PRACTICA 4

Herencia, Interfaces y Excepciones

Alejandro Antonio Martín Almansa
Mario García Roque

Apartado 1:

En el apartado 1 de esta práctica desarrollamos las interfaces, clases abstractas, e implementaciones de las clases que modelan los tipos de datos de la aplicación y las distintas entidades. Estas clases e interfaces son: IEntity, ITypeDescriptor (con una clase NTypeDescriptor para cada elemento de la tienda, donde N es el nombre del artículo o de las personas), junto con todas las clases de la tienda y sus correspondientes clases abstractas para mejorar el diseño.

Código de la declaración de la interfaz IEntity:

```
package DAO;

/**
 * Interfaz para leer y escribir informacion de los objetos
 *
 * @author Alejandro Antonio Martin Almansa
 * @author Mario Garcia Roque
 *
 */
public interface IEntity {

	/**
	 * Devuelve el id de la entidad
	 *
	 * @return id de la entidad
	 */
	public Long getId();

	/**
	 * Asigna un id a la entidad
	 *
	 * @param id
	 *            a asignar a la entidad
	 */
	public void setId(Long id);

	/**
	 * Devuelve el nombre del tipo de datos
	 *
	 * @return nombre del tipo de datos
	 */
	public String getType();

	/**
	 * Devuelve el valor de una propiedad de una entidad
	 *
	 * @param property
	 *            de la entidad
	 * @return el objeto con el valor de la propiedad
	 */
	public Object getProperty(String property);

	/**
	 * Modifica el valor de una propiedad
	 *
	 * @param property
	 *            propiedad a modifcar
	 * @param value
	 *            valor para asignar
	 */
	public void setProperty(String property, Object value);
}
```

Código de la declaración de la interfaz ITypeDescriptor:

```java
package DAO;

import java.util.List;

/**
 * Interfaz para trabajar con un unico tipo de dato
 *
 * @author Alejandro Antonio Martin Almansa
 * @author Mario Garcia Roque
 *
 */
public interface ITypeDescriptor {

	public enum Type {
		LONG, DOUBLE, STRING;
	}

	/**
	 * Devuelve el nombre del tipo de datos
	 *
	 * @return nombre del tipo de datos
	 */
	public String getName();

	/**
	 * Devuelve las propiedades del tipo de datos
	 *
	 * @return lista de propiedades del tipo de datos
	 */
	public List<String> getProperties();

	/**
	 * Indica el tipo de la propiedad
	 *
	 * @param property
	 *            del tipo de dato
	 * @return tipo de dato de la propiedad
	 */
	public Type getType(String property);

	/**
	 * Creador de nueva entidad
	 *
	 * @return la entidad creada
	 */
	public IEntity newEntity();

}
```

Código de la clase AbstractEntity:

```java
package tienda;

import DAO.*;

/**
 * Clase para implementar las funciones del id general a todas las entidades
 *
 * @author Alejandro Antonio Martin Almansa
 * @author Mario Garcia Roque
 *
 */
public abstract class AbstractEntity implements IEntity {
```

```
	private Long id;

	/**
	 * Devuelve el id de la entidad
	 *
	 * @return id de la entidad
	 */
	@Override
	public Long getId() {
		return id;
	}

	/**
	 * Asigna un id a la entidad
	 *
	 * @param id
	 *            a asignar a la entidad
	 */
	@Override
	public void setId(Long id) {
		this.id = id;
	}

}
```

Código de la clase Artículo:

```
package tienda;

/**
 * Se describe la clase Articulo para que la tienda pueda gestionar los
 * articulos
 *
 * @author Mario Garcia Roque
 * @author Alejandro Antonio Martin Almansa
 *
 */
public abstract class Articulo extends AbstractEntity {

	private String titulo;

	/**
	 * Constructor de la clase articulo
	 */
	public Articulo() {

	}

	/**
	 * Metodo para obtener el titulo del articulo
	 *
	 * @return el titulo del articulo
	 */
	public String getTitulo() {
		return titulo;
	}

	/**
	 * Metodo para asignar un titulo a un articulo
	 *
	 * @param titulo
	 *            que se le va a asignar al articulo
	 */
	public void setTitulo(String titulo) {
		this.titulo = titulo;
	}
```

```
}
```

Código de la clase Disco:

```
package tienda;

import descriptores.DiscoTypeDescriptor;

/**
 * Clase Disco
 *
 * @author Mario Garcia Roque
 * @author Alejandro Antonio Martin Almansa
 *
 */
public class Disco extends Articulo {

	private Long interprete;
	private Long year;
	private static DiscoTypeDescriptor disco = new DiscoTypeDescriptor();

	/**
	 * Constructor de la clase Disco
	 */
	public Disco() {
	}

	/**
	 * Devuelve el interprete de un disco
	 *
	 * @return el interprete de un disco
	 */
	public Long getInterprete() {
		return interprete;
	}

	/**
	 * Asigna un interprete a un disco
	 *
	 * @param interprete
	 *            a asignar al disco
	 */
	public void setInterprete(Long interprete) {
		this.interprete = interprete;
	}

	/**
	 * Devuelve el anyo de un disco
	 *
	 * @return el anyo del disco
	 */
	public Long getYear() {
		return year;
	}

	/**
	 * Asigna un anyo a un disco
	 *
	 * @param year
	 *            anyo a asignar al disco
	 */
	public void setYear(Long year) {
		this.year = year;
	}

	/**
```

```
	 * Devuelve el nombre del tipo de datos
	 *
	 * @return nombre del tipo de datos disco
	 */
	@Override
	public String getType() {
		return "Disco";
	}

	/**
	 * Metodo para obtener el typo de dato Disco
	 *
	 * @return tipo de dato disco
	 */
	public static DiscoTypeDescriptor getDescriptor() {
		return disco;
	}

	/**
	 * Devuelve el valor de una propiedad de una entidad
	 *
	 * @param property
	 *            de la entidad
	 * @return el objeto con el valor de la propiedad
	 */
	@Override
	public Object getProperty(String property) {
		if (property.equalsIgnoreCase("interprete"))
			return interprete;
		else if (property.equalsIgnoreCase("year"))
			return year;
		else if (property.equalsIgnoreCase("id"))
			return getId();
		else if (property.equalsIgnoreCase("titulo"))
			return getTitulo();

		return null;
	}

	/**
	 * Modifica el valor de una propiedad
	 *
	 * @param property
	 *            propiedad a modifcar
	 * @param value
	 *            valor para asignar
	 */
	@Override
	public void setProperty(String property, Object value) {
		if (property.equalsIgnoreCase("interprete"))
			interprete = (Long) value;
		else if (property.equalsIgnoreCase("year"))
			year = (Long) value;
		else if (property.equalsIgnoreCase("id"))
			setId((Long) value);
		else if (property.equalsIgnoreCase("titulo"))
			setTitulo((String) value);

	}
}
```

Código de la clase Libro:

```
package tienda;

import descriptores.LibroTypeDescriptor;

/**
```

```java
 * Clase Autor (Autor de un libro)
 *
 * @author Mario Garcia Roque
 * @author Alejandro Antonio Martin Almansa
 *
 */
public class Libro extends Articulo {

	private Long autor;
	private String editorial;
	private static LibroTypeDescriptor libro = new LibroTypeDescriptor();

	/**
	 * Constructor de la clase Libro
	 */
	public Libro() {

	}

	/**
	 * Devuelve el autor de un libro
	 *
	 * @return el autor del libro
	 */
	public Long getAutor() {
		return autor;
	}

	/**
	 * Asigna un autor a un libro
	 *
	 * @param autor
	 *            a asignar al libro
	 */
	public void setAutor(Long autor) {
		this.autor = autor;
	}

	/**
	 * Devuelve la editorial de un libro
	 *
	 * @return la editorial del libro
	 */
	public String getEditorial() {
		return editorial;
	}

	/**
	 * Asigna una editorial a un libro
	 *
	 * @param editorial
	 *            a asignar al libro
	 */
	public void setEditorial(String editorial) {
		this.editorial = editorial;
	}

	/**
	 * Metodo para obtener el typo de dato Libro
	 *
	 * @return tipo de dato libro
	 */
	public static LibroTypeDescriptor getDescriptor() {
		return libro;
	}

	/**
```

```
	 * toString de la clase Libro
	 */
	public String toString() {
		return "[" + getId() + "]LIBRO: " + getTitulo() + ". " + this.autor
				+ ". " + this.editorial + "";
	}

	/**
	 * Devuelve el nombre del tipo de datos
	 *
	 * @return nombre del tipo de datos libro
	 */
	@Override
	public String getType() {
		return "Libro";
	}

	/**
	 * Devuelve el valor de una propiedad de una entidad
	 *
	 * @param property
	 *            de la entidad
	 * @return el objeto con el valor de la propiedad
	 */
	@Override
	public Object getProperty(String property) {
		if (property.equalsIgnoreCase("autor"))
			return this.autor;
		else if (property.equalsIgnoreCase("editorial"))
			return this.editorial;
		else if (property.equalsIgnoreCase("id"))
			return getId();
		else if (property.equalsIgnoreCase("titulo"))
			return getTitulo();

		return null;
	}

	/**
	 * Modifica el valor de una propiedad
	 *
	 * @param property
	 *            propiedad a modifcar
	 * @param value
	 *            valor para asignar
	 */
	@Override
	public void setProperty(String property, Object value) {
		if (property.equalsIgnoreCase("autor"))
			this.autor = (Long) value;
		else if (property.equalsIgnoreCase("editorial"))
			this.editorial = (String) value;
		else if (property.equals("id"))
			setId((Long) value);
		else if (property.equalsIgnoreCase("titulo"))
			setTitulo((String) value);

	}

}
```

Código de la clase Película:

```java
package tienda;

import descriptores.PeliculaTypeDescriptor;

/**
 * Clase Pelicula
 *
 * @author Mario Garcia Roque
 * @author Alejandro Antonio Martin Almansa
 *
 */
public class Pelicula extends Articulo {

	private String genero;
	private Long director;
	private static PeliculaTypeDescriptor pelicula = new PeliculaTypeDescriptor();

	/**
	 * Constructor de la clase Pelicula
	 */
	public Pelicula() {
	}

	/**
	 * Constructor de la clase Pelicula
	 *
	 * @param id
	 *            de la pelicula
	 * @param titulo
	 *            de la pelicula
	 * @param genero
	 *            de la pelicula
	 * @param director
	 *            de la pelicula
	 */
	public Pelicula(int id, String titulo, String genero, String director) {
	}

	/**
	 * Devuelve el genero de una pelicula
	 *
	 * @return el genero de una pelicula
	 */
	public String getGenero() {
		return this.genero;
	}

	/**
	 * Asigna un genero a una pelicula
	 *
	 * @param genero
	 *            a asignar a la pelicula
	 */
	public void setGenero(String genero) {
		this.genero = genero;
	}

	/**
	 * Asigna un director a una pelicula
	 *
	 * @param director
	 *            a asignar a la pelicula
	 */
	public void setDirector(Long director) {
		this.director = director;
	}
```

```java
/**
 * Devuelve el director de una pelicula
 *
 * @return el director de una pelicula
 */
public Long getDirector() {
	return this.director;
}

/**
 * toString de la clase Pelicula
 */
public String toString() {
	return "[" + getId() + "]PELICULA: " + getTitulo() + ". ("
			+ getGenero() + ") Dir: " + getDirector();
}

/**
 * Devuelve el nombre del tipo de datos
 *
 * @return nombre del tipo de datos pelicula
 */
@Override
public String getType() {
	return "Pelicula";
}

/**
 * Metodo para obtener el typo de dato Pelicula
 *
 * @return tipo de dato pelicula
 */
public static PeliculaTypeDescriptor getDescriptor() {
	return pelicula;
}

/**
 * Devuelve el valor de una propiedad de una entidad
 *
 * @param property
 *            de la entidad
 * @return el objeto con el valor de la propiedad
 */
@Override
public Object getProperty(String property) {
	if (property.equalsIgnoreCase("genero"))
		return getGenero();
	else if (property.equalsIgnoreCase("director"))
		return getDirector();
	else if (property.equalsIgnoreCase("id"))
		return getId();
	else if (property.equalsIgnoreCase("titulo"))
		return getTitulo();

	return null;
}

/**
 * Modifica el valor de una propiedad
 *
 * @param property
 *            propiedad a modifcar
 * @param value
 *            valor para asignar
 */
@Override
public void setProperty(String property, Object value) {
```

```
            if (property.equalsIgnoreCase("genero"))
                  setGenero((String) value);
            else if (property.equalsIgnoreCase("director"))
                  setDirector((long) value);
            else if (property.equalsIgnoreCase("id"))
                  setId((long) value);
            else if (property.equalsIgnoreCase("titulo"))
                  setTitulo((String) value);

      }
}
```

Código de la clase Persona:

```
package tienda;

/**
 * Clase Persona (Autor, Director)
 *
 * @author Mario Garcia Roque
 * @author Alejandro Antonio Martin Almansa
 *
 */

public abstract class Persona extends AbstractEntity {

      private String nombre;
      private String apellidos;

      /**
       * Constructor de la clase persona
       */
      public Persona() {

      }

      /**
       * Devuelve el nombre de una persona
       *
       * @return el nombre de una persona
       */
      public String getNombre() {
            return nombre;
      }

      /**
       * Asigna un nombre a una persona
       *
       * @param nombre
       *            a asignar a la persona
       */
      public void setNombre(String nombre) {
            this.nombre = nombre;
      }

      /**
       * Devuelve los apellidos de una persona
       *
       * @return los apellidos de una persona
       */
      public String getApellidos() {
            return apellidos;
      }

      /**
       * Asigna apellidos a una persona
       *
       * @param apellidos
```

```
	 *            a asignar a la persona
	 */
	public void setApellidos(String apellidos) {
		this.apellidos = apellidos;
	}

	/**
	 * Devuelve el valor de una propiedad de una entidad
	 *
	 * @param property
	 *            de la entidad
	 * @return el objeto con el valor de la propiedad
	 */
	@Override
	public Object getProperty(String property) {
		if (property.equalsIgnoreCase("nombre"))
			return this.getNombre();
		else if (property.equalsIgnoreCase("apellidos"))
			return this.getApellidos();
		else if (property.equals("id"))
			return this.getId();
		return null;
	}

	/**
	 * Modifica el valor de una propiedad
	 *
	 * @param property
	 *            propiedad a modifcar
	 * @param value
	 *            valor para asignar
	 */
	@Override
	public void setProperty(String property, Object value) {
		if (property.equals("nombre"))
			this.setNombre((String) value);
		else if (property.equals("apellidos"))
			this.setApellidos((String) value);
		else if (property.equals("id"))
			this.setId((Long) value);
	}

}
```

Código de la clase Autor:

```
package tienda;

import descriptores.AutorTypeDescriptor;

/**
 * Clase Autor (Autor de un libro)
 *
 * @author Mario Garcia Roque
 * @author Alejandro Antonio Martin Almansa
 *
 */
public class Autor extends Persona {

	private static AutorTypeDescriptor autor = new AutorTypeDescriptor();

	/**
	 * Constructor de la clase Autor
	 */
	public Autor() {

	}
```

```
	/**
	 * Metodo para obtener el typo de dato Autor
	 *
	 * @return tipo de dato autor
	 */
	public static AutorTypeDescriptor getDescriptor() {
		return autor;
	}

	/**
	 * Devuelve el nombre del tipo de datos
	 *
	 * @return nombre del tipo de datos autor
	 */
	@Override
	public String getType() {
		return "Autor";
	}

	/**
	 * toString de la clase autor
	 */
	public String toString() {
		return "[" + getId() + "]AUTOR: " + getNombre() + " " + getApellidos();
	}
}
```

Código de la clase Director:

```
package tienda;

import descriptores.DirectorTypeDescriptor;

/**
 * Clase Director (Director de una pelicula)
 *
 * @author Mario Garcia Roque
 * @author Alejandro Antonio Martin Almansa
 *
 */
public class Director extends Persona {

	private static DirectorTypeDescriptor director = new DirectorTypeDescriptor();

	/**
	 * Constructor de la clase Director
	 */
	public Director() {

	}

	/**
	 * Metodo para obtener el typo de dato Director
	 *
	 * @return tipo de dato director
	 */
	public static DirectorTypeDescriptor getDescriptor() {
		return director;
	}

	/**
	 * Devuelve el nombre del tipo de datos
	 *
	 * @return nombre del tipo de datos director
	 */
```

```
	@Override
	public String getType() {
		return "Director";
	}

	/**
	 * toString de la clase Director
	 */
	public String toString() {
		return "[" + getId() + "]DIRECTOR: " + getNombre() + " "
				+ getApellidos();
	}

}
```

Código de la clase DiscoTypeDescriptor:

```
package descriptores;

import java.util.ArrayList;
import java.util.List;
import tienda.Autor;
import DAO.*;

/**
 * Clase para trabajar con el tipo de dato Disco
 *
 * @author Alejandro Antonio Martin Almansa
 * @author Mario Garcia Roque
 *
 */
public class DiscoTypeDescriptor implements ITypeDescriptor {

	/**
	 * Devuelve el nombre del tipo de datos
	 *
	 * @return nombre del tipo de datos
	 */
	@Override
	public String getName() {
		return "Articulo";
	}

	/**
	 * Devuelve las propiedades del tipo de datos Disco
	 *
	 * @return lista de propiedades del tipo de datos
	 */
	@Override
	public List<String> getProperties() {
		List<String> pr = new ArrayList<String>();
		pr.add("id");
		pr.add("titulo");
		pr.add("interprete");
		pr.add("year");
		return pr;
	}

	/**
	 * Indica el tipo de la propiedad
	 *
	 * @param property
	 *            del tipo de dato
	 * @return tipo de dato de la propiedad
	 */
	@Override
	public Type getType(String property) {
		if (property.equalsIgnoreCase("interprete"))
```

```
                return Type.LONG;
            else if (property.equalsIgnoreCase("year"))
                return Type.LONG;
            else if (property.equalsIgnoreCase("id"))
                return Type.LONG;
            else if (property.equalsIgnoreCase("titulo"))
                return Type.STRING;
            return null;
        }

        /**
         * Creador de nueva entidad
         *
         * @return la entidad creada
         */
        @Override
        public IEntity newEntity() {
            return new Autor();
        }

}
```

Código de la clase LibroTypeDescriptor:

```
package descriptores;

import java.util.ArrayList;
import java.util.List;
import tienda.Libro;
import DAO.*;

/**
 * Clase para trabajar con el tipo de dato Libro
 *
 * @author Alejandro Antonio Martin Almansa
 * @author Mario Garcia Roque
 *
 */
public class LibroTypeDescriptor implements ITypeDescriptor {

        /**
         * Devuelve el nombre del tipo de datos
         *
         * @return nombre del tipo de datos
         */
        @Override
        public String getName() {
            return "Libro";
        }

        /**
         * Devuelve las propiedades del tipo de datos Libro
         *
         * @return lista de propiedades del tipo de datos
         */
        @Override
        public List<String> getProperties() {
            List<String> pr = new ArrayList<String>();
            pr.add("id");
            pr.add("titulo");
            pr.add("autor");
            pr.add("editorial");
            return pr;
        }
```

```
	/**
	 * Indica el tipo de la propiedad
	 *
	 * @param property
	 *            del tipo de dato
	 * @return tipo de dato de la propiedad
	 */
	@Override
	public Type getType(String property) {
		if (property.equalsIgnoreCase("autor"))
			return Type.LONG;
		else if (property.equalsIgnoreCase("editorial"))
			return Type.STRING;
		else if (property.equalsIgnoreCase("id"))
			return Type.LONG;
		else if (property.equalsIgnoreCase("titulo"))
			return Type.STRING;
		return null;
	}

	/**
	 * Creador de nueva entidad
	 *
	 * @return la entidad creada
	 */
	@Override
	public IEntity newEntity() {

		return new Libro();
	}

}
```

Código de la clase PeliculaTypeDescriptor:

```
package descriptores;

import java.util.ArrayList;
import java.util.List;
import tienda.Pelicula;
import DAO.*;

/**
 * Clase para trabajar con el tipo de dato Pelicula
 *
 * @author Alejandro Antonio Martin Almansa
 * @author Mario Garcia Roque
 *
 */
public class PeliculaTypeDescriptor implements ITypeDescriptor {

	/**
	 * Devuelve el nombre del tipo de datos
	 *
	 * @return nombre del tipo de datos
	 */
	@Override
	public String getName() {
		return "Pelicula";
	}

	/**
	 * Devuelve las propiedades del tipo de datos Pelicula
	 *
	 * @return lista de propiedades del tipo de datos
	 */
	@Override
```

```
	public List<String> getProperties() {
		List<String> pr = new ArrayList<String>();
		pr.add("id");
		pr.add("titulo");
		pr.add("director");
		pr.add("genero");
		return pr;
	}

	/**
	 * Indica el tipo de la propiedad
	 *
	 * @param property
	 *            del tipo de dato
	 * @return tipo de dato de la propiedad
	 */
	@Override
	public Type getType(String property) {
		if (property.equalsIgnoreCase("director"))
			return Type.LONG;
		else if (property.equalsIgnoreCase("genero"))
			return Type.STRING;
		else if (property.equalsIgnoreCase("id"))
			return Type.LONG;
		else if (property.equalsIgnoreCase("titulo"))
			return Type.STRING;
		return null;
	}

	/**
	 * Creador de nueva entidad
	 *
	 * @return la entidad creada
	 */
	@Override
	public IEntity newEntity() {

		return new Pelicula();
	}

}
```

Código de la clase PersonaTypeDescriptor:

```
package descriptores;

import java.util.ArrayList;
import java.util.List;
import DAO.ITypeDescriptor;

/**
 * Clase para trabajar con los tipos de dato que heredan de Persona
 *
 * @author Alejandro Antonio Martin Almansa
 * @author Mario Garcia Roque
 *
 */
public abstract class PersonaTypeDescriptor implements ITypeDescriptor {

	/**
	 * Devuelve las propiedades del tipo de datos Persona (Autpr, Director)
	 *
	 * @return lista de propiedades del tipo de datos
	 */
	@Override
	public List<String> getProperties() {
		List<String> propiedades = new ArrayList<>();
```

```
			propiedades.add("nombre");
			propiedades.add("apellidos");
			propiedades.add("id");

			return propiedades;
		}

		/**
		 * Indica el tipo de la propiedad
		 *
		 * @param property
		 *            del tipo de dato
		 * @return tipo de dato de la propiedad
		 */
		@Override
		public Type getType(String property) {
			if (property.equalsIgnoreCase("nombre"))
				return Type.STRING;
			else if (property.equalsIgnoreCase("apellidos"))
				return Type.STRING;
			else if (property.equals("id"))
				return Type.LONG;

			return null;
		}

}
```

Código de la clase AutorTypeDescriptor:

```
package descriptores;

import tienda.Autor;
import DAO.*;

/**
 * Clase para trabajar con el tipo de dato Autor
 *
 * @author Alejandro Antonio Martin Almansa
 * @author Mario Garcia Roque
 *
 */
public class AutorTypeDescriptor extends PersonaTypeDescriptor {

		/**
		 * Devuelve el nombre del tipo de datos
		 *
		 * @return nombre del tipo de datos
		 */
		@Override
		public String getName() {
			return "Autor";
		}
		/**
		 * Creador de nueva entidad
		 *
		 * @return la entidad creada
		 */
		@Override
		public IEntity newEntity() {

			return new Autor ();
		}
}
```

Código de la clase DirectorTypeDescriptor:

```
package descriptores;
```

```
import tienda.Director;
import DAO.IEntity;

/**
 * Clase para trabajar con el tipo de dato Director
 *
 * @author Alejandro Antonio Martin Almansa
 * @author Mario Garcia Roque
 *
 */
public class DirectorTypeDescriptor extends PersonaTypeDescriptor {

	/**
	 * Devuelve el nombre del tipo de datos
	 *
	 * @return nombre del tipo de datos
	 */
	@Override
	public String getName() {
		return "Director";
	}

	/**
	 * Creador de nueva entidad
	 *
	 * @return la entidad creada
	 */
	@Override
	public IEntity newEntity() {

		return new Director();
	}

}
```

Para probar las clases implementadas realizamos un test basado en el que se nos da en el guion de la de la práctica.

Código de la clase de prueba apto1:

```
package tests;

import tienda.*;
import DAO.*;
import DAO.ITypeDescriptor.Type;

/**
 * Prueba de las clases de la tienda, TypeDescriptor y Entity
 *
 * @author Alejandro Antonio Martin Almansa
 * @author Mario Garcia Roque
 *
 */
public class apto1 {
	public static void main(String[] args) {

		String string;
		Type tipo;

		Libro l1 = new Libro();

		System.out
				.println("Prueba getProperty de Entity: (Salida esperada: El
Quijote)");
		l1.setTitulo("El Quijote");
		String titulo = (String) l1.getProperty("titulo");
```

```
            System.out.println("Titulo: " + titulo);

            ITypeDescriptor t1 = Libro.getDescriptor();

            System.out
                        .println("Prueba getName de TypeDescriptor: (Salida esperada:
Libro)");
            string = t1.getName();
            System.out.println("Tipo del t1: " + string);

            System.out
                        .println("Prueba getType de TypeDescriptor: (Salida esperada:
String)");
            tipo = t1.getType("titulo");
            System.out.println("Tipo del titulo de t1: " + tipo);

            System.out
                        .println("Prueba getType del TypeDescriptor: (Salida esperada:
Long)");
            tipo = t1.getType("autor");
            System.out.println("Tipo del autor de t1: " + tipo);

            Autor a1 = new Autor();

            System.out
                        .println("Prueba setId y getId de Entity: (Salida esperada:
1)");
            a1.setId(1L);
            System.out.println("ID del autor a1: " + a1.getId());

            System.out
                        .println("Prueba setProperty de Entity: (Salida esperada:
Cervantes)");
            a1.setProperty("apellidos", "Cervantes");
            string = a1.getApellidos(); // devuelve Cervantes
            System.out.println("Apellidos del autor a1: " + string);

            System.out.println("Prueba getId y setAutor: (Salida esperada: 1)");
            l1.setAutor(a1.getId());
            System.out.println("Autor del libro l1: " + l1.getAutor());
      }
}
```

Salida: La salida esperada como podemos comprobar:

```
<terminated> apto1 [Java Application] C:\Program Files\Java\jre1.8.0_40\bin\javaw.exe (9/4/2015 18:15:3
Prueba getProperty de Entity: (Salida esperada: El Quijote)
Titulo: El Quijote
Prueba getName de TypeDescriptor: (Salida esperada: Libro)
Tipo del t1: Libro
Prueba getType de TypeDescriptor: (Salida esperada: String)
Tipo del titulo de t1: STRING
Prueba getType del TypeDescriptor: (Salida esperada: Long)
Tipo del autor de t1: LONG
Prueba setId y getId de Entity: (Salida esperada: 1)
ID del autor a1: 1
Prueba setProperty de Entity: (Salida esperada: Cervantes)
Apellidos del autor a1: Cervantes
Prueba getId y setAutor: (Salida esperada: 1)
Autor del libro l1: 1
```

Apartado 2:

En este apartado desarrollamos la interfaz IIndex para poder buscar en la base de datos de la práctica. Para ello creamos una clase nueva Index que implemente a la Interfaz IIndex y así desarrollar las funciones correspondientes.

Código de la declaración de la interfaz IIndex:

```
package DAO;

import java.util.Collection;

/**
 * Interfaz para crear un sistema de indexacion
 *
 * @author Alejandro Antonio Martin Almansa
 * @author Mario Garcia Roque
 *
 */
public interface IIndex {

	/**
	 * Funcion para relacionar value con key
	 *
	 * @param key
	 *            para relacionar
	 * @param value
	 *            relacionado con key
	 */
	void add(Object key, Long value);

	/**
	 * Funcion para eliminar la relacion entre value y key
	 *
	 * @param key
	 *            para eliminar su relacion
	 * @param value
	 *            para eliminar de key
	 */
	void delete(Object key, Long value);

	/**
```

```
	 * Lista los valores relacionados con key
	 *
	 * @param key
	 *            para buscar
	 * @return coleccion con los valores relacionados con key
	 */
	Collection<Long> search(Object key);

	/**
	 * Devuelve la lista de valores desde from a to
	 *
	 * @param from
	 *            valor desde el cual se va a buscar
	 * @param to
	 *            valor hasta el cual se va a buscar
	 * @return coleccion con los valores encontrados
	 */
	Collection<Long> search(Object from, Object to);
}
```

Código de la clase Index que implementa la interfaz IIndex:

```
package tienda;

import java.util.*;
import DAO.IIndex;
import tienda.AllowNullComparator;

/**
 * Clase Index para implementar la interfaz IIndex
 *
 * @author Mario Garcia Roque
 * @author Alejandro Antonio Martin Almansa
 *
 */
public class Index implements IIndex {

	private SortedMap<Object, Set<Long>> indices;

	/**
	 * Constructor de la clase Index
	 */
	public Index() {

		this.indices = new TreeMap<>(AllowNullComparator.Instance);

	}

	/**
	 * Funcion para relacionar value con key
	 *
	 * @param key
	 *            para relacionar
	 * @param value
	 *            relacionado con key
	 */
	@Override
	public void add(Object key, Long value) {

		Set<Long> set = indices.get(key);
		if (set == null) {
			set = new HashSet<>();
			indices.put(key, set);
		}
		set.add(value);

	}
```

```java
/**
 * Funcion para eliminar la relacion entre value y key
 *
 * @param key
 *            para eliminar su relacion
 * @param value
 *            para eliminar de key
 */
@Override
public void delete(Object key, Long value) {

	Set<Long> setkey = indices.get(key);
	setkey.remove(value);
	if (setkey.size() == 0) {
		indices.remove(key);
	}
}

/**
 * Lista los valores relacionados con key
 *
 * @param key
 *            para buscar
 * @return coleccion con los valores relacionados con key
 */
@Override
public Collection<Long> search(Object key) {

	if (key == null) {
		return null;
	}

	Set<Long> setkey = indices.get(key);

	if (setkey == null) {
		return Collections.emptySet();
	}

	return Collections.unmodifiableSet(setkey);
}

/**
 * Devuelve la lista de valores desde from a to
 *
 * @param from
 *            valor desde el cual se va a buscar
 * @param to
 *            valor hasta el cual se va a buscar
 * @return coleccion con los valores encontrados
 */
@Override
public Collection<Long> search(Object from, Object to) {

	Collection<Set<Long>> set;
	Collection<Long> retorno = new HashSet<Long>();
	String c = null;
	int i;
	if (to == null) {
		c = indices.lastKey().toString();
		i = Integer.valueOf(c);
		i++;
		c = Integer.toString(i);
	}

	if (from == null && to == null) {
		set = indices.subMap(indices.firstKey(), (Object) c).values();
		for (Iterator<Set<Long>> iterador = set.iterator(); iterador
```

```
                        .hasNext();) {
                    retorno.addAll(iterador.next());
                }
            } else if (from == null) {
                set = indices.subMap(indices.firstKey(), to).values();
                for (Iterator<Set<Long>> iterador = set.iterator(); iterador
                        .hasNext();) {
                    retorno.addAll(iterador.next());
                }
            } else if (to == null) {
                set = indices.subMap(from, (Object) c).values();
                for (Iterator<Set<Long>> iterador = set.iterator(); iterador
                        .hasNext();) {
                    retorno.addAll(iterador.next());
                }
            } else {
                set = indices.subMap(from, to).values();
                for (Iterator<Set<Long>> iterador = set.iterator(); iterador
                        .hasNext();) {
                    retorno.addAll(iterador.next());
                }
            }

            return Collections.unmodifiableSet((Set<Long>) retorno);
        }

}
```

En este apartado también realizamos una clase AllowNullComparator, que nos ayudará a comprobar objetos que se le pasen a las funciones de search.

Código de la clase AllowNullComparator:

```
package tienda;

import java.util.*;

/**
 * Clase para comparar objetos
 *
 * @author Alejandro Antonio Martin Almansa
 * @author Mario Garcia Roque
 *
 */
public class AllowNullComparator implements Comparator<Object> {

    public static final AllowNullComparator Instance = new AllowNullComparator();

    /**
     * Metodo para comparar dos objetos
     *
     * @param a
     *            primer objeto a comparar
     * @param b
     *            segundo objeto a comparar
     * @return entero que devuelve la comparacion
     */
    @SuppressWarnings("unchecked")
    @Override
    public int compare(Object a, Object b) {
        if (a == null) {
            return b == null ? 0 : 1;
        } else if (b == null) {
            return 1;
        } else {
            return ((Comparable<Object>) a).compareTo(b);
        }
```

```
	}

}
```

Para probar las distintas implementaciones de este apartado creamos una clase TestIndex.

Código de la clase de Prueba TestIndex:

```
package tests;

import java.util.*;
import tienda.*;

/**
 * Prueba de las clases de la tienda, TypeDescriptor, Entity y de Index
 *
 * @author Alejandro Antonio Martin Almansa
 * @author Mario Garcia Roque
 *
 */
public class TestIndex {

	public static void main(String[] args) {

		Index indice = new Index();
		Collection<Long> coleccion = new HashSet<Long>();
		Collection<Long> coleccion_search = new HashSet<Long>();
		Collection<Long> coleccion_error = new HashSet<Long>();

		indice.add("12", 1L);
		indice.add("12", 2L);
		indice.add("14", 3L);
		indice.add("15", 4L);
		indice.add("16", 5L);

		System.out.println("Prueba add y search: (Salida esperada: 1, 2)");
		coleccion = indice.search("12");
		for (Iterator<Long> iterador = coleccion.iterator(); iterador.hasNext();) {
			System.out.println("Elemento coleccion: " + iterador.next());
		}

		System.out
				.println("Prueba search: (Salida esperada: La clave 1 no
contiene 1L)");
		coleccion_error = indice.search("1");
		if (coleccion_error.isEmpty()) {
			System.out.println("La clave 1 no contiene 1L");
		}

		System.out.println("Prueba delete y search: (Salida esperada: false)");
		indice.delete("16", 5L);
		System.out.println("Coleccion: " + indice.search("16").contains(5L));

		System.out
				.println("Prueba search (from, to): (Salida esperada: 1, 2, 3,
4)");
		coleccion_search = indice.search("11", "17");
		for (Iterator<Long> iterador = coleccion_search.iterator(); iterador
				.hasNext();) {
			System.out.println("Elemento coleccion_search: " + iterador.next());
		}

	}

}
```

Salida: La esperada como podemos comprobar en la siguiente foto.

<terminated> TestIndex [Java Application] C:\Program Files\Java\jre1.8.0_40\bin\javaw.exe (9/4/2015 18:30:

```
Prueba add y search: (Salida esperada: 1, 2)
Elemento coleccion: 1
Elemento coleccion: 2
Prueba search: (Salida esperada: La clave 1 no contiene 1L)
La clave 1 no contiene 1L
Prueba delete y search: (Salida esperada: false)
Coleccion: false
Prueba search (from, to): (Salida esperada: 1, 2, 3, 4)
Elemento coleccion_search: 1
Elemento coleccion_search: 2
Elemento coleccion_search: 3
Elemento coleccion_search: 4
```

Apartado 3:

En el apartado 3 hemos desarrollado la interfaz ITable de la misma forma que la de IIndex, es decir creando una nueva clase que la implemente para desarrollar sus métodos, para así poder guardar los objetos y poder trabajar con ellos de forma más simple.

Código de la declaración de la interfaz ITable:

```
package DAO;

import java.util.*;

/**
 * Interfaz para trabajar con un unico tipo de dato
 *
 * @author Alejandro Antonio Martin Almansa
 * @author Mario Garcia Roque
 *
 */
public interface ITable {

	/**
	 * Devuelve el tipo de dato para el que se ha creado la tabla
	 *
	 * @return tipo de dato de la tabla
	 */
	public ITypeDescriptor getType();

	/**
	 * Lee el objeto de la tabla y lo convierte en una Entity
	 *
	 * @param id
	 *            del objeto de la tabla
	 * @return entidad convertida a partir del id
	 */
	public IEntity getEntity(Long id);

	/**
	 * Actualiza la entidad y los indices, asignandole un ID si este era null, y
	 * devuelve el ID
	 *
	 * @param e
	 *            entidad a actualizar
	 * @return id de la entidad actualizada
	 */
	public long updateEntity(IEntity e);

	/**
	 * Elimina la entidad, si existe, actualizando los indices si es necesario
	 *
	 * @param e
	 *            entidad a eliminar
	 */
	public void delete(IEntity e);

	/**
	 * Busca las entidades que tengan el valor indicado
	 *
	 * @param property
	 *            propiedad a buscar
	 * @param value
	 *            valor de la propiedad que se quiere buscar
	 * @return coleccion con los valores encontrados
	 */
	public Collection<Long> search(String property, Object value);
```

```
	/**
	 * Devuelve la lista de entidades que tienen la propiedad property, entre el
	 * valor from y to
	 *
	 * @param property
	 *            propiedad a buscar
	 * @param from
	 *            valor desde el cual se va a buscar
	 * @param to
	 *            valor hasta el que se va a buscar
	 * @return coleccion con los valores encontrados
	 */
	public Collection<Long> search(String property, Object from, Object to);
}
```

Código de la clase Table que implementa a la interfaz ITable:

```
package tienda;

import java.util.*;
import DAO.*;

/**
 * Clase para implementar la interfaz ITable
 *
 * @author Alejandro Antonio Martin Almansa
 * @author Mario Garcia Roque
 *
 */
public class Table implements ITable {

	private Map<Long, Map<String, Object>> m;
	private ITypeDescriptor type;
	private static Long contador = 0L;
	private Map<String, IIndex> indices;

	/**
	 * Constructor de la clase Table
	 *
	 * @param td
	 *            tipo de dato de la tabla
	 */
	public Table(ITypeDescriptor td) {
		this.m = new HashMap<Long, Map<String, Object>>();
		this.type = td;
		this.indices = new HashMap<String, IIndex>();
	}

	/**
	 * Devuelve el tipo de dato para el que se ha creado la tabla
	 *
	 * @return tipo de dato de la tabla
	 */
	@Override
	public ITypeDescriptor getType() {
		return this.type;
	}

	/**
	 * Lee el objeto de la tabla y lo convierte en una Entity
	 *
	 * @param id
	 *            del objeto de la tabla
	 * @return entidad convertida a partir del id
	 */
	@Override
	public IEntity getEntity(Long id) {
```

```
		IEntity entidad = this.getType().newEntity();
		List<String> propiedades = this.getType().getProperties();

		for (String p : propiedades) {
			entidad.setProperty(p, m.get(id).get(p));
		}

		return entidad;
	}

	/**
	 * Actualiza la entidad y los indices, asignandole un ID si este era null, y
	 * devuelve el ID
	 *
	 * @param e
	 *            entidad a actualizar
	 * @return id de la entidad actualizada
	 */

	@Override
	public long updateEntity(IEntity e) {

		Map<String, Object> n_map = new HashMap<String, Object>();
		List<String> propiedades = this.type.getProperties();
		String propiedad;
		int i;
		Long cont_aux = contador;

		if (e.getId() != null) {
			this.m.remove(e.getId());
		}

		e.setId(contador);

		for (i = 0; i < propiedades.size(); i++) {

			Index ind_aux = new Index();
			propiedad = propiedades.get(i);
			n_map.put(propiedad, e.getProperty(propiedad));
			ind_aux.add(e.getProperty(propiedad), contador);
			indices.put(propiedad, ind_aux);
		}
		this.m.put(e.getId(), n_map);
		contador++;

		return cont_aux;

	}

	/**
	 * Elimina la entidad, si existe, actualizando los indices si es necesario
	 *
	 * @param e
	 *            entidad a eliminar
	 */
	@Override
	public void delete(IEntity e) {

		List<String> propiedades = this.type.getProperties();

		for (String prop : propiedades) {
			this.indices.get(prop).delete(e.getProperty(prop), e.getId());
		}

		this.m.remove(e.getId());
	}
```

```
	/**
	 * Busca las entidades que tengan el valor indicado
	 *
	 * @param property
	 *            propiedad a buscar
	 * @param value
	 *            valor de la propiedad que se quiere buscar
	 * @return coleccion con los valores encontrados
	 */
	@Override
	public Collection<Long> search(String property, Object value) {

		if (property == null || value == null) {
			return null;
		}

		return indices.get(property).search(value);
	}

	/**
	 * Devuelve la lista de entidades que tienen la propiedad property, entre el
	 * valor from y to
	 *
	 * @param property
	 *            propiedad a buscar
	 * @param from
	 *            valor desde el cual se va a buscar
	 * @param to
	 *            valor hasta el que se va a buscar
	 * @return coleccion con los valores encontrados
	 */
	@Override
	public Collection<Long> search(String property, Object from, Object to) {

		return indices.get(property).search(from, to);
	}

}
```

Para este apartado como para los anteriores también desarrollamos un test, TestTable, para comprobar la correcta implementación de la interfaz ITable.

Código de la clase de prueba TestTable:

```
package tests;

import java.util.Collection;
import java.util.HashSet;
import DAO.*;
import tienda.*;

/**
 * Prueba de las clases de la tienda, TypeDescriptor, Entity, Index y Table
 *
 * @author Alejandro Antonio Martin Almansa
 * @author Mario Garcia Roque
 *
 */
public class TestTable {

	public static void main(String[] args) {

		ITypeDescriptor td = Libro.getDescriptor();
		ITable tabla = new Table(td);
		Libro l0 = new Libro();
		Libro l1 = new Libro();
```

```
		Libro l2 = new Libro();
		Collection<Long> coleccion1 = new HashSet<Long>();
		Collection<Long> coleccion2 = new HashSet<Long>();
		Collection<Long> coleccion3 = new HashSet<Long>();

		System.out
				.println("Prueba updateEntity y getId: (Salida esperada: 0,
0)");
		l0.setTitulo("El Quijote");
		System.out.println("Id Libro l0: " + tabla.updateEntity(l0));
		System.out.println("Id Libro l0: " + l0.getId());

		System.out
				.println("Prueba updateEntity y getId: (Salida esperada: 1,
1)");
		l1.setEditorial("sma");
		System.out.println("Id Libro l1: " + tabla.updateEntity(l1));
		System.out.println("Id Libro l1: " + l1.getId());

		System.out
				.println("Prueba updateEntity y getId: (Salida esperada: 2,
2)");
		l2.setEditorial("santillana");
		System.out.println("Id Libro l2: " + tabla.updateEntity(l2));
		System.out.println("Id Libro l2: " + l2.getId());

		System.out
				.println("Prueba getType y getName: (Salida esperada: Libro)");
		System.out.println("Tipo de la tabla: " + tabla.getType().getName());

		System.out
				.println("Prueba getEntity y getProperty: (Salida esperada: El
Quijote)");
		Libro l_aux0 = (Libro) tabla.getEntity(0L);
		System.out.println("Titulo del Libro l_aux0: "
				+ l_aux0.getProperty("titulo"));

		System.out
				.println("Prueba getEntity y getProperty: (Salida esperada:
sma)");
		Libro l_aux1 = (Libro) tabla.getEntity(1L);
		System.out.println("Editorial del Libro l_aux1: "
				+ l_aux1.getProperty("editorial"));

		System.out
				.println("Prueba updateEntity y search: (Salida esperada:
true)");
		tabla.updateEntity(l_aux0);
		tabla.updateEntity(l_aux1);
		coleccion1 = tabla.search("id", 4L);
		if (coleccion1 == null) {
			System.out.println("Error en tabla search");
		} else {
			System.out.println("Search funciona correctamente: "
					+ coleccion1.contains(4L));
		}

		System.out.println("Prueba delete y search: (Salida esperada: false)");
		tabla.delete(l_aux1);
		coleccion2 = tabla.search("id", 4L);
		if (coleccion2.isEmpty()) {
			System.out.println("Encuentra el id 4 en tabla: "
					+ coleccion2.contains(4L));
		} else {
			System.out.println("Error en tabla delete");
		}
```

```
            // La funcion search(property, from, to) no funciona correctamente
            System.out.println("Prueba search de from-to: (Salida esperada: true)");
            coleccion3 = tabla.search("titulo", "El Quijote", "El Quijote");
            if (coleccion3.isEmpty()) {
                  System.out.println("Error en funcion delete from to");
            } else {
                  System.out.println("Search from-to funciona correctamente: "
                              + coleccion3.contains(3L));
            }

      }

}
```

Salida: La esperada, salvo por una función. La función search from-to no funciona correctamente, no sabemos muy bien el porqué dado que la Index si que realiza lo que tiene que hacer.

```
<terminated> TestTable [Java Application] C:\Program Files\Java\jre1.8.0_40\bin\javaw.exe (9/4/2015 18:37
Prueba updateEntity y getId: (Salida esperada: 0, 0)
Id Libro l0: 0
Id Libro l0: 0
Prueba updateEntity y getId: (Salida esperada: 1, 1)
Id Libro l1: 1
Id Libro l1: 1
Prueba updateEntity y getId: (Salida esperada: 2, 2)
Id Libro l2: 2
Id Libro l2: 2
Prueba getType y getName: (Salida esperada: Libro)
Tipo de la tabla: Libro
Prueba getEntity y getProperty: (Salida esperada: El Quijote)
Titulo del Libro l_aux0: El Quijote
Prueba getEntity y getProperty: (Salida esperada: sma)
Editorial del Libro l_aux1: sma
Prueba updateEntity y search: (Salida esperada: true)
Search funciona correctamente: true
Prueba delete y search: (Salida esperada: false)
Encuentra el id 4 en tabla: false
Prueba search de from-to: (Salida esperada: true)
Error en funcion delete from to
```

Apartado 4:

En el apartado 4 hemos desarrollado la interfaz IDAO de la misma forma que las de IIndex e ITable, es decir creando una nueva clase que la implemente para desarrollar sus métodos, para así poder operar con los distintos tipos de datos.

Código de la declaración de la interfaz IDAO:

```
package DAO;

import java.util.*;

/**
 * Interfaz para operar con los tipos de datos
 *
 * @author Alejandro Antonio Martin Almansa
 * @author Mario Garcia Roque
 *
 */
public interface IDAO {

      /**
```

```
     * Registra un nuevo tipo en la base de datos
     *
     * @param t
     *            tipo de dato a registrar
     */
    public void registerType(ITypeDescriptor t);

    /**
     * Lee el objeto de la base de datos, o de cache, si se encuentra en memoria
     *
     * @param type
     *            tipo de dato a leer de la base de datos
     * @param id
     *            id del dato que se quiere leer
     * @return la entidad leida
     */
    public IEntity getEntity(String type, Long id);

    /**
     * Actualiza la entidad, asignandole un ID si este era null, y devuelve el
     * ID
     *
     * @param e
     *            entidad que se quiere actualizar
     * @return id de la entidad
     */
    public long updateEntity(IEntity e);

    /**
     * Elimina la entidad
     *
     * @param e
     *            entidad a eliminar
     * @throws InstantiationException
     *             excepcion si no se instancia
     * @throws IllegalAccessException
     *             excepcion si se accede a memoria a la que no hay permiso
     */
    public void delete(IEntity e) throws InstantiationException,
                IllegalAccessException;

    /**
     * Busca las entidades que tengan el valor indicado
     *
     * @param type
     *            tipo de la entidad que se quiere buscar
     * @param property
     *            propiedad que se quiere buscar
     * @param value
     *            valor de la propiedad
     * @return coleccion de ids de los datos encontrados, vacia si no se
     *         encuentra nada
     */
    public Collection<Long> search(String type, String property, Object value);

    /**
     * Devuelve la lista de entidades que tienen la propiedad property, entre el
     * valor from y to
     *
     * @param type
     *            tipo de la entidad que se quiere buscar
     * @param property
     *            propiedad que se quiere buscar
     * @param from
     *            valor desde el cual se quiere empezar a buscar
     * @param to
     *            valor hasta el que se quiere buscar
```

```
         * @return coleccion de ids de los datos encontrados, vacia si no se
         *         encuentra nada
         */
        public Collection<Long> search(String type, String property, Object from,
                    Object to);
}
```

Código de la clase DAO que implementa a la interfaz IDAO:

```
package tienda;

import java.util.*;
import DAO.*;

/**
 * Clase para implementar la interfaz IDAO
 *
 * @author Alejandro Antonio Martin Almansa
 * @author Mario Garcia Roque
 *
 */
public class DAO implements IDAO {

      private List<Table> tablas;

      /**
       * Constructor de la clase DAO
       */
      public DAO() {
            this.tablas = new ArrayList<Table>();
      }

      /**
       * Registra un nuevo tipo en la base de datos
       *
       * @param t
       *            tipo de dato a registrar
       */
      @Override
      public void registerType(ITypeDescriptor t) {

            this.tablas.add(new Table(t));
      }

      /**
       * Lee el objeto de la base de datos, o de cache, si se encuentra en memoria
       *
       * @param type
       *            tipo de dato a leer de la base de datos
       * @param id
       *            id del dato que se quiere leer
       * @return la entidad leida
       */
      @Override
      public IEntity getEntity(String type, Long id) {

            for (Table tab : tablas) {
                  if (tab.getType().getName().equals(type)) {
                        return tab.getEntity(id);
                  }
            }

            return null;
      }

      /**
       * Actualiza la entidad, asignandole un ID si este era null, y devuelve el
       * ID
```

```
 *
 * @param e
 *            entidad que se quiere actualizar
 * @return id de la entidad
 */
@Override
public long updateEntity(IEntity e) {

	for (Table tab : tablas) {
		if (tab.getType().getName().equals(e.getType())) {
			return tab.updateEntity(e);
		}
	}

	return -1L;
}

/**
 * Elimina la entidad
 *
 * @param e
 *            entidad a eliminar
 * @throws InstantiationException
 *             excepcion si no se instancia
 * @throws IllegalAccessException
 *             excepcion si se accede a memoria a la que no hay permiso
 */
@Override
public void delete(IEntity e) throws InstantiationException,
		IllegalAccessException {

	for (Table tab : tablas) {
		if (tab.getType().getName().equals(e.getType())) {
			tab.delete(e);
		}
	}

}

/**
 * Busca las entidades que tengan el valor indicado
 *
 * @param type
 *            tipo de la entidad que se quiere buscar
 * @param property
 *            propiedad que se quiere buscar
 * @param value
 *            valor de la propiedad
 * @return coleccion de ids de los datos encontrados, vacia si no se
 *         encuentra nada
 */
@Override
public Collection<Long> search(String type, String property, Object value) {

	for (Table tab : tablas) {
		if (tab.getType().getName().equals(type)) {
			return tab.search(property, value);
		}
	}

	return Collections.emptySet();
}

/**
 * Devuelve la lista de entidades que tienen la propiedad property, entre el
 * valor from y to
 *
```

```
	 * @param type
	 *            tipo de la entidad que se quiere buscar
	 * @param property
	 *            propiedad que se quiere buscar
	 * @param from
	 *            valor desde el cual se quiere empezar a buscar
	 * @param to
	 *            valor hasta el que se quiere buscar
	 * @return coleccion de ids de los datos encontrados, vacia si no se
	 *         encuentra nada
	 */
	@Override
	public Collection<Long> search(String type, String property, Object from,
			Object to) {

		for (Table tab : tablas) {
			if (tab.getType().getName().equals(type)) {
				return tab.search(property, from, to);
			}
		}

		return Collections.emptySet();
	}

}
```

Como para los apartados anteriores, en este también realizamos una clase de prueba, TestDAO, para poder probar la funcionalidad de la base de datos al completo.

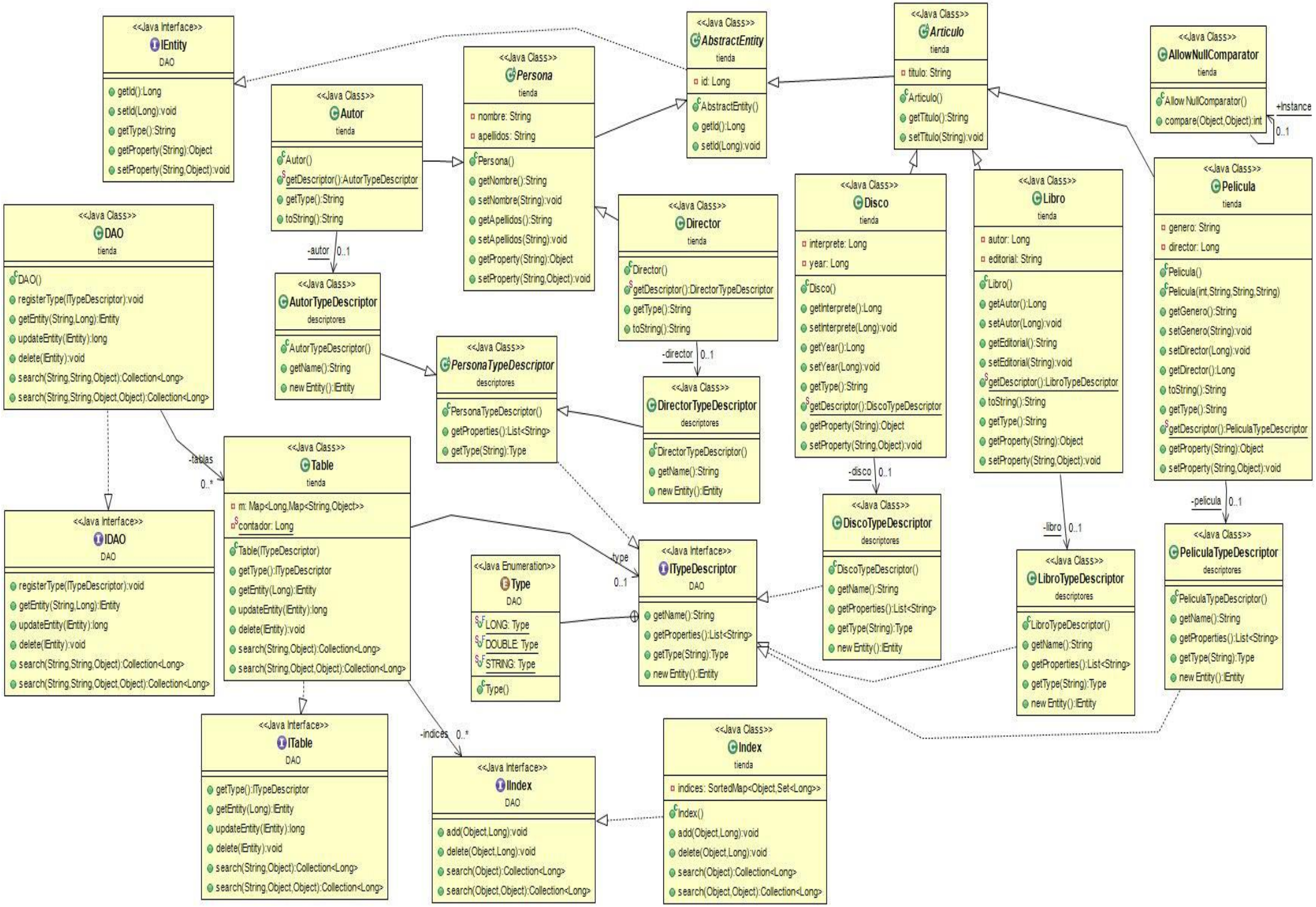
Código de la clase de prueba TestDAO:

```
package tests;

import tienda.*;
import java.util.*;
import descriptores.*;

/**
 * Prueba de las clases de la tienda, TypeDescriptor, Entity, Index, Table y DAO
 *
 * @author Alejandro Antonio Martin Almansa
 * @author Mario Garcia Roque
 *
 */
public class TestDAO {

	public static void main(String[] args) {

		DAO dao = new DAO();
		Collection<Long> coleccion;

		try {

			PeliculaTypeDescriptor tdpeli = Pelicula.getDescriptor();
			DirectorTypeDescriptor tddirector = Director.getDescriptor();

			Pelicula p = (Pelicula) tdpeli.newEntity();
			Director d = (Director) tddirector.newEntity();

			p.setTitulo("TLOTR");
			p.setProperty("genero", "Ciencia F");

			d.setNombre("nombre");
			d.setApellidos("apellidos");

			dao.registerType(tddirector);
			dao.updateEntity(d);
```

```
                    p.setProperty("director", d.getId());

                    dao.registerType(tdpeli);
                    dao.updateEntity(p);

                    System.out
                                .println("Prueba getEntity, registerType y updateEntity:
");
                    System.out.println("Salida esperada: ");
                    System.out
                                .println("  Pelicula: id=1, titulo=TLOTR,
genero=CienciaF, Director=0");
                    System.out
                                .println("  Director: id=0, nombre=nombre,
apellidos=apellidos");
                    System.out.println("Pelicula: "
                                + dao.getEntity("Pelicula", p.getId()));
                    System.out.println("Director de la pelicula: "
                                + dao.getEntity("Director", d.getId()));

                    coleccion = dao.search("Director", "nombre", "nombre");
                    if (coleccion.isEmpty()) {
                          System.out.println("Busqueda de nombre sin exito");
                    }

                    System.out.println("Prueba search: (Salida esperada: 0)");
                    for (Iterator<Long> iterador = coleccion.iterator(); iterador
                                .hasNext();) {
                          System.out.println("Elemento coleccion: " + iterador.next());
                    }

                    System.out.println("Prueba search: (Salida esperada: 1)");
                    System.out.println("Busqueda de pelicula: "
                                + dao.search("Pelicula", "titulo", "TLOTR").toString());

                    System.out.println("Prueba delete: (Salida esperada: false)");
                    dao.delete(p);
                    System.out.println("Busqueda de pelicula: "
                                + dao.search("Pelicula", "titulo",
"TLOTR").contains(1L));

              } catch (Exception excep) {
                    excep.printStackTrace();
              }

        }

}
```

Salida: La esperada, salvo por una función. La función search from-to no funciona correctamente y en este caso no la introducimos en el test porque hay un null pointer que no sabemos solucionar.

```
<terminated> TestDAO [Java Application] C:\Program Files\Java\jre1.8.0_40\bin\javaw.exe (9/4/2015 18:44:5
Prueba getEntity, registerType y updateEntity:
Salida esperada:
        Pelicula: id=1, titulo=TLOTR, genero=CienciaF, Director=0
        Director: id=0, nombre=nombre, apellidos=apellidos
Pelicula: [1]PELICULA: TLOTR. (Ciencia F) Dir: 0
Director de la pelicula: [0]DIRECTOR: nombre apellidos
Prueba search: (Salida esperada: 0)
Elemento coleccion: 0
Prueba search: (Salida esperada: 1)
Busqueda de pelicula: [1]
Prueba delete: (Salida esperada: false)
Busqueda de pelicula: false
```


<<Java Interface>>
IEntity
DAO
getId():Long
setId(Long):void
getType():String
getProperty(String):Object
setProperty(String,Object):void
<<Java Class>>
AbstractEntity
tienda
id: Long
AbstractEntity()
getId():Long
setId(Long):void
<<Java Class>>
Articulo
tienda
titulo: String
Articulo()
getTitulo():String
setTitulo(String):void
<<Java Class>>
AllowNullComparator
tienda
AllowNullComparator()
compare(Object,Object):int
+Instance
0..1
<<Java Class>>
Persona
tienda
nombre: String
apellidos: String
Persona()
getNombre():String
setNombre(String):void
getApellidos():String
setApellidos(String):void
getProperty(String):Object
setProperty(String,Object):void
<<Java Class>>
Autor
tienda
Autor()
getDescriptor():AutorTypeDescriptor
getType():String
toString():String
-autor
0..1
<<Java Class>>
Director
tienda
Director()
getDescriptor():DirectorTypeDescriptor
getType():String
toString():String
-director
0..1
<<Java Class>>
Disco
tienda
interprete: Long
year: Long
Disco()
getInterprete():Long
setInterprete(Long):void
getYear():Long
setYear(Long):void
getType():String
getDescriptor():DiscoTypeDescriptor
getProperty(String):Object
setProperty(String,Object):void
<<Java Class>>
Libro
tienda
autor: Long
editorial: String
Libro()
getAutor():Long
setAutor(Long):void
getEditorial():String
setEditorial(String):void
getDescriptor():LibroTypeDescriptor
toString():String
getType():String
getProperty(String):Object
setProperty(String,Object):void
<<Java Class>>
Pelicula
tienda
genero: String
director: Long
Pelicula()
Pelicula(int,String,String,String)
getGenero():String
setGenero(String):void
setDirector(Long):void
getDirector():Long
toString():String
getType():String
getDescriptor():PeliculaTypeDescriptor
getProperty(String):Object
setProperty(String,Object):void
<<Java Class>>
DAO
tienda
DAO()
registerType(ITypeDescriptor):void
getEntity(String,Long):IEntity
updateEntity(IEntity):long
delete(IEntity):void
search(String,String,Object):Collection<Long>
search(String,String,Object,Object):Collection<Long>
-tablas
0..*
<<Java Class>>
AutorTypeDescriptor
descriptores
AutorTypeDescriptor()
getName():String
newEntity():IEntity
<<Java Class>>
PersonaTypeDescriptor
descriptores
PersonaTypeDescriptor()
getProperties():List<String>
getType(String):Type
<<Java Class>>
DirectorTypeDescriptor
descriptores
DirectorTypeDescriptor()
getName():String
newEntity():IEntity
-disco
0..1
<<Java Class>>
DiscoTypeDescriptor
descriptores
DiscoTypeDescriptor()
getName():String
getProperties():List<String>
getType(String):Type
newEntity():IEntity
-libro
0..1
<<Java Class>>
LibroTypeDescriptor
descriptores
LibroTypeDescriptor()
getName():String
getProperties():List<String>
getType(String):Type
newEntity():IEntity
-pelicula
0..1
<<Java Class>>
PeliculaTypeDescriptor
descriptores
PeliculaTypeDescriptor()
getName():String
getProperties():List<String>
getType(String):Type
newEntity():IEntity
<<Java Interface>>
IDAO
DAO
registerType(ITypeDescriptor):void
getEntity(String,Long):IEntity
updateEntity(IEntity):long
delete(IEntity):void
search(String,String,Object):Collection<Long>
search(String,String,Object,Object):Collection<Long>
<<Java Class>>
Table
tienda
m: Map<Long,Map<String,Object>>
contador: Long
Table(ITypeDescriptor)
getType():ITypeDescriptor
getEntity(Long):IEntity
updateEntity(IEntity):long
delete(IEntity):void
search(String,Object):Collection<Long>
search(String,Object,Object):Collection<Long>
type
0..1
<<Java Interface>>
ITypeDescriptor
DAO
getName():String
getProperties():List<String>
getType(String):Type
newEntity():IEntity
<<Java Enumeration>>
Type
DAO
LONG: Type
DOUBLE: Type
STRING: Type
Type()
<<Java Interface>>
ITable
DAO
getType():ITypeDescriptor
getEntity(Long):IEntity
updateEntity(IEntity):long
delete(IEntity):void
search(String,Object):Collection<Long>
search(String,Object,Object):Collection<Long>
-indices
0..*
<<Java Interface>>
IIndex
DAO
add(Object,Long):void
delete(Object,Long):void
search(Object):Collection<Long>
search(Object,Object):Collection<Long>
<<Java Class>>
Index
tienda
indices: SortedMap<Object,Set<Long>>
Index()
add(Object,Long):void
delete(Object,Long):void
search(Object):Collection<Long>
search(Object,Object):Collection<Long>