UT04 Práctica: Gestión de restaurantes

Vamos a construir la estructura de objetos necesaria para implementar un pequeño framework que represente la estructura de datos para soportar la **gestión de los platos de una cadena de restaurantes**. En concreto, el sistema estará preparado gestionar los platos y los menús de la cadena de restaurantes, los cuales estarán organizados en categorías e indicando los posibles alérgenos que pudieran tener.

A continuación, se detalla los objetos que debemos implementar junto con su funcionalidad. Tenemos que tener en cuenta que debemos implementar también la estructura de objetos necesaria para gestionar las excepciones que genere la aplicación.

Para el desarrollo de la práctica y su superación debemos utilizar la herramienta de gestión de versiones **GIT** e ir guardando una copia de la versión realizada en **GitHub** de forma obligatoria. La práctica podrá considerarse no válida si no se ha utilizado dichas herramientas.

1. Listado de objetos

El siguiente listado son objetos de entidad, es decir, objetos planos que tan solo tienen propiedades que almacenar, pero no tienen relación con otros objetos. Cada objeto debe disponer de las propiedades getter y setter correspondientes y añadir los métodos que pudieran ser de utilidad, en principio será obligatorio un `toString()`.

1.1. Objeto Dish

Objeto para identifica los datos de un plato. Sus propiedades y métodos serán:

Propiedad	Tipo	Obligatorio	Descripción
name	String	Si	Nombre del plato.
description	String	No	Descripción
ingredients	[String]	No	Array con los posibles ingredientes que componen el plato.
image	String	No	String con la ruta donde está ubicada la imagen del plato.

Tabla 1 Descripción del objeto Dish

1.2. Objeto Category

Con este objeto podemos crear la estructura de categorías. Sus propiedades y métodos serán:

Propiedad	Tipo	Obligatorio	Descripción
name	String	Si	Nombre de la categoría.
description	String	No	Descripción de la categoría.

Tabla 2 Descripción del objeto Category

1.3. Objeto Allergen

Representa los alérgenos que pueden tener un determinado plato.

Propiedad	Tipo	Obligatorio	Descripción
name	String	Si	Nombre del alérgeno.
description	String	No	Descripción del alérgeno.

Tabla 3 Descripción del objeto Allergen

1.4. Objeto Menu

Se trata de una agregación de platos. Los platos que componen el menú se implementarán desde el gestor de restaurante.

Propiedad	Tipo	Obligatorio	Descripción
name	String	Si	Nombre del menú.
description	String	No	Descripción del menú.

Tabla 4 Descripción del objeto Menu

1.5. Objeto Restaurant

Representa un recurso restaurante para formar parte de la cadena de restaurantes a gestionar.

Propiedad	Tipo	Obligatorio	Descripción
name	String	Si	Nombre del restaurante.
description	String	No	Descripción del restaurante.
location	Coordinate	No	Ubicación del restaurante en forma de coordenadas.

Tabla 5 Descripción del objeto Restaurant

1.6. Objeto Coordinate

Son coordenadas para localizar una ubicación.

Propiedad	Tipo	Obligatorio	Descripción
latitude	Number	si	Latitud de la ubicación.
longitude	Number	si	Longitud de la ubicación.

Tabla 6 Descripción de Coordinate.

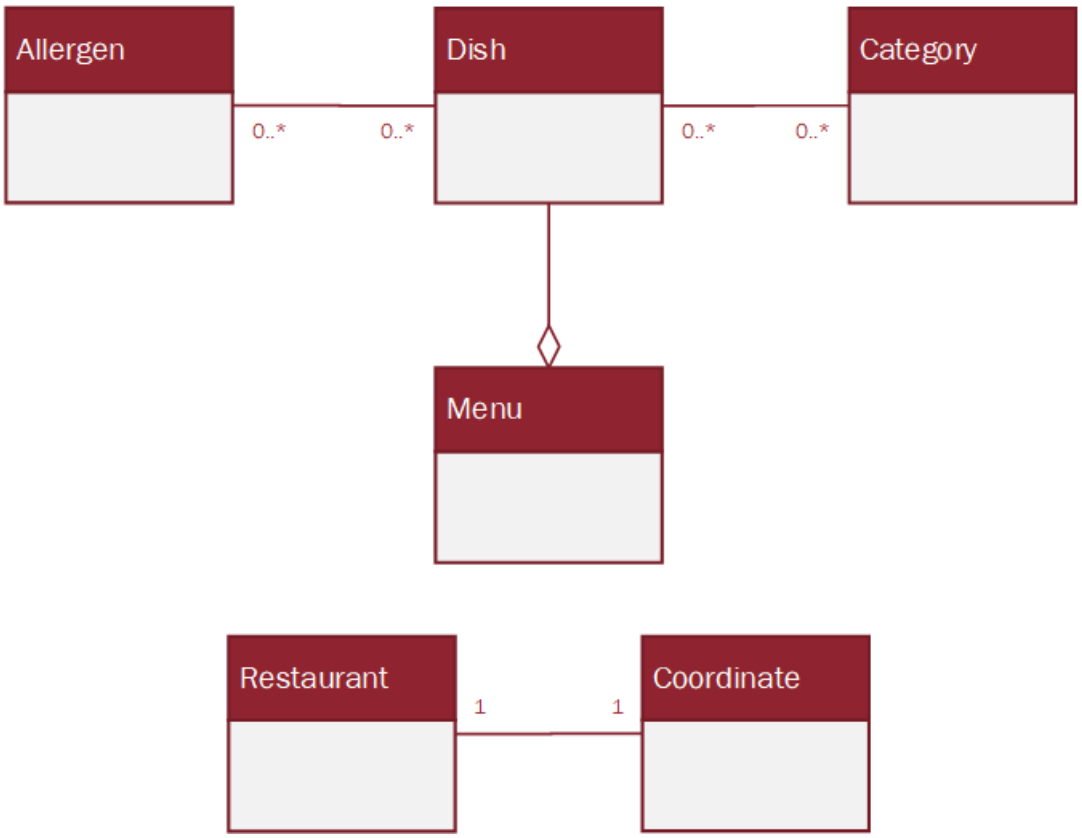
2. Gestión de restaurantes

En un objeto **RestaurantsManager** vamos a mantener el estado del propio sistema, donde vamos a relacionar todos los objetos anteriores.

La información que debe mantener es:

- Nombre del sistema.
- Colección de categorías de platos. Los platos pueden pertenecer a más de una categoría.
- Colección de tipos a alérgenos. Los platos pueden tener asociado más de un alérgeno.
- Colección de platos.
- Colección de menús. Se trata de una agregación de platos.
- Colección de restaurantes.

En la siguiente imagen podemos ver el diagrama UML con las clases que formaran parte del objeto.



A continuación, definimos los métodos que deben estar implementados en el objeto. El objeto debe implementar una estructura Singleton y ser un objeto Flyweight como se puede ver en la descripción de los métodos.

Los métodos de añadir, borrar o asignar deben poder aceptar multiargumentos.

Los métodos son:

Tabla 7 Relación de métodos del objeto RestaurantsManager

Método	Funcionalidad	Argumentos	Retorno	Excepciones
Getter categories	Devuelve un iterador que permite recorrer las categorías del sistema.	-	Iterador de categorías	
Getter menus	Devuelve un iterador que permite recorrer los menus del sistema.	-	Iterador de menus	
Getter allergens	Devuelve un iterador que permite recorrer los alérgenos del sistema.	-	Iterador de alérgenos	
Getter restaurants	Devuelve un iterador que permite recorrer los restaurantes del sistema.	-	Iterador de restaurantes	
addCategory	Añade una nueva categoría.	Objeto Category	Se debe poder encadenar.	- La categoría no puede ser null o no es un objeto Category. - La categoría ya existe.
removeCategory	Elimina una categoría. Los platos quedarán desasignados de la categoría.	Objeto Category	Se debe poder encadenar.	- La categoría no está registrada.
addMenu	Añade un nuevo menú.	Objeto Menu	Se debe poder encadenar.	- El menú no puede ser null o no es un objeto Menu. - El menú ya existe.
removeMenu	Elimina un menú.	Objeto Menu	Se debe poder encadenar.	- El menú no está registrado.
addAllergen	Añade un nuevo alérgeno.	Objeto Allergen	Se debe poder encadenar.	- El alérgeno no puede ser null o no es un objeto Allergen. - El alérgeno ya existe.
removeAllergen	Elimina un alérgeno.	Objeto Allergen	Se debe poder encadenar.	- El alérgeno no está registrado.
addDish	Añade un nuevo plato.	Objeto Dish	Se debe poder encadenar.	- El plato no puede ser null o no es un objeto Dish. - El plato ya existe.
removeDish	Elimina un plato y todas sus asignaciones a categorías, alérgenos y menús.	Objeto Dish	Se debe poder encadenar.	- El plato no está registrado.

addRestaurant	Añade un nuevo restaurante.	Objeto Restaurant	Se debe poder encadenar.	- El restaurante no puede ser null o no es un objeto Restaurant. - El restaurante ya existe.
removeRestaurant	Elimina un restaurante.	Objeto Restaurant	Se debe poder encadenar.	- El restaurante no está registrado.
assignCategoryToDish	Asigna un plato a una categoría. Si el objeto Category o Dish no existen se añaden al sistema.	-Category -Dish	Se debe poder encadenar.	- Category es null. - Dish es null.
deassignCategoryToDish	Desasigna un plato de una categoría.	-Category -Dish	Se debe poder encadenar.	- Category es null o no está registrada. - Dish es null o no está registrado.
assignAllergenToDish	Asigna un alérgeno a un plato. Si algún argumento no existe se añade al sistema.	-Allergen -Dish	Se debe poder encadenar.	- Allergen es null - Dish es null.
deassignAllergenToDish	Desasigna un alérgeno.	-Allergen -Dish	Se debe poder encadenar.	- Allergen es null o no está registrado. - Dish es null o no está registrado.
assignDishToMenu	Asigna un plato a un menú. Si algún argumento no existe se añade al sistema.	-Menu -Dish	Se debe poder encadenar.	- Dish es null - Menu es null.
deassignDishToMenu	Desasigna un plato de un menú.	-Menu -Dish	Se debe poder encadenar.	- Dish es null o no está registrado. - Menu es null o no está registrado.
changeDishesPositionsInMenu	Intercambia las posiciones de dos platos en un menú	- Menu - Dish - Dish	Se debe poder encadenar.	- Menu es null o no está registrado - Dish es null o no está registrado

				- Dish no está asigando en le menú
getDishesInCategory	Obtiene un iterador con la relación de los platos a una categoría. El iterador puede estar ordenado.	- Category - Function	iterador	- Category es null o no está registrada.
getDishesWithAllergen	Obtiene un iterador con los platos que tiene un determinado alérgeno. El iterador puede estar ordenado.	- Allergen - Function	iterador	- Allergen es null o no está registrado.
findDishes	Obtiene un iterador que cumpla un criterio concreto en base a una función de callback. El iterador puede estar ordenado.	- Dish - Function - Function	iterador	- Dish es null o no está registrado.
createDish	Devuelve un objeto Dish si está registrado, o crea un nuevo.	- Argumentos del constructor	Dish	
createMenu	Devuelve un objeto Menu si está registrado, o crea un nuevo.	- Argumentos del constructor	Menu	
createAllergen	Devuelve un objeto Allergen si está registrado, o crea un nuevo.	- Argumentos del constructor	Allergen	
createCategory	Devuelve un objeto Category si está registrado, o crea un nuevo.	- Argumentos del constructor	Category	
createRestaurant	Devuelve un objeto Restaurant si está registrado, o crea un nuevo.	- Argumentos del constructor	Restaurant	

3. Relación entre objetos

Las bases de datos **NOSQL** son aquellas que utilizan almacenes de objetos para contener la información, en lugar de utilizar relaciones entre tablas como ocurre con las relacionales. Un ejemplo es **MongoDB** la cual almacena documento en un formato parecido a **JSON**. Como veremos en próximas unidades, un documento **JSON** es un *objeto literal* traducido a *string* para que pueda ser portable. Estos documentos son fácilmente interpretables por un humano, por lo que no necesitan ser procesados. Por último, pueden ser transferibles a través de red para comunicar componentes de una aplicación, o de forma más habitual, entre un cliente y servidor.

Un ejemplo de documento JSON podría ser utilizado para representar un usuario. El siguiente ejemplo vemos cómo podemos utilizar esta representación para un usuario con una propiedad *_id*, *name*, *contact*, y *dob* con la fecha de nacimiento.

```
{
    "_id": "52ffc33cd85242f436000001",
    "name": "Tom Hanks",
    "contact": "987654321",
    "dob": "01-01-1991"
 }
```

Otro ejemplo podría ser utilizado para representar una dirección.

```
{
    "_id": "52ffc4a5d85242602e000000",
    "building": "22 A, Indiana Apt",
    "pincode": 123456,
    "city": "Los Angeles",
    "state": "California"
}
```

Como vemos, ambos siguen una estructura similar a la de un objeto literal. Ambos, JSON y los objetos literales son formatos intercambiables.

MongoDB utiliza los dos modelos de relación de objetos citados. Veamos unos ejemplos.

3.1. Modelo embebido o de relación embebida

En este modelo, los objetos subordinados están embebidos dentro del objeto principal. Un ejemplo es el siguiente, donde tenemos un usuario y un array con todas sus posibles direcciones.

```
{
    "_id": "52ffc33cd85242f436000001",
    "contact": "987654321",
    "dob": "01-01-1991",
    "name": "Tom Benzamin",
    "address": [
       {
          "building": "22 A, Indiana Apt",
          "pincode": 123456,
          "city": "Los Angeles",
          "state": "California"
       },
```

```
        {
            "building": "170 A, Acropolis Apt",
            "pincode": 456789,
            "city": "Chicago",
            "state": "Illinois"
        }
    ]
}
```

Esto nos crea la **ventaja** de que si queremos acceder al contenido se encuentra disponible inmediatamente. No tenemos que hacer ningún paso extra para recuperarlo. El **inconveniente** viene en el mantenimiento de los datos y su repetición. Dos usuarios podrían tener la misma dirección, si queremos modificar un dato de la dirección tendríamos que ir objeto tras objeto comprobando si es la dirección que estamos buscando para actualizar el dato.

3.2. Modelo relación por referencia

En este caso los objetos subordinados residen en otro documento o estructura, y son referenciados mediante una propiedad con forma de identificador. En este ejemplo, las direcciones son referenciadas por un identificador.

```
{
    "_id":"52ffc33cd85242f436000001",
    "contact": "987654321",
    "dob": "01-01-1991",
    "name": "Tom Benzamin",
    "address_ids": [
        "52ffc4a5d85242602e000000",
        "52ffc4a5d85242602e000001"
    ]
}
```

La **ventaja** de este modelo es el mantenimiento, ya que los datos están centralizados, pero el **inconveniente** es que tendremos que realizar varios pasos para encontrar la información, el usuario, y una vez encontrado, habría que buscar todas las direcciones que le pueden pertenecer a través de los *id* aportados por el objeto del usuario.

Nunca hay soluciones perfectas, todo depende de las necesidades de nuestra aplicación, y qué beneficia más a nuestra lógica de negocio.

4. Consejos de implementación

Para la implementación de la práctica necesitaremos diseñar una estructura que nos permita mantener todos los objetos que vayamos creando, restaurantes, platos, categorías, alérgenos y menús. La estructura podría implementarse con arrays, mapas, una implementación mixta o con la clase SortedMap.

Solución 1

Las relaciones las creamos en las colecciones de categorías, menús y alérgenos. Las colecciones, aunque estén representadas con un array, podrían ser cualquier tipo de colección.

```
{
  category,
  dishes: []
}

{
  allerge,
  dishes: []
}

{
  menu,
  dishes: []
}
```

Solución 2

En esta solución la relación la implementamos en la colección de platos y de menús. Las colecciones, aunque estén representadas con un array, podrían ser cualquier tipo de colección.

```
{
  dish,
  categories: [],
  allergens: []
}

{
  menu,
  dishes: []
}
```

Nota

Implementa una función de testeo de toda la funcionalidad implementada en la aplicación a través de la consola. **Esta funcionalidad es imprescindible para corregir la práctica. Si la función no está implementada la nota final será de 0.**

Podrás realizar cualquier cambio en la funcionalidad propuesta siempre y cuando esté justificada y mejore la funcionalidad propuesta.

Si consideras el diseño de los objetos, excepciones o argumentos de entrada y salida de los diferentes métodos también lo pueden realizar, siempre y cuando estén justificados.

Calificación

Criterio de evaluación	Puntos
Implementación de la aplicación. Verificación de funcionamiento. - Métodos de inserción y borrado: 1 punto - Métodos de asignación: 1 punto - Métodos de búsqueda: 1 punto - Métodos de creación: 1 punto - Intercambio de posiciones de los platos en el menú: 1 punto	5 puntos
Estructura OO. Verificación del código siga el paradigma de orientación a objetos.	1 punto
Comentarios. Deberás comentar el código que has implementado.	1 punto
Uso de patrones de diseño y características avanzadas de objetos - Transformación de los arrays por iteradores. (1,5 puntos) - Uso del patrón Singleton (1 puntos) - Empaquetado en módulos (0,5 puntos)	3 puntos

Tabla 8 Calificación