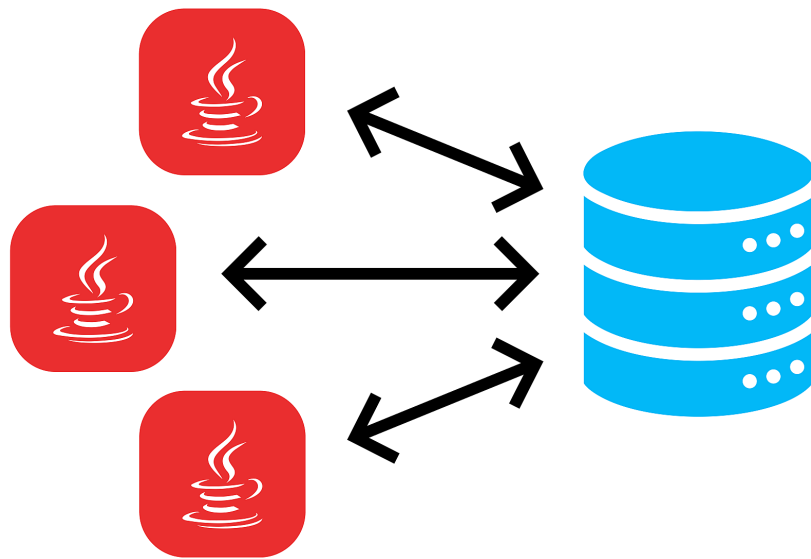


Taller UD10_3: Patron DAO (CRUD completo)



1. **Introducción**
2. **Esquema de la BBDD**
3. **Implementación paso a paso**
4. **Actividades**
5. **Fuentes de información**

1. Introducción

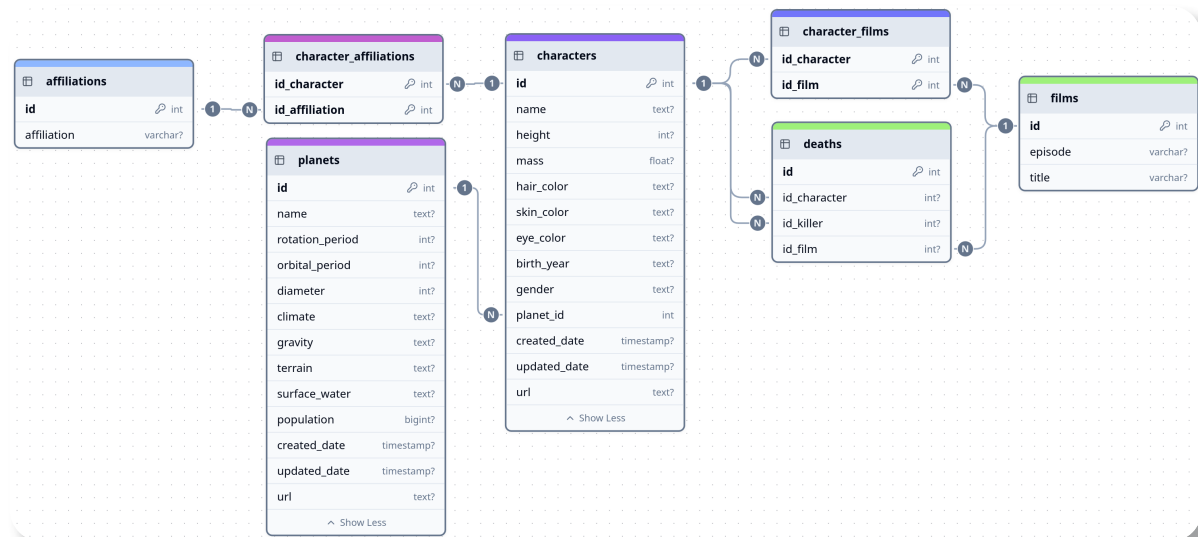
En el apartado correspondiente de teoría, ya estudiamos los patrones `Singleton` y `DAO`.

Importante

Para realizar el ejemplo que seguiremos en este apartado usaremos la BBDD starwars. Pero tu deberás replicar este ejemplo para la BBDD que creaste en el anterior taller [UD10_T2_AWS_Intellij_ES](#).

En programación existen una serie de estándares denominados [Patrones de Diseño](#) que debes conocer para poder programar según estos patrones y no reinventar la rueda.

2. Esquema de la BBDD



3. Implementación paso a paso

Paso 0: Definir la clase `DbConnect` (`Singleton`), usando [HikariCP](#)

Para simplificar la implementación y conexión a la base de datos podemos crear la clase `DbConnect` (como ya hemos implementado anteriormente en algunos ejercicios):

```

1  package es.martinezpenya.ejemplos.UD10.starwars;
2
3  import com.zaxxer.hikari.HikariConfig;
4  import com.zaxxer.hikari.HikariDataSource;
5  import java.sql.Connection;
6  import java.sql.SQLException;
7
8  public class DBConnect {
9
10     private static HikariDataSource dataSource;
11
12     // Configuración del pool de conexiones
13     static {
14         HikariConfig config = new HikariConfig();
15         config.setJdbcUrl("jdbc:mariadb://databasedmp.cipxbdkxiaqy.us-east-1.rds.amazonaws.com:6000/starwars"); // URL de la base de datos
16         config.setUsername("admin"); // Usuario de la base de datos
17         config.setPassword("123456Ab$"); // Contraseña de la base de datos
18         config.setMaximumPoolSize(20); // Número máximo de conexiones en el pool
19         config.setMinimumIdle(2); // Número mínimo de conexiones inactivas en el pool
20         config.setIdleTimeout(3000); // Tiempo de espera para conexiones inactivas (en milisegundos)
21         config.setMaxLifetime(180000); // Tiempo máximo de vida de una conexión (en milisegundos)
22         config.setConnectionTimeout(300000); // Tiempo de espera para obtener una conexión (en milisegundos)
23
24         dataSource = new HikariDataSource(config);
25     }
26
27     // Método para obtener una conexión del pool
28     public static Connection getConnection() throws SQLException {
29         return dataSource.getConnection();
30     }
31
32     // Método para cerrar el pool de conexiones (opcional)
33     public static void closeDataSource() {
34         if (dataSource != null) {
35             dataSource.close();
36         }
37     }
38 }

```

Paso 1: Definir las clases de las entidades

Definiremos las clases `Film`, `Planet`, `Character` y `Characterfilm`, pero tu deberás hacerlo con todas tus tablas (o al menos 3 de ellas, y 2 deben estar relacionadas), deberás incluir todos los campos de las tablas, los constructores que estimes oportuno, los getters y setters y un método `toString()`:

Film

```

1  package es.martinezpenya.ejemplos.UD10.starwars;
2
3  public class Film {
4      private int id;
5      private String episode;
6      private String title;
7
8      public Film(int id, String episode, String title) {
9          this.id = id;
10         this.episode = episode;
11         this.title = title;
12     }

```

```

13
14     public Film(int id, String title) {
15         this.id = id;
16         this.title = title;
17     }
18
19     // Getters y setters
20     public int getId() {
21         return id;
22     }
23
24     public void setId(int id) {
25         this.id = id;
26     }
27
28     public String getEpisode() {
29         return episode;
30     }
31
32     public void setEpisode(String episode) {
33         this.episode = episode;
34     }
35
36     public String getTitle() {
37         return title;
38     }
39
40     public void setTitle(String title) {
41         this.title = title;
42     }
43
44     @Override
45     public String toString() {
46         return "Film{" +
47             "id=" + id +
48             ", episode='" + episode + '\'' +
49             ", title='" + title + '\'' +
50             '}';
51     }
52 }

```

Planet

```

1  package es.martinezpenya.ejemplos.UD10.starwars;
2
3  public class Planet {
4      private int id;
5      private String name;
6      private String rotation_period;
7      private String orbital_period;
8      private String diameter;
9      private String climate;
10     private String gravity;
11     private String terrain;
12     private String surface_water;
13     private String population;
14     private String created_date;
15     private String updated_date;
16     private String url;
17
18     public Planet(int id, String name, String rotation_period, String orbital_period,
19 String diameter, String climate, String gravity, String terrain, String surface_water,
20 String population) {
21         this.id = id;
22         this.name = name;
23         this.rotation_period = rotation_period;
24         this.orbital_period = orbital_period;
25         this.diameter = diameter;
26         this.climate = climate;
27         this.gravity = gravity;
28         this.terrain = terrain;
29         this.surface_water = surface_water;
30         this.population = population;
31     }
32 }

```

```

30
31     public Planet(int id, String name) {
32         this.id = id;
33         this.name = name;
34     }
35
36     // Getters y setters
37     public int getId() {
38         return id;
39     }
40
41     public void setId(int id) {
42         this.id = id;
43     }
44
45     public String getName() {
46         return name;
47     }
48
49     public void setName(String name) {
50         this.name = name;
51     }
52
53     public String getRotation_period() {
54         return rotation_period;
55     }
56
57     public void setRotation_period(String rotation_period) {
58         this.rotation_period = rotation_period;
59     }
60
61     public String getOrbital_period() {
62         return orbital_period;
63     }
64
65     public void setOrbital_period(String orbital_period) {
66         this.orbital_period = orbital_period;
67     }
68
69     public String getDiameter() {
70         return diameter;
71     }
72
73     public void setDiameter(String diameter) {
74         this.diameter = diameter;
75     }
76
77     public String getClimate() {
78         return climate;
79     }
80
81     public void setClimate(String climate) {
82         this.climate = climate;
83     }
84
85     public String getGravity() {
86         return gravity;
87     }
88
89     public void setGravity(String gravity) {
90         this.gravity = gravity;
91     }
92
93     public String getTerrain() {
94         return terrain;
95     }
96
97     public void setTerrain(String terrain) {
98         this.terrain = terrain;
99     }
100
101     public String getSurface_water() {
102         return surface_water;
103     }
104

```

```

105     public void setSurface_water(String surface_water) {
106         this.surface_water = surface_water;
107     }
108
109     public String getPopulation() {
110         return population;
111     }
112
113     public void setPopulation(String population) {
114         this.population = population;
115     }
116
117     public String getCreated_date() {
118         return created_date;
119     }
120
121     public void setCreated_date(String created_date) {
122         this.created_date = created_date;
123     }
124
125     public String getUpdated_date() {
126         return updated_date;
127     }
128
129     public void setUpdated_date(String updated_date) {
130         this.updated_date = updated_date;
131     }
132
133     public String getUrl() {
134         return url;
135     }
136
137     public void setUrl(String url) {
138         this.url = url;
139     }
140
141     @Override
142     public String toString() {
143         return "Planet{" +
144             "id=" + id +
145             ", name='" + name + '\'' +
146             ", rotation_period='" + rotation_period + '\'' +
147             ", orbital_period='" + orbital_period + '\'' +
148             ", diameter='" + diameter + '\'' +
149             ", climate='" + climate + '\'' +
150             ", gravity='" + gravity + '\'' +
151             ", terrain='" + terrain + '\'' +
152             ", surface_water='" + surface_water + '\'' +
153             ", population='" + population + '\'' +
154             ", created_date='" + created_date + '\'' +
155             ", updated_date='" + updated_date + '\'' +
156             ", url='" + url + '\'' +
157             '}';
158     }
159 }

```

Character

```

1  package es.martinezpenya.ejemplos.UD10.starwars;
2
3  import java.time.LocalDateTime;
4
5  public class Character {
6      private int id;
7      private String name;
8      private int height;
9      private double mass;
10     private String hair_color;
11     private String skin_color;
12     private String eye_color;
13     private String birth_year;
14     private String gender;
15     private Planet planet_id;
16     private LocalDateTime created_date;

```



```

17     private LocalDateTime updated_date;
18     private String url;
19
20     public Character(int id, String name, int height, double mass, String hair_color,
String skin_color, String eye_color, String birth_year, String gender, Planet planet_id)
{
21         this.id = id;
22         this.name = name;
23         this.height = height;
24         this.mass = mass;
25         this.hair_color = hair_color;
26         this.skin_color = skin_color;
27         this.eye_color = eye_color;
28         this.birth_year = birth_year;
29         this.gender = gender;
30         this.planet_id = planet_id;
31     }
32
33     public Character(int id, String name) {
34         this.id = id;
35         this.name = name;
36     }
37
38     // Getters y setters
39     public int getId() {
40         return id;
41     }
42
43     public void setId(int id) {
44         this.id = id;
45     }
46
47     public String getName() {
48         return name;
49     }
50
51     public void setName(String name) {
52         this.name = name;
53     }
54
55     public int getHeight() {
56         return height;
57     }
58
59     public void setHeight(int height) {
60         this.height = height;
61     }
62
63     public double getMass() {
64         return mass;
65     }
66
67     public void setMass(double mass) {
68         this.mass = mass;
69     }
70
71     public String getHair_color() {
72         return hair_color;
73     }
74
75     public void setHair_color(String hair_color) {
76         this.hair_color = hair_color;
77     }
78
79     public String getSkin_color() {
80         return skin_color;
81     }
82
83     public void setSkin_color(String skin_color) {
84         this.skin_color = skin_color;
85     }
86
87     public String getEye_color() {
88         return eye_color;
89     }

```

```

90
91     public void setEye_color(String eye_color) {
92         this.eye_color = eye_color;
93     }
94
95     public String getBirth_year() {
96         return birth_year;
97     }
98
99     public void setBirth_year(String birth_year) {
100         this.birth_year = birth_year;
101     }
102
103     public String getGender() {
104         return gender;
105     }
106
107     public void setGender(String gender) {
108         this.gender = gender;
109     }
110
111     public Planet getPlanet_id() {
112         return planet_id;
113     }
114
115     public void setPlanet_id(Planet planet_id) {
116         this.planet_id = planet_id;
117     }
118
119     public LocalDateTime getCreated_date() {
120         return created_date;
121     }
122
123     public void setCreated_date(LocalDateTime created_date) {
124         this.created_date = created_date;
125     }
126
127     public LocalDateTime getUpdated_date() {
128         return updated_date;
129     }
130
131     public void setUpdated_date(LocalDateTime updated_date) {
132         this.updated_date = updated_date;
133     }
134
135     public String getUrl() {
136         return url;
137     }
138
139     public void setUrl(String url) {
140         this.url = url;
141     }
142
143     @Override
144     public String toString() {
145         return "Character{" +
146             "id=" + id +
147             ", name='" + name + '\'' +
148             ", height=" + height +
149             ", mass=" + mass +
150             ", hair_color='" + hair_color + '\'' +
151             ", skin_color='" + skin_color + '\'' +
152             ", eye_color='" + eye_color + '\'' +
153             ", birth_year='" + birth_year + '\'' +
154             ", gender='" + gender + '\'' +
155             ", planet='" + planet_id + '\'' +
156             ", created_date=" + created_date +
157             ", updated_date=" + updated_date +
158             ", url='" + url + '\'' +
159             '}';
160
161     }
162 }

```

CharacterFilm

```

1 package es.martinezpenya.ejemplos.UD10.starwars;
2
3 public class CharacterFilm {
4     private Character character;
5     private Film film;
6
7     public CharacterFilm(Character character, Film film) {
8         this.character = character;
9         this.film = film;
10    }
11
12    public Character getCharacter() {
13        return character;
14    }
15
16    public void setCharacter(Character character) {
17        this.character = character;
18    }
19
20    public Film getFilm() {
21        return film;
22    }
23
24    public void setFilm(Film film) {
25        this.film = film;
26    }
27
28    @Override
29    public String toString() {
30        return "CharacterFilm{" +
31            "character=" + character +
32            ", film=" + film +
33            '}';
34    }
35 }

```

Repite el proceso anterior para todas las clases de tu BD.

Paso 2: Definir la interface DAO

Luego, definiremos la interfaz DAO para después implementarlas en cada entidad.

DAO

```

1 package es.martinezpenya.ejemplos.UD10.starwars;
2
3 import java.sql.SQLException;
4 import java.util.ArrayList;
5
6 public interface DAO<T> {
7     void crear(T entidad) throws SQLException;;           // C: Create
8     T obtener(int id) throws SQLException;;               // R: Read
9     ArrayList<T> obtenerTodos() throws SQLException;;     // R: Read (all)
10    void actualizar(T entidad) throws SQLException;;       // U: Update
11    void eliminar(int id) throws SQLException;;            // D: Delete
12 }

```

Paso 3: Implementar las clases DAO

Luego, implementaremos las clases DAO para cada entidad e interactuando con la base de datos.

FilmDAO

```

1 package es.martinezpenya.ejemplos.UD10.starwars;
2
3 import java.sql.*;
4 import java.util.ArrayList;
5
6 public class FilmDAO implements DAO<Film> {
7     private static final String INSERT_QUERY = "INSERT INTO films (id, episode, title)
VALUES (?, ?, ?)";

```

```

8     private static final String SELECT_BY_ID_QUERY = "SELECT * FROM films WHERE id = ?";
9     private static final String SELECT_ALL_QUERY = "SELECT * FROM films";
10    private static final String UPDATE_QUERY = "UPDATE films SET episode = ?, title = ?
WHERE id = ?";
11    private static final String DELETE_QUERY = "DELETE FROM films WHERE id = ?";
12    private Connection con;
13
14    public FilmDAO () {
15        try {
16            con = DBConnect.getConnection();
17        } catch (SQLException e) {
18            System.out.println("ERROR al conectar: " + e.getMessage());
19        }
20    }
21
22    @Override
23    public void crear(Film film) throws SQLException {
24        try (PreparedStatement pst = con.prepareStatement(INSERT_QUERY,
Statement.RETURN_GENERATED_KEYS)) {
25
26            pst.setInt(1, film.getId());
27            pst.setString(2, film.getEpisode());
28            pst.setString(3, film.getTitle());
29
30            int filasInsertadas = pst.executeUpdate();
31            /* El siguiente fragmento serviría para el caso en que la tabla tuviera un
campo autoincremental
32            con pst.getGeneratedKeys() obtenemos el id generado y se lo podemos asignar
al objeto
33            if (filasInsertadas > 0) {
34                ResultSet rs = pst.getGeneratedKeys();
35                if (rs.next()) {
36                    int id = rs.getInt(1);
37                    film.setId(id);
38                }
39            }
40            */
41        }
42    }
43
44    @Override
45    public Film obtener(int id) throws SQLException {
46        Film usuario = null;
47        try (PreparedStatement pst = con.prepareStatement(SELECT_BY_ID_QUERY)) {
48
49            pst.setInt(1, id);
50            ResultSet rs = pst.executeQuery();
51            if (rs.next()) {
52                int idDevuelto = rs.getInt("id");
53                String episode = rs.getString("episode");
54                String title = rs.getString("title");
55                usuario = new Film(id, episode, title);
56            }
57        }
58        return usuario;
59    }
60
61    @Override
62    public ArrayList<Film> obtenerTodos() throws SQLException {
63        ArrayList<Film> films = new ArrayList<>();
64        try (Statement st = con.createStatement()) {
65            ResultSet rs = st.executeQuery(SELECT_ALL_QUERY) {
66                while (rs.next()) {
67                    int id = rs.getInt("id");
68                    String episode = rs.getString("episode");
69                    String title = rs.getString("title");
70                    Film film = new Film(id, episode, title);
71                    films.add(film);
72                }
73            }
74            return films;
75        }
76
77    @Override
78    public void actualizar(Film usuario) throws SQLException {

```

```

79         try (PreparedStatement pst = con.prepareStatement(UPDATE_QUERY)) {
80             pst.setString(1, usuario.getEpisode());
81             pst.setString(2, usuario.getTitle());
82             pst.setInt(3, usuario.getId());
83             pst.executeUpdate();
84         }
85     }
86
87     @Override
88     public void eliminar(int id) throws SQLException {
89         try (PreparedStatement pst = con.prepareStatement(DELETE_QUERY)) {
90             pst.setInt(1, id);
91             pst.executeUpdate();
92         }
93     }
94 }

```

PlanetDAO

```

1  package es.martinezpenya.ejemplos.UD10.starwars;
2
3  import java.sql.*;
4  import java.util.ArrayList;
5
6  public class PlanetDAO implements DAO<Planet> {
7      private static final String INSERT_QUERY = "INSERT INTO planets (id, name,
8      rotation_period, orbital_period, diameter, climate, gravity, terrain, surface_water,
9      population) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";
10     private static final String SELECT_BY_ID_QUERY = "SELECT * FROM planets WHERE id =
11     ?";
12     private static final String SELECT_ALL_QUERY = "SELECT * FROM planets";
13     private static final String UPDATE_QUERY = "UPDATE planets SET name = ?,
14     rotation_period = ?, orbital_period = ?, diameter = ?, climate = ?, gravity = ?, terrain
15     = ?, surface_water = ?, population = ? WHERE id = ?";
16     private static final String DELETE_QUERY = "DELETE FROM planets WHERE id = ?";
17     private Connection con;
18
19     public PlanetDAO () {
20         try {
21             con = DBConnect.getConnection();
22         } catch (SQLException e) {
23             System.out.println("ERROR al conectar: " + e.getMessage());
24         }
25     }
26
27     public void crear(Planet planet) throws SQLException {
28         try (PreparedStatement pst = con.prepareStatement(INSERT_QUERY,
29         Statement.RETURN_GENERATED_KEYS)) {
30
31             pst.setInt(1, planet.getId());
32             pst.setString(2, planet.getName());
33             pst.setString(3, planet.getRotation_period());
34             pst.setString(4, planet.getOrbital_period());
35             pst.setString(5, planet.getDiameter());
36             pst.setString(6, planet.getClimate());
37             pst.setString(7, planet.getGravity());
38             pst.setString(8, planet.getTerrain());
39             pst.setString(9, planet.getSurface_water());
40             pst.setString(10, planet.getPopulation());
41
42             int filasInsertadas = pst.executeUpdate();
43             if (filasInsertadas > 0) {
44                 ResultSet rs = pst.getGeneratedKeys();
45                 if (rs.next()) {
46                     int id = rs.getInt(1);
47                     planet.setId(id);
48                 }
49             }
50         }
51     }
52
53     public Planet obtener(int id) throws SQLException {
54         Planet planet = null;
55         try (PreparedStatement pst = con.prepareStatement(SELECT_BY_ID_QUERY)) {
56

```

```

50
51         pst.setInt(1, id);
52         ResultSet rs = pst.executeQuery();
53         if (rs.next()) {
54             int idDevuelto = rs.getInt("id");
55             String name = rs.getString("name");
56             String rotation_period = rs.getString("rotation_period");
57             String orbital_period = rs.getString("orbital_period");
58             String diameter = rs.getString("diameter");
59             String climate = rs.getString("climate");
60             String gravity = rs.getString("gravity");
61             String terrain = rs.getString("terrain");
62             String surface_water = rs.getString("surface_water");
63             String population = rs.getString("population");
64             Planet planet = new Planet(id, name, rotation_period, orbital_period, diameter,
climate, gravity, terrain, surface_water, population);
65         }
66     }
67     return planet;
68 }
69
70 @Override
71 public ArrayList<Planet> obtenerTodos() throws SQLException {
72     ArrayList<Planet> planets = new ArrayList<>();
73     try (Statement st = con.createStatement();
74         ResultSet rs = st.executeQuery(SELECT_ALL_QUERY)) {
75         while (rs.next()) {
76             int id = rs.getInt("id");
77             String name = rs.getString("name");
78             String rotation_period = rs.getString("rotation_period");
79             String orbital_period = rs.getString("orbital_period");
80             String diameter = rs.getString("diameter");
81             String climate = rs.getString("climate");
82             String gravity = rs.getString("gravity");
83             String terrain = rs.getString("terrain");
84             String surface_water = rs.getString("surface_water");
85             String population = rs.getString("population");
86             Planet planet = new Planet(id, name, rotation_period, orbital_period,
diameter, climate, gravity, terrain, surface_water, population);
87             planets.add(planet);
88         }
89     }
90     return planets;
91 }
92
93 public void actualizar(Planet planet) throws SQLException {
94     try (PreparedStatement pst = con.prepareStatement(UPDATE_QUERY)) {
95
96         pst.setString(1, planet.getName());
97         pst.setString(2, planet.getRotation_period());
98         pst.setString(3, planet.getOrbital_period());
99         pst.setString(4, planet.getDiameter());
100        pst.setString(5, planet.getClimate());
101        pst.setString(6, planet.getGravity());
102        pst.setString(7, planet.getTerrain());
103        pst.setString(8, planet.getSurface_water());
104        pst.setString(9, planet.getPopulation());
105        pst.setInt(10, planet.getId());
106
107        pst.executeUpdate();
108    }
109 }
110
111 public void eliminar(int id) throws SQLException {
112     try (PreparedStatement pst = con.prepareStatement(DELETE_QUERY)) {
113
114         pst.setInt(1, id);
115         pst.executeUpdate();
116     }
117 }
118 }

```

CharacterDAO

```

1 package es.martinezpenya.ejemplos.UD10.starwars;
2
3 import java.sql.*;
4 import java.util.ArrayList;
5
6 public class CharacterDAO implements DAO<Character> {
7     private static final String INSERT_QUERY = "INSERT INTO characters (id, name, height,
8 mass, hair_color, skin_color, eye_color, birth_year, gender, planet_id) VALUES (?, ?, ?,
9 ?, ?, ?, ?, ?, ?, ?)";
10    private static final String SELECT_BY_ID_QUERY = "SELECT * FROM characters WHERE id =
11 ?";
12    private static final String SELECT_ALL_QUERY = "SELECT * FROM characters";
13    private static final String UPDATE_QUERY = "UPDATE characters SET name = ?, height =
14 ?, mass = ?, hair_color = ?, skin_color = ?, eye_color = ?, birth_year = ?, gender = ?,
15 planet_id = ? WHERE id = ?";
16    private static final String DELETE_QUERY = "DELETE FROM characters WHERE id = ?";
17    private Connection con;
18
19    public CharacterDAO() {
20        try {
21            con = DBConnect.getConnection();
22        } catch (SQLException e) {
23            System.out.println("ERROR al conectar: " + e.getMessage());
24        }
25    }
26
27    public void crear(Character character) throws SQLException {
28        try (PreparedStatement pst = con.prepareStatement(INSERT_QUERY,
29 Statement.RETURN_GENERATED_KEYS)) {
30
31            pst.setInt(1, character.getId());
32            pst.setString(2, character.getName());
33            pst.setInt(3, character.getHeight());
34            pst.setDouble(4, character.getMass());
35            pst.setString(5, character.getHair_color());
36            pst.setString(6, character.getSkin_color());
37            pst.setString(7, character.getEye_color());
38            pst.setString(8, character.getBirth_year());
39            pst.setString(9, character.getGender());
40            pst.setInt(10, character.getPlanet_id().getId());
41
42            int filasInsertadas = pst.executeUpdate();
43            if (filasInsertadas > 0) {
44                ResultSet rs = pst.getGeneratedKeys();
45                if (rs.next()) {
46                    int id = rs.getInt(1);
47                    character.setId(id);
48                }
49            }
50        }
51    }
52
53    @Override
54    public Character obtener(int id) throws SQLException {
55        Character character = null;
56        try (PreparedStatement pst = con.prepareStatement(SELECT_BY_ID_QUERY)) {
57
58            pst.setInt(1, id);
59            PlanetDAO planetDAO = new PlanetDAO();
60            ResultSet rs = pst.executeQuery();
61            if (rs.next()) {
62                int idDevuelto = rs.getInt("id");
63                String name = rs.getString("name");
64                int height = rs.getInt("height");
65                double mass = rs.getDouble("mass");
66                String hair_color = rs.getString("hair_color");
67                String skin_color = rs.getString("skin_color");
68                String eye_color = rs.getString("eye_color");
69                String birth_year = rs.getString("birth_year");
70                String gender = rs.getString("gender");
71                Planet planet = planetDAO.obtener(rs.getInt("planet_id")); //buscamos el
72 planeta por su id
73                character = new Character(id, name, height, mass, hair_color, skin_color,
74 eye_color, birth_year, gender, planet);
75            }
76        }
77    }

```

```

68     }
69     return character;
70 }
71
72 @Override
73 public ArrayList<Character> obtenerTodos() throws SQLException {
74     ArrayList<Character> characters = new ArrayList<>();
75     PlanetDAO planetDAO = new PlanetDAO();
76     try (Statement st = con.createStatement();
77         ResultSet rs = st.executeQuery(SELECT_ALL_QUERY)) {
78         while (rs.next()) {
79             int id = rs.getInt("id");
80             String name = rs.getString("name");
81             int height = rs.getInt("height");
82             double mass = rs.getDouble("mass");
83             String hair_color = rs.getString("hair_color");
84             String skin_color = rs.getString("skin_color");
85             String eye_color = rs.getString("eye_color");
86             String birth_year = rs.getString("birth_year");
87             String gender = rs.getString("gender");
88             int planet_id = rs.getInt("planet_id");
89             Planet planet = null;
90             if (!rs.isNull()) {
91                 planet = planetDAO.obtener(planet_id); //buscamos el planeta por su
92                 id
93             }
94             Character character = new Character(id, name, height, mass, hair_color,
95                 skin_color, eye_color, birth_year, gender, planet);
96             characters.add(character);
97         }
98     }
99     return characters;
100 }
101
102 @Override
103 public void actualizar(Character entidad) throws SQLException {
104     try (PreparedStatement pst = con.prepareStatement(UPDATE_QUERY)) {
105         pst.setString(1, entidad.getName());
106         pst.setInt(2, entidad.getHeight());
107         pst.setDouble(3, entidad.getMass());
108         pst.setString(4, entidad.getHair_color());
109         pst.setString(5, entidad.getSkin_color());
110         pst.setString(6, entidad.getEye_color());
111         pst.setString(7, entidad.getBirth_year());
112         pst.setString(8, entidad.getGender());
113         pst.setInt(9, entidad.getPlanet_id().getId());
114         pst.setInt(10, entidad.getId());
115         pst.executeUpdate();
116     }
117 }
118
119 @Override
120 public void eliminar(int id) throws SQLException {
121     try (PreparedStatement pst = con.prepareStatement(DELETE_QUERY)) {
122         pst.setInt(1, id);
123         pst.executeUpdate();
124     }
125 }

```

CharacterFilmDAO

```

1 package es.martinezpenya.ejemplos.UD10.starwars;
2
3 import java.sql.*;
4 import java.util.ArrayList;
5
6 public class CharacterFilmDAO implements DAO<CharacterFilm> {
7     private static final String INSERT_QUERY = "INSERT INTO character_films
8     (id_character, id_film) VALUES (?, ?)";
9     private static final String SELECT_ALL_QUERY_BY_CHARACTER = "SELECT * FROM
10     character_films WHERE id_character = ?";

```



```

9      private static final String SELECT_ALL_QUERY_BY_FILM = "SELECT * FROM character_films
WHERE id_film = ?";
10     private static final String DELETE_QUERY_BY_CHARACTER = "DELETE FROM character_films
WHERE id_character = ?";
11     private static final String DELETE_QUERY_BY_FILM = "DELETE FROM character_films WHERE
id_film = ?";
12     public static final String DELETE_QUERY_BY_CHARACTER_AND_FILM = "DELETE FROM
character_films WHERE id_character = ? AND id_film = ?";
13
14     private Connection con;
15
16     public CharacterFilmDAO() {
17         try {
18             con = DBConnect.getConnection();
19         } catch (SQLException e) {
20             System.out.println("ERROR al conectar: " + e.getMessage());
21         }
22     }
23     @Override
24     public void crear(CharacterFilm entidad) throws SQLException {
25         try (PreparedStatement pst = con.prepareStatement(INSERT_QUERY,
Statement.RETURN_GENERATED_KEYS)) {
26
27             pst.setInt(1, entidad.getCharacter().getId());
28             pst.setInt(2, entidad.getFilm().getId());
29
30             ResultSet rs = pst.executeQuery();
31         }
32     }
33
34     @Override
35     public CharacterFilm obtener(int id) throws SQLException {
36         //No tiene sentido en una tabla N a N
37         return null;
38     }
39
40     @Override
41     public ArrayList<CharacterFilm> obtenerTodos() throws SQLException {
42         //No tiene sentido en una tabla N a N
43         return null;
44     }
45
46     public ArrayList<CharacterFilm> obtenerTodosPorCharacter(int id_character) throws
SQLException {
47         ArrayList<CharacterFilm> characterFilms = new ArrayList<>();
48         try (PreparedStatement pst = con.prepareStatement(SELECT_ALL_QUERY_BY_CHARACTER))
{
49             pst.setInt(1, id_character);
50             ResultSet rs = pst.executeQuery();
51             CharacterDAO characterDAO = new CharacterDAO();
52             Character character = characterDAO.obtener(id_character);
53             FilmDAO filmDAO = new FilmDAO();
54             while (rs.next()) {
55                 int id_film = rs.getInt("id_film");
56                 Film film = filmDAO.obtener(id_film);
57                 characterFilms.add(new CharacterFilm(character, film));
58             }
59         }
60         return characterFilms;
61     }
62
63     public ArrayList<CharacterFilm> obtenerTodosPorFilm(int id_film) throws SQLException
{
64         ArrayList<CharacterFilm> characterFilms = new ArrayList<>();
65         try (PreparedStatement pst = con.prepareStatement(SELECT_ALL_QUERY_BY_FILM)) {
66             pst.setInt(1, id_film);
67             ResultSet rs = pst.executeQuery();
68             CharacterDAO characterDAO = new CharacterDAO();
69             FilmDAO filmDAO = new FilmDAO();
70             Film film = filmDAO.obtener(id_film);
71             while (rs.next()) {
72                 int id_character = rs.getInt("id_character");
73                 Character character = characterDAO.obtener(id_character);
74                 characterFilms.add(new CharacterFilm(character, film));
75             }

```

```

76     }
77     return characterFilms;
78 }
79
80 @Override
81 public void actualizar(CharacterFilm entidad) throws SQLException {
82     //No tiene sentido en una tabla N a N
83 }
84
85 @Override
86 public void eliminar(int id) throws SQLException {
87     //No tiene sentido en una tabla N a N
88 }
89
90 public void eliminarPorCharacterYFilm(int idCharacter, int idFilm) throws
SQLException {
91     try (PreparedStatement pst =
con.prepareStatement(DELETE_QUERY_BY_CHARACTER_AND_FILM)) {
92         pst.setInt(1, idCharacter);
93         pst.setInt(2, idFilm);
94         pst.executeUpdate();
95     }
96 }
97
98 public void eliminarPorCharacter(int id) throws SQLException {
99     try (PreparedStatement pst = con.prepareStatement(DELETE_QUERY_BY_CHARACTER)) {
100         pst.setInt(1, id);
101         pst.executeUpdate();
102     }
103 }
104
105 public void eliminarPorFilm(int id) throws SQLException {
106     try (PreparedStatement pst = con.prepareStatement(DELETE_QUERY_BY_FILM)) {
107         pst.setInt(1, id);
108         pst.executeUpdate();
109     }
110 }
111 }

```

Paso 4: Implementar la lógica de la aplicación

Finalmente, implementaremos la lógica de la aplicación en una clase principal `Main` donde podremos interactuar con los DAOs y la base de datos.

```

1  package es.martinezpenya.ejemplos.UD10.starwars;
2
3  import java.sql.Connection;
4  import java.sql.SQLException;
5  import java.util.ArrayList;
6  import java.util.Scanner;
7
8  public class Main {
9      public static void main(String[] args) {
10         Scanner entrada = new Scanner(System.in);
11         try (Connection con = DBConnect.getConnection()) {
12             while (true) {
13                 menuPrincipal();
14                 int opcion = entrada.nextInt();
15                 entrada.nextLine(); // Limpiar el buffer del entrada
16                 switch (opcion) {
17                     case 1:
18                         gestionarFilms();
19                         break;
20                     case 2:
21                         gestionarCharacters();
22                         break;
23                     case 3:
24                         gestionarCharacterFilms();
25                         break;
26                     case 4:
27                         gestionarPlanetas();
28                         break;
29                     case 0:
30                         System.out.println("Saliendo...");

```

```

31         entrada.close();
32         System.exit(0);
33         default:
34             System.out.println("Opción inválida. Intenta de nuevo.");
35     }
36 }
37 } catch (SQLException e) {
38     System.out.println("Error en la conexión con la base de datos: " +
e.getMessage());
39 }
40 }
41
42 private static void gestionarCharacterFilms() {
43     Scanner entrada = new Scanner(System.in);
44     CharacterFilmDAO characterFilmDAO = new CharacterFilmDAO();
45     while (true) {
46         menuCharacterFilms();
47         int opcion = entrada.nextInt();
48         entrada.nextLine(); // Limpiar el buffer del entrada
49         switch (opcion) {
50             case 1:
51                 System.out.print("Introduce el id del character: ");
52                 int id_character = entrada.nextInt();
53                 entrada.nextLine();
54                 System.out.print("Introduce el id del film: ");
55                 int id_film = entrada.nextInt();
56                 entrada.nextLine();
57                 Character character = null;
58                 Film film = null;
59                 try {
60                     character = new CharacterDAO().obtener(id_character);
61                 } catch (SQLException e) {
62                     System.out.println("Error al obtener el character: " +
e.getMessage());
63                 }
64                 try {
65                     film = new FilmDAO().obtener(id_film);
66                 } catch (SQLException e) {
67                     System.out.println("Error al obtener el film: " +
e.getMessage());
68                 }
69                 if (character == null || film == null) {
70                     System.out.println("No se pudo crear el CharacterFilm porque el
Character o el Film no existen.");
71                 }
72                 else {
73                     CharacterFilm nuevoCharacterFilm = new CharacterFilm(character,
film);
74                     try {
75                         characterFilmDAO.crear(nuevoCharacterFilm);
76                         System.out.println("CharacterFilm creado");
77                     } catch (SQLException e) {
78                         System.out.println("Error al crear characterFilm: " +
e.getMessage());
79                     }
80                 }
81                 break;
82             case 2:
83                 System.out.print("Introduce el ID del Character para ver todos los
Films en los que aparece: ");
84                 int idCharacter = entrada.nextInt();
85                 entrada.nextLine(); // Limpiar el buffer del entrada
86                 try {
87                     ArrayList<CharacterFilm> characterFilms =
characterFilmDAO.obtenerTodosPorCharacter(idCharacter);
88                     if (!characterFilms.isEmpty()) {
89                         System.out.println("Listado de CharacterFilm:");
90                         for (CharacterFilm cf : characterFilms) {
91                             System.out.println(cf);
92                         }
93                     } else {
94                         System.out.println("No hay CharacterFilm registrados.");
95                     }
96                 } catch (SQLException e) {

```

```

97         System.out.println("Error al obtener todos los CharacterFilm del
Character: " + e.getMessage());
98     }
99     break;
100     case 3:
101         System.out.print("Introduce el ID del Film para ver todos los
Characters que aparecen: ");
102         int idFilm = entrada.nextInt();
103         entrada.nextLine(); // Limpiar el buffer del entrada
104         try {
105             ArrayList<CharacterFilm> characterFilms =
characterFilmDAO.obtenerTodosPorFilm(idFilm);
106             if (!characterFilms.isEmpty()) {
107                 System.out.println("Listado de CharacterFilm:");
108                 for (CharacterFilm cf : characterFilms) {
109                     System.out.println(cf);
110                 }
111             } else {
112                 System.out.println("No hay CharacterFilm registrados.");
113             }
114         } catch (SQLException e) {
115             System.out.println("Error al obtener todos los CharacterFilm del
Film: " + e.getMessage());
116         }
117         break;
118     case 4:
119         System.out.print("Introduce el ID del Character para eliminar todos
los CharacterFilm asociados: ");
120         int idCharacterEliminar = entrada.nextInt();
121         entrada.nextLine(); // Limpiar el buffer del entrada
122         try {
123             characterFilmDAO.eliminarPorCharacter(idCharacterEliminar);
124             System.out.println("CharacterFilm eliminados correctamente.");
125         } catch (SQLException e) {
126             System.out.println("Error al eliminar CharacterFilm: " +
e.getMessage());
127         }
128         break;
129     case 5:
130         System.out.print("Introduce el ID del Film para eliminar todos los
CharacterFilm asociados: ");
131         int idFilmEliminar = entrada.nextInt();
132         entrada.nextLine(); // Limpiar el buffer del entrada
133         try {
134             characterFilmDAO.eliminarPorFilm(idFilmEliminar);
135             System.out.println("CharacterFilm eliminados correctamente.");
136         } catch (SQLException e) {
137             System.out.println("Error al eliminar CharacterFilm: " +
e.getMessage());
138         }
139         break;
140     case 6:
141         System.out.print("Introduce el ID del Character del CharacterFilm a
eliminar: ");
142         int idCharacterCharacterFilmEliminar = entrada.nextInt();
143         System.out.print("Introduce el ID del Film del CharacterFilm a
eliminar: ");
144         int idFilmCharacterFilmEliminar = entrada.nextInt();
145         entrada.nextLine(); // Limpiar el buffer del entrada
146         try {
147             characterFilmDAO.eliminarPorCharacterYFilm(idCharacterCharacterFilmEliminar,
idFilmCharacterFilmEliminar);
148             System.out.println("CharacterFilm eliminado correctamente.");
149         } catch (SQLException e) {
150             System.out.println("Error al eliminar CharacterFilm: " +
e.getMessage());
151         }
152         break;
153     }
154 }
155 }
156
157 private static void gestionarFilms() {
158     Scanner entrada = new Scanner(System.in);

```

```

159     FilmDAO filmDAO = new FilmDAO();
160     while (true) {
161         menuFilms();
162         int opcion = entrada.nextInt();
163         entrada.nextLine(); // Limpiar el buffer del entrada
164         switch (opcion) {
165             case 1:
166                 System.out.print("Introduce el id del film: ");
167                 int id = entrada.nextInt();
168                 entrada.nextLine();
169                 System.out.print("Introduce el episode del film: ");
170                 String episode = entrada.nextLine();
171                 System.out.print("Introduce el title del film: ");
172                 String title = entrada.nextLine();
173                 Film nuevoFilm = new Film(id, episode, title); // El ID se genera
automáticamente
174                 try {
175                     filmDAO.crear(nuevoFilm);
176                     System.out.println("Usuario creado con ID: " +
nuevoFilm.getId());
177                 } catch (SQLException e) {
178                     System.out.println("Error al crear usuario: " + e.getMessage());
179                 }
180                 break;
181             case 2:
182                 System.out.print("Introduce el ID del film a consultar: ");
183                 int idFilm = entrada.nextInt();
184                 entrada.nextLine(); // Limpiar el buffer del entrada
185                 try {
186                     Film film = filmDAO.obtener(idFilm);
187                     if (film != null) {
188                         System.out.println("Film encontrado:");
189                         System.out.println(film);
190                     } else {
191                         System.out.println("No se encontró ningún film con ese ID.");
192                     }
193                 } catch (SQLException e) {
194                     System.out.println("Error al obtener el film: " +
e.getMessage());
195                 }
196                 break;
197             case 3:
198                 try {
199                     ArrayList<Film> films = filmDAO.obtenerTodos();
200                     if (!films.isEmpty()) {
201                         System.out.println("Listado de Usuarios:");
202                         for (Film u : films) {
203                             System.out.println(u);
204                         }
205                     } else {
206                         System.out.println("No hay films registrados.");
207                     }
208                 } catch (SQLException e) {
209                     System.out.println("Error al obtener todos los films: " +
e.getMessage());
210                 }
211                 break;
212             case 4:
213                 System.out.print("Introduce el ID del film a actualizar: ");
214                 int idActualizar = entrada.nextInt();
215                 entrada.nextLine(); // Limpiar el buffer del entrada
216                 try {
217                     Film filmActualizar = filmDAO.obtener(idActualizar);
218                     if (filmActualizar != null) {
219                         System.out.print("Introduce el nuevo episode del film (deja
en blanco para mantener el actual): ");
220                         String nuevoEpisode = entrada.nextLine();
221                         if (!nuevoEpisode.isEmpty()) {
222                             filmActualizar.setEpisode(nuevoEpisode);
223                         }
224                         System.out.print("Introduce el nuevo title del film (deja en
blanco para mantener el actual): ");
225                         String nuevoTitle = entrada.nextLine();
226                         if (!nuevoTitle.isEmpty()) {
227                             filmActualizar.setTitle(nuevoTitle);

```

```

228         }
229         filmDAO.actualizar(filmActualizar);
230         System.out.println("Film actualizado correctamente.");
231     } else {
232         System.out.println("No se encontró ningún Film con ese ID.");
233     }
234 } catch (SQLException e) {
235     System.out.println("Error al actualizar Film: " +
e.getMessage());
236     }
237     break;
238     case 5:
239         System.out.print("Introduce el ID del film a eliminar: ");
240         int idEliminar = entrada.nextInt();
241         entrada.nextLine(); // Limpiar el buffer del entrada
242         try {
243             Film filmEliminar = filmDAO.obtener(idEliminar);
244             if (filmEliminar != null) {
245                 // eliminar primero los character_films y deaths asociados al
film
246                 //
Character_filmDAO.eliminarCharacter_filmsPorFilm(idEliminar);
247                 // luego eliminar el film
248                 filmDAO.eliminar(idEliminar);
249                 System.out.println("Film eliminado correctamente.");
250             } else {
251                 System.out.println("No se encontró ningún film con ese ID.");
252             }
253         } catch (SQLException e) {
254             System.out.println("Error al eliminar film: " + e.getMessage());
255         }
256         break;
257     case 0:
258         return;
259     default:
260         System.out.println("Opción inválida. Intenta de nuevo.");
261     }
262 }
263 }
264
265 private static void gestionarPlanetas() {
266     Scanner entrada = new Scanner(System.in);
267     PlanetDAO planetDAO = new PlanetDAO();
268     while (true) {
269         menuPlanets();
270         int opcion = entrada.nextInt();
271         entrada.nextLine(); // Limpiar el buffer del entrada
272         switch (opcion) {
273             case 1:
274                 System.out.print("Introduce el id del planeta: ");
275                 int id = entrada.nextInt();
276                 entrada.nextLine();
277                 System.out.print("Introduce el nombre del planeta: ");
278                 String name = entrada.nextLine();
279                 System.out.print("Introduce el rotation_period del planeta: ");
280                 String rotation_period = entrada.nextLine();
281                 System.out.print("Introduce el orbital_period del planeta: ");
282                 String orbital_period = entrada.nextLine();
283                 System.out.print("Introduce el diameter del planeta: ");
284                 String diameter = entrada.nextLine();
285                 System.out.print("Introduce el climate del planeta: ");
286                 String climate = entrada.nextLine();
287                 System.out.print("Introduce el gravity del planeta: ");
288                 String gravity = entrada.nextLine();
289                 System.out.print("Introduce el terrain del planeta: ");
290                 String terrain = entrada.nextLine();
291                 System.out.print("Introduce el surface_water del planeta: ");
292                 String surface_water = entrada.nextLine();
293                 System.out.print("Introduce el population del planeta: ");
294                 String population = entrada.nextLine();
295                 Planet nuevoPlanet = new Planet(id, name, rotation_period,
orbital_period, diameter, climate, gravity, terrain, surface_water, population); // El ID
se genera automáticamente
296                 try {
297                     planetDAO.crear(nuevoPlanet);

```

```

298         System.out.println("Planeta creado con ID: " +
nuevoPlanet.getId());
299     } catch (SQLException e) {
300         System.out.println("Error al crear planeta: " + e.getMessage());
301     }
302     break;
303     case 2:
304         System.out.print("Introduce el ID del planeta a consultar: ");
305         int idPlanet = entrada.nextInt();
306         entrada.nextLine(); // Limpiar el buffer del entrada
307         try {
308             Planet planet = planetDAO.obtener(idPlanet);
309             if (planet != null) {
310                 System.out.println("Planeta encontrado:");
311                 System.out.println(planet);
312             } else {
313                 System.out.println("No se encontró ningún planeta con ese
ID.");
314             }
315         } catch (SQLException e) {
316             System.out.println("Error al obtener el planeta: " +
e.getMessage());
317         }
318         break;
319     case 3:
320         try {
321             ArrayList<Planet> planets = planetDAO.obtenerTodos();
322             if (!planets.isEmpty()) {
323                 System.out.println("Listado de Planetas:");
324                 for (Planet u : planets) {
325                     System.out.println(u);
326                 }
327             } else {
328                 System.out.println("No hay planetas registrados.");
329             }
330         } catch (SQLException e) {
331             System.out.println("Error al obtener todos los planetas: " +
e.getMessage());
332         }
333         break;
334     case 4:
335         System.out.print("Introduce el ID del planeta a actualizar: ");
336         int idActualizar = entrada.nextInt();
337         entrada.nextLine(); // Limpiar el buffer del entrada
338         try {
339             Planet planetActualizar = planetDAO.obtener(idActualizar);
340             if (planetActualizar != null) {
341                 System.out.print("Introduce el nuevo nombre del planeta (deja
en blanco para mantener el actual): ");
342                 String nuevoName = entrada.nextLine();
343                 if (!nuevoName.isEmpty()) {
344                     planetActualizar.setName(nuevoName);
345                 }
346                 System.out.print("Introduce el nuevo rotation_period del
planeta (deja en blanco para mantener el actual): ");
347                 String nuevoRotation_period = entrada.nextLine();
348                 if (!nuevoRotation_period.isEmpty()) {
349                     planetActualizar.setRotation_period(nuevoRotation_period);
350                 }
351                 System.out.print("Introduce el nuevo orbital_period del
planeta (deja en blanco para mantener el actual): ");
352                 String nuevoOrbital_period = entrada.nextLine();
353                 if (!nuevoOrbital_period.isEmpty()) {
354                     planetActualizar.setOrbital_period(nuevoOrbital_period);
355                 }
356                 System.out.print("Introduce el nuevo diameter del planeta
(deja en blanco para mantener el actual): ");
357                 String nuevoDiameter = entrada.nextLine();
358                 if (!nuevoDiameter.isEmpty()) {
359                     planetActualizar.setDiameter(nuevoDiameter);
360                 }
361                 System.out.print("Introduce el nuevo climate del planeta
(deja en blanco para mantener el actual): ");
362                 String nuevoClimate = entrada.nextLine();

```

```

363         if (!nuevoClimate.isEmpty()) {
364             planetActualizar.setClimate(nuevoClimate);
365         }
366         System.out.print("Introduce el nuevo gravity del planeta
(deja en blanco para mantener el actual): ");
367         String nuevoGravity = entrada.nextLine();
368         if (!nuevoGravity.isEmpty()) {
369             planetActualizar.setGravity(nuevoGravity);
370         }
371         System.out.print("Introduce el nuevo terrain del planeta
(deja en blanco para mantener el actual): ");
372         String nuevoTerrain = entrada.nextLine();
373         if (!nuevoTerrain.isEmpty()) {
374             planetActualizar.setTerrain(nuevoTerrain);
375         }
376         System.out.print("Introduce el nuevo surface_water del
planeta (deja en blanco para mantener el actual): ");
377         String nuevoSurface_water = entrada.nextLine();
378         if (!nuevoSurface_water.isEmpty()) {
379             planetActualizar.setSurface_water(nuevoSurface_water);
380         }
381         System.out.print("Introduce el nuevo population del planeta
(deja en blanco para mantener el actual): ");
382         String nuevoPopulation = entrada.nextLine();
383         if (!nuevoPopulation.isEmpty()) {
384             planetActualizar.setPopulation(nuevoPopulation);
385         }
386         planetDAO.actualizar(planetActualizar);
387         System.out.println("Planeta actualizado correctamente.");
388     } else {
389         System.out.println("No se encontró ningún Planeta con ese
ID.");
390     }
391 } catch (SQLException e) {
392     System.out.println("Error al actualizar Planeta: " +
e.getMessage());
393 }
394 break;
395 case 5:
396     System.out.print("Introduce el ID del planeta a eliminar: ");
397     int idEliminar = entrada.nextInt();
398     entrada.nextLine(); // Limpiar el buffer del entrada
399     try {
400         Planet planetEliminar = planetDAO.obtener(idEliminar);
401         if (planetEliminar != null) {
402             // todo: Revisar si quiero eliminar Characters al eliminar su
Planeta Origen.
403             planetDAO.eliminar(idEliminar);
404             System.out.println("Planeta eliminado correctamente.");
405         } else {
406             System.out.println("No se encontró ningún planeta con ese
ID.");
407         }
408     } catch (SQLException e) {
409         System.out.println("Error al eliminar planeta: " +
e.getMessage());
410     }
411     break;
412 case 0:
413     return;
414 default:
415     System.out.println("Opción inválida. Intenta de nuevo.");
416 }
417 }
418 }
419
420 private static void gestionarCharacters() {
421     Scanner entrada = new Scanner(System.in);
422     CharacterDAO characterDAO = new CharacterDAO();
423     while (true) {
424         menuCharacters();
425         PlanetDAO planetDAO = new PlanetDAO();
426         int opcion = entrada.nextInt();
427         entrada.nextLine(); // Limpiar el buffer del entrada
428         switch (opcion) {

```



```

429         case 1:
430             System.out.print("Introduce el id del character: ");
431             int id = entrada.nextInt();
432             entrada.nextLine();
433             System.out.print("Introduce el name del character: ");
434             String name = entrada.nextLine();
435             System.out.print("Introduce el height del character: ");
436             int height = Integer.parseInt(entrada.nextLine());
437             System.out.print("Introduce el mass del character: ");
438             int mass = Integer.parseInt(entrada.nextLine());
439             System.out.print("Introduce el hair_color del character: ");
440             String hair_color = entrada.nextLine();
441             System.out.print("Introduce el skin_color del character: ");
442             String skin_color = entrada.nextLine();
443             System.out.print("Introduce el eye_color del character: ");
444             String eye_color = entrada.nextLine();
445             System.out.print("Introduce el birth_year del character: ");
446             String birth_year = entrada.nextLine();
447             System.out.print("Introduce el gender del character: ");
448             String gender = entrada.nextLine();
449             System.out.print("Introduce el id del planeta del character: ");
450             int planet_id = Integer.parseInt(entrada.nextLine());
451             Character nuevoCharacter = null;
452             try {
453                 nuevoCharacter = new Character(id, name, height, mass,
skin_color, hair_color, eye_color, birth_year, gender, planetDAO.obtener(planet_id));
454             } catch (SQLException e) {
455                 System.out.println("Error al obtener el planeta: " +
e.getMessage());
456             }
457             try {
458                 characterDAO.crear(nuevoCharacter);
459                 System.out.println("Character creado con ID: " +
nuevoCharacter.getId());
460             } catch (SQLException e) {
461                 System.out.println("Error al crear character: " +
e.getMessage());
462             }
463             break;
464         case 2:
465             System.out.print("Introduce el ID del character a consultar: ");
466             int idCharacter = entrada.nextInt();
467             entrada.nextLine(); // Limpiar el buffer del entrada
468             try {
469                 Character character = characterDAO.obtener(idCharacter);
470                 if (character != null) {
471                     System.out.println("Character encontrado:");
472                     System.out.println(character);
473                 } else {
474                     System.out.println("No se encontró ningún character con ese
ID.");
475                 }
476             } catch (SQLException e) {
477                 System.out.println("Error al obtener el character: " +
e.getMessage());
478             }
479             break;
480         case 3:
481             try {
482                 ArrayList<Character> characters = characterDAO.obtenerTodos();
483                 if (!characters.isEmpty()) {
484                     System.out.println("Listado de Characters:");
485                     for (Character u : characters) {
486                         System.out.println(u);
487                     }
488                 } else {
489                     System.out.println("No hay characters registrados.");
490                 }
491             } catch (SQLException e) {
492                 System.out.println("Error al obtener todos los characters: " +
e.getMessage());
493             }
494             break;
495         case 4:
496             System.out.print("Introduce el ID del character a actualizar: ");

```

```

497         int idActualizar = entrada.nextInt();
498         entrada.nextLine(); // Limpiar el buffer del entrada
499         try {
500             Character characterActualizar =
characterDAO.obtener(idActualizar);
501             if (characterActualizar != null) {
502                 System.out.print("Introduce el nuevo name del character (deja
en blanco para mantener el actual): ");
503                 String nuevoName = entrada.nextLine();
504                 if (!nuevoName.isEmpty()) {
505                     characterActualizar.setName(nuevoName);
506                 }
507                 System.out.print("Introduce el nuevo height del character
(deja en blanco para mantener el actual): ");
508                 String nuevoHeight = entrada.nextLine();
509                 if (!nuevoHeight.isEmpty()) {
510                     characterActualizar.setHeight(Integer.parseInt(nuevoHeight));
511                 }
512                 System.out.print("Introduce el nuevo mass del character (deja
en blanco para mantener el actual): ");
513                 String nuevoMass = entrada.nextLine();
514                 if (!nuevoMass.isEmpty()) {
515                     characterActualizar.setMass(Double.parseDouble(nuevoMass));
516                 }
517                 System.out.print("Introduce el nuevo hair_color del character
(deja en blanco para mantener el actual): ");
518                 String nuevoHair_color = entrada.nextLine();
519                 if (!nuevoHair_color.isEmpty()) {
520                     characterActualizar.setHair_color(nuevoHair_color);
521                 }
522                 System.out.print("Introduce el nuevo skin_color del character
(deja en blanco para mantener el actual): ");
523                 String nuevoSkin_color = entrada.nextLine();
524                 if (!nuevoSkin_color.isEmpty()) {
525                     characterActualizar.setSkin_color(nuevoSkin_color);
526                 }
527                 System.out.print("Introduce el nuevo eye_color del character
(deja en blanco para mantener el actual): ");
528                 String nuevoEye_color = entrada.nextLine();
529                 if (!nuevoEye_color.isEmpty()) {
530                     characterActualizar.setEye_color(nuevoEye_color);
531                 }
532                 System.out.print("Introduce el nuevo birth_year del character
(deja en blanco para mantener el actual): ");
533                 String nuevoBirth_year = entrada.nextLine();
534                 if (!nuevoBirth_year.isEmpty()) {
535                     characterActualizar.setBirth_year(nuevoBirth_year);
536                 }
537                 System.out.print("Introduce el nuevo gender del character
(deja en blanco para mantener el actual): ");
538                 String nuevoGender = entrada.nextLine();
539                 if (!nuevoGender.isEmpty()) {
540                     characterActualizar.setGender(nuevoGender);
541                 }
542                 System.out.print("Introduce el nuevo id del planeta del
character (deja en blanco para mantener el actual): ");
543                 String nuevoPlanet_id = entrada.nextLine();
544                 if (!nuevoPlanet_id.isEmpty()) {
545                     Planet p;
546                     if
((p=planetDAO.obtener(Integer.parseInt(nuevoPlanet_id))!=null) {
547                         characterActualizar.setPlanet_id(p);
548                     }
549                     characterDAO.actualizar(characterActualizar);
550                     System.out.println("Character actualizado correctamente.");
551                 } else {
552                     System.out.println("No se encontró ningún Character con ese
ID.");
553                 }
554             }
555         } catch (SQLException e) {
556             System.out.println("Error al actualizar Character: " +
e.getMessage());

```

```

557         }
558         break;
559     case 5:
560         System.out.print("Introduce el ID del character a eliminar: ");
561         int idEliminar = entrada.nextInt();
562         entrada.nextLine(); // Limpiar el buffer del entrada
563         try {
564             Character characterEliminar = characterDAO.obtener(idEliminar);
565             if (characterEliminar != null) {
566                 characterDAO.eliminar(idEliminar);
567                 System.out.println("Character eliminado correctamente.");
568             } else {
569                 System.out.println("No se encontró ningún character con ese
ID.");
570             }
571         } catch (SQLException e) {
572             System.out.println("Error al eliminar character: " +
e.getMessage());
573         }
574         break;
575     case 0:
576         return;
577     default:
578         System.out.println("Opción inválida. Intenta de nuevo.");
579     }
580 }
581 }
582
583 // MENÚ PRINCIPAL: menuPrincipal()
584 private static void menuPrincipal() {
585     System.out.println("\nMenú Principal:");
586     System.out.println("1. Gestionar Films");
587     System.out.println("2. Gestionar Characters");
588     System.out.println("3. Gestionar Character_films");
589     System.out.println("4. Gestionar Planets");
590     System.out.println("0. Salir");
591     System.out.print("Selecciona una opción: ");
592 }
593
594 // MENÚ SECUNDARIO: menuFilms()
595 private static void menuFilms() {
596     System.out.println("\nMenú de Films:");
597     System.out.println("1. Crear film");
598     System.out.println("2. Consultar film por ID");
599     System.out.println("3. Listar todos los films");
600     System.out.println("4. Actualizar film");
601     System.out.println("5. Eliminar film");
602     System.out.println("0. Volver al menú principal");
603     System.out.print("Selecciona una opción: ");
604 }
605
606 //MENU SECUNDARIO: menuPlanets()
607 private static void menuPlanets() {
608     System.out.println("\nMenú de Planets:");
609     System.out.println("1. Crear planeta");
610     System.out.println("2. Consultar planeta por ID");
611     System.out.println("3. Listar todos los planetas");
612     System.out.println("4. Actualizar planeta");
613     System.out.println("5. Eliminar planeta");
614     System.out.println("0. Volver al menú principal");
615     System.out.print("Selecciona una opción: ");
616 }
617
618 // MENÚ SECUNDARIO: menuCharacters()
619 private static void menuCharacters() {
620     System.out.println("\nMenú de Characters:");
621     System.out.println("1. Crear character");
622     System.out.println("2. Consultar character por ID");
623     System.out.println("3. Listar todos los characters");
624     System.out.println("4. Actualizar character");
625     System.out.println("5. Eliminar character");
626     System.out.println("0. Volver al menú principal");
627     System.out.print("Selecciona una opción: ");
628 }
629

```

```

630 // MENÚ SECUNDARIO: menuCharacterFilms()
631 private static void menuCharacterFilms() {
632     System.out.println("\nMenú de CharacterFilm:");
633     System.out.println("1. Crear characterFilm");
634     System.out.println("2. Consultar characterFilm por Character");
635     System.out.println("3. Consultar characterFilm por Film");
636     System.out.println("4. Eliminar characterFilm por Character");
637     System.out.println("5. Eliminar characterFilm por Film");
638     System.out.println("6. Eliminar characterFilm por Character y Film");
639     System.out.println("0. Volver al menú principal");
640     System.out.print("Selecciona una opción: ");
641 }
642 }

```

Consideraciones

- **HikariCP y SQL Queries:** Hemos utilizado HikariCP para conectarnos y realizar operaciones en la base de datos. Es importante manejar excepciones y cerrar correctamente las conexiones y recursos.
- **Patrón DAO:** Este patrón nos ayuda a mantener un código organizado y aislado, separando la lógica de acceso a datos de la lógica de negocio.
- **Lógica de la aplicación:** En la clase `Main`, hemos implementado un menú interactivo que permite al usuario gestionar *films*, *characters*, *planets* y *character_films* utilizando los métodos proporcionados por los DAO's.
- **Adaptabilidad:** Puedes expandir este ejemplo completando las tablas que faltan, añadiendo más funcionalidades o haciendo ajustes según los requisitos de tu aplicación.
- **Fuera de la norma:** Fíjate en la lógica de la tabla `character_films` (tabla N a N), fíjate que no sigue exactamente el patrón DAO, pero eso no es un problema, la interfaz DAO marca un mínimo, pero no es completa, la podemos expandir con nuevas funcionalidades.
- **Eliminación y actualización en cascada:** En este ejemplo no se han contemplado estos casos, se podrían gestionar directamente con el gestor de las bases de datos, al definir las tablas y su comportamiento, o bien simular esta funcionalidad desde sentencias SQL en la lógica del programa.
- **Autonuméricos:** El ejemplo que se ha desarrollado más arriba no tiene ninguna tabla con un campo autoincremental (como suelen tener algunas claves primarias), pero en el código fuente del método `crear` de la clase `FilmDAO` puedes ver un fragmento comentado sobre como podrias gestionar el echo de que no sepas el identificador del registro que insertas en la BBDD hasta que realmente se haya insertado, como recuperarlo y mostrarlo al usuario.
- **Tablas Relacionadas:** Fíjate en el comportamiento que tiene la clase `Character` respecto del `Planet` (o el resto de relaciones), no solo se muestra el `planet_id` de la tabla `Character`, sino que se recupera también el resto de información del planeta.
- **Localización de la BBDD.** En este caso he apostado por alojar la BBDD en un servidor AWS, pero si lo prefieres puedes hacerlo localmente. Eso sí, asegurate que le das toda la información necesaria al docente para que pueda reproducir tu entorno y poder evaluar que tu proyecto funciona como se espera.
- **Rendimiento:** En según que casos, si las tablas tienen muchísima información pueden provocar salidas muy lentas, o incluso errores de *timeout*.

4. Actividades

Teniendo todo lo anterior en cuenta debes...

1. Crear un nuevo proyecto con un nombre identificativo que incluya tu nombre y/o apellidos. En mi caso podría llamarse "`StarWarsDavid`".
2. Sigue las instrucciones para crear las clases básicas para cada una de las tablas de tu BBDD.
3. Genera la clase de conexión a la BBDD.
4. Define la interfaz para gestionar el patrón `DAO`.
5. Implementa las clases que a su vez implementan la interfaz `DAO` para cada una de las tablas.
6. Genera una clase con el `main` que hará que todo funcione con menús y gestión completa del CRUD de tu BBDD.
7. Añade un archivo `.sql` que permita reconstruir la BBDD desde cero, debe incluir tanto el DDL como la inserción de información en las tablas.

Envía a la tarea de Aules:

1. Una memoria en fichero **pdf** explicando los pasos seguidos (con capturas), explicando la estructura de la BBDD, las tablas que vas a implementar, y toda la información que creas relevante, así como tu opinión personal sobre el proyecto (dificultades, futuras ampliaciones, limitaciones conocidas, etc.)
2. Un fichero **zip** con el proyecto gestionado con `maven` en **IntelliJ**. Asegurate de incluir todo lo necesario para que el profesor pueda reproducir y poner en funcionamiento el proyecto. No olvides el fichero `.sql` (o similar) para poder generar la BBDD de nuevo. Este archivo generalmente lo puedes obtener exportando la BBDD o generando una copia de seguridad en SQL.

5. Fuentes de información

- [Wikipedia](#)
- [Programación \(Grado Superior\) - Juan Carlos Moreno Pérez \(Ed. Ra-ma\)](#)
- Apuntes IES Henri Matisse (Javi García Jimenez?)
- Apuntes AulaCampus
- [Apuntes José Luis Comesaña](#)
- [Apuntes IOC Programació bàsica \(Joan Arnedo Moreno\)](#)
- [Apuntes IOC Programació Orientada a Objectes \(Joan Arnedo Moreno\)](#)
- <https://arturoblasco.github.io/prg>