

```
1  package opo.informatica.modelos;
2
3  import java.io.File;
4  import java.util.Date;
5
6  /**
7   * RedSocial
8   */
9  public class RedSocial {
10
11     private String nombre;
12     private Date fechaLanzamiento;
13     private File logo;
14
15     public RedSocial(){}
16
17     public RedSocial(String nombre, Date fecha){
18         this.nombre = nombre;
19         this.fechaLanzamiento = fecha;
20     }
21
22     public void setNombre(String nom) {
23         this.nombre = nom;
24     }
25
26     public String getNombre() {
27         return this.nombre;
28     }
29
30     public void setFechaLanzamiento(Date fecha) {
31         this.fechaLanzamiento = fecha;
32     }
33
34     public Date getFechaLanzamiento() {
35         return this.fechaLanzamiento;
36     }
37
38     public void setLogo(File logo) {
39         this.logo = logo;
40     }
41
42     public File getLogo() {
43         return this.logo;
44     }
45 }
```

```
1 package opo.informatica.modelos;
2
3 import java.util.Date;
4
5 public class Usuario {
6
7     private int usrId;
8     private String nombre;
9     private String apellido1;
10    private String apellido2;
11    private String pais;
12    private int telefono;
13    private String email;
14    private Date fechaNacimiento;
15
16    public Usuario(){}
17
18    public Usuario(int id, String nombre, String apellido1, String apellido2, String pais, int
    tlf, Date fechaNac, String email){
19        this.usrId = id;
20        this.nombre = nombre;
21        this.apellido1 = apellido1;
22        this.apellido2 = apellido2;
23        this.pais = pais;
24        this.telefono = tlf;
25        this.fechaNacimiento = fechaNac;
26        this.email = email;
27    }
28
29    public void setUsrId(int id){
30        this.usrId = id;
31    }
32
33    public int getUsrId(){
34        return this.usrId;
35    }
36
37    public void setNombre(String nom) {
38        this.nombre = nom;
39    }
40
41    public String getNombre() {
42        return this.nombre;
43    }
44
45    public void setApellido1(String apellido1) {
46        this.apellido1 = apellido1;
47    }
48
49    public String getApellido1(){
50        return this.apellido1;
51    }
52
53    public void setApellido2(String apellido2) {
54        this.apellido2 = apellido2;
55    }
56
57    public String getApellido2() {
58        return this.apellido2;
59    }
60
61    public void setPais(String pais) {
62        this.pais = pais;
63    }
64
65    public String getPais(){
66        return this.pais;
67    }
68
```

```
69     public void setTelefono(int tlf) {
70         this.telefono = tlf;
71     }
72
73     public int getTelefono() {
74         return this.telefono;
75     }
76
77     public void setEmail(String email) {
78         this.email = email;
79     }
80
81     public String getEmail() {
82         return this.email;
83     }
84
85     public void setFechaNacimiento(Date fecha) {
86         this.fechaNacimiento = fecha;
87     }
88
89     public Date getFechaNacimiento() {
90         return this.fechaNacimiento;
91     }
92 }
93
```

```
1 package opo.informatica.modelos;
2
3 /**
4  * https://www.baeldung.com/java-new-custom-exception
5  */
6 public class SeguidorExcepcion extends Exception {
7
8     public SeguidorExcepcion(){
9         super();
10    }
11
12    public SeguidorExcepcion(String mensaje) {
13        super(mensaje);
14    }
15
16    public SeguidorExcepcion(String mensaje, Throwable err){
17        super(mensaje, err);
18    }
19 }
20
```

```
1 package opo.informatica.accesodatos;
2
3 import java.util.List;
4
5 import opo.informatica.modelos.RedSocial;
6 import opo.informatica.modelos.Usuario;
7
8 public interface UsuarioDAO {
9
10     public void alta(Usuario usr);
11     public void baja(Usuario usr);
12     public Usuario buscar(int id);
13     public void modificar(Usuario usr);
14     public boolean sigueA(Usuario usr1, Usuario usr2, RedSocial redSoc);
15     public void empezarASeguir(Usuario usr1, Usuario usr2, RedSocial redSoc);
16     public void dejarDeSeguir(Usuario usr1, Usuario usr2, RedSocial redSoc);
17     public List<Usuario> obtenerSeguidores(Usuario usr, RedSocial redSoc);
18     public List<Usuario> obtenerSeguidosPor(Usuario usr, RedSocial redSoc);
19 }
20
```

```

1 package opo.informatica.accesodatos;
2
3 import java.sql.Connection;
4 import java.sql.DriverManager;
5 import java.sql.PreparedStatement;
6 import java.sql.ResultSet;
7 import java.sql.SQLException;
8 import java.sql.Statement;
9 import java.util.ArrayList;
10 import java.util.Date;
11 import java.util.List;
12
13 import opo.informatica.modelos.RedSocial;
14 import opo.informatica.modelos.SeguidorExcepcion;
15 import opo.informatica.modelos.Usuario;
16
17 public class JDBCUsuarioDAO implements UsuarioDAO {
18
19     private static final String jdbcURL = "jdbc:mysql://localhost:3307/bdRedesSociales";
20     private static final String jdbcUsername = "oposicion";
21     private static final String jdbcPassword = "1234";
22
23     private static final String SQL_Obtener_Seguidores = " SELECT u.* \r\n" + //
24                                                         " FROM Usuario u \r\n" + //
25                                                         " WHERE u.usrId IN ( SELECT
26                                                         s.seguidorId \r\n" + //
27                                                         FROM
28                                                         Seguidores s \r\n" + //
29                                                         WHERE
30                                                         s.seguidoId = ? \r\n" + //
31                                                         AND
32                                                         s.nomRS = ?)";
33
34     private static final String SQL_Esta_Suscrito = " SELECT count(*) \r\n" + //
35                                                         " FROM Suscripcion sus \r\n" + //
36                                                         " WHERE sus.usrId = ? \r\n" + //
37                                                         " AND sus.nomRs = ?";
38
39     private static final String SQL_Obtener_Usuarios = " SELECT * \r\n" +
40                                                         " FROM Usuario ";
41
42     private static final String SQL_Obtener_Redes_Sociales = " SELECT * \r\n" +
43                                                         " FROM RedSocial ";
44
45     @Override
46     public void alta(Usuario usr) {
47         // TODO Auto-generated method stub
48         throw new UnsupportedOperationException("Unimplemented method 'alta'");
49     }
50
51     @Override
52     public void baja(Usuario usr) {
53         // TODO Auto-generated method stub
54         throw new UnsupportedOperationException("Unimplemented method 'baja'");
55     }
56
57     @Override
58     public Usuario buscar(int id) {
59         // TODO Auto-generated method stub
60         throw new UnsupportedOperationException("Unimplemented method 'buscar'");
61     }
62
63     @Override
64     public void modificar(Usuario usr) {
65         // TODO Auto-generated method stub
66         throw new UnsupportedOperationException("Unimplemented method 'modificar'");
67     }
68
69     @Override

```

```

66     public boolean sigueA(Usuario usr1, Usuario usr2, RedSocial redSoc) {
67         // TODO Auto-generated method stub
68         throw new UnsupportedOperationException("Unimplemented method 'sigueA'");
69     }
70
71     @Override
72     public void empezarASeguir(Usuario usr1, Usuario usr2, RedSocial redSoc) {
73         // TODO Auto-generated method stub
74         throw new UnsupportedOperationException("Unimplemented method 'empezarASeguir'");
75     }
76
77     @Override
78     public void dejarDeSeguir(Usuario usr1, Usuario usr2, RedSocial redSoc) {
79         // TODO Auto-generated method stub
80         throw new UnsupportedOperationException("Unimplemented method 'dejarDeSeguir'");
81     }
82
83     /**
84      * Obtener, mediante unconector ODBC (JDBC o similar), una lista de seguidores de un
85      * usuario
86      * dado en una red social dada.
87      * En particular, si un usuario no está suscrito a una redsocial y tratamos de obtener
88      * sus
89      * seguidores en dicha red social, deberá lanzarse la excepción SeguidorExcepcion.
90      */
91     @Override
92     public List<Usuario> obtenerSeguidores(Usuario usr, RedSocial redSoc) {
93         List<Usuario> seguidores = new ArrayList<>();
94
95         // Obtenemos la conexión JDBC
96         // Utilizamos el try-with-resources para evitar cerrar nosotros los recursos en caso
97         // de Excepción
98         try (Connection connection = this.getConnection();
99             // Creamos la sentencia a ejecutar por la BBDD
100             PreparedStatement ps = connection.prepareStatement(SQL_Obtener_Seguidores);
101         ) {
102
103             // En particular, si un usuario no está suscrito a una redsocial y tratamos de
104             // obtener sus
105             // seguidores en dicha red social, deberá lanzarse la excepción SeguidorExcepcion.
106             if (!this.estaSuscrito(usr, redSoc)) {
107                 String msgError = "El usuario "+usr.getNombre()+ " "+usr.getApellido1() + " "
108                     +usr.getApellido2() + " no está suscrito a " + redSoc.getNombre() + ". Por lo
109                     que no puede tener seguidores en esta Red Social.";
110                 throw new SeguidorExcepcion(msgError);
111             }
112
113             // Pasamos los parámetros para filtrar la sentencia a la BBDD
114             ps.setInt(1, usr.getUsrId());
115             ps.setString(2, redSoc.getNombre());
116
117             // Pedimos a la BBDD que ejecute la consulta
118             // y nos devuelva un objeto con el conjunto de resultados
119             ResultSet rs = ps.executeQuery();
120
121             // Procesamos el conjunto de resultados para devolverlos en la
122             // estructura de salida de este método
123             while(rs.next()) {
124                 int id = rs.getInt("usrId");
125                 String nom = rs.getString("nombre");
126                 String apellido1 = rs.getString("apellido1");
127                 String apellido2 = rs.getString("apellido1");
128                 String pais = rs.getString("pais");
129                 int tlf = rs.getInt("tel");
130                 Date fechaNac = new Date(rs.getDate("fechaNac").getTime());
131                 String email = rs.getString("email");
132
133                 seguidores.add(new Usuario(id, nom, apellido1, apellido2, pais, tlf, fechaNac,
134                     email));
135             }
136         }
137     }

```

```

128         }
129
130     } catch (Exception e) {
131         e.printStackTrace();
132     }
133     return seguidores;
134 }
135
136 @Override
137 public List<Usuario> obtenerSeguidosPor(Usuario usr, RedSocial redSoc) {
138     // TODO Auto-generated method stub
139     throw new UnsupportedOperationException("Unimplemented method 'obtenerSeguidosPor'");
140 }
141
142 /**
143  * Obtener un conector ODBC (JDBC o similar)
144  * @return
145  */
146 private Connection getConnection(){
147     Connection connection = null;
148     try {
149         Class.forName("com.mysql.jdbc.Driver");
150         connection = DriverManager.getConnection(jdbcURL,jdbcUsername,jdbcPassword);
151     } catch (SQLException | ClassNotFoundException e) {
152         e.printStackTrace();
153     }
154     return connection;
155 }
156
157 /**
158  * Comprueba si un usuario está suscrito a una redsocial
159  * @param usr
160  * @param redSoc
161  * @return
162  */
163 private boolean estaSuscrito(Usuario usr, RedSocial redSoc) {
164     boolean result = false;
165
166     // Obtenemos la conexión JDBC
167     // Utilizamos el try-with-resources para evitar cerrar nosotros los recursos en caso
    de Excepción
168     try (Connection connection = this.getConnection();
169         // Creamos la sentencia a ejecutar por la BBDD
170         PreparedStatement ps = connection.prepareStatement(SQL_Esta_Suscrito);
171     ) {
172         // Pasamos los parámetros para filtrar la consulta a la BBDD
173         ps.setInt(1, usr.getUsrId());
174         ps.setString(2, redSoc.getNombre());
175
176         // Pedimos a la BBDD que ejecute la consulta
177         // y nos devuelva un objeto con el conjunto de resultados
178         ResultSet rs = ps.executeQuery();
179
180         // Procesamos el conjunto de resultados para devolverlos en la
181         // estructura de salida de este método
182         while(rs.next()){
183             // https://www.tutorialspoint.com/how-to-get-the-row-count-in-jdbc
184             // Al ser un SELECT COUNT(*) pedimos el resultado de la 1ª columna
185             int respuesta = rs.getInt(1);
186             if (respuesta > 0){
187                 result = true;
188             }
189         }
190
191     } catch (Exception e) {
192         e.printStackTrace();
193     }
194     return result;
195 }

```



```

196
197
198 public List<Usuario> obtenerUsuarios(){
199     List<Usuario> usuarios = new ArrayList<>();
200
201     // Obtenemos la conexión JDBC
202     // Utilizamos el try-with-resources para evitar cerrar nosotros los recursos en caso
    de Excepción.
203     try (Connection connection = this.getConnection();
204         // Creamos la sentencia a ejecutar en la BBDD
205         Statement statement = connection.createStatement();
206     ) {
207         // Pedimos a la BBDD que ejecute la consulta que le pasamos en la sentencia
208         // y nos devuelva un objeto con el conjunto de resultados
209         ResultSet rs = statement.executeQuery(SQL_Obtener_Usuarios);
210
211         // Procesamos el conjunto de resultados para devolverlos en la
212         // estructura de salida de este método
213         while(rs.next()){
214             int id = rs.getInt("usrId");
215             String nom = rs.getString("nombre");
216             String apellido1 = rs.getString("apellido1");
217             String apellido2 = rs.getString("apellido1");
218             String pais = rs.getString("pais");
219             int tlf = rs.getInt("tel");
220             Date fechaNac = new Date(rs.getDate("fechaNac").getTime());
221             String email = rs.getString("email");
222
223             usuarios.add(new Usuario(id, nom, apellido1, apellido2, pais, tlf, fechaNac,
                email));
224         }
225
226     } catch (Exception e){
227         e.printStackTrace();
228     }
229
230     return usuarios;
231 }
232
233 public List<RedSocial> obtenerRedesSociales() {
234     List<RedSocial> redes = new ArrayList<>();
235
236     // Obtenemos la conexión JDBC
237     // Utilizamos el try-with-resources para evitar cerrar nosotros los recursos en caso
    de Excepción.
238     try (Connection connection = this.getConnection();
239         // Creamos la sentencia a ejecutar en la BBDD
240         Statement statement = connection.createStatement();
241     ){
242         // Pedimos a la BBDD que ejecute la consulta que le pasamos en la sentencia
243         // y nos devuelva un objeto con el conjunto de resultados
244         ResultSet rs = statement.executeQuery(SQL_Obtener_Red_Sociales);
245
246         // Procesamos el conjunto de resultados para devolverlos en la
247         // estructura de salida de este método
248         while(rs.next()){
249             String nombre = rs.getString("nombre");
250             //String url = rs.getString("url");
251             Date fechaLanzamiento = new Date(rs.getDate("fechaLanzamiento").getTime());
252             // https://www.mysqltutorial.org/mysql-jdbc-blob
253             //File logo = new File
254
255             redes.add(new RedSocial(nombre, fechaLanzamiento));
256         }
257
258     } catch (Exception e) {
259         e.printStackTrace();
260     }
261

```

```
262         return redes;  
263     }  
264 }  
265
```

```
1 package opo.informatica;
2
3 import java.util.List;
4
5 import opo.informatica.accesodatos.JDBCUsuarioDAO;
6 import opo.informatica.modelos.RedSocial;
7 import opo.informatica.modelos.Usuario;
8
9 public class Prueba {
10
11     public static void main(String[] args) {
12         JDBCUsuarioDAO dao = new JDBCUsuarioDAO();
13
14         List<Usuario> usuarios = dao.obtenerUsuarios();
15         List<RedSocial> redeRedSociales = dao.obtenerRedesSociales();
16
17         System.out.println("Mostramos la lista de seguidores para cada usuario y red Social");
18         for (Usuario usuario : usuarios) {
19             int orden = 1;
20             for (RedSocial redSocial : redeRedSociales) {
21                 System.out.println("## ***** ##");
22                 System.out.println("Seguidores de "+usuario.getNombre()+" "+usuario.getApellido1()
23                     +" en "+redSocial.getNombre());
24
25                 List<Usuario> seguidores = dao.obtenerSeguidores(usuario, redSocial);
26                 for (Usuario seguidor : seguidores) {
27                     System.out.println(orden++ +" - " + seguidor.getNombre() + " " + seguidor.
28                         getApellido1() +" - " +seguidor.getPais() );
29                 }
30             }
31         }
32     }
```

```

1  SALIDA:
2
3  PS C:\Proyectos\Java\Servlets\ExamenInformatica2021Ej3> c:: cd
   'c:\Proyectos\Java\Servlets\ExamenInformatica2021Ej3'; & 'C:\Program
   Files\Java\jdk1.8.0_361\bin\java.exe' '-cp'
   'C:\Users\toppe\AppData\Local\Temp\cp_aw6cmul7wod0w16r9icy51fx0.jar' 'opo.informatica.Prueba'
4  Loading class `com.mysql.jdbc.Driver'. This is deprecated. The new driver class is
   `com.mysql.cj.jdbc.Driver'. The driver is automatically registered via the SPI and manual
   loading of the driver class is generally unnecessary.
5  Mostramos la lista de seguidores para cada usuario y red Social
6  ## ***** ##
7  Seguidores de Alberto MartÃ-nez en facebook
8  1 - JosÃ MarÃ-a ViÃtuellas - esp
9  2 - Cristina Lorente - esp
10 ## ***** ##
11 Seguidores de Alberto MartÃ-nez en instagram
12 opo.informatica.modelos.SeguidorExcepcion: El usuario Alberto MartÃ-nez MartÃ-nez no estÃ
   suscrito a instagram. Por lo que no puede tener seguidores en esta Red Social.
13     at opo.informatica.accesodatos.JDBCUsuarioDAO.obtenerSeguidores(JDBCUsuarioDAO.java:
   104)
14     at opo.informatica.Prueba.main(Prueba.java:24)
15     Suppressed: java.lang.IllegalArgumentException: Self-suppression not permitted
16         at java.lang.Throwable.addSuppressed(Throwable.java:1082)
17         at opo.informatica.accesodatos.JDBCUsuarioDAO.obtenerSeguidores(JDBCUsuarioDAO
   .java:130)
18         ... 1 more
19     [CIRCULAR REFERENCE:opo.informatica.modelos.SeguidorExcepcion: El usuario Alberto Mart
   Ã-nez MartÃ-nez no estÃ suscrito a instagram. Por lo que no puede tener seguidores en
   esta Red Social.]
20 ## ***** ##
21 Seguidores de Alberto MartÃ-nez en linkedIn
22 opo.informatica.modelos.SeguidorExcepcion: El usuario Alberto MartÃ-nez MartÃ-nez no estÃ
   suscrito a linkedIn. Por lo que no puede tener seguidores en esta Red Social.
23     at opo.informatica.accesodatos.JDBCUsuarioDAO.obtenerSeguidores(JDBCUsuarioDAO.java:
   104)
24     at opo.informatica.Prueba.main(Prueba.java:24)
25     Suppressed: java.lang.IllegalArgumentException: Self-suppression not permitted
26         at java.lang.Throwable.addSuppressed(Throwable.java:1082)
27         at opo.informatica.accesodatos.JDBCUsuarioDAO.obtenerSeguidores(JDBCUsuarioDAO
   .java:130)
28         ... 1 more
29     [CIRCULAR REFERENCE:opo.informatica.modelos.SeguidorExcepcion: El usuario Alberto Mart
   Ã-nez MartÃ-nez no estÃ suscrito a linkedIn. Por lo que no puede tener seguidores en
   esta Red Social.]
30 ## ***** ##
31 Seguidores de MarÃ-a Ãngeles Robledo en facebook
32 opo.informatica.modelos.SeguidorExcepcion: El usuario MarÃ-a Ãngeles Robledo Robledo no estÃ
   suscrito a facebook. Por lo que no puede tener seguidores en esta Red Social.
33     at opo.informatica.accesodatos.JDBCUsuarioDAO.obtenerSeguidores(JDBCUsuarioDAO.java:
   104)
34     at opo.informatica.Prueba.main(Prueba.java:24)
35     Suppressed: java.lang.IllegalArgumentException: Self-suppression not permitted
36         at java.lang.Throwable.addSuppressed(Throwable.java:1082)
37         at opo.informatica.accesodatos.JDBCUsuarioDAO.obtenerSeguidores(JDBCUsuarioDAO
   .java:130)
38         ... 1 more
39     [CIRCULAR REFERENCE:opo.informatica.modelos.SeguidorExcepcion: El usuario MarÃ-a Ã
   ngeles Robledo Robledo no estÃ suscrito a facebook. Por lo que no puede tener
   seguidores en esta Red Social.]
40 ## ***** ##
41 Seguidores de MarÃ-a Ãngeles Robledo en instagram
42 1 - Cristina Lorente - esp
43 ## ***** ##
44 Seguidores de MarÃ-a Ãngeles Robledo en linkedIn
45 opo.informatica.modelos.SeguidorExcepcion: El usuario MarÃ-a Ãngeles Robledo Robledo no estÃ
   suscrito a linkedIn. Por lo que no puede tener seguidores en esta Red Social.
46     at opo.informatica.accesodatos.JDBCUsuarioDAO.obtenerSeguidores(JDBCUsuarioDAO.java:
   104)
47     at opo.informatica.Prueba.main(Prueba.java:24)

```

```

48     Suppressed: java.lang.IllegalArgumentException: Self-suppression not permitted
49         at java.lang.Throwable.addSuppressed(Throwable.java:1082)
50         at opo.informatica.accesodatos.JDBCUsuarioDAO.obtenerSeguidores(JDBCUsuarioDAO
        .java:130)
51         ... 1 more
52     [CIRCULAR REFERENCE:opo.informatica.modelos.SeguidorExcepcion: El usuario MarÃ-a Ã-
ngeles Robledo Robledo no estÃ suscrito a linkedIn. Por lo que no puede tener
seguidores en esta Red Social.]
53     ## ***** ##
54     Seguidores de JosÃ MarÃ-a ViÃ±uelas en facebook
55     1 - Alberto MartÃ-nez - esp
56     ## ***** ##
57     Seguidores de JosÃ MarÃ-a ViÃ±uelas en instagram
58     opo.informatica.modelos.SeguidorExcepcion: El usuario JosÃ MarÃ-a ViÃ±uelas ViÃ±uelas no est
Ã suscrito a instagram. Por lo que no puede tener seguidores en esta Red Social.
59         at opo.informatica.accesodatos.JDBCUsuarioDAO.obtenerSeguidores(JDBCUsuarioDAO.java:
104)
60         at opo.informatica.Prueba.main(Prueba.java:24)
61     Suppressed: java.lang.IllegalArgumentException: Self-suppression not permitted
62         at java.lang.Throwable.addSuppressed(Throwable.java:1082)
63         at opo.informatica.accesodatos.JDBCUsuarioDAO.obtenerSeguidores(JDBCUsuarioDAO
        .java:130)
64         ... 1 more
65     [CIRCULAR REFERENCE:opo.informatica.modelos.SeguidorExcepcion: El usuario JosÃ MarÃ-a
ViÃ±uelas ViÃ±uelas no estÃ suscrito a instagram. Por lo que no puede tener
seguidores en esta Red Social.]
66     ## ***** ##
67     Seguidores de JosÃ MarÃ-a ViÃ±uelas en linkedIn
68     opo.informatica.modelos.SeguidorExcepcion: El usuario JosÃ MarÃ-a ViÃ±uelas ViÃ±uelas no est
Ã suscrito a linkedIn. Por lo que no puede tener seguidores en esta Red Social.
69         at opo.informatica.accesodatos.JDBCUsuarioDAO.obtenerSeguidores(JDBCUsuarioDAO.java:
104)
70         at opo.informatica.Prueba.main(Prueba.java:24)
71     Suppressed: java.lang.IllegalArgumentException: Self-suppression not permitted
72         at java.lang.Throwable.addSuppressed(Throwable.java:1082)
73         at opo.informatica.accesodatos.JDBCUsuarioDAO.obtenerSeguidores(JDBCUsuarioDAO
        .java:130)
74         ... 1 more
75     [CIRCULAR REFERENCE:opo.informatica.modelos.SeguidorExcepcion: El usuario JosÃ MarÃ-a
ViÃ±uelas ViÃ±uelas no estÃ suscrito a linkedIn. Por lo que no puede tener seguidores
en esta Red Social.]
76     ## ***** ##
77     Seguidores de Cristina Lorente en facebook
78     ## ***** ##
79     Seguidores de Cristina Lorente en instagram
80     1 - MarÃ-a Ã-geles Robledo - esp
81     ## ***** ##
82     Seguidores de Cristina Lorente en linkedIn
83     opo.informatica.modelos.SeguidorExcepcion: El usuario Cristina Lorente Lorente no estÃ
suscrito a linkedIn. Por lo que no puede tener seguidores en esta Red Social.
84         at opo.informatica.accesodatos.JDBCUsuarioDAO.obtenerSeguidores(JDBCUsuarioDAO.java:
104)
85         at opo.informatica.Prueba.main(Prueba.java:24)
86     Suppressed: java.lang.IllegalArgumentException: Self-suppression not permitted
87         at java.lang.Throwable.addSuppressed(Throwable.java:1082)
88         at opo.informatica.accesodatos.JDBCUsuarioDAO.obtenerSeguidores(JDBCUsuarioDAO
        .java:130)
89         ... 1 more
90     [CIRCULAR REFERENCE:opo.informatica.modelos.SeguidorExcepcion: El usuario Cristina
Lorente Lorente no estÃ suscrito a linkedIn. Por lo que no puede tener seguidores en
esta Red Social.]
91     PS C:\Proyectos\Java\Servlets\ExamenInformatica2021Ej3>

```

```
1  package net.javaguides.usermanagement.model;
2
3  public class User {
4      protected int id;
5      protected String name;
6      protected String email;
7      protected String country;
8
9      public User() {
10     }
11
12     public User(String name, String email, String country) {
13         this.name = name;
14         this.email = email;
15         this.country = country;
16     }
17
18     public User(int id, String name, String email, String country) {
19         this.id = id;
20         this.name = name;
21         this.email = email;
22         this.country = country;
23     }
24
25     public int getId() {
26         return id;
27     }
28
29     public void setId(int id) {
30         this.id = id;
31     }
32
33     public String getName() {
34         return name;
35     }
36
37     public void setName(String name) {
38         this.name = name;
39     }
40
41     public String getEmail() {
42         return email;
43     }
44
45     public void setEmail(String email) {
46         this.email = email;
47     }
48
49     public String getCountry() {
50         return country;
51     }
52
53     public void setCountry(String country) {
54         this.country = country;
55     }
56
57 }
```

```

1  package net.javaguides.usermanagement.dao;
2
3  import java.sql.Connection;
4  import java.sql.DriverManager;
5  import java.sql.PreparedStatement;
6  import java.sql.ResultSet;
7  import java.sql.SQLException;
8  import java.util.ArrayList;
9  import java.util.List;
10
11  import net.javaguides.usermanagement.model.User;
12
13  public class UserDao {
14      private static final String jdbcURL = "jdbc:mysql://localhost:3307/demo";
15      private static final String jdbcUsername = "javaguides";
16      private static final String jdbcPassword = "1234";
17
18      private static final String INSERT_USERS_SQL = "INSERT INTO usuarios (name, email,
19      country) "
20      + "VALUES (?, ?, ?);";
21      private static final String SELECT_USER_BY_ID = "SELECT * FROM usuarios WHERE id = ?";
22      private static final String SELECT_ALL_USERS = "SELECT * FROM usuarios";
23      private static final String DELETE_USERS_SQL = "DELETE FROM usuarios WHERE id = ?";
24      private static final String UPDATE_USERS_SQL = "UPDATE usuarios SET name = ?, email = ?,
25      country = ? WHERE id = ?";
26
27      public UserDao() {}
28
29      protected Connection getConnection() {
30          Connection connection = null;
31          try {
32              Class.forName("com.mysql.jdbc.Driver");
33              connection = DriverManager.getConnection(jdbcURL, jdbcUsername, jdbcPassword);
34          } catch (SQLException | ClassNotFoundException e) {
35              e.printStackTrace();
36          }
37          return connection;
38      }
39
40      public void insertUser(User user) throws SQLException {
41          System.out.println(INSERT_USERS_SQL);
42          // try-with-resource statement will auto close the connection.
43          try (Connection connection = getConnection(); PreparedStatement preparedStatement =
44          connection.prepareStatement(INSERT_USERS_SQL)) {
45              preparedStatement.setString(1, user.getName());
46              preparedStatement.setString(2, user.getEmail());
47              preparedStatement.setString(3, user.getCountry());
48              System.out.println(preparedStatement);
49              preparedStatement.executeUpdate();
50          } catch (SQLException e) {
51              printSQLException(e);
52          }
53      }
54
55      public User selectUser(int id) {
56          User user = null;
57          // Step 1: Establishing a Connection
58          try (Connection connection = getConnection();
59          PreparedStatement preparedStatement = connection.prepareStatement(
60          SELECT_USER_BY_ID)) {
61              // Step 2: Create a statement using connection object
62              preparedStatement.setInt(1, id);
63              System.out.println(preparedStatement);
64              // Step 3: Execute the query or update query
65              ResultSet rs = preparedStatement.executeQuery();
66
67              // Step 4: Process the ResultSet object.

```

```

66         while (rs.next()) {
67             String name = rs.getString("name");
68             String email = rs.getString("email");
69             String country = rs.getString("country");
70             user = new User(id, name, email, country);
71         }
72     } catch (SQLException e) {
73         printSQLException(e);
74     }
75
76     return user;
77 }
78
79 public List<User> selectAllUsers() {
80
81     // using try-with-resources to avoid closing resources (boiler plate code)
82     List<User> users = new ArrayList<>();
83     // Step 1: Establishing a Connection
84     try (Connection connection = getConnection();
85         // Step 2: Create a statement using connection object
86         PreparedStatement preparedStatement = connection.prepareStatement(
87             SELECT_ALL_USERS)
88     ) {
89         System.out.println(preparedStatement);
90         // Step 3: Execute the query or update query
91         ResultSet rs = preparedStatement.executeQuery();
92
93         // Step 4: Process the ResultSet object.
94         while (rs.next()) {
95             int id = rs.getInt("id");
96             String name = rs.getString("name");
97             String email = rs.getString("email");
98             String country = rs.getString("country");
99             users.add(new User(id, name, email, country));
100         }
101     } catch (SQLException e) {
102         printSQLException(e);
103     }
104
105     return users;
106 }
107
108 public boolean deleteUser(int id) throws SQLException {
109     boolean rowDeleted = false;
110     try (Connection connection = getConnection(); PreparedStatement statement = connection
111         .prepareStatement(DELETE_USERS_SQL)) {
112         statement.setInt(1, id);
113         rowDeleted = statement.executeUpdate() > 0;
114     } catch (SQLException e) {
115         printSQLException(e);
116     }
117
118     return rowDeleted;
119 }
120
121 public boolean updateUser(User user) throws SQLException {
122     boolean rowUpdated = false;
123     try (Connection connection = getConnection(); PreparedStatement statement = connection
124         .prepareStatement(UPDATE_USERS_SQL)) {
125         statement.setString(1, user.getName());
126         statement.setString(2, user.getEmail());
127         statement.setString(3, user.getCountry());
128         statement.setInt(4, user.getId());
129
130         rowUpdated = statement.executeUpdate() > 0;
131     } catch (SQLException e) {
132         printSQLException(e);
133     }

```



```
132
133     return rowUpdated;
134 }
135
136 private void printSQLException(SQLException ex) {
137     for (Throwable e: ex) {
138         if (e instanceof SQLException) {
139             e.printStackTrace(System.err);
140             System.err.println("SQLState: " + ((SQLException)e).getSQLState());
141             System.err.println("Error Code: " + ((SQLException)e).getErrorCode());
142             System.err.println("Message: " + e.getMessage());
143             Throwable t = ex.getCause();
144             while (t != null) {
145                 System.out.println("Cause: " + t);
146                 t = t.getCause();
147             }
148         }
149     }
150 }
151
152
```

```

1  package net.javaguides.usermanagement.web;
2
3  import java.io.IOException;
4  import java.sql.SQLException;
5  import java.util.List;
6
7  import javax.servlet.RequestDispatcher;
8  import javax.servlet.ServletException;
9  import javax.servlet.annotation.WebServlet;
10 import javax.servlet.http.HttpServlet;
11 import javax.servlet.http.HttpServletRequest;
12 import javax.servlet.http.HttpServletResponse;
13
14 import net.javaguides.usermanagement.dao.UserDAO;
15 import net.javaguides.usermanagement.model.User;
16
17 @WebServlet("/")
18 public class UserService extends HttpServlet {
19     private static final long serialVersionUID = 1L;
20     private UserDAO userDAO;
21
22     public void init() {
23         userDAO = new UserDAO();
24     }
25
26     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
27         doGet(request, response);
28     }
29
30     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
31         String action = request.getServletPath();
32
33         try {
34             switch (action) {
35                 case "/new":
36                     showNewForm(request, response);
37                     break;
38                 case "/insert":
39                     insertUser(request, response);
40                     break;
41                 case "/delete":
42                     deleteUser(request, response);
43                     break;
44                 case "/edit":
45                     showEditForm(request, response);
46                     break;
47                 case "/update":
48                     updateUser(request, response);
49                     break;
50                 default:
51                     listUser(request, response);
52                     break;
53             }
54         } catch (SQLException ex) {
55             throw new ServletException(ex);
56         }
57     }
58
59     private void listUser(HttpServletRequest request, HttpServletResponse response) throws
SQLException, ServletException, IOException {
60         List<User> listUser = userDAO.selectAllUsers();
61         request.setAttribute("listUser", listUser);
62         RequestDispatcher dispatcher = request.getRequestDispatcher("user-list.jsp");
63         dispatcher.forward(request, response);
64     }
65
66     private void showNewForm(HttpServletRequest request, HttpServletResponse response) throws

```

```

ServletException, IOException {
67     RequestDispatcher dispatcher = request.getRequestDispatcher("user-form.jsp");
68     dispatcher.forward(request, response);
69 }
70
71 private void showEditForm(HttpServletRequest request, HttpServletResponse response) throws
SQLException, ServletException, IOException {
72     int id = Integer.parseInt(request.getParameter("id"));
73     User existingUser = userDao.selectUser(id);
74     RequestDispatcher dispatcher = request.getRequestDispatcher("user-form.jsp");
75     request.setAttribute("user", existingUser);
76     dispatcher.forward(request, response);
77 }
78
79 private void insertUser(HttpServletRequest request, HttpServletResponse response) throws
SQLException, IOException {
80     String name = request.getParameter("name");
81     String email = request.getParameter("email");
82     String country = request.getParameter("country");
83     User newUser = new User(name, email, country);
84     userDao.insertUser(newUser);
85     response.sendRedirect("list");
86 }
87
88 private void updateUser(HttpServletRequest request, HttpServletResponse response) throws
SQLException, IOException {
89     int id = Integer.parseInt(request.getParameter("id"));
90     String name = request.getParameter("name");
91     String email = request.getParameter("email");
92     String country = request.getParameter("country");
93
94     User user = new User(id, name, email, country);
95     userDao.updateUser(user);
96     response.sendRedirect("list");
97 }
98
99 private void deleteUser(HttpServletRequest request, HttpServletResponse response) throws
SQLException, IOException {
100     int id = Integer.parseInt(request.getParameter("id"));
101     userDao.deleteUser(id);
102     response.sendRedirect("list");
103 }
104 }
105

```