**Como crear una clase Genérica que simule las operaciones de un CRUD en memoria**

**Veamos la siguiente clase Genérica llamada InMemoryRepository<T> en el mismo paquete donde reside el main:**

public class InMemoryRepository<T> {  
 protected Map<Long, T> data = new HashMap<>();  
 protected AtomicLong idGenerator = new AtomicLong();  
  
 public T save(T entity) {  
 long id = idGenerator.incrementAndGet();  
 *// Suponiendo que las entidades tienen un método setId* try {  
 String clase;  
 entity.getClass().getMethod("setId", Long.class).invoke(entity, id);  
 clase = entity.getClass().getName();  
 System.*out*.println(clase + " id :" + id);  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
  
 data.put(id, entity);  
 return entity;  
 }  
  
 public Optional<T> findById(Long id) {  
 return Optional.*ofNullable*(data.get(id));  
 }  
  
  
  
 public List<T> findAll() {  
 return new ArrayList<>(data.values());  
 }  
  
  
 public Optional<T> genericUpdate(Long id, T updatedEntity) {  
 if (!data.containsKey(id)) {  
 return Optional.*empty*();  
 }  
  
 try {  
 *// Establecer el mismo ID en la entidad actualizada para mantener la coherencia* Method setIdMethod = updatedEntity.getClass().getMethod("setId", Long.class);  
 setIdMethod.invoke(updatedEntity, id);  
  
 data.put(id, updatedEntity);  
 return Optional.*of*(updatedEntity);  
 } catch (Exception e) {  
 e.printStackTrace();  
 return Optional.*empty*();  
 }  
 }  
  
 public Optional<T> genericDelete(Long id) {  
 if (!data.containsKey(id)) {  
 return Optional.*empty*();  
 }  
 return Optional.*ofNullable*(data.remove(id));  
 }  
  
 public List<T> genericFindByField(String fieldName, Object value) {  
 List<T> results = new ArrayList<>();  
 try {  
 for (T entity : data.values()) {  
 Method getFieldMethod = entity.getClass().getMethod("get" + capitalize(fieldName));  
 Object fieldValue = getFieldMethod.invoke(entity);  
 if (fieldValue != null && fieldValue.equals(value)) {  
 results.add(entity);  
 }  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 return results;  
 }  
  
 private String capitalize(String str) {  
 if (str == null || str.isEmpty()) {  
 return str;  
 }  
 return str.substring(0, 1).toUpperCase() + str.substring(1);  
 }  
}

EXPLICACION

**La clase InMemoryRepository<T> es un repositorio genérico en memoria que permite gestionar entidades de cualquier tipo T. Esta clase proporciona métodos para guardar, buscar, actualizar y eliminar entidades, así como para encontrar entidades por campos específicos. Aquí está una explicación detallada de cada componente y método de la clase:**

**Componentes de la Clase**

1. **protected Map<Long, T> data = new HashMap<>();**
   * Un mapa que almacena las entidades en memoria. La clave es un Long que representa el ID de la entidad y el valor es la propia entidad.
2. **protected AtomicLong idGenerator = new AtomicLong();**
   * Un generador de IDs que se incrementa automáticamente cada vez que se guarda una nueva entidad, asegurando que cada entidad tenga un ID único.

**Métodos de la Clase**

1. **public T save(T entity)**
   * Guarda una entidad en el repositorio.
   * Genera un nuevo ID para la entidad utilizando idGenerator.
   * Utiliza reflexión para invocar el método setId de la entidad y asignarle el ID generado.
   * Almacena la entidad en el mapa data.
   * Retorna la entidad guardada.

public T save(T entity) {

long id = idGenerator.incrementAndGet();

try {

entity.getClass().getMethod("setId", Long.class).invoke(entity, id);

System.out.println(entity.getClass().getName() + " id :" + id);

} catch (Exception e) {

e.printStackTrace();

}

data.put(id, entity);

return entity;

}

1. **public Optional<T> findById(Long id)**
   * Busca una entidad por su ID.
   * Retorna un Optional que puede contener la entidad si se encuentra, o estar vacío si no se encuentra.

public Optional<T> findById(Long id) {

return Optional.ofNullable(data.get(id));

}

1. **public List<T> findAll()**
   * Retorna una lista con todas las entidades almacenadas en el repositorio.

public List<T> findAll() {

return new ArrayList<>(data.values());

}

1. **public Optional<T> genericUpdate(Long id, T updatedEntity)**
   * Actualiza una entidad existente en el repositorio.
   * Comprueba si el ID existe en el mapa data.
   * Si existe, utiliza reflexión para establecer el mismo ID en la entidad actualizada.
   * Almacena la entidad actualizada en el mapa y retorna un Optional que contiene la entidad actualizada.
   * Si no existe, retorna un Optional.empty().

public Optional<T> genericUpdate(Long id, T updatedEntity) {

if (!data.containsKey(id)) {

return Optional.empty();

}

try {

Method setIdMethod = updatedEntity.getClass().getMethod("setId", Long.class);

setIdMethod.invoke(updatedEntity, id);

data.put(id, updatedEntity);

return Optional.of(updatedEntity);

} catch (Exception e) {

e.printStackTrace();

return Optional.empty();

}

}

1. **public Optional<T> genericDelete(Long id)**
   * Elimina una entidad del repositorio por su ID.
   * Retorna un Optional que puede contener la entidad eliminada si se encuentra, o estar vacío si no se encuentra.

public Optional<T> genericDelete(Long id) {

if (!data.containsKey(id)) {

return Optional.empty();

}

return Optional.ofNullable(data.remove(id));

}

1. **public List<T> genericFindByField(String fieldName, Object value)**
   * Busca entidades en el repositorio por un campo específico y su valor.
   * Utiliza reflexión para obtener el valor del campo especificado en cada entidad.
   * Retorna una lista de entidades que tienen el valor especificado en el campo.

public List<T> genericFindByField(String fieldName, Object value) {

List<T> results = new ArrayList<>();

try {

for (T entity : data.values()) {

Method getFieldMethod = entity.getClass().getMethod("get" + capitalize(fieldName));

Object fieldValue = getFieldMethod.invoke(entity);

if (fieldValue != null && fieldValue.equals(value)) {

results.add(entity);

}

}

} catch (Exception e) {

e.printStackTrace();

}

return results;

}

1. **private String capitalize(String str)**
   * Capitaliza la primera letra de una cadena. Este método es útil para construir nombres de métodos getter basados en nombres de campos.

private String capitalize(String str) {

if (str == null || str.isEmpty()) {

return str;

}

return str.substring(0, 1).toUpperCase() + str.substring(1);

}