Començat el	dimecres, 22 setembre 2021, 19:40
Estat	Acabat
Completat el	dimarts, 19 octubre 2021, 18:27
Temps emprat	26 dies 22 hores
Qualificació	**7,75** sobre 10,00 (**78**%)

CFGS Desenvolupament d'Aplicacions Web

Mòdul 7 B2 – Desenvolupament web en entorn servidor

UF3 – Tècniques d'accés a dades

Unitat 4 – Tècniques d'accés a dades

EAC4

(Curs 2021–22 / 1r semestre)

Presentació i resultats d'aprenentatge

Aquest exercici d'avaluació continuada (EAC) es correspon amb els continguts treballats a la unitat 4 **"Tècniques d'accés a dades"**

Els **resultats d'aprenentatge** que es plantegen són:

- Desenvolupa aplicacions d'accés a magatzems de dades, aplicant mesures per mantenir la seguretat i la integritat de la informació.

Criteris d'avaluació

La puntuació màxima assignada a cada activitat s'indica a l'enunciat.

Els criteris que es tindran en compte per avaluar el treball de l'alumnat són els següents:

- S'han analitzat les tecnologies que permeten l'accés mitjançant programació a la informació disponible en magatzems de dades.
- S'han creat aplicacions que estableixin connexions amb bases de dades.
- S'ha recuperat informació emmagatzemada en bases de dades.
- S'ha publicat en aplicacions web la informació recuperada.
- S'han utilitzat conjunts de dades per emmagatzemar la informació.
- S'han creat aplicacions web que permetin l'actualització i l'eliminació d'informació disponible en una base de dades.
- S'han utilitzat transaccions per mantenir la consistència de la informació.
- S'han provat i subsidiat les aplicacions.

Forma de lliurament

Us demanem que, a part de contestar aquest qüestionari, copieu totes les preguntes i respostes en un document *.odt.

En aquest document hi heu d'afegir totes les preguntes i respostes de TOTS els qüestionaris de l'EAC en curs i, finalment, entregar-ho a la tasca EACx corresponent.

(si copieu les respostes al document sempre tindreu una còpia en cas que s'acabi la sessió del navegador també. ;D)

1

Puntuació 0,00 sobre 0,75

Completa

Quin és el motiu pel qual els tests unitaris són acumulatius, és a dir, un cop un test passa no es treu del conjunt de test, si no que es deixa?

Fes-ne un raonament curt però tan precís com puguis.

Podria ser pel tema de la refactorització, ja que l'essència d'aquesta tècnica consisteix a aplicar una sèrie de petits canvis en el codi mantenint el seu comportament. Cadascun d'aquests canvis ha de ser tan petit que pugui ser completament controlat per nosaltres sense por a equivocar-nos. És l'efecte acumulatiu de totes aquestes modificacions el que fa de la Refactorització una potent tècnica. L'objectiu final d'refactorizar és mantenir el nostre codi senzill i ben estructurat.

Comentari:

2

Puntuació 0,75 sobre 1,00

Completa

Teniu la següent interfície que defineix un Repository:

```
public interface CotxesRepository {
  Cotxe getCotxeByName(String name); // Retorna el Cotxe que coincideix amb el nom name
   List<Cotxe> getAllCotxes();  // Retorna un llistat amb tots els Cotxes a la BD
 void changeCotxeStock(Cotxe cotxe, int numUnits); // Canvia l'estoc del cotxe, substituïnt numUnitatsDisp per el valor de numUnits
}
```

amb el model:

```
public class Cotxe {
   private String nom;
   private Double preu;
   private int numUnitatsDisp;
  // Getters i setters
}
```

Desenvolupeu un DAO utilitzant JDBC. Podeu assumir que teniu la connexió establerta i disponible. Es valorarà que refactoritzeu el codi per reutilitzar parts comunes.

// Evitant Injecció SQL

```
  public Cotxe getCotxeByName(String name) throws Exception {
        String qry = "select * from cotxes where nom =?";
        PreparedStatement preparedStatement = getPreparedStatement(qry);
        preparedStatement.setString(1, name);
        return findUniqueResult(preparedStatement);
    }
```

```
    public  List<Cotxe> getAllCotxes() throws SQLException {
        String qry = "select * from cotxes";
        PreparedStatement preparedStatement = getPreparedStatement(qry);
        List<Cotxes> cotxes = executeQuery(preparedStatement);
        return cotxes;
    }
```

```
    public changeCotxeStock(Cotxe cotxe, int numUnits)throws Exception {
        String qry = "UPDATE cotxes SET numUnitatsDisp= ?  WHERE nom= ? ";
        PreparedStatement preparedStatement = getPreparedStatement(qry);
        preparedStatement.setInt(1, numUnits);
        preparedStatement.setString(2, cotxe.getNom());
        createOrUpdateCotxe(cotxe.getNom(), preparedStatement);
    }
```

```
    private  createOrUpdateCotxe(String nom, PreparedStatement preparedStatement) throws Exception {
```

```
        int result = executeUpdateQuery(preparedStatement);
        if (result == 0) {
            throw new Exception("Error creating cotxe");
        }
         findCotxeByCotxename(nom);
    }
```

```
    private Cotxe findUniqueResult(PreparedStatement preparedStatement) throws Exception {
        List<Cotxe> cotxes = executeQuery(preparedStatement);
        if (cotxes.isEmpty()) {
            return null;
        }
        if (cotxes.size() > 1) {
            throw new Exception("Only one result expected");
        }
        return cotxes.get(0);
    }
```

```
  private List<Cotxes> executeQuery(PreparedStatement preparedStatement) {
List<Cotxe> cotxes = new ArrayList<>();

try (
ResultSet rs = preparedStatement.executeQuery()) {
while (rs.next()) {
Cotxe cotxe = buildCotxeFromResultSet(rs);
cotxes.add(cotxe);
}
} catch (SQLException e) {
e.printStackTrace();
}
return cotxes;
}
```

```
private PreparedStatement getPreparedStatement(String query) throws SQLException {
if (getConnection() == null) {
try {
setConnection(dBConnection.getConnection());
} catch (SQLException | IOException e) {
e.printStackTrace();
}
}
return getConnection().prepareStatement(query);
}
```

```
private int executeUpdateQuery(PreparedStatement preparedStatement) {
int result = 0;
if (getConnection() == null) {
try {
setConnection(dBConnection.getConnection());
} catch (SQLException | IOException e) {
e.printStackTrace();
}
}
try {
result = preparedStatement.executeUpdate();
} catch (SQLException e) {
e.printStackTrace();
}
return result;
}
```

```
private Cotxe buildCotxeFromResultSet(ResultSet rs) throws SQLException {
String nom = rs.getInt("nom");
Double preu = rs.getDouble("preu");
Int numUnitatsDisp = rs.getInt("numUnitatsDisp");
Cotxe cotxe = new Cotxe(nom, preu,numUnitatsDisp);
return cotxe;
}
```

```
public Connection getConnection() {
return connection;
}

public void setConnection(Connection connection) {
this.connection = connection;
}
```

Comentari:
No és classe

Puntuació 0,00 sobre 1,00

Completa

3

Tenim el següent mètode per desactivar un usuari:

```
public void deleteUser(User user) throws Exception {
   String qry = "UPDATE users SET active = false where user_id = ?";
   PreparedStatement preparedStatement = getPreparedStatement(qry);
   preparedStatement.setInt(1, user.getUserId());
   createOrUpdateUser(user.getUsername(), preparedStatement);
}
```

Creeu un test unitari per comprovar-ne el funcionament correcte. Podeu assumir-ne el DAO fet.

```
@Test
public void deleteUser() throws Exception {
    String username = "testUser";
    String name = "Tester User";
    String email = "Tester@email.com";
    User createdUser = userDAO.createUser(username, name, email);
    Assert.assertNotNull(createdUser);
    assertTrue( createdUser.isActive());
    User deletedUser = userDAO.findUserByUsername(username);
    Assert.assertNotNull(deletedUser);
    Assert.assertFalse(deletedUser.isActive());
}
```

Comentari:
No borres l'usuari al test

4

Puntuació 0,25 sobre 0,25

Correcte

Per assegurar que un objecte no té assignat un valor s'utilitza l'expressió...

Trieu-ne una o més:

- [ ] a. Assert.assertNotNull
- [x] b. Object obj = ""; ✗
- [x] c. Assert.assertNull ✓
- [ ] d. @Assert.null

La resposta és correcta.

La resposta correcta és: Assert.assertNull

5

Puntuació 1,00 sobre 1,00

Completa

Tenim la següent interface:

```
public interface VideojocRepository {
  List<Videojoc> getAllVideojocs();
}
```

Escriu el mètode del DAO en el supòsit que treballem amb JDBC (assumint la connexió feta) i i si treballem amb JPA (EntityManager llest per utilitzar)

```
//JDBC
public List<Videojoc> getAllVideojocs() {
String qry = "select * from Videojoc";
List<Videojoc> videojocs = new ArrayList<>();
try (
Connection conn = dBConnection.getConnection();
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery(qry);) {
while (rs.next()) {
int VideojocsId = rs.getInt("Videojocs_id");
String VideojocsName = rs.getString("Videojocs_name");
int puntuacio = rs.getInt("puntuacio");
```

```
Videojocs videojocs = new Videojosc(VideojocsId, VideojocsName, puntuacio);
```

```
videojocs.add(videojocs);
```

```
}
} catch (SQLException | IOException e) {
e.printStackTrace();
}
return videojocs;
}
```

```

```

```
//JPA
```

public List <Videojoc> getAllVideojocs ( ) {

return (List <Videojoc> entityManager.createQuery (" Select * from Videojoc").getResultList( );

}

Comentari:

6

Puntuació 0,75 sobre 1,00

Completa

Ens encarreguen el desenvolupament d'una app web per al lloguer de vivendes.

Implementeu la següent entitat:

mitjançant un POJO JPA, amb les restriccions adients.

```
@Entity
@Table(name = "Vivenda")
public class Vivenda implements Serializable {

private static final long serialVersionUID = 1L;

@Id
@NotNull
@Column(name = "idVivenda")
private
Int idVivenda;
```

@NotNull

@Size(max = 255)
@Column(name = "carrer")
private String carrer;

@Column(name = "numero") private Int numero;
@NotNull
@Size(max = 45)
@Column(name = "poblacio")
private String poblacio;

```
@NotNull
@Size(max = 45)
@Column(name = "provincia")
private String provincia;
```

public Int getIdVivenda() {
return idVivenda;
}

public void setIdVivenda(Int idVivenda) {
this.idVivenda = idVivenda;
}

---

public String getIdVivenda() {
return carrer;
}

public void setCarrer(String carrer) {
this.carrer = carrer;
}

```
public Int getNumero() {
return numero;
}

public void setNumero(Int numero) {
this.numero = numero;
}
```

```
public String getIdPoblacio() {
return poblacio;
}

public void setPoblacio(String poblacio) {
this.poblacio = poblacio;
}
```

```
public String getProvincia() {
return provincia;
}

public void setProvincia(String provincia) {
this.provincia = provincia;
}
}
```

Comentari:
@Table innecesari

7

Puntuació 0,50 sobre 0,50

Correcte

JPA neix per solucionar diversos problemes. Marca aquells que siguin correctes

Trieu-ne una o més:

- [x] a. Els drivers utilitzats no tenen tota la funcionalitat necessària i cal afegir una capa transparent al driver ✗
- [x] b. Cada BD utilitza sentències pròpies i caldria reescriure el codi si canviem de BD ✓
- [x] c. Estalviem molt codi repetitiu ✓
- [ ] d. Utilitzant JDBC ja aconseguim objectes en comptes de valors escalars

La resposta és correcta.

Les respostes correctes són: Cada BD utilitza sentències pròpies i caldria reescriure el codi si canviem de BD, Estalviem molt codi repetitiu

---

8

Puntuació 0,50 sobre 0,50

Correcte

Trieu l'opció correcta

Trieu-ne una:

- a. @NotNull indica que un objecte no pot ser null abans de guardar-se a la base de dades ✔
- b. @Size determina només la mida mínima que pot tenir un atribut quan es guarda a la base de dades
- c. @Column indica el nom de la taula on es guardaran les dades d'un objecte Java

La resposta és correcta.

La resposta correcta és: @NotNull indica que un objecte no pot ser null abans de guardar-se a la base de dades

9

Puntuació 2,00 sobre 2,00

Completa

Tenim la següent interfície:

```
public interface VolsRepository {
   Vol getVolByDesti(String desti);
   List<Vol> getAllVols();
   List<Vol> getVolByCompany(String company);
   void addVol(Vol vol);
   void updateVol(Vol vol);
}
```

Escriu el DAO Hibernate que l'implementi

```
//DAO Hibernate
```

```
import org.hibernate.Criteria;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.criterion.Order;
import org.hibernate.criterion.Restrictions;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;
import javax.transaction.Transactional;
```

```

@Transactional
@Repository("VolHibernateDAO")
public class VolHibernateDAO implements VolsRepository {
```

```
 @Autowired
 private SessionFactory sessionFactory;
 @Override
```

```

public Vol getVolByDesti(String desti) {
       Criteria criteria = createEntityCriteria();
       criteria.add(Restrictions.eq("disti", desti));
       return (Vol) criteria.uniqueResult();
   }
```

```

@Override
public void addVol(Vol vol) {
       getSession().saveOrUpdate(vol);
   }
```

```

```

```
 @Override
```

```
public List<Vol> getAllVols() {
 return getSession().createQuery("from Vols").list();
}
```

```
  @Override
  public List<Vol> getVolByCompany(String company) {
     Criteria criteria = createEntityCriteria();
     criteria.add(Restrictions.eq("company", company));
     return (List<Vol>) criteria.list();
  }
```

```
    @Override
    public void updateVol(Vol vol) {
         getSession().merge(vol);
    }
```

```
    protected Session getSession() {
        return sessionFactory.getCurrentSession();
    }
    private Criteria createEntityCriteria() {
        return getSession().createCriteria(Vol.class);
    }
}
```

```
```

```
//Vol
```

```
package org.ioc.daw.vol;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToMany;
import javax.persistence.Table;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;
import java.io.Serializable;
import java.util.List;
import org.ioc.daw.answer.Answer;

@Entity
@Table(name = "vol")
public class Vol implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @NotNull
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Integer volId;

    @NotNull
    @Size(max = 45)
    @Column(name = "desti")
    private String desti;

   @NotNull
    @Size(max = 45)
    @Column(name = "company")
    private String company;

    @OneToMany(cascade = {CascadeType.ALL}, fetch = FetchType.EAGER)
    private List<Answer> answers;

    public Vol() {
    }
```

```
public Vol(Integer volId, String desti, String company) {
        this.volId = volId;
        this.desti = desti;
        this.company = company;

    }


    public Integer getVolId() {
        return volId;
    }

    public void setVolId(Integer volId) {
        this.volId = volId;
    }

    public String getDesti() {
        return desti;
    }

    public void setDesti(String desti) {
        this.desti = desti;
    }

    public String getCompany() {
        return company;
    }

    public void setCompany(String company) {
        this.company = company;
    }

    public List<Answer> getAnswers() {
        return answers;
    }

    public void setAnswers(List<Answer> answers) {
        this.answers = answers;
    }


}
```

Comentari:

10

Puntuació 1,00 sobre 1,00

Completa

Com puc modificar el pom.xml per no haver de modificar moltes vegades el valor de la versió de les dependències de Spring?

Fent servir un placeholder "${spring.version}" configurant el seu valor en el block de propietats:

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<springframework.version>4.3.4.RELEASE</springframework.version>

<mysql.connector.version>5.1.40</mysql.connector.version>

<junit.version>4.12</junit.version>

<mockito.version>1.10.19</mockito.version>

<h2.version>1.4.190</h2.version>

</properties>

<dependencies>

...

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>**${springframework.version}**</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

<version>**${springframework.version}**</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>**${springframework.version}**</version>

...

</dependencies>

Comentari:

11

Puntuació 0,50 sobre 0,50

Correcte

Només una d'aquestes afirmacions és certa

Trieu-ne una:

- a. Hibernate s'encarrega d'injectar els objectes d'entitat que utilizeu en una aplicació mitjançant l'anotació @Autowired
- b. Si escriviu correctament les entitats d'una aplicació i les seves relacions, Hibernate pot crear les taules a la BD encara que no existeixin ✔
- c. Una relació OneToMany es tradueix en una List de l'entitat amb la que es relaciona una entitat en qüestió
- d. Hibernate obliga a carregar les dades un cop configurem la connexió

La resposta correcta és: Si escriviu correctament les entitats d'una aplicació i les seves relacions, Hibernate pot crear les taules a la BD encara que no existeixin

12

Puntuació 0,50 sobre 0,50

Correcte

Selecciona les respostes correctes

Trieu-ne una o més:

- a. Una connexió Eager requereix mantenir la sessió oberta en cada petició per completar les dades ✘
- b. La connexió Lazy necessita que la sessió es mantingui oberta per completar les dades ✔
- c. Eager recupera totes les dades al moment i Lazy només quan fa falta ✔
- d. Lazy recupera totes les dades cada petició i Eager només les necessàries

La resposta és correcta.

Les respostes correctes són: Eager recupera totes les dades al moment i Lazy només quan fa falta, La connexió Lazy necessita que la sessió es mantingui oberta per completar les dades

◀ MATERIALS. U4. APARTAT 1. ACCÉS A DADES AMB JDBC.

Salta a...

MATERIALS. U4. APARTAT 2. ACCÉS A DADES AMB JAVA ENTERPRISE EDITION ▶