
Grup-6

¿A qué jugamos? Estructura Nucli Java

Versió 3.0

G6

| | |
|-----------------------|------------------|
| ¿A qué jugamos ? | Version: 3.0 |
| Estructura Nucli Java | Data: 24/04/2018 |
| | |

Històric de revisions

| Data | Versió | Descripció | Autor |
|------------|--------|---------------------------------|-----------------------------------|
| 20/03/2018 | 1.0 | Primer Plantejament nucli Java | Alejandro García i Sergi Portero. |
| 10/04/2018 | 2.0 | Segon Plantejament nucli Java | Alejandro García i Sergi Portero. |
| 24/04/2018 | 3.0 | Tercer plantejament nucli Java. | Alejandro García. |

| | |
|-----------------------|------------------|
| ¿A qué jugamos ? | Version: 3.0 |
| Estructura Nucli Java | Data: 24/04/2018 |
| | |

Taula de continguts

| | |
|---|---|
| 1. Informació estructura Nucli Java | 4 |
| 2. Sistema de logs | 7 |
| 3. Canvis 24/04/2018 | 8 |

| | |
|-----------------------|------------------|
| ¿A qué jugamos ? | Version: 3.0 |
| Estructura Nucli Java | Data: 24/04/2018 |
| | |

1. Informació estructura Nucli Java

He utilitzat una capa DAO (Data Acces Object) per aïllar la implementació de la base de dades que utilitzem. D'aquesta forma, complim el principi de **"single responsibility"**, i aconseguim un codi amb menys acoblament i alhora, més cohesionat ja que cada classe DAO tindrà els mètodes per gestionar les seves respectives classes concretes, i no tindrem totes les funcions una sola classe de aplicacionBD.

```
public abstract class DAOFactory {

    /** numero para identificar que queremos usar una conexion Mysql */
    public static final int MYSQL = 0;

    /** Abstract method para el JocAlternatiuDAO. */
    public abstract JocAlternatiuDAO getJocAlternatiuDAO();
    /** Abstract method para el JocDAO. */
    public abstract JocDAO getJocDAO();
    /** Abstract method para el JocTaulaDAO. */
    public abstract JocTaulaDAO getJocTaulaDAO();
    /** Abstract method para el PartidaDAO. */
    public abstract PartidaDAO getPartidaDAO();
    /** Abstract method para el UsuarioDAO. */
    public abstract UsuarioDAO getUsuarioDAO();

    /** Aqui podremos añadir tantas bases de datos diferentes como queramos,
     * desde el main solo haria falta cambiar el numero de la base de datos a
     * utilizar y adaptariamos todo**
     */

    public static DAOFactory getDAOFactory(int databaseType) {
        switch(databaseType) {
            case MYSQL:
                return new MysqlDAOFactory();

            default:
                return null;
        }
    }
}
```

La classe DAOFactory l'única cosa que farà serà crear objectes DAO (objectes que accedeixin a la base de dades i facin operacions amb ella) amb funcions abstractes per retornar un objecte de tipus interfície (general) de cada tipus d'objecte sense tenir en compte si usen Mysql, Oracle o qualsevol altra base de dades i un mètode per especificar que tipus de base de dades volem usar, a través d'aquest mètode en el main triem que base de dades volem usar:

```
public class Main {

    public static void main(String[] args) throws SQLException {

        DAOFactory mysqlFactory = DAOFactory.getDAOFactory(DAOFactory.MYSQL);
    }
}
```

Simplement hem de passar-li per paràmetre el nombre de la base de dades a usar (en aquest cas he definit un int estàtic a la pròpia classe per usar-ho des de qualsevol costat, cridat igual que la base de dades a usar), i tot el nostre codi estarà adaptat sense tocar res més en el codi.

| | |
|-----------------------|------------------|
| ¿A qué jugamos ? | Version: 3.0 |
| Estructura Nucli Java | Data: 24/04/2018 |
| | |

```

public MysqlDAOFactory() {};

/** Usaremos este metodo cada vez que queramos una conexio de tipo mysql */
public static Connection crearConexio() {
    Connection conn = null;
    MysqlDataSource dataSource;
    dataSource = new MysqlDataSource();
    dataSource.setUser("root");
    dataSource.setServerName("127.0.0.1");
    dataSource.setPassword("");
    dataSource.setDatabaseName("AQueJugamos");

    try {
        conn = dataSource.getConnection();
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return conn;
}

```

A la classe MysqlDAOFactory implementem el mètode estàtic que retorna una connexió de tipus Mysql.

```

@Override
public JocAlternatiuDAO getJocAlternatiuDAO() {
    return new JocAlternatiuDAOMysqlImp();
}

```

I els mètodes que fan override als de la classe DAOFactory, per retornar una connexió de tipus Mysql. Cada objecte DAO (usuari, joc, partida, etc) té una interfície per aïllar la implementació de cadascun d'aquests, i d'aquesta manera poder tractar per igual un DAO de mysql, que un de oracle o qualsevol altre accés a dades. Això dona una gran escalabilitat i flexibilitat.

```

public interface UsuarioDAO {

    public void carregarUsuarios();
    public boolean comprobarUsuario(String mail, String password);
    public void agregarListaPreferits(int idJoc, int idUsuario);
    public void consultaDadesUsuario();
    public void insertarDadesUsuario(String nomUsuario, String password, String mail, String fe
    public boolean comprobarUsuarioLogejat();
    public void logout();
    public List<Usuario> getListaUsuarios();

}

```

(Exemple d'interfície, aquests mètodes els tindran tots els objectes DAO sigui el que sigui la seva implementació)

Finalment queda implementar cada DAO segons la seva base de dades (estan en el paquet com.aquejugamos.DAOImplementation, en aquestes classes podem treballar com sempre hem fet, no hem de canviar gens mes), nosaltres de moment solament necessitem la de Mysql, així que fem la implementació per Mysql (ja la teníem feta, solament he separat per classes les funcions que ja teníem).

| | |
|-----------------------|------------------|
| ¿A qué jugamos ? | Version: 3.0 |
| Estructura Nucli Java | Data: 24/04/2018 |
| | |

```

public class UsuarioDAOImpl implements UsuarioDAO{

    /** Logger */
    private static Logger logger = LoggerFactory.getLogger(UsuarioDAOImpl.class);

    private List<Usuari> llistaUsuaris = new ArrayList<Usuari>();

    public void carregarUsuaris() {
        Connection conn = MysqlDAOFactory.crearConexion();
        try {
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT U.idUsuario, U.Email, U.Nombre, U.password, U.Activo, U.bloqueado, U.isAdmin, U.grupo, U.fecha, U.provincia FROM Usuario U");

            while(rs.next())
            {
                int idUsuario=rs.getInt(1);
                String email=rs.getString(2);
                String nombre = rs.getString(3);
                String password=rs.getString(4);
                int activo = rs.getInt(5);
                int bloqueado = rs.getInt(6);
                int isAdmin = rs.getInt(7);
                String grupo = rs.getString(8);
                String fecha = rs.getString(9);
                String provincia = rs.getString(10);
                Usuari usuari = new Usuari(idUsuario,nombre,password,email,activo,bloqueado, isAdmin, grupo, fecha, provincia);
                llistaUsuaris.add(usuari);
            }

            stmt.close();
            rs.close();
        } catch (SQLException e) {
            logger.error("Error al cargar usuarios: " + e.getMessage());
        }
    }
}

```

En el main solamente falta instanciar las clases:

```

public class Main {

    public static void main(String[] args) throws SQLException {

        DAOFactory mysqlFactory = DAOFactory.getDAOFactory(DAOFactory.MYSQL);
        UsuarioDAO usuariDAO = mysqlFactory.getUsuarioDAO();
        JocAlternatiuDAO jocAlternatiuDAO = mysqlFactory.getJocAlternatiuDAO();
        JocTaulaDAO jocTaulaDAO = mysqlFactory.getJocTaulaDAO();
        PartidaDAO partidaDAO = mysqlFactory.getPartidaDAO();
        JocDAO jocDAO = mysqlFactory.getJocDAO();

        jocDAO.mostrarJocs();
        jocAlternatiuDAO.carregarJocsAlternatius();
        jocTaulaDAO.carregarJocsTaula();
        usuariDAO.carregarUsuaris();
    }
}

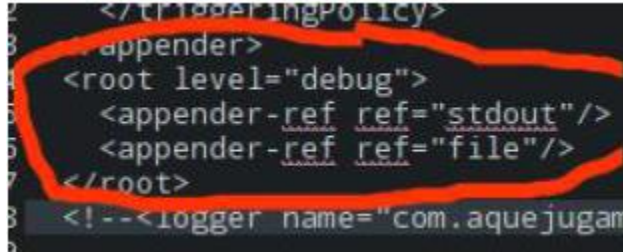
```

(UsuarioDAO tendrá los métodos de base de datos relacionados con usuario: login, registrar, veure favorits usuari, etc...), el de JocDAO el relacionado con Joc, etc...) Si necesitamos añadir una nueva función solamente hay que añadirla a la clase a la que pertenece y ya.

| | |
|-----------------------|------------------|
| ¿A qué jugamos ? | Version: 3.0 |
| Estructura Nucli Java | Data: 24/04/2018 |
| | |

2. Sistema de logs

També he afegit les llibreries necessàries per poder utilitzar logs i he configurat l'arxiu XML per personalitzar-ho. En aquest arxiu he posat que els logs es guardin en un fitxer de text i que surtin per consola.



```

<root level="debug">
  <appender-ref ref="stdout"/>
  <appender-ref ref="file"/>
</root>

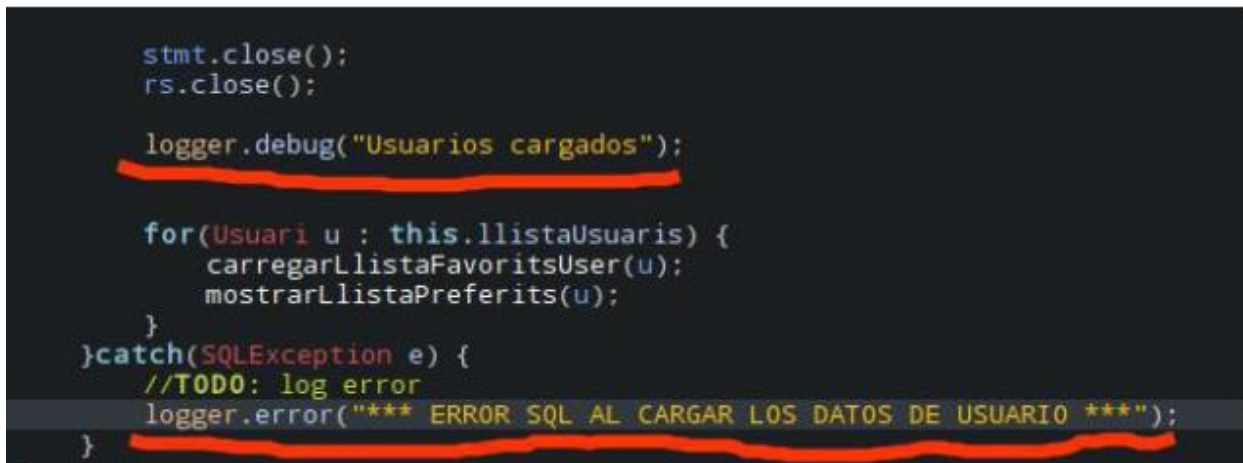
```

Solament hem de modificar aquests valors, si volem solament escriure'ls en fitxer podem comentar la línia que té un appender "stdout" (que és la que especifica que es mostrin per consola).

Aquests són els nivells de log que ens proporciona la llibreria de logback:

| level of request <i>p</i> | effective level <i>q</i> | | | | | |
|---------------------------|--------------------------|-------|------|------|-------|-----|
| | TRACE | DEBUG | INFO | WARN | ERROR | OFF |
| TRACE | YES | NO | NO | NO | NO | NO |
| DEBUG | YES | YES | NO | NO | NO | NO |
| INFO | YES | YES | YES | NO | NO | NO |
| WARN | YES | YES | YES | YES | NO | NO |
| ERROR | YES | YES | YES | YES | YES | NO |

Es pot especificar en el XML (on surt root level="debug") que nivell de logs volem mostrar/guardar, si solament ens interessen els errors, posarem el nivell a "ERROR". Àdhuc no he posat gairebé cap log, solament he implementat el sistema, aquí una petita mostra de com funciona:



```

stmt.close();
rs.close();

logger.debug("Usuarios cargados");

for(Usuari u : this.llistaUsuaris) {
    cargarLlistaFavoritsUser(u);
    mostrarLlistaPreferits(u);
}
}catch(SQLException e) {
    //TODO: log error
    logger.error("*** ERROR SQL AL CARGAR LOS DATOS DE USUARIO ***");
}

```

| | |
|-----------------------|------------------|
| ¿A qué jugamos ? | Version: 3.0 |
| Estructura Nucli Java | Data: 24/04/2018 |
| | |

Posem logger.nivell-de-importància-del-log("missatge");

En el fitxer log4j-application.log podem veure els logs guardats.

```
1 2018-04-19 00:56:02 DEBUG c.a.D.UsuarioDAOImpl:52 - Usuarios cargados
2 2018-04-19 00:57:48 DEBUG c.a.D.UsuarioDAOImpl:52 - Usuarios cargados
```

3. Canvis 24/04/2018

He començat a fer algun test i he hagut de fer alguns canvis. He pensat bastant sobre com testear els DAO ja que les seves funcions es basen en trucades a la base de dades, però no tenen molta mes lògica, per la qual cosa no t'ènia sentit fer un mockObject (el qual simula una tornada de dades i s'aplica la lògica sobre aquestes dades). He mirat diferents opcions per internet però sembla ser que és un tema complicat, hi ha gent que diu que el referent a la base de dades se sol testear en les proves d'integració i uns altres que també s'ha de fer unit testing.

Al final he optat pel següent:

-Tenir una base de dades en local que sigui una còpia de l'original (no en dades, solament en estructura), on puguem fer les proves sabent les dades que hi ha en ella i sense que puguin haver-hi efectes col·laterals en la base de dades original.

Per a això, he creat una classe que s'encarregarà de crear connexions Mysql a la base de dades de testeig:

```
public class MysqlDAOFactoryWithTestDB extends DAOFactory{
    /** Logger */
    private static Logger logger = LoggerFactory.getLogger(MysqlDAOFactoryWithTestDB.class);

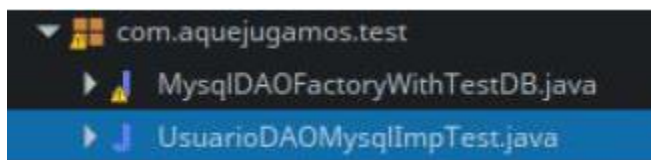
    public MysqlDAOFactoryWithTestDB() {}

    /** Usaremos este metodo cada vez que queramos una conexion de tipo mysql */
    public static Connection crearConexion() {
        Connection conn = null;
        MysqlDataSource dataSource;
        dataSource = new MysqlDataSource();
        dataSource.setUser("root");
        dataSource.setServerName("127.0.0.1");
        dataSource.setPassword("");
        dataSource.setDatabaseName("aquejugamostest"); //cambiamos a una base de datos local de prueba

        try {
            conn = dataSource.getConnection();
        } catch (SQLException e) {
            e.printStackTrace();
        }

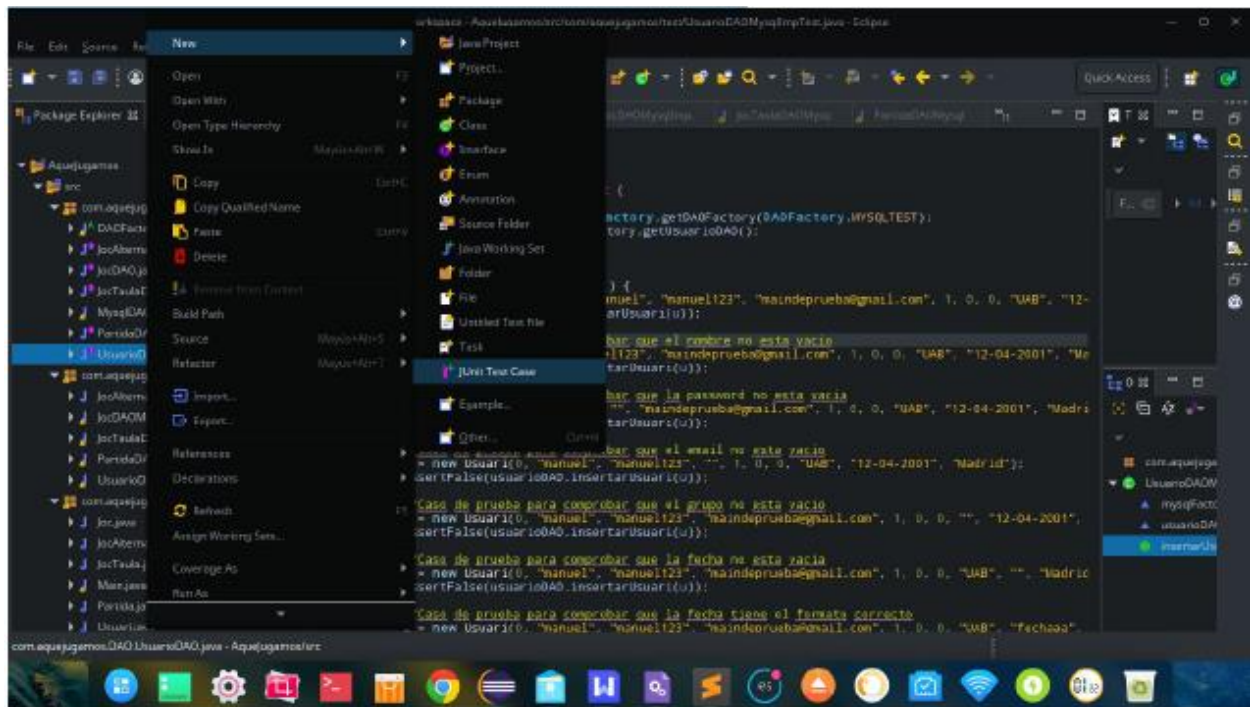
        return conn;
    }
}
```

I per a cada classe DAO d'implementació (les que contenen Impl) farem un fitxer de testing amb Junit (de moment solament he començat el de Usuari)



| | |
|-----------------------|------------------|
| ¿A qué jugamos ? | Version: 3.0 |
| Estructura Nucli Java | Data: 24/04/2018 |
| | |

Per crear els fitxers de testing hem de seleccionar amb clic dret sobre la classe a testear i seleccionar Junit Test Case:



I ens sortirà una finestra com aquesta:

| | |
|-----------------------|------------------|
| ¿A qué jugamos ? | Version: 3.0 |
| Estructura Nucli Java | Data: 24/04/2018 |
| | |

JUnit Test Case

⚠ Warning: Class under test 'com.aquejugamos.DAO.UsuarioDAO' is an interface.

☐ New JUnit 3 test ☒ New JUnit 4 test

Source folder: Aquejugamos/src Browse...

Package: com.aquejugamos.test Browse...

Name: UsuarioDAOTest

Superclass: java.lang.Object Browse...

Which method stubs would you like to create?

☐ setUpBeforeClass() ☐ tearDownAfterClass()
☒ setUp() ☐ tearDown()
☐ constructor

Do you want to add comments? (Configure templates and default value [here](#))

☐ Generate comments

Class under test: com.aquejugamos.DAO.UsuarioDAO Browse...

? < Back Next > Cancel Finish

Hem de seleccionar que ho posarem en el paquet de test, i com a convenció podem cridar als fitxers de test igual que el fitxer a testear afegint un "Test" al final.

Finalment ja podem implementar els test casi:

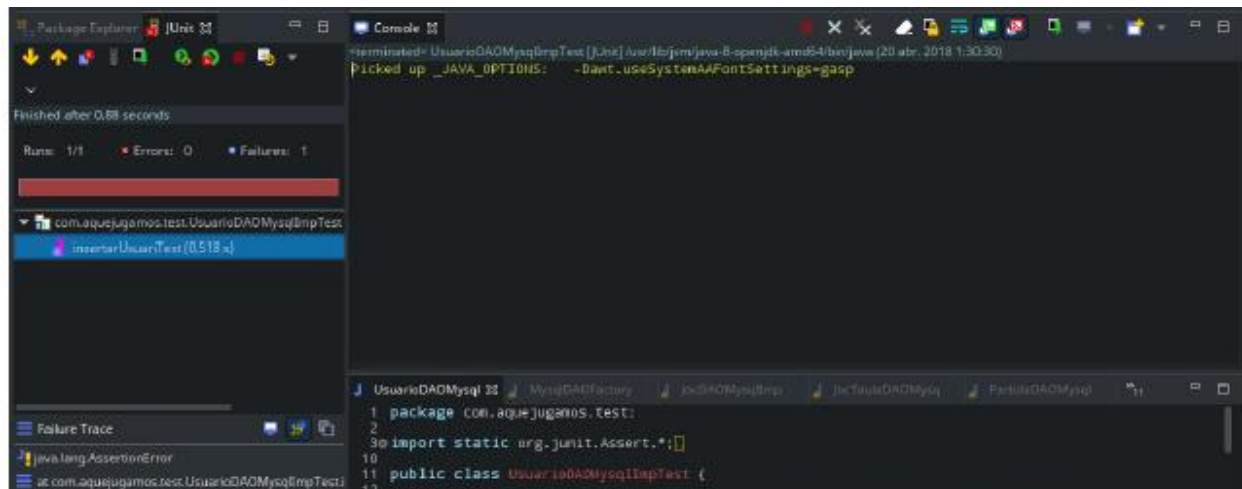
| | |
|-----------------------|------------------|
| ¿A qué jugamos ? | Version: 3.0 |
| Estructura Nucli Java | Data: 24/04/2018 |
| | |

```

1 package com.aquejugamos.test;
2
3 import static org.junit.Assert.*;
4
5 public class UsuarioDAOMySQLImpTest {
6
7     DAOFactory mysqlFactory = DAOFactory.getDAOFactory(DAOFactory.MYSQLTEST);
8     UsuarioDAO usuarioDAO = mysqlFactory.getUsuarioDAO();
9
10    @Test
11    public void insertarUsuarioTest() {
12        Usuario u = new Usuario(0, "manuel", "manuel123", "maindeprueba@gmail.com", 1, 0, 0, "UAB", "12-04-2001", "Madrid");
13        assertTrue(usuarioDAO.insertarUsuario(u));
14
15        //Caso de prueba para comprobar que el nombre no esta vacio
16        u = new Usuario(0, "", "manuel123", "maindeprueba@gmail.com", 1, 0, 0, "UAB", "12-04-2001", "Madrid");
17        assertFalse(usuarioDAO.insertarUsuario(u));
18
19        //Caso de prueba para comprobar que la password no esta vacia
20        u = new Usuario(0, "manuel", "", "maindeprueba@gmail.com", 1, 0, 0, "UAB", "12-04-2001", "Madrid");
21        assertFalse(usuarioDAO.insertarUsuario(u));
22
23        //Caso de prueba para comprobar que el email no esta vacio
24        u = new Usuario(0, "manuel", "manuel123", "", 1, 0, 0, "UAB", "12-04-2001", "Madrid");
25        assertFalse(usuarioDAO.insertarUsuario(u));
26
27        //Caso de prueba para comprobar que el grupo no esta vacio
28        u = new Usuario(0, "manuel", "manuel123", "maindeprueba@gmail.com", 1, 0, 0, "", "12-04-2001", "Madrid");
29        assertFalse(usuarioDAO.insertarUsuario(u));
30
31        //Caso de prueba para comprobar que la fecha no esta vacia
32        u = new Usuario(0, "manuel", "manuel123", "maindeprueba@gmail.com", 1, 0, 0, "UAB", "", "Madrid");
33        assertFalse(usuarioDAO.insertarUsuario(u));
34    }
35}

```

Hem d'indicar que usarem la base de dades de test, i per a cada funció de la classe que testegem, creem una funció de test, on posarem tots els casos a cobrir, i compararem si les dades que retorna concorden amb el que hauria de retornar. Si fem clic dret sobre el fitxer de test, i li donem a Run as < Junit Test, s'executarà el test.



En aquest cas surt en vermell, indicant que no ha passat el test. Això és a causa que en el codi àdhuc no tenim implementades les validacions de les dades que introdueix l'usuari, per la qual cosa de moment el test troba els errors.

Finalment he fet uns petits canvis en el sistema de logs:

| | |
|-----------------------|------------------|
| ¿A qué jugamos ? | Version: 3.0 |
| Estructura Nucli Java | Data: 24/04/2018 |
| | |

```
<?xml version="1.0" encoding="UTF-8"?>

<configuration>
  <appender name="stdout" class="ch.qos.logback.core.ConsoleAppender">
    <Target>System.out</Target>
    <encoder>
      <pattern>%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n</pattern>
    </encoder>
  </appender>
  <appender name="file" class="ch.qos.logback.core.rolling.RollingFileAppender">
    <!-- See also http://logback.qos.ch/manual/appenders.html#RollingFileAppender -->
    <!--<File>log4j-application.log</File>-->
    <encoder>
      <pattern>%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n</pattern>
    </encoder>
    <!--<rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
      <maxIndex>10</maxIndex>
      <FileNamePattern>log4j-application.log.%i</FileNamePattern>
    </rollingPolicy>-->
    <!--<file>${USER_HOME}/logfile.log</file>-->
    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
      <fileNamePattern>logs/aquejugamosLOG.%d.log</fileNamePattern>
    </rollingPolicy>
    <triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
      <MaxFileSize>5MB</MaxFileSize>
    </triggeringPolicy>
  </appender>
  <root level="debug">
    <!--<appender-ref ref="stdout"/>-->
    <appender-ref ref="file"/>
  </root>
```

He modificat el XML per guardar un arxiu log en una carpeta anomenada logs, on es crearà un arxiu de log diferent per cada dia, i cada fitxer tindrà com a nom aquejugamosLOG"*fechaDeCuandoSeCreo".log, per la qual cosa podrem buscar els logs per dia mes fàcilment.