }
34
35 }

```

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

```

37  /**
38   * Constructor de la clase sin la fecha de apertura. Se establece fecha
39   * actual
40   *
41   * @param _nombreTienda Nombre de la tienda de tipo String
42   * @param _articuloTienda Artículos de la tienda de tipo ArrayList
43   * @throws FechaException Si la fecha de apertura es mayor a 90 días después
44   * de la fecha actual
45   */
46  public Tienda(String _nombreTienda, ArrayList<Articulo> _articuloTienda) throws FechaException {
47      creaTienda(_nombreTienda, _fechaapertura: fechaAct(), _articuloTienda);
48  }
49
50
51

```

- 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 a $(4 * 13,10) + (2 * 29,25) = 52,4 + 58,5 = 110,90$.

```

68  /**
69   * Método que calcula el valor total de género del que dispone
70   *
71   * @return Valor total del género de tipo double
72   */
73
74  public double valorTienda() {
75
76      double total = 0;
77
78      for (Articulo a : articuloTienda) {
79          total += a.totalValor();
80      }
81
82      return total;
83
84  }
85
86
87

```

- 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).

```

88  /**
89   * Método que muestra la información de la tienda por pantalla
90   */
91
92  public void muestraInfo() {
93
94      System.out.println("**** Artículos de la tienda ****");
95
96      for (Articulo a : articuloTienda) {
97          System.out.println(a + ", Tipo de Objeto: " + a.getClass().getSimpleName());
98      }
99
100  }
101
102

```


○ Métodos auxiliares de Tienda

```

52  /**
53   * Método auxiliar para crear la tienda en el constructor
54   *
55   * @param _nombreTienda Nombre de la tienda
56   * @param _fechaApertura Fecha de apertura
57   * @param _articuloTienda Artículos de la tienda
58   * @throws FechaException Si la fecha de apertura es mayor a 90 días después
59   * de la fecha actual
60   */
61  private void creaTienda(String _nombreTienda, String _fechaApertura, ArrayList<Articulo> _articuloTienda) throws FechaException {
62
63      this.nombreTienda = _nombreTienda;
64      setFechaApertura(_fechaApertura);
65      this.articuloTienda = new ArrayList<>(_articuloTienda);
66
67  }
68

```

```

103 /**
104  * Método auxiliar para convertir fechas de tipo dd/MM/yyyy a tipo LocalDate
105  *
106  * @param fecha Fecha de tipo String dd/MM/yyyy
107  * @return Fecha de tipo LocalDate yyyy-MM-dd
108  */
109  private LocalDate fechaEsp(String fecha) {
110
111      DateTimeFormatter fechaCadena = DateTimeFormatter.ofPattern(pattern: "dd/MM/yyyy");
112
113      return LocalDate.parse(text: fecha, formatter: fechaCadena);
114
115  }
116
117 /**
118  * Método auxiliar para calcular la fecha actual de tipo String dd/MM/yyyy
119  *
120  * @return Fecha actual en String dd/MM/yyyy
121  */
122  private String fechaAct() {
123
124      LocalDate fechaLocalDate = LocalDate.now();
125      DateTimeFormatter formato = DateTimeFormatter.ofPattern(pattern: "dd/MM/yyyy");
126      String fecha = fechaLocalDate.format(formatter: formato);
127
128      return fecha;
129
130  }
131
132 /**
133  * Método auxiliar para calcular la fecha entre un día pasado por parámetro
134  * y la fecha actual
135  *
136  * @param fecha Fecha de tipo LocalDate
137  * @return Diferencia entre fecha actual y la pasada por parámetro en días
138  */
139  private static long diferenciaDias(LocalDate fecha) {
140
141      LocalDate fechaAct = LocalDate.now();
142      LocalDate fechaIn = fecha;
143
144      long dias = ChronoUnit.DAYS.between(temporal: fechaAct, temporal2Exclusive: fechaIn);
145
146      return dias;
147
148  }

```

○ Getter y setters

```

150  /**
151   * Método para obtener el nombre de la tienda
152   *
153   * @return Nombre de la tienda de tipo String
154   */
155  public String getNombreTienda() {
156      return nombreTienda;
157  }
158
159  /**
160   * Método para establecer un nuevo nombre a la tienda
161   *
162   * @param nombreTienda Nuevo nombre de la tienda de tipo String
163   */
164  public void setNombreTienda(String nombreTienda) {
165      this.nombreTienda = nombreTienda;
166  }
167
168  /**
169   * Método para obtener la fecha de apertura de la tienda
170   *
171   * @return Fecha de apertura de la tienda de tipo LocalDate
172   */
173  public LocalDate getFechaApertura() {
174      return fechaApertura;
175  }
176
177  /**
178   * Método para establecer una nueva fecha de apertura de la tienda
179   *
180   * @param fechaApertura Nueva fecha de apertura de tipo String dd/MM/yyyy
181   * @throws FechaException Si la fecha introducida no es del tipo dd/MM/yyyy
182   */
183  public void setFechaApertura(String _fechaApertura) throws FechaException {
184
185      try {
186
187          LocalDate fechaRecibida = fechaEsp(_fechaApertura);
188
189          if (diferenciaDias(fechaEsp(fechaApertura), fechaRecibida) < 90) {
190
191              this.fechaApertura = fechaRecibida;
192
193          } else {
194
195              throw new FechaException(message: "Error. La fecha de apertura tiene que ser como máximo 90 días posterior a la fecha actual");
196
197          }
198
199      } catch (DateTimeParseException fechaEx) {
200
201          throw new FechaException(message: "Error. El formato de la fecha debe ser dd/MM/yyyy");
202
203      }
204
205  }
206
207  /**
208   * Método para obtener los artículos de la tienda
209   *
210   * @return Artículos de la tienda de tipo ArrayList
211   */
212  public ArrayList<Articulo> getArticuloTienda() {
213      return articuloTienda;
214  }
215
216  /**
217   * Método para establecer nuevos artículos a la tienda
218   *
219   * @param articuloTienda Nuevos artículos de la tienda de tipo ArrayList
220   */
221  public void setArticuloTienda(ArrayList<Articulo> articuloTienda) {
222      this.articuloTienda = articuloTienda;
223  }
224
225  }
226

```

○ Clase utilizada para las excepciones de la tienda

```

1  package excepciones;
2
3  /**
4   *
5   * @author Fran
6   */
7  public class FechaException extends Exception {
8
9      public FechaException(String message) {
10         super(message);
11     }
12
13
14

```

○ Pruebas de tienda

```

1 package com.myccompany.garcia_cutillas_franciscojose_actut10_perrera;
2
3 import clases.Articulo;
4 import clases.Bolso;
5 import clases.Tienda;
6 import clases.Zapato;
7 import excepciones.ArticuloException;
8 import excepciones.FechaException;
9 import java.util.ArrayList;
10
11 /**
12  *
13  * @author Fran
14  */
15 public class Garcia_Cutillas_FranciscoJose_ActUti0_Tienda {
16
17     public static void main(String[] args) {
18
19         try {
20
21             Zapato z1 = new Zapato(_codigo: "1", _nombre: "Zapato boda", _material: "Piel", _color: "Negro", _precio: 84.99, _stock: 5, _numero: 43, _tipo: "Formal");
22             Zapato z2 = new Zapato(_codigo: "2", _nombre: "Zapato estándar", _material: "Polipiel", _color: "Marrón", _precio: 39.99, _stock: 50, _numero: 41, _tipo: "Casual");
23
24             z2.mostrarInfo();
25
26             Bolso b1 = new Bolso(_codigo: "23", _nombre: "Bolso beige", _material: "Esparto", _color: "Marrón", _precio: 19.99, _stock: 10, _marca: "Bomba y Loli", _tipo: "Fiesta");
27             Bolso b2 = new Bolso(_codigo: "24", _nombre: "Bolso piel", _material: "Polipiel", _color: "Rojo", _precio: 59.99, _stock: 3, _marca: "Adidas", _tipo: "Mochila");
28
29             System.out.println("");
30             b1.mostrarInfo();
31             System.out.println("");
32             System.out.println("Total stock bolso con código " + b1.getCodigo() + ": " + b1.getTotalValor() + " Euros");
33
34             ArrayList<Articulo> articulos = new ArrayList<> ();
35
36             articulos.add(z1);
37             articulos.add(z2);
38             articulos.add(b1);
39             articulos.add(b2);
40
41
42             Tienda t1 = new Tienda (_nombreTienda: "Boutique Archena", _fechaApertura: "23/05/2023", _articulosTienda: articulos);
43             System.out.println("");
44             t1.mostrarInfo();
45             System.out.println("");
46             System.out.println("Valor del género de la tienda: " + t1.valorTienda() + " Euros");
47
48
49             } catch (ArticuloException aEx) {
50
51                 System.out.println(aEx.getMessage());
52
53             } catch (FechaException fEx) {
54
55                 System.out.println(fEx.getMessage());
56
57             }
58         }
59     }
60 }

```

```

--- exec-maven-plugin:3.0.0:exec (default-cli) @ Garcia_Cutillas_FranciscoJose_ActUti0_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

```