Módulo Programación.

1º DAW.

PRÁCTICA 1: Estructuras de datos avanzadas y excepciones.
UNIDAD DE TRABAJO 6.
Profesor: Pedro Antonio Santiago Santiago.

1. Introducción.

Las colecciones, la generalización, las interfaces funcionales y los “Streams” son características claves y muy usadas para la creación de aplicaciones. En esta práctica se crea una aplicación sencilla para la gestión de una tienda reducida en el que los datos se almacenan en colecciones en memoria.

2. Enunciado.

Se desea crear una aplicación para la gestión de una tienda, la tienda posee un conjunto de categorías de productos, es posible crear, borrar y actualizar categorías, de igual forma se han de poder crear, borrar y actualizar productos, cada uno de ellos asociados a una categoria, además se desea poder gestionar tanto de forma independiente los productos como los productos asociados a una categoria.

Además se tiene clientes, estos clientes pueden realizar pedidos que pasan por diferentes fases antes de la entrega, estos pedidos poseen a su vez línea de pedido compuestas de un producto y un pedido,

La tienda dispone de un conjunto de productos que poseen:

- Name de tipo “String”.
- Id de tipo “int”.
- Image_path de tipo “String”.
- Active de tipo “boolean”.
- Description de tipo “String”.
- Category de tipo “Category”.
- Price de tipo “float”.

Además estos productos se encuentra clasificados por categorias, cada categoria se define como:

- Name de tipo “String”.
- Id de tipo “int”

- Image_path de tipo “String”
- Products que es una colección de productos.

También se tienen clientes, de los que se gestiona:

- Name de tipo “String”.
- Surname de tipo “String”.
- Password de tipo “String”
- Active de tipo “boolean”.
- Id de tipo “int”.
- Colección de pedidos “Orders”.

Por supuesto cada cliente puede realizar diferentes pedidos, cada pedido a su vez posee:

- State de tipo enumerado con los valores WAITING, PREPARING, READY, DELIVERY, DELIVERED, CANCELED.
- Date de tipo “date” que es la fecha en que se realizo el pedido.
- Id de tipo “int”
- Client de tipo “Client”.
- Colección de líneas de pedido que contienen objetos de la clase “OrderLine”.

Por último se tiene las línea de pedido con los atributos siguientes:

- Product de tipo “Product”.
- Order de tipo “Order”.
- Price de tipo “float”, lo obtiene de Product, pero puede cambiar en el producto en el futuro.
- Units de tipo “int”.
- Id de tipo “int”.

En la aplicación se va a usar una arquitectura que permita gestionar todas las entidades de la forma más sencilla que se pueda, y que se encuentre preparada en la medida de lo posible para los cambios. Por ejemplo, en este caso los datos se encuentran en memoria,

pero en un futuro se quiere almacenar la información en ficheros o en una base de datos, o en vez de utilizar la línea de comandos como interfaz con el usuario, cambiar a una aplicación gráfica o incluso a una web,

El primer elemento de esta arquitectura son los repositorios, que se pueden ver como almacenes de las entidades o gestores de estos almacenes, la gestión de dicho almacen es resposabilidad de esta clase pudiendo ser en memoria, en una base de datos relacional, NO SQL, servidor remoto… de forma que no sea necesario cambiar nada en el resto de la aplicación cuando se cambie el origen de los datos.

Se define una interfaz para los repositorios llamada Irepository:

```
public interface IRepository<T, K, ID> {
    public void add(T item);
    public void update(T item);
    public void removeById(ID id);
    public void remove(T item);
    public T getById(ID key);
    public List<T> getAll();
    public int count();
    public boolean existsById(ID key);
    public List<T> find(K find);
    public List<T> find(Object... arg);
}
```

La interfaz utiliza generics en el que el primero (T) es la clase que se va a gestionar, el segundo (K) es el tipo del atributo más destacado de la entidad, y el último el ID que identifica de forma única al objeto, normalmente un “Integer” aunque puede ser otro o incluso compuesto.

Sobre esta interfaz se definen el resto de repositorios, por ejemplo para las categorías u otros elementos se ha definido un repositorio en memoria que internamente es una Map de Java:

```
public  abstract class AMapRepository<T, K,ID> implements IRepository<T, K,ID>
{
   private HashMap<K, T> elements;
    public AMapRepository() {
        this.elements= new HashMap<>();
```

```
    }
    protected HashMap<K, T> getElements() {
        return elements;
    }
    protected void setElements(HashMap<K, T> elements) {
        this.elements = elements;
    }
    public T getByKey(K key){
        return this.elements.get(key);
    }

    @Override
    public List<T> getAll() {
        return new ArrayList<T>((Collection<? extends T>)
this.getElements().values());
    }

    @Override
    public int count() {
        return this.getElements().size();
    }
    @Override
    public void update(T item){
    }
}
```

La siguiente capa se encarga de interactuar con los repositorios y realizar la lógica de la aplicación, como puede ser comprobar que un producto no se encuentra ya en un pedido, o validar los datos antes de introducirlos en el repositorio como puede ser comprobar que al dar de alta un usuario en nombre no es nulo. Se definen como servicios, e internamente poseen uno o varios repositorios y de forma opcionar otros servicios que sean necesarios, por ejemplo al añadir un nuevo producto, también se ha de añadir a las categorias.

Uno de los problemas que puede surgir es la de existncia de varios repositorios que representen lo mismo ya que al hacer new se crea un nuevo repositorio, existen diferentes alternativas para impedir esto, una de ellas es el uso del patrón “Singleton” que garantiza que solo va a existir una instancia de una clase en toda la VM. Se puede hacer tanto el

repositorio como el servicio Singleton, en este caso se opta por hacer Singleton los servicios.

Un extracto (faltan métodos y atributos) del servicio de Categories, que se encarga de la gestión de las categorias:

```
public class CategoriesService {
    private static CategoriesService service;
    private ProductsService products_service;
    private CategoryRepository repository;

    private CategoriesService() { //constructor privado
        super();
        this.repository = new CategoryRepository();
        this.products_service = ProductsService.getIntance();
        this.init();
    }
    //valores iniciales
    private void init() {
        Category tempo;

        tempo = new Category();
        tempo.setImage_path("uno.png");
        tempo.setName("electrodomésticos");
        tempo.setId(tempo.getName().hashCode());
        this.add(tempo);

        Product p = new Product();
        p.setActive(true);
        p.setCategory(tempo);
        p.setDescription("Lavadora de 5 Kg blanca");
        p.setImage_path("imagen.png");
        p.setPrice(354.55f);
        p.setName("Lavadora NG");
        this.products_service.add(p);
        tempo.add(p);

        p = new Product();
        p.setActive(true);
```

```
        p.setCategory(tempo);
        p.setDescription("Frigorífico dos puertas");
        p.setImage_path("imagen.png");
        p.setPrice(610.22f);
        p.setName("Frigorífico Nuzassi");

        this.products_service.add(p);
        tempo.add(p);

        //nueva categoria
        tempo = new Category();
        tempo.setImage_path("dos.png");
        tempo.setName("ordenadores");
        tempo.setId(tempo.getName().hashCode());
        this.add(tempo);
        //se anyade producto
        p = new Product();
        p.setActive(true);
        p.setCategory(tempo);
        p.setDescription("Dell sobremesa I5 con 16 GB de RAM y SSD de 1TB");
        p.setImage_path("imagen.png");
        p.setPrice(510.22f);
        p.setName("Dell");
        this.products_service.add(p);
        tempo.add(p);

        //se anyade producto
        p = new Product();
        p.setActive(true);
        p.setCategory(tempo);
        p.setDescription("Lenovo Ryzen 7, 12 GB de RAM y SSD de 2TB");
        p.setImage_path("imagen.png");
        p.setPrice(610.22f);
        p.setName("Lenovo");
        this.products_service.add(p);
        tempo.add(p);
    }
    //método estático de la clase para que sea Singleton
    public static CategoriesService getIntance() {
        if (service == null) {
```

```
            service = new CategoriesService();
        }
        return service;
    }

    public List<Category> getAll() {
        return this.repository.getAll();
    }

    public Category getById(int id) {
        return this.repository.getById(id);
    }

    public Category getByName(String name) {
        return this.repository.getByKey(name);
    }

    public List<ValidateError> add(Category item) {
        ArrayList<ValidateError> errors = new ArrayList();
        if (item.getImage_path() == null) {
            errors.add(new ValidateError("ImagePath", " is null"));
        }
        if (item.getName() == null) {
            errors.add(new ValidateError("Name", " is null"));
        }
        if (errors.isEmpty()) {
            item.setId(item.getName().hashCode());
            this.repository.add(item);
        }
        return errors;
    }

    public List<Category> find(String s) {

        return this.repository.find(s);
    }

    public int count() {
        return this.repository.count();
    }
```

```
    public void remove(Category item) {

        this.repository.remove(item);
        //se borran los productos asociados
        item.getProducts().forEach(p -> {
            this.products_service.remove(p);
        });
    }

    public void remove(String name) {
        Category item = this.repository.getByKey(name);
        if (item != null) {
            this.repository.removeByKey(name);
            item.getProducts().forEach(p -> {
                this.products_service.remove(p);
            });
        }
    }

    public void update(Category item) {
        this.repository.removeById(item.getId());
        this.repository.add(item);
    }

}
```

Esta aplicación es sencilla y la lógica de la misma hace que el servicio se limite poco más que a llamar a la interfaz Irepository, pero por ejemplo en el método “add” se valida el objeto antes de insertarlo en el repositorio, devolviendo una lista de errores.

```
 public List<ValidateError> add(Category item) {
        ArrayList<ValidateError> errors = new ArrayList();
        if (item.getImage_path() == null) {
            errors.add(new ValidateError("ImagePath", " is null"));
        }
        if (item.getName() == null) {
            errors.add(new ValidateError("Name", " is null"));
        }
        if (errors.isEmpty()) {
            item.setId(item.getName().hashCode());
```

```
            this.repository.add(item);
        }
        return errors;
    }
```

Dependiendo del tipo de entidad del modelo (categoría, cliente, pedido, producto o linea de pedido) se tiene dos servicios, uno o ninguno. El el caso de categorias, se dispone del servicio CategoriesServices, encargado de la gestión de todas las categorias y CategoryService encargado de una categoria en concreto, encargandose entre otras funciones de la gestión de los productos asociados. La clase CategoryService:

```
public class CategoryService {
    private static CategoryService service;
    private Category selected;
    private ProductsService productservice;

    private CategoryService() {
        super();
        this.productservice = ProductsService.getIntance();
    }
      public static CategoryService getIntance() {
        if (service == null) {
            service = new  CategoryService();
        }
        return service;
    }
      public Category getSelected(){
          return this.selected;
      }
    public Product getProductById(int id) {
        return this.selected.getProductById(id);
    }
    public Product getProductByName(String name){
        return this.selected.getProductByName(name);
    }

    public void addProduct(Product item) {
        this.selected.add(item);
        item.setCategory(selected);
```

```
       this.productservice.add(item);
    }
     public List<ValidateError> addProduct(String name, String
description,String image, boolean active, float price){
          ArrayList<ValidateError> errors = new ArrayList();
        if (name == null) {
            errors.add(new ValidateError("Name", " is null"));
        }
        if (description == null) {
            errors.add(new ValidateError("Description", " is null"));
        }

        if (image == null) {
            errors.add(new ValidateError("Image", " is null"));
        }

        if (price<= 0) {
            errors.add(new ValidateError("Price", " is  less or equals 0"));
        }
        if (errors.isEmpty()) {
            //se crea el producto y se llama para que lo inserte
            Product p= new Product();
            p.setActive(active);

            p.setDescription(description);
            p.setImage_path(image);
            p.setName(name);
            p.setPrice(price);
            this.addProduct(p);

        }
        return errors;
    }
    public List<Product> getProducts(){
        return this.selected.getProducts();
    }

    public List<Product> find(String s) {

        return this.selected.findProductsByName(s);
```

```
    }

    public int countProducts() {
        return this.selected.getProducts().size();
    }

    public void removeProduct(Product item) {
        this.selected.removeProduct(item);
        this.productservice.remove(item);
    }

    public void removeProduct(String name){

        this.selected.removeProduct(this.selected.getProductByName(name));
        this.productservice.remove(name);
    }
    public void setSelected(Category category){
        this.selected=category;
    }
}
```

Por último se tiene el controlador, en este caso se encarga de adaptar las peticiones del exterior para que las pueda recibir el servicio. La interfaz con el usuario es la línea de comandos haciendo uso de la librería asg.cliche, en Maven: https://mvnrepository.com/artifact/com.googlecode.clichemaven/cliche/110413, que se basa en la definición de menús y submenus en línea de comandos, definiciendo métodos que se transforman en comando de forma sencilla.

El programa principal:

```
public class DAWShop implements ShellDependent {

    private Shell shell;
    private CategoriesShellController categoriescontroller;
    private ProductsShellController productscontroller;
    private ClientsShellController clientscontroller;
    @Command
   public void categories() throws IOException{
       if(this.categoriescontroller==null)
```

```
            this.categoriescontroller=new CategoriesShellController();
        ShellFactory.createSubshell("Categories", this.shell, "Categories",
                this.categoriescontroller).commandLoop();

    }
    @Command
    public void clients() throws IOException{
       if(this.clientscontroller==null)
            this.clientscontroller=new ClientsShellController();
        ShellFactory.createSubshell("Clients", this.shell, "Clients",
                this.clientscontroller).commandLoop();

    }
     @Command
    public void products() throws IOException{
        if(this.productscontroller==null)
            this.productscontroller=new ProductsShellController();
        ShellFactory.createSubshell("Products", this.shell, "Products",
                this.productscontroller).commandLoop();

    }

     public static void main(String[] args) throws IOException {
         ShellFactory.createConsoleShell("DAW>", "?list to list all
commands\\\"", new DAWShop()) .commandLoop();
     }

     @Override
     public void cliSetShell(Shell shell) {
         this.shell = shell;
     }
}
```

El controlador para las categorias:

```
1. public class CategoriesShellController implements ShellDependent {
2.     private Shell shell;
3.     private String name = "categorias";
4.     private CategoryShellController categoryshell;
5.     private CategoriesService service;
6.
```

```
7.     public CategoriesShellController() {
8.         this.service = CategoriesService.getIntance();
9.
10.          }
11.
12.          @Command
13.          public void add(String name, String image_path) {
14.              Category c = new Category();
15.              c.setName(name);
16.              c.setImage_path(image_path);
17.              this.service.add(c);
18.          }
19.
20.          @Command
21.          public List<Category> getAll() {
22.              return this.service.getAll();
23.          }
24.
25.          @Command(description = "Devuelve una categoría por id")
26.          public Category get(
27.                  @Param(name = "Id", description = "Id de la categoria")
   int id) {
28.              return this.service.getById(id);
29.          }
30.
31.          @Command
32.          public void remove(String c) {
33.              this.service.remove(c);
34.          }
35.
36.          @Command(description = "Selecciona una categoria en funcíon del
   nombre")
37.          public void select(String name) {
38.              Category c = this.service.getByName(name);
39.              if (c != null) {
40.                  if (this.categoryshell == null) {
41.                      try {
42.                          this.categoryshell = new
   CategoryShellController();
43.                          this.categoryshell.setSelected(c);
44.
```

```
45.                         ShellFactory.createSubshell("Category",
   this.shell, name,
46.                                 this.categoryshell).commandLoop();
47.                     } catch (IOException ex) {
48.         Logger.getLogger(CategoriesShellController.class.getName()).log(Leve
   l.SEVERE, null, ex);
49.                     }
50.                 }
51.             }
52.         }
53.             @Command
54.             public List<Category> findByName(String name) {
55.                 return this.service.find(name);
56.             }
57.             @Command
58.             public void updateCategoryName(Category item) {
59.                 this.service.update(item);
60.
61.             }
62.             @Override
63.             public void cliSetShell(Shell shell) {
64.                 this.shell = shell;
65.             }
66.
67.         }
```

Observar en la línea 36 como al seleccionar una categoría se entra en el subshell de categoria. Un ejemplo de ejecución:

```
?list to list all commands\"
DAW>> ?list
abbrev      name  params
p     products    ()
c     categories  ()
cl    clients     ()
DAW>> c
Categories
DAW>/Categories> ga
ID:-1172194798 NAME:electrodomésticos IMAGE:uno.png
ID:-1533487042 NAME:ordenadores IMAGE:dos.png
DAW>/Categories> s ordenadores
ordenadores
```

```
DAW>/Categories/Category> ?list
abbrev      name  params
gs    get-selected      ()
ap    add-product (p1, p2, p3, p4, p5)
gp    get-products      ()
rp    remove-product    (p1)
fpbn  find-products-by-name   (p1)
DAW>/Categories/Category> gp
ID:2126305 NAME:Dell DESCRIPTION:Dell sobremesa I5 con 16 GB de RAM y SSD de
1TB ACTIVE:true IMAGE:imagen.png CATEGORYordenadores
ID:-2022488749 NAME:Lenovo DESCRIPTION:Lenovo Ryzen 7, 12 GB de RAM y SSD de
2TB ACTIVE:true IMAGE:imagen.png CATEGORYordenadores
DAW>/Categories/Category>
```

3. Desarrollo de la práctica.

3.1. Crear las entidades del dominio.

- Categoria.
- Producto.
- Cliente.
- Pedido.
- Linea de pedido.

Justificar la elección de las estructuras internas, por ejemplo, una categoría posee una colección de productos. Las clases que tienen colecciones internas deberán proveer de métodos para la gestión de estas clases internas.

3.2. Definición de los repositorios.

A partir de la interfaz IRepository crear repositorios abstractos para la gestión de las entidades, internamente los repositorios abstractos implementarán alguna de las colecciones vistas en teoría, justificar la elección, por ejemplo en el caso de categorías se utiliza un Map por el nombre de la misma.

Los repositorios a crear son:

- Categorías.
- Productos.
- Clientes.
- Pedidos.

De líneas de orden no se crea repositorio.

3.3. Definición de los servicios.

Crear ahora los servicios para la aplicación, recordar que los servicios poseen la lógica de la aplicación. Se han de crear:

- CategoriasServices.
- CategoriaService.
- ProductsServices.
- ProductService.
- ClientsService.
- ClientService.
- OrdersService.
- OrderService.

Dependiendo de cada servicio, se tendrá que hacer referencia a otros servicios, y contendrá uno o varios repositorios en función de las necesidades.

3.4. Definición de los controladores.

- Será necesario crear controladores con comandos para los servicios anteriores.

3.5. Uso de streams.

Es necesario añadir el código en los servicios y controladores para usando stream obtener:

- Clientes con al menos un pedido.
- Categorias activas.

- Listado de nombres de categorias inactivas.
- Productos con un precio entre 2 valores.
- El total gastado por un cliente.
- Pedidos que se encuentran en un estado concreto.
- Pedidos que superan una cantidad.
- Pedidos con más de 10 unidades en total.

4. Entrega.

La práctica se entrega en formato ZIP con 2 ficheros (código y documento en formato PDF, en el campus virtual ww.aules.edu.gva.es (pendiente de matricula e inicio de curso). Se fijará la fecha en AULES.

El fichero 1 será el código generado comprimido a su vez también en zip, **importante comentar el código.** Cada una de las partes en un proyecto Maven **(en caso de no ser correcto no se corrige la práctica).**

El fichero 2 es el documento de entrega ha de tener los siguientes puntos.

1. Portada. (Título de la práctica y autor).
2. Introducción.
3. Desarrollo de la práctica. **Documento de texto con las partes más importantes comentadas, estrategias aplicacadas, capturas de ejecución y respuestas a las preguntas si las hay.**
4. Conclusiones. (Pequeño comentario sobre la práctica: dificultad, problemas encontrados…).

5. Evaluación.

Unos días después de la entrega se realizá la corrección de forma presencial donde el profesor preguntará cuestiones sencillas sobre la práctica para comprobar la autoría de la misma.

CRITERIOS DE EVALUACIÓN:

- CE6b. Se han reconocido las librerías de clases relacionadas con tipos de datos avanzados.
- CE6c. Se han utilizado listas para almacenar y procesar información.
- CE6d. Se han utilizado iteradores para recorrer los elementos de las listas.
- CE6e. Se han reconocido las características y ventajas de cada una de la colecciones de datos disponibles.
- CE6f. Se han creado clases y métodos genéricos.

FUNCIONALIDAD	PUNTUACIÓN
Se implementan todos los elementos del dominio, justiicando las estructuras y colecciones seleccionadas.	2 puntos
Se definen los repositorios necesarios haciendo uso de la herencia, las colecciones y las clases genericas.	2,25 puntos
Gestiona la lógica del dominio desde los servicios.	1 punto
La apliación cumple con la funcionalidad esperada y gestiona correctamente los elementos usando los servicios, repositorios y colecciones.	1,5 puntos
Usando los streams completa las peticiones de información solicitadas.	1,5 puntos.
El borrado de alguna de las entidades del dominio se hace correctamente.	1 punto
Define los controladores y se puede realizar una navegación por las opciones.	0,5 puntos
Comenta el código	0,25 puntos