Connexió amb bases de dades relacionals en Java

Persistència i serialització

La **persistència** és el mecanisme mitjançant el qual s'aconsegueix desar dades perquè no es perdin en finalitzar l'aplicació i posteriorment recuperar-les.

Si l'aplicació gestiona petites quantitats de dades, pot ser suficient emmagatzemar-les en un fitxer de text en disc. Si, en canvi, és necessari manejar grans quantitats de dades, l'opció serà emmagatzemar-les en una base de dades.

Podem convertir un objecte en bytes per poder enviar-lo a través de xarxa, desar-lo en un arxiu, reconstruir-lo a l'altra banda d'una xarxa, llegir-lo de fitxer, etc. Aquest procés de conversió a bytes s'anomena **serialització**. Per fer que un objecte es pugui serialitzar en Java només cal que implementi la interface **Serializable**, la qual no té mètodes. Els atributs de les classes que implementin Serializable han de ser dades primitives o bé classes que també implementin la interface.

Si necessitem que la serialització es faci d'una manera especial, hem de definir els mètodes que java cridarà per convertir l'objecte a bytes i per recuperar-lo:

private void **readObject**(java.io.ObjectInputStream stream) throws IOException, ClassNotFoundException
private void **writeObject**(java.io.ObjectOutputStream stream) throws IOException

Quan enviem objectes a través d'una xarxa, és possible que les versions de l'arxiu .class corresponents siguin diferents, impossibilitant la recuperació. Java pot detectar si les versions són iguals o no si la classe té un atribut privat de versió definit de la següent manera:

```
private static final long serialVersionUID = 6678793847896547166L;
```

El mecanisme de persistència que utilitzarem en aquest capítol és la base de dades relacional. Una **base de dades relacional** està formada per un conjunt de **taules** o **entitats**. Cada instància d'una entitat constitueix una **fila** o **registre** de la taula. Els atributs de l'entitat són les **columnes** o **camps** de la taula.

Les taules s'associen mitjançant **relacions** entre camps. Cada taula pot tenir una **clau principal** o **clau primària**, que identifica de forma unívoca cada registre de la taula, i que està formada per un camp o un grup de camps. Alguns camps poden ser **clau forana**, que fa referència a un camp d'una altra taula i serveix per relacionar registres de dues taules.

Drivers i API de les bases de dades

Java proporciona la biblioteca **JDBC** amb interfície (**driver**) diferent per a cada base de dades.

Tots els drivers JDBC tenen en comú les següents classes:

DriverManager	Carrega els drivers i gestiona les connexions
Driver	Tradueix les invocacions de l'API a les de la base de dades concreta
Connection	Encapsula una sessió entre l'aplicació i la base de dades
Statement	Classe per a sentència SQL

Metadata	Informació sobre la base de dades
ResultSet	Files i columnes resultat d'una consulta

La versió de l'API JDBC que descriurem aquí és la 2.0.

El **procediment** per treballar amb bases de dades consisteix en les següents operacions:

1. Carregar el driver de la base de dades i registrar-lo al **DriverManager**
2. Obtenir una connexió (classe **Connection**) amb el mètode **getConnection()** de DriverManager
3. Crear una sentència (classes **Statement**, **PreparedStatement**, **CallableStatement**)
4. Executar la sentència i, si escau, obtenir els resultats (classe **ResultSet**, entre altres)

Connexió amb base de dades ODBC

Anem a programar la connexió i realització d'una consulta amb una base de dades Access a través d'un pont JDBC-ODBC. El pont és molt fàcil d'instal·lar, però requereix configurar la màquina client i té limitacions (per exemple, amb els applets).

La base de dades conté una única taula paisos. La creació de la taula es fa amb la sentència SQL següent:

```
CREATE TABLE `paisos` (
   `id` int(4) NOT NULL auto_increment,
   `nom` varchar(40) collate latin1_spanish_ci default NULL,
   `capital` varchar(40) collate latin1_spanish_ci default NULL,
   `superficie_km2` double default NULL,
   `poblacio` double default NULL,
   `pib_dolars` double default NULL,
   `esp_vida` double default NULL,
   PRIMARY KEY  (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_spanish_ci
AUTO_INCREMENT=41 ;
```

Les dades de la taula s'obtenen amb les sentències d'inserció següents:

```
INSERT INTO `paisos` (`id`, `nom`, `capital`, `superficie_km2`, `poblacio`,
`pib_dolars`, `esp_vida`) VALUES
(1, 'Albania', 'Tirana', 28748, 3100000, 3189, 73),
(2, 'Alemania Federal', 'BerlÃn', 357021, 82000000, 23742, 77),
(3, 'Andorra', 'AndorraLaVieja', 453, 75000, 17420, 78),
(4, 'Austria', 'Viena', 83859, 8100000, 25089, 77),
(5, 'Belarús', 'Minsk', 207600, 10100000, 6876, 68),
(6, 'Bélgica', 'Bruselas', 30528, 10300000, 25443, 78),
(7, 'BosniayHerzegovina', 'Sarajevo', 51129, 4100000, 1086, 56),
(8, 'Bulgaria', 'Sofia', 110994, 7900000, 5071, 70),
(9, 'Croacia', 'Zagreb', 56610, 4700000, 7387, 73),
(10, 'Dinamarca', 'Copenhague', 44493, 5300000, 25869, 76),
(11, 'Eslovaquia', 'Bratislava', 49035, 5400000, 10591, 73),
(12, 'Eslovenia', 'Lubiana', 20258, 2000000, 15977, 75),
(13, 'España', 'Madrid', 505989, 39900000, 18079, 78),
(14, 'Estonia', 'Tallin', 45227, 1400000, 8355, 70),
(15, 'FederacióndeRusia', 'Moscú', 17075400, 144700000, 7473, 66),
(16, 'Finlandia', 'Helsinki', 338145, 5200000, 23, 77),
(17, 'Francia', 'París', 543965, 59500000, 22897, 78),
(18, 'Grecia', 'Atenas', 131957, 10600000, 15414, 78),
(19, 'HungrÃa', 'Budapest', 93030, 9900000, 11430, 71),
(20, 'Irlanda', 'Dublin', 70285, 3800000, 25918, 76),
```

```
(21, 'Islandia', 'Reykjavik', 102819, 281000, 27835, 79),
(22, 'Italia', 'Roma', 301308, 57500000, 22172, 78),
(23, 'Letonia', 'Riga', 64610, 2400000, 6264, 70),
(24, 'Liechtenstein', 'Vaduz', 160, 32015, 37000, 72),
(25, 'Lituania', 'Vilnius', 65300, 3700000, 6656, 71),
(26, 'Luxemburgo', 'Luxemburgo', 2586, 442000, 42769, 77),
(27, 'Macedonia', 'Skopje', 25713, 2000000, 4651, 73),
(28, 'Malta', 'La Valletta', 316, 392000, 15189, 77),
(29, 'Moldova, Rep. de', 'Kisinev', 33700, 4300000, 2037, 66),
(30, 'Mónaco', 'Mónaco', 2, 33000, 26470, 78),
(31, 'Noruega', 'Oslo', 323758, 4500000, 28433, 78),
(32, 'Países Bajos', 'Amsterdam', 41526, 15900000, 24215, 78),
(33, 'Polonia', 'Varsovia', 312685, 38600000, 8450, 73),
(34, 'Portugal', 'Lisboa', 91831, 10000000, 16064, 75),
(35, 'Reino Unido de Gran Bretaña', 'Londres', 244110, 59500000, 22093, 77),
(36, 'República Checa', 'Praga', 78866, 10300000, 13018, 74),
(37, 'Rumania', 'Bucarest', 238391, 22400000, 6041, 69),
(38, 'Suecia', 'Estocolmo', 449964, 8800000, 22636, 79),
(39, 'Suiza', 'Berna', 41285, 7200000, 27171, 78),
(40, 'Ucrania', 'Kiev', 603700, 49100000, 3458, 68);
```

Cal establir l'origen de dades al sistema operatiu Windows, mitjançant *Eines administratives > Orígens de dades (ODBC)* , al tauler de control. Cal afegir al separador *DNS de sistema* la base de dades Access, de nom paisos.mdb.

El codi de l'aplicació és el següent:

```
/**
 * MostraPaisosOdbc.java
 * Exemple de connexió amb base de dades de paisos
 * utilitzant el driver ODBC
 * @author Jose Moreno
 * @version
*/
import java.sql.*;
public class MostraPaisosOdbcAccess
{
   public static void main(String[] args) throws Exception
   {
      final String DRIVER = "sun.jdbc.odbc.JdbcOdbcDriver";
      final String BD_URL = "jdbc:odbc:Paisos";
      final String USUARI = "";
      final String PASSWORD = "";
      Connection conn = null;
      try
      {
         //carregar el driver
         Class.forName(DRIVER);
         //obtenir una connexió amb la base de dades
         conn = DriverManager.getConnection(BD_URL, USUARI, PASSWORD);
         if (conn != null)
         {
            //crear un Statement per a les consultes sql
            Statement stmt = conn.createStatement();
            //executar la consulta i obtenir el resultat en un ResulSet
            ResultSet res = stmt.executeQuery("SELECT * FROM Paisos");
            //escriure encapçalament dels resultats
            System.out.println(
               "id \t nom \t capital \t superficie_km2 \t poblacio \t pib_dolars \t
esp_vida");
```

```
            //recórrer el ResultSet i escriure el contingut
            while(res.next())
            {
               //convertir dades a format de dades de java
               int id = res.getInt("id");
               String nom = res.getString("nom");
               String capital = res.getString("capital");
               double superficie_km2 = res.getDouble("superficie_km2");
               double poblacio = res.getDouble("poblacio");
               double pib_dolars = res.getDouble("pib_dolars");
               double esp_vida = res.getDouble("esp_vida");
               //escriure dades
               System.out.println(
                  id+" \t"+nom +" \t "+capital+" \t "+superficie_km2
                  +" \t "+poblacio+" \t "+pib_dolars+" \t "+esp_vida);
            }
            res.close();
            stmt.close();
            conn.close();
         }
      }
      catch(SQLException ex)
      {
         System.out.println(ex);
      }
      catch(ClassNotFoundException ex)
      {
         System.out.println(ex);
      }
   }
}
```

Amb **Class.forName()** carreguem el driver sun.jdbc.odbc.JdbcOdbcDriver. A partir d'aquest moment, la classe **DriverManager** podrà utilitzar-lo per a la connexió amb bases de dades que tinguin aquest driver. Es llança **ClassNotFoundException** si no es troba la classe del driver.

Utilitzem el mètode **getConnection()** de la classe DriverManager per obtenir un objecte de tipus **Connection**. La classe Connection encapsula la connexió amb la base de dades. Els paràmetres que accepta depenen del tipus de base de dades. En aquest cas, li passem la URL de la base de dades (jdbc:odbc:Paisos) així com l'usuari i la clau o contrasenya de connexió. El mètode getConnection() llança **SQLException** si es produeix algun error en l'accés a la base de dades.

Les sentències SQL es representen amb la classe **Statement**. Obtenim un Statement a partir d'un objecte Connection utilitzant el mètode **createStatement()**. En cas d'error es llança SQLException.

El mètode createStatement() està sobrecarregat per admetre paràmetres, els quals permeten modificar el comportament del ResultSet que es retorna en executar la sentència SQL. Veurem les diferents possibilitats més endavant.

La classe Statement disposa de diversos mètodes per executar les sentències SQL:

ResultSet executeQuery(String sql)	Consultes que retornen un ResultSet (Select)
int executeUpdate(String sql)	Consultes que no retornen un ResulSet (Insert, Update, Delete)

boolean execute(String sql)	Consultes que poden retornar múltiples resultats

En el nostre cas, efectuem una consulta (select) sobre la taula Paisos. Obtenim un ResultSet, que no és res més que un conjunt de files amb els camps de la sentència select.

La classe **ResultSet** disposa de mètodes per desplaçar el cursor i recórrer les files (absolute(), afterLast(), beforeFirst(), first(), next(), previous(), relative()). Segons el tipus de ResultSet, alguns o tots els mètodes estaran disponibles. Per obtenir la informació continguda a les columnes, proporciona mètodes **getXXX()** per a diferents tipus de dades (getBoolean(), getDate(), getDouble(), getFloat(), getInt(), getLong(), getObject(), getString(), getTime(), etc.), els quals admeten com a paràmetre el número de columna (començant per 1) o el nom de la columna (tal i com es va etiquetar a la consulta SQL).

Un cop finalitzat l'accés a les dades, cal tancar els recursos amb el mètode **close()**. Cal tenir present que quan es tanca un Statement, també es tanca el ResultSet obtingut. Cada vegada que executem una consulta amb un mateix objecte Statement, es perd el ResulSet anterior. El mateix passa quan tanquem una connexió.

Connexió amb base de dades JDBC

Les limitacions del pont JDBC-ODBC fan preferible utilitzar drivers natius JDBC. Anem a programar la mateixa aplicació de l'apartat anterior, però ara l'origen de dades serà una base de dades **MySql**.

Si no disposem del driver per a bases de dades Mysql, el descarregarem d'Internet i el desarem en una ruta accessible al CLASSPATH. La base de dades es diu daic6 i la creació de la taula i inserció de dades es fa amb el codi SQL descrit a l'apartat anterior.

El codi de l'aplicació és el següent:

```
/**
 * MostraPaisosJdbcMySql.java
 * Exemple de connexió amb base de dades de paisos
 * utilitzant el driver JODBC
 * @author Jose Moreno
 * @version
*/
import java.sql.*;
public class MostraPaisosJdbcMySql
{
   public static void main(String[] args) throws Exception
   {
      final String DRIVER = "com.mysql.jdbc.Driver";
      final String BD_URL = "jdbc:mysql://"+"localhost/daic6";
      final String USUARI = "root";
      final String PASSWORD = "";
      Connection conn = null;
      try
      {
         //carregar el driver
         Class.forName(DRIVER);
         //obtenir una connexió amb la base de dades
         conn = DriverManager.getConnection(BD_URL, USUARI, PASSWORD);
         if (conn != null)
         {
            //crear un statement per a les consultes sql
```

```
            Statement stmt = conn.createStatement();
            ResultSet res = stmt.executeQuery("SELECT * FROM paisos");
            //escriure encapçalament dels resultats
            System.out.println(
               "id \t nom \t capital \t superficie_km2 \t poblacio \t pib_dolars \t
esp_vida");
            //recórrer el ResultSet i escriure el contingut
            while(res.next())
            {
               //convertir dades a format de dades de java
               int id = res.getInt("id");
               String nom = res.getString("nom");
               String capital = res.getString("capital");
               double superficie_km2 = res.getDouble("superficie_km2");
               double poblacio = res.getDouble("poblacio");
               double pib_dolars = res.getDouble("pib_dolars");
               double esp_vida = res.getDouble("esp_vida");
               //escriure dades
               System.out.println(
                  id+" \t"+nom +" \t "+capital+" \t "+superficie_km2
                  +" \t "+poblacio+" \t "+pib_dolars+" \t "+esp_vida);
            }
            res.close();
            stmt.close();
            conn.close();
         }
      }
      catch(SQLException ex)
      {
         System.out.println(ex);
      }
      catch(ClassNotFoundException ex)
      {
         System.out.println(ex);
      }
   }
}
```

Podem comprovar que el codi és idèntic, només cal canviar les dades de connexió amb la base de dades.

Amb **Class.forName()** carreguem el driver com.mysql.jdbc.Driver. A partir d'aquest moment, la classe **DriverManager** podrà utilitzar-lo per a la connexió amb bases de dades que tinguin aquest driver. Es llança **ClassNotFoundException** si no es troba la classe del driver.

Utilitzem el mètode **getConnection()** de la classe DriverManager per obtenir un objecte de tipus **Connection**. La classe Connection encapsula la connexió amb la base de dades. Els paràmetres que accepta depenen del tipus de base de dades. En aquest cas, li passem la URL de la base de dades (jdbc:mysql://localhost/daic6) així com l'usuari i la clau o contrasenya de connexió. El mètode getConnection() llança **SQLException** si es produeix algun error en l'accés a la base de dades.

Obtenció d'informació sobre el ResultSet: interface ResultSetMetaData

Els objectes de tipus **ResultSetMetaData** proporcionen mètodes per obtenir informació sobre els tipus i propietats de les columnes d'un ResultSet. Per obtenir un ResultSetMetaData invoquem el mètode **getMetaData()** de l'objecte ResultSet.

Alguns dels mètodes que podem utilitzar són els següents:

int getColumnCount()	nombre de columnes
String getColumnLabel(int column)	títol de la columna
String getColumnName(int column)	nom de la columna
int getColumnType(int column)	tipus SQL
String getColumnTypeName(int column)	nom del tipus SQL
boolean isAutoIncrement(int column)	indica si és un camp autonumèric
boolean isNullable(int column)	indica si el camp pot tenir valor nul
int getPrecision(int column)	nombre de xifres decimals

ResultSet arrossegable i/o modificable

Amb el JDBC 2.0, es disposa d'una altra versió de createStatement:

```
Statement createStatement ( int resultSetType, int resultSetConcurrency )
```

Els paràmetres són constants estàtiques de ResultSet. Els valors per a resultSetType són:

- TYPE_FORWARD_ONLY: moviment només cap endavant; és el valor per defecte
- TYPE_SCROLL_INSENSITIVE: cursor bidireccional insensible a canvis en la BD
- TYPE_SCROLL_SENSITIVE: cursos bidireccional sensible a canvis en la BD

Els valors per a resultSetConcurrency són:

- CONCUR_READ_ONLY: només lectura, no admet modificacions
- CONCUR_UPDATABLE: permet actualitzacions

Anem a programar una aplicació per gestionar una base de dades de països. La descripció i contingut de la taula de països és la descrita als apartats anteriors. La nostra aplicació permetrà explorar (*browse*) els registres de la taula de països, avançant, retrocedint i posicionant el cursos de la taula segons les peticions de l'usuari. En aquesta versió inicial, la interfície amb l'usuari serà de tipus text. A més, permetrà modificar el registre actiu i inserir nous registres.

En primer lloc, programem la classe que encapsula l'objecte Pais.

```
/**
 * Pais.java
 * classe per als objectes pais
 * @author Jose Moreno
 * @version
 */
public class Pais {
   private Long id;
   private String nom;
```

```
   private String capital;
   private Double superficieKm2;
   private Double poblacio;
   private Double pibDolars;
   private Double espVida;
   public Pais() {}
   public Pais( String nom, String capital, Double superficieKm2,
                Double poblacio, Double pibDolars, Double espVida )
   {
      this.nom = nom;
      this.capital = capital;
      this.superficieKm2 = superficieKm2;
      this.poblacio = poblacio;
      this.pibDolars = pibDolars;
      this.espVida = espVida;
   }
   public Long getId() { return id; }
   public void setId(Long id) { this.id = id; }
   public String getNom() { return nom; }
   public void setNom(String nom) { this.nom = nom; }
   public String getCapital() { return capital; }
   public void setCapital(String capital) { this.capital = capital; }
   public Double getSuperficieKm2() { return superficieKm2; }
   public void setSuperficieKm2(Double superficieKm2) {
      this.superficieKm2 = superficieKm2;
   }
   public Double getPoblacio() { return poblacio; }
   public void setPoblacio(Double poblacio) { this.poblacio = poblacio; }
   public Double getPibDolars() { return pibDolars; }
   public void setPibDolars(Double pibDolars) { this.pibDolars = pibDolars; }
   public Double getEspVida() { return espVida; }
   public void setEspVida(Double espVida) { this.espVida = espVida; }
   public boolean equals(Object obj) {
      if (!(obj instanceof Pais)) {
         return false;
      }
      Pais other = (Pais) obj;
      if ( !nom.equals(other.getNom()) || !capital.equals(other.getCapital()) ||
         !superficieKm2.equals(other.getSuperficieKm2()) ||
         !poblacio.equals(other.getPoblacio()) ||
         !pibDolars.equals(other.getPibDolars()) ||
         !espVida.equals(other.getEspVida()) )
      {
         return false;
      }
      return true;
   }
   public String toString() {
      return "["+nom+"]["+capital+"]["+superficieKm2+"]["+poblacio+"]
["+pibDolars+"]["+espVida+"]";
   }
}
```

La definició de la classe Pais és idèntica a la de qualsevol altre objecte del nostre model de negoci, llevat de la incorporació d'un atribut id per desar el camp autonumèric que identifica els registres de la taula Paisos.

```
/**
 * PaisosUpdate.java
 * Exemple d'actualització amb Statement
 * @author Jose Moreno
 * @version
*/
import java.sql.*;
import java.io.*;
public class PaisosUpdate
{
   public static void main(String[] args) throws Exception
   {
      //Dades connexió base de dades mysql
      final String DRIVER = "com.mysql.jdbc.Driver";
      final String BD_URL = "jdbc:mysql://"+"localhost/daic6";
      final String USUARI = "root";
      final String PASSWORD = "";
      //Dades connexió base de dades Access
/*    final String DRIVER = "sun.jdbc.odbc.JdbcOdbcDriver";
      final String BD_URL = "jdbc:odbc:Paisos";
      final String USUARI = "";
      final String PASSWORD = "";
*/
      Connection conn = null;
      try {
         //carregar el driver
         Class.forName(DRIVER);
         //obtenir una connexió amb la base de dades
         conn = DriverManager.getConnection(BD_URL, USUARI, PASSWORD);
         if (conn != null) {
            //crear un statement arrossegable i actualitzable per a les consultes sql
            Statement stmt = conn.createStatement(
               ResultSet.TYPE_SCROLL_INSENSITIVE,
                ResultSet.CONCUR_UPDATABLE );
            //executar la consulta i obtenir el resultat en un ResulSet
            String sql = "SELECT * FROM paisos";
            ResultSet res = stmt.executeQuery( sql );
            /*if ( res.getConcurrency()!=ResultSet.CONCUR_UPDATABLE )
            System.out.println( "ResultSet no actualitzable" );*/
            char opcio; //opció per al menú
            Pais pais = new Pais(); //dades a mostrar o desar
            Long id; //valor id del registre a actualitzar
            do { //iterar mentre no sortir
               //presentar menu i llegir opcio
               opcio = Menu();
               //tractar opcio
               switch ( opcio ) {
                  case 'Q': //sortir
                     break;
                  case 'F': //cursor a primer registre
                     res.first();
                     pais = ResultSetToPais(res);
                     System.out.println( pais.toString() );
                     break;
                  case 'P': //cursor a registre anterior
                     res.previous();
                     pais = ResultSetToPais(res);
```

```
                    System.out.println( pais.toString() );
                    break;
                case 'N': //cursor a registre següent
                    res.next();
                    pais = ResultSetToPais(res);
                    System.out.println( pais.toString() );
                    break;
                case 'L': //cursor a últim registre
                    res.last();
                    pais = ResultSetToPais(res);
                    System.out.println( pais.toString() );
                    break;
                case 'I': //inserir registre
                    pais=llegirPais();
                    insertRow( res, pais );
                    System.out.println( pais.toString() );
                    break;
                case 'U': //modificar registre
                    pais=llegirPais();
                    updateRow( res, pais );
                    System.out.println( pais.toString() );
                    break;
                default: break;
              }
            } while ( opcio!='Q' );
            //tancar recursos
            res.close();
            stmt.close();
            conn.close();
        }
      }
      catch(SQLException ex) {
         System.out.println(ex);
      }
      catch(ClassNotFoundException ex) {
         System.out.println(ex);
      }
   }
   /**
    * Menu()
    * presenta el menú i retorna l'opció escollida
    * @return opcio (char)
    */
   private static char Menu()
   {
      BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
      String [] opcions = {"Quit", "First", "Previous", "Next", "Last", "Insert",
"Update"};
      for (int i=0; i<opcions.length; i++) {
         System.out.println("["+opcions[i].charAt(0)+"]"+" - "+opcions[i]);
      }
      char op = 'Q';
      try {
         op = ((br.readLine()).toUpperCase()).charAt(0);
      } catch (IOException ioe) { }
       return op;
```

```
    }
    /**
     * insertRow()
     * inserta una fila nova a rs amb els valors de p
     * @param rs ResultSet
     * @param p Pais
     */
    private static void insertRow(ResultSet rs, Pais p)
    {
       try {
          //posicionar cursor en fila inserció
          rs.moveToInsertRow();
          //modificar camps
          //rs.updateLong( "id", p.getId() );
          rs.updateString( "nom", p.getNom() );
          rs.updateString( "capital", p.getCapital() );
          rs.updateDouble( "superficie_km2", p.getSuperficieKm2() );
          rs.updateDouble( "poblacio", p.getPoblacio() );
          rs.updateDouble( "pib_dolars", p.getPibDolars() );
          rs.updateDouble( "esp_vida", p.getEspVida() );
          //insertar fila
          rs.insertRow();
          //posicionar cursor a la fila actual
          rs.moveToCurrentRow();
       } catch (SQLException ioe) { System.out.println(ioe); }
    }
    /**
     * updateRow()
     * actualitza la fila actual de rs amb els valors de p
     * @param rs ResultSet
     * @param p Pais
     */
    private static void updateRow(ResultSet rs, Pais p)
    {
       try {
          //modificar camps
          //rs.updateLong( "id", p.getId() );
          rs.updateString( "nom", p.getNom() );
          rs.updateString( "capital", p.getCapital() );
          rs.updateDouble( "superficie_km2", p.getSuperficieKm2() );
          rs.updateDouble( "poblacio", p.getPoblacio() );
          rs.updateDouble( "pib_dolars", p.getPibDolars() );
          rs.updateDouble( "esp_vida", p.getEspVida() );
          //enviar els canvis
          rs.updateRow();
       } catch (SQLException ioe ) { System.out.println(ioe); }
    }
    /**
     * ResultSetToPais()
     * converteix la fila actual d'un RecordSet en un objecte Pais
     * @param res ResultSet del qual obtenir dades
     * @return objecte Pais
     */
    private static Pais ResultSetToPais( ResultSet res ) throws SQLException {
       Pais pais = new Pais();
       //recuperar valors dels camps
       Long id = res.getLong("id");
```

```
      String nom  = res.getString("nom");
      String capital   = res.getString("capital");
      Double superficieKm2 = res.getDouble("superficie_km2");
      Double poblacio = res.getDouble("poblacio");
      Double pibDolars = res.getDouble("pib_dolars");
      Double espVida = res.getDouble("esp_vida");
      //construeix l'objecte
      pais = new Pais( nom, capital, superficieKm2, poblacio, pibDolars, espVida );
       return pais;
   }
   /**
    * llegirPais()
    * demana per l'entrada estàndard valors per als atributs d'un país
    * @return Pais
    */
   private static Pais llegirPais()
   {
      BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
      try {
         System.out.print( "Nom: " );
         String nom = br.readLine();
         System.out.print( "Capital: " );
         String capital = br.readLine();
         System.out.print( "Superficie: " );
         Double superficieKm2 = Double.parseDouble(br.readLine());
         System.out.print( "Població: " );
         Double poblacio = Double.parseDouble(br.readLine());
         System.out.print( "PIB: " );
         Double pibDolars = Double.parseDouble(br.readLine());
         System.out.print( "Esp.Vida: " );
         Double espVida = Double.parseDouble(br.readLine());
         Pais pais = new Pais( nom, capital, superficieKm2, poblacio, pibDolars,
espVida );
         return pais;
      } catch (Exception e) {return new Pais();}
   }
}
```