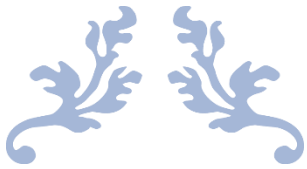


PRÁCTICA BASE DE DATOS

SAÚL MELLADO Y ENEKO REBOLLO

Contenido

Paquete Model

Paquete Pojo

En este paquete tenemos nuestras clases pojo.

Clase Commit

```java
@Data
public class Commit {
    private String id;
    private String titulo;
    private String mensaje;
    private String fecha;
    private String id_repositorio;
    private String id_proyecto;
    private String id_autor;
    private String id_issue;

    public Commit(String id, String titulo, String mensaje, String fecha, String id_repositorio, String id_proyecto, String id_autor, String id_issue) {
        this.id = id;
        this.titulo = titulo;
        this.mensaje = mensaje;
        this.fecha = fecha;
        this.id_repositorio = id_repositorio;
        this.id_proyecto = id_proyecto;
        this.id_autor = id_autor;
        this.id_issue = id_issue;
    }

    public Commit() {
    }
}
```

Clase Departamento

```java
@Data
public class Departamento {
    private String id;
    private String nombre;
    private String id_jefe;
    private double presupuesto;

    public Departamento(String id, String nombre, String id_jefe, double presupuesto) {
        this.id = id;
        this.nombre = nombre;
        this.id_jefe = id_jefe;
        this.presupuesto = presupuesto;
    }

    public Departamento() {
    }
}
```

Clase Issue

```java
@Data
public class Issue {
    private String id;
    private String titulo;
    private String texto;
    private String fecha;
    private String id_proyecto;
    private String id_repositorio;
    private boolean solucionado;

    public Issue(String id, String titulo, String texto, String fecha, String id_proyecto, String id_repositorio, boolean solucionado) {
        this.id = id;
        this.titulo = titulo;
        this.texto = texto;
        this.fecha = fecha;
        this.id_proyecto = id_proyecto;
        this.id_repositorio = id_repositorio;
        this.solucionado = solucionado;
    }

    public Issue() {

    }
}
```

Clase Programador

```java
@Data
@AllArgsConstructor
public class Programador {
    private String id;
    private String nombre;
    private String alta;
    private Double salario;
    private String id_departamento;
    private String tecnologias;

    public Programador(String id, String nombre, String alta, Double salario) {
        this.id = id;
        this.nombre = nombre;
        this.alta = alta;
        this.salario = salario;
    }

    public Programador() {
    }
}
```

Clase Proyecto

```java
@Data
public class Proyecto {
    private String id;
    private double presupuestoAnual;
    private String id_jefe;
    private String nombre;
    private String inicio;
    private String fin;
    private String id_repositorio;

    public Proyecto(String id, double presupuestoAnual, String id_jefe, String nombre, String inicio, String fin, String id_repositorio) {
        this.id = id;
        this.presupuestoAnual = presupuestoAnual;
        this.id_jefe = id_jefe;
        this.nombre = nombre;
        this.inicio = inicio;
        this.fin = fin;
        this.id_repositorio = id_repositorio;
    }

    public Proyecto() {
    }

    public Proyecto(String id, double presupuestoAnual, String nombre, String inicio, String fin) {
        this.id = id;
        this.presupuestoAnual = presupuestoAnual;
        this.nombre = nombre;
        this.inicio = inicio;
        this.fin = fin;
    }
}
```

Clase Repositorio

```java
@Data
public class Repositorio {
    private String id;
    private String nombre;
    private String fecha;
    private String id_proyecto;

    public Repositorio(String id, String nombre, String fecha, String id_proyecto) {
        this.id = id;
        this.nombre = nombre;
        this.fecha = fecha;
        this.id_proyecto = id_proyecto;
    }

    public Repositorio() {
    }

    public Repositorio(String id, String nombre, String fecha) {
        this.id = id;
        this.nombre = nombre;
        this.fecha = fecha;
    }
}
```

Paquete PojoDTO

En este paquete tenemos nuestros dto

Clase CommitDTO

```java
@Data
@Builder
public class CommitDTO {
    private String id;
    private String titulo;
    private String mensaje;
    private String fecha;
    private Repositorio repo;
    private Proyecto proyect;
    private Programador autor;
    private Issue issue;

}
```

Clase DepartamentoDTO

```java
@Data
@Builder
public class DepartamentoDTO {
    private String id;
    private String nombre;
    private Programador jefe;
    private double presupuesto;
    List<Proyecto> proyectosTerminados;
    List<Proyecto> proyectosDesarrollo;
    List<Programador> historicoJefes;

}
```

Clase IssueDTO

```java
@Data
@Builder
public class IssueDTO {
    private String id;
    private String titulo;
    private String texto;
    private String fecha;
    private Proyecto poyect;
    private Repositorio repo;
    private boolean solucionado;
    List<Programador> solucionadores;
}
```

Clase ProgramadorDTO

```java
@Data
@Builder
public class ProgramadorDTO {
    private String id_programador;
    private String nombre;
    private String alta;
    private Departamento trabajo;
    private List<Proyecto> proyectos;
    private List<Commit> commits;
    private List<Issue> issues;
    private List<String> tecnologias;
    private double salario;
}
```

Clase ProyectoDTO

```java
@Data
@Builder
public class ProyectoDTO {
    private String id;
    private double presupuestoAnual;
    private Programador jefe;
    private String nombre;
    private String inicio;
    private String fin;
    private Repositorio repo;
    List<String> tecnologias;
}
```

Clase RepositorioDTO

```java
@Data
@Builder
public class RepositorioDTO {
    private String id;
    private String nombre;
    private String fecha;
    private Proyecto proyect;
    List<Commit> commits;
    List<Issue> issues;
}
```

Paquete Driver

Aquí tenemos nuestro driver de la base de datos.

Definimos las variables que vamos a utilizar

```
public class SQLiteDriver {

    private String ruta;
    private static SQLiteDriver controller;
    @NonNull
    private String serverUrl;
    @NonNull
    private String serverPort;
    @NonNull
    private String dataBaseName;
    @NonNull
    private String user;
    @NonNull
    private String password;
    @NonNull
    private String jdbcDriver;
    @NonNull
    private Connection connection;
    @NonNull
    private PreparedStatement preparedStatement;
```

El metodo initConfig inicializa la base de datos con una configuración predeterminada.

El metodo open abre la conexión del servidor con la base de datos. En nuestro caso como estamos utilizando sqlite le metemos el driver correspondiente.

```
private void initConfig() {
    serverUrl = "localhost";
    serverPort = "3306";
    dataBaseName = "accessData";
    jdbcDriver = "org.sqlite.JDBC";
    user = "accessData";
    password = "accessData1234";
}

/**
 * Open the conexion of the server with the database
 *
 * @throws SQLException Server not accessible due to connection problems or incorrect access data
 */
public void open() throws SQLException {
    String url = "jdbc:sqlite:"+this.ruta;
    connection = DriverManager.getConnection(url, user, password);
}
```

Creamos dos consultas preparadas a la base de datos de tipo select con parámetros opcionales si son necesarios.

```java
/**
 * Performs a query to the database in a prepared way
 * @param querySQL select SQL query
 * @param params   params of the query
 * @return ResultSet of the query
 * @throws SQLException The query could not be performed or the table does not exist
 */
private ResultSet executeQuery(@NonNull String querySQL, Object... params) throws SQLException {
    preparedStatement = connection.prepareStatement(querySQL);
    for (int i = 0; i < params.length; i++) {
        preparedStatement.setObject( parameterIndex: i + 1, params[i]);
    }
    return preparedStatement.executeQuery();
}

/**
 * Performs a select query to the database in a parepared way with the optional params
 * @param querySQL select SQL query
 * @param params   params of the query
 * @return ResultSet of the query
 * @throws SQLException The query could not be performed or the table does not exist
 */
public Optional<ResultSet> select(@NonNull String querySQL, Object... params) throws SQLException {
    return Optional.of(executeQuery(querySQL, params));
}
```

Creamos otra consulta de tipo select en la que le establecemos el desplazamiento y el numero de registros que tiene.

```java
public Optional<ResultSet> select(@NonNull String querySQL, int limit, int offset, Object... params) throws SQLException {
    String query = querySQL + " LIMIT " + limit + " OFFSET " + offset;
    return Optional.of(executeQuery(query, params));
}
```

Creamos también consultas para insertar , actualizar y eliminar registros de nuestra base de datos.

```java
public Optional<ResultSet> insert(@NonNull String insertSQL, Object... params) throws SQLException {
    preparedStatement = connection.prepareStatement(insertSQL, preparedStatement.RETURN_GENERATED_KEYS);
    for (int i = 0; i < params.length; i++) {
        preparedStatement.setObject( parameterIndex: i + 1, params[i]);
    }
    preparedStatement.executeUpdate();
    return Optional.of(preparedStatement.getGeneratedKeys());
}

/**
 * Performs an update query to the database in a parepared way with the optional params
 * @param updateSQL update SQL query
 * @param params params of the query
 * @return number of updated records
 * @throws SQLException table does not exist or operation could not be performed
 */
public int update(@NonNull String updateSQL, Object... params) throws SQLException {
    return updateQuery(updateSQL, params);
}

/**
 * Performs a delete query to the database in a parepared way with the optional params
 * @param deleteSQL delete SQL query
 * @param params params of the query
 * @return number of deleted records
 * @throws SQLException table does not exist or operation could not be performed
 */
public int delete(@NonNull String deleteSQL, Object... params) throws SQLException {
    return updateQuery(deleteSQL, params);
}
```

Por último creamos otra consulta de update pero en este caso preparada utilizando prepared Statement.

```java
private int updateQuery(@NonNull String genericSQL, Object... params) throws SQLException {
    preparedStatement = connection.prepareStatement(genericSQL);
    for (int i = 0; i < params.length; i++) {
        preparedStatement.setObject( parameterIndex: i + 1, params[i]);
    }
    return preparedStatement.executeUpdate();
}
```

Paquete Mapper

En esta clase creamos nuestros Mapper.

Clase RepositoryMapper

Creamos un metodo para crear Programadores a partir de valores pasados por parámetros

```
public Programador datosToPrgramadorPOJO(String id, String nombre, String alta, double salario){
    Programador returner = new Programador();

    returner.setId(id);
    returner.setNombre(nombre);
    returner.setAlta(alta);
    returner.setSalario(salario);

    return returner;
}
```

Creamos un metodo para crear Commit a partir de valores pasados por parámetros

```
public Commit datosToCommitPOJO(String id, String titulo, String mensaje, String fecha, String idRepo, String idProyect, String idAutor, String idIssue){
    Commit returner = new Commit();

    returner.setId(id);
    returner.setTitulo(titulo);
    returner.setMensaje(mensaje);
    returner.setFecha(fecha);
    returner.setId_repositorio(idRepo);
    returner.setId_proyecto(idProyect);
    returner.setId_autor(idAutor);
    returner.setId_issue(idIssue);

    return returner;
```

Creamos un metodo para crear Departamento a partir de valores pasados por parámetros

```
public Departamento datosToDepartamentoPOJO(String id, String nombre, String idJefe,double presupuesto){
    Departamento returner = new Departamento();

    returner.setId(id);
    returner.setNombre(nombre);
    returner.setId_jefe(idJefe);
    returner.setPresupuesto(presupuesto);

    return returner;
}
```

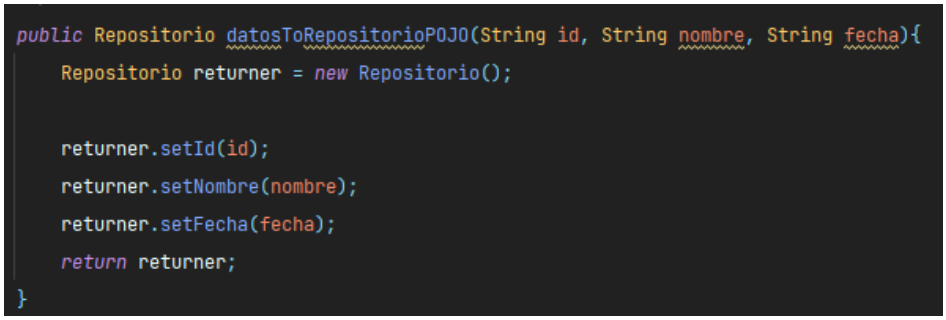
Creamos un metodo para crear Departamento a partir de valores pasados por parámetros

```
public Issue datosToRepositorioPOJO(String id, String titulo, String texto, String fecha, String idProyecto, String idRepo, boolean solucionado){
    Issue returner = new Issue();

    returner.setId(id);
    returner.setTitulo(titulo);
    returner.setTexto(texto);
    returner.setFecha(fecha);
    returner.setId_proyecto(idProyecto);
    returner.setId_repositorio(idRepo);
    returner.setSolucionado(solucionado);

    return returner;
}
```

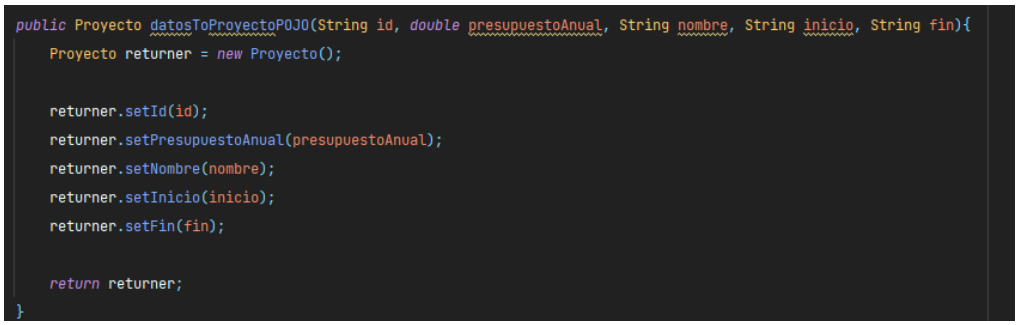
Creamos un metodo para crear Repositorio a partir de valores pasados por parámetros



```
public Repositorio datosToRepositorioPOJO(String id, String nombre, String fecha){
    Repositorio returner = new Repositorio();

    returner.setId(id);
    returner.setNombre(nombre);
    returner.setFecha(fecha);
    return returner;
}
```

Creamos un metodo para crear Proyecto a partir de valores pasados por parámetros



```
public Proyecto datosToProyectoPOJO(String id, double presupuestoAnual, String nombre, String inicio, String fin){
    Proyecto returner = new Proyecto();

    returner.setId(id);
    returner.setPresupuestoAnual(presupuestoAnual);
    returner.setNombre(nombre);
    returner.setInicio(inicio);
    returner.setFin(fin);

    return returner;
}
```

Clase CommitMapper

Creamos un método en nuestro mapper que hace pasa de pojo a dto.

```
public class CommitMapper {
    /**
     * Creation of the method that introduces the information from th
     * @param commit
     * @return builder of CommitDTO
     */
    public CommitDTO fromPojo(Commit commit){
        return CommitDTO.builder()
                .id(commit.getId())
                .titulo(commit.getTitulo())
                .mensaje(commit.getMensaje())
                .fecha(commit.getFecha())
                .repo(getRepo(commit.getId_repositorio()))
                .proyect(getProject(commit.getId_proyecto()))
                .autor(getProgramer(commit.getId_autor()))
                .issue(getIssue(commit.getId_issue()))
                .build();
    }
```

Este método llama a estos otros métodos para sacar la id del repositorio, del proyecto, de los programadores y de las issues.

```
private Repositorio getRepo(String id){
    return RepositorioRepository.getInstance().getRepositoriosList().stream().filter(x→ Objects.equals(x.getId(), id)).collect(Collectors.toList()).get(0);
}

private Proyecto getProject(String id){
    return ProyectoRepository.getInstance().getProyectosList().stream().filter(x→ Objects.equals(x.getId(), id)).collect(Collectors.toList()).get(0);
}

private Programador getProgramer(String id){
    return ProgramadorRepository.getInstance().getProgramadoresList().stream().filter(x→ Objects.equals(x.getId(), id)).collect(Collectors.toList()).get(0);
}

private Issue getIssue(String id){
    return IssueRepository.getInstance().getIssuesList().stream().filter(x→ Objects.equals(x.getId(), id)).collect(Collectors.toList()).get(0);
}
```

Clase DepartamentoMapper

Creamos un método en nuestro mapper que hace pasa de pojo a dto.

```
public class DepartamentoMapper {
    /**
     * Creation of the method that introduces the information from the class Dep
     * @param departamento
     * @return builder of DepartamentoDTO
     */
    public DepartamentoDTO fromPojo(Departamento departamento){
        return DepartamentoDTO.builder()
                .id(departamento.getId())
                .nombre(departamento.getNombre())
                .jefe(getJefe(departamento.getId_jefe()))
                .presupuesto(departamento.getPresupuesto())
                .proyectosTerminados(getTerminados(departamento.getId()))
                .proyectosDesarrollo(getEnDesarrollo(departamento.getId()))
                .build();
    }
```

Este método llama a estos otros métodos para sacar la id del jefe, la lista de proyectos termnados, y en desarrollo.

```
private Programador getJefe(String id){
    return ProgramadorRepository.getInstance().getProgramadoresList().stream().filter(x→ Objects.equals(x.getId(), id)).collect(Collectors.toList()).get(0);
}

private List<Proyecto> getTerminados(String id){
    return ProyectoRepository.getInstance().getProyectosList().stream().filter(x→ Objects.equals(x.getId(), id) && x.isFinalizado()).collect(Collectors.toList());
}

private List<Proyecto> getEnDesarrollo(String id){
    return ProyectoRepository.getInstance().getProyectosList().stream().filter(x→ Objects.equals(x.getId(), id) && x.isFinalizado()==false).collect(Collectors.toList())
}
```

Clase IssueMapper

Creamos un método en nuestro mapper que hace pasa de pojo a dto.

```java
public class IssueMapper {
    /**
     * Creation of the method that introduces the information from the class I
     * @param issue
     * @return builder of IssueDTO
     */
    public IssueDTO fromPojo(Issue issue){
        return IssueDTO.builder()
                .id(issue.getId())
                .titulo(issue.getTitulo())
                .texto(issue.getTexto())
                .fecha(issue.getFecha())
                .poyect(getProject(issue.getId_proyecto()))
                .repo(getRepo(issue.getId_repositorio()))
                .solucionado(issue.isSolucionado())
                .build();
    }
```

Este método llama a estos otros métodos para sacar la id del proyecto, y el repositorio.

```java
private Proyecto getProject(String id){
    return ProyectoRepository.getInstance().getProyectosList().stream().filter(x→ Objects.equals(x.getId(), id)).collect(Collectors.toList()).get(0);
}

private Repositorio getRepo(String id){
    return RepositorioRepository.getInstance().getRepositoriosList().stream().filter(x→ Objects.equals(x.getId(), id)).collect(Collectors.toList()).get(0);
}
```

Clase ProgramadorMapper

Clase ProyectoMapper

Clase RepositorioMapper

Paquete Repository

En este paquete creamos todos nuestros repositorios.

Clase CommitRepository

En esta clase creamos todas nuestras CRUD

```
public List<Commit> selectAll() throws SQLException {
    List<Commit> returner = new ArrayList<>();
    String query = "select * from commits";

    driver.open();
    Optional<ResultSet> rs = driver.select(query);
    while (rs.get().next()) {

        returner.add(rm.datosToCommitPOJO(rs.get().getString( s: "id"), rs.get().getString( s: "titulo"), rs.get().getString( s: "mensaje"),
                rs.get().getString( s: "fecha"), rs.get().getString( s: "idRepo"),rs.get().getString( s: "idProyect"),
                rs.get().getString( s: "idAutor"), rs.get().getString( s: "idIssue")));
    }

    driver.close();

    commitsList = returner;
    return returner;
}
```

```
public Commit insert(Commit c) throws SQLException {
    Commit returner = null;
    String query = "insert into commits (id, titulo, mensaje, fecha, idRepo, idProyect, idAutor, idIssue) values (?,?,?,?,?,?,?,?)";

    driver.open();
    Optional<ResultSet> rs = driver.insert(query,c.getId(),c.getTitulo(),c.getMensaje(),c.getFecha(),c.getId_repositorio(),c.getId_proyecto(),c.getId_autor(),c.getId_is
    while(rs.get().next()){
        if(rs.get().getInt( i: 1)>0){
            returner = c;
        }else{
            returner = null;
        }
    }

    driver.close();
    return returner;
}
```

Este método es llamado por el método insert de los test.

```
public Commit insert(String id, String titulo, String mensaje, String fecha, String idRepo, String idProyect, String idAutor, String idIssue) throws SQLException {
    return insert(new Commit(id,titulo,mensaje,fecha,idRepo, idProyect, idAutor, idIssue));
}
```

```
public String update(Commit c) throws SQLException {
    String query = "update commits set titulo=?, mensaje=?, fecha=?, idRepo=?, idProyect=?, idAutor=?, idIssue=? where id=?";

    driver.open();
    int rs = driver.update(query,c.getTitulo(),c.getMensaje(),c.getFecha(),c.getId_repositorio(),c.getId_proyecto(),c.getId_autor(),
            c.getId_issue(),c.getId());
    driver.close();

    if(rs==0){
        return null;
    }

    commitsList.removeIf(x-> Objects.equals(x.getId(), c.getId()));
    commitsList.add(c);
    return c.getId();
}
```

```java
public String delete(String id) throws SQLException {
    String query = "delete from commits where id=?";

    driver.open();
    int rs = driver.delete(query,id);
    driver.close();

    if(rs==0){
        return null;
    }

    commitsList.removeIf(x-> Objects.equals(x.getId(), id));
    return id;
}
```

Clase DepartamentoRepository

En esta clase creamos todas nuestras CRUD

```java
public List<Departamento> selectAll() throws SQLException {
    List<Departamento> returner = new ArrayList<>();
    String query = "select * from departamento";

    driver.open();
    Optional<ResultSet> rs = driver.select(query);

    while (rs.get().next()) {

        returner.add(rm.datosToDepartamentoPOJO(rs.get().getString( s: "id"),
                rs.get().getString( s: "nombre"), rs.get().getString( s: "idJefe"),
                rs.get().getDouble( s: "presupuesto")));
    }
    driver.close();

    departamentosList = returner;
    return returner;
}
```

```java
public Departamento insert(Departamento d) throws SQLException {
    Departamento returner = null;
    String query = "insert into departamento (id, nombre, idJefe, presupuesto) values (?,?,?,?)";

    driver.open();
    Optional<ResultSet> rs = driver.insert(query,d.getId(),d.getNombre(),d.getId_jefe(),d.getPresupuesto());
    while(rs.get().next()){
        if(rs.get().getInt( i: 1)>0){
            returner = d;
        }else{
            returner = null;
        }
    }
    driver.close();

    return returner;
}
```

```java
public Departamento insert(String id, String nombre, String idJefe, double presupuesto) throws SQLException {
    return insert(new Departamento(id,nombre,idJefe,presupuesto));
}
```

```java
public String update(Departamento d) throws SQLException {
    String query = "update departamento set nombre=?, idJefe=?, presupuesto=? where id=?";

    driver.open();
    int rs = driver.update(query,d.getNombre(),d.getId_jefe(),d.getPresupuesto(),d.getId());
    driver.close();

    if(rs==0){
        return null;
    }

    departamentosList.removeIf(x→ Objects.equals(x.getId(), d.getId()));
    departamentosList.add(d);
    return d.getId();
}
```

```java
public String delete(String id) throws SQLException {
    String query = "delete from departamento where id=?";

    driver.open();
    int rs = driver.delete(query,id);
    driver.close();

    if(rs==0){
        return null;
    }

    departamentosList.removeIf(x-> Objects.equals(x.getId(), id));
    return id;
}
```

Clase IssueRepository

En esta clase creamos todas nuestras CRUD

```java
public List<Issue> selectAll() throws SQLException {
    List<Issue> returner = new ArrayList<>();
    String query = "select * from issue";

    driver.open();
    Optional<ResultSet> rs = driver.select(query);

    while (rs.get().next()) {

        returner.add(rm.datosToIssuePOJO(rs.get().getString( s: "id"),
                rs.get().getString( s: "titulo"), rs.get().getString( s: "texto"), rs.get().getString( s: "fecha"),
                rs.get().getString( s: "idProyecto"), rs.get().getString( s: "idRepo"),rs.get().getBoolean( s: "solucionado")));
    }
    driver.close();

    issuesList = returner;
    return returner;
}
```

```java
public Issue insert(Issue i) throws SQLException {
    Issue returner = null;
    String query = "insert into issue (id, titulo, texto, fecha, idProyecto, idRepo, solucionado) values (?,?,?,?,?,?,?)";

    driver.open();
    Optional<ResultSet> rs = driver.insert(query,i.getId(),i.getTitulo(),i.getTexto(),i.getFecha(),i.getId_proyecto(),i.getId_repositorio(),i.isSolucionado());
    while(rs.get().next()){
        if(rs.get().getInt( i: 1)>0){
            returner = i;
        }else{
            returner = null;
        }
    }
    driver.close();

    return returner;
}
```

```java
public Issue insert(String id,String titulo,String texto,String fecha,String idProyecto,String idRepo,boolean solucionado) throws SQLException {
    return insert(new Issue(id,titulo, texto, fecha, idProyecto, idRepo, solucionado));
}
```

```java
public String update(Issue i) throws SQLException {
    String query = "update issue set titulo=?, texto=?, fecha=?, idProyecto=?, idRepo=?, solucionado=? where id=?";

    driver.open();
    int rs = driver.update(query,i.getTitulo(),i.getTexto(),i.getFecha(),i.getId_proyecto(),i.getId_repositorio(),i.isSolucionado(),i.getId());
    driver.close();

    if(rs==0){
        return null;
    }

    issuesList.removeIf(x-> Objects.equals(x.getId(), i.getId()));
    issuesList.add(i);
    return i.getId();
}
```

```java
public String delete(String id) throws SQLException {
    String query = "delete from issue where id=?";

    driver.open();
    int rs = driver.delete(query,id);
    driver.close();

    if(rs==0){
        return null;
    }

    issuesList.removeIf(x-> Objects.equals(x.getId(), id));
    return id;
}
```

Clase ProgramadorRepository

En esta clase creamos todas nuestras CRUD

```java
public List<Programador> selectAll() throws SQLException {
    List<Programador> returner = new ArrayList<>();
    String query = "select * from programador";

    driver.open();
    Optional<ResultSet> rs = driver.select(query);

    while (rs.get().next()) {

        returner.add(rw.datosToPrgramadorPOJO(rs.get().getString( s: "id"),
                rs.get().getString( s: "nombre"), rs.get().getString( s: "alta"),
                rs.get().getDouble( s: "salario")));
    }
    driver.close();

    programadoresList = returner;
    return returner;
}
```

```java
public Programador insert(Programador p) throws SQLException {
    Programador returner = null;
    String query = "insert into programador (id, nombre, alta, salario) values (?,?,?,?)";

    driver.open();
    Optional<ResultSet> rs = driver.insert(query,p.getId(),p.getNombre(),p.getAlta(),p.getSalario());
    while(rs.get().next()){
        //probar a crear variables sueltas y meterlas en el metodo de crear pojo
        if(rs.get().getInt( i: 1)>0){
            returner = p;
        }else {
            returner = null;
        }
    }
    driver.close();
    return returner;
}
```

```java
public Programador insert(String id, String nombre, String alta, double salario) throws SQLException {
    return insert(new Programador(id,nombre,alta,salario));
}
```

```java
public String update(Programador p) throws SQLException {
    String query = "update programador set nombre=?, alta=?, salario=? where id=?";

    driver.open();
    int rs = driver.update(query,p.getNombre(),p.getAlta(),p.getSalario(),p.getId());
    driver.close();

    if(rs==0){
        return null;
    }

    programadoresList.removeIf(x→ Objects.equals(x.getId(), p.getId()));
    programadoresList.add(p);
    return p.getId();
}
```

```java
public String delete(String id) throws SQLException {
    String query = "delete from programador where id=?";

    driver.open();
    int rs = driver.delete(query,id);
    driver.close();

    if(rs==0){
        return null;
    }

    programadoresList.removeIf(x-> Objects.equals(x.getId(), id));
    return id;
}
```

Clase ProyectoRepository

```java
public List<Proyecto> selectAll() throws SQLException {
    List<Proyecto> returner = new ArrayList<>();
    String query = "select * from proyecto";

    driver.open();
    Optional<ResultSet> rs = driver.select(query);

    while (rs.get().next()) {

        returner.add(rw.datosToProyectoPOJO(rs.get().getString( s: "id"),
                rs.get().getDouble( s: "presupuesto"),
                rs.get().getString( s: "nombre"),rs.get().getString( s: "inicio"),rs.get().getString( s: "fin"),rs.get().getBoolean( s: "finalizado")));
    }
    driver.close();

    proyectosList = returner;
    return returner;
}
```

```java
public Proyecto insert(Proyecto p) throws SQLException {
    Proyecto returner = null;
    String query = "insert into proyecto (id,presupuesto, idJefe, nombre, inicio, fin, idRepo, finalizado) values (?,?,?,?,?,?,?,?)";

    driver.open();
    Optional<ResultSet> rs = driver.insert(query,p.getId(),p.getPresupuestoAnual(),p.getId_jefe(),p.getNombre(),p.getInicio(),p.getFin(),p.getId_repositorio(),p.getFin
    while(rs.get().next()){
        if(rs.get().getInt( i: 1)>0){
            returner = p;
        }else {
            returner = null;
        }

    }
    driver.close();
    return returner;
}
```

```java
public Proyecto insert(String id, double presupuestoAnual, String idJefe,String nombre, String inicio, String fin,String idRepo, boolean finalizado) throws SQLException
    return insert(new Proyecto(id,presupuestoAnual,idJefe,nombre,inicio,fin,idRepo,finalizado));
}
```

```java
public String update(Proyecto p) throws SQLException {
    String query = "update proyecto set presupuesto=?, idJefe=?, nombre=?, inicio=?, fin=?, idRepo=?, finalizado=? where id=?";

    driver.open();
    int rs = driver.update(query,p.getPresupuestoAnual(),p.getId_jefe(),p.getNombre(),p.getInicio(),p.getFin(),p.getId_repositorio(),p.getFin(),p.getId());
    driver.close();

    if(rs==0){
        return null;
    }

    proyectosList.removeIf(x-> Objects.equals(x.getId(), p.getId()));
    proyectosList.add(p);
    return p.getId();
}
```

```java
public String delete(String id) throws SQLException {
    String query = "delete from proyecto where id=?";

    driver.open();
    int rs = driver.delete(query,id);
    driver.close();

    if(rs==0){
        return null;
    }

    proyectosList.removeIf(x-> Objects.equals(x.getId(), id));
    return id;
```

Clase RepositorioRepository

```java
public List<Repositorio> selectAll() throws SQLException {
    List<Repositorio> returner = new ArrayList<>();
    String query = "select * from repositorio";

    driver.open();
    Optional<ResultSet> rs = driver.select(query);

    while (rs.get().next()) {

        returner.add(rw.datosToRepositorioPOJO(rs.get().getString( s: "id"),
                rs.get().getString( s: "nombre"),
                rs.get().getString( s: "fecha")));
    }
    driver.close();

    repositoriosList = returner;
    return returner;
}
```

```java
public Repositorio insert(Repositorio r) throws SQLException {
    Repositorio returner = null;
    String query = "insert into repositorio (id, nombre, fecha, idProyecto) values (?,?,?,?)";

    driver.open();
    Optional<ResultSet> rs = driver.insert(query,r.getId(),r.getNombre(),r.getFecha(),r.getId_proyecto());
    while(rs.get().next()){
        if(rs.get().getInt( i: 1)>0){
            returner = r;
        }else{
            returner = null;
        }
    }
    driver.close();

    return returner;
}
```

```java
public Repositorio insert(String id, String nombre, String fecha,String idProject) throws SQLException {
    return insert(new Repositorio(id,nombre,fecha,idProject));
}
```

```java
public String update(Repositorio r) throws SQLException {
    String query = "update repositorio set nombre=?, fecha=?, idProyecto=? where id=?";

    driver.open();
    int rs = driver.update(query,r.getNombre(),r.getFecha(),r.getId_proyecto(),r.getId());
    driver.close();

    if(rs==0){
        return null;
    }

    repositoriosList.removeIf(x→ Objects.equals(x.getId(), r.getId()));
    repositoriosList.add(r);
    return r.getId();
}
```

```java
public String delete(String id) throws SQLException {
    String query = "delete from repositorio where id=?";

    driver.open();
    int rs = driver.delete(query,id);
    driver.close();

    if(rs==0){
        return null;
    }

    repositoriosList.removeIf(x→ Objects.equals(x.getId(), id));
    return id;
}
```

Paquete Controller

Todas las clases

En este paquete nos creamos varias clases y todas hacen lo mismo por lo que explicaremos una de ellas.

Este metodo lo que hace es llamar al metodo insertar un nuevo repositorio (anteriormente explicado) si no existe en la lista de repositorios ya creada, además mediante un boolean decimos si queremos que nos saque formato json o xml.

```
public void newRepositorio(Repositorio c, boolean JSon) throws SQLException, JAXBException {
    repositorio.selectAll();
    if(!repositorio.getRepositoriosList().contains(c)){
        Repositorio ans = repositorio.insert(c);

        if(ans≠null){
            if(JSon){
                export.toJSon(ans);
            }else{
                export.toXML(ans, tipo: "repo");
            }
        }else{
            if(JSon){
                export.toJSon( o: "Hubo un problema al aniadir el Repositorio");
            }
            else export.toXML( o: "Hubo un problema al aniadir el Repositorio", tipo: "error");
        }
    }else{
        if(JSon){
            export.toJSon( o: "no se ha podido crear el Repositorio porque ya existe");
        }
        else export.toXML( o: "no se ha podido crear el Repositorio porque ya existe", tipo: "error");
    }
}
```

Este metodo lo que hace es llamar al metodo update (anteriormente comentado) y si existe en la lista de repositorios lo actualiza y nos lo saca en formato json o xml.

```
public void updateRepositorio(Repositorio c, boolean JSon) throws SQLException, JAXBException {
    repositorio.selectAll();
    if(repositorio.getRepositoriosList().contains(c)){
        String ans = repositorio.update(c);

        if(ans≠null){
            if(JSon){
                export.toJSon(c);
            }else{
                export.toXML(c, tipo: "repo");
            }
        }else{
            if(JSon){
                export.toJSon( o: "Hubo un problema al actualizar el Repositorio");
            }
            else export.toXML( o: "Hubo un problema al actualizar el Repositorio", tipo: "error");
        }
    }else{
        if(JSon){
            export.toJSon( o: "no se ha podido actualizar el Repositorio porque no existe");
        }
        else export.toXML( o: "no se ha podido actualizar el Repositorio porque no existe", tipo: "error");
    }
}
```

En este metodo lo que hacemos es borrar un repositorio si la id coincide con la misma que hay en la lista de repositorios existentes. Saca el elemento borrado en json o xml.

```
public void deleteRepositorio(String id, boolean JSon) throws SQLException, JAXBException {
    repositorio.selectAll();
    if(repositorio.getRepositoriosList().stream().filter(x -> Objects.equals(x.getId(), id)).count() !=0){
        String ans = repositorio.delete(id);

        if(ans!=null){
            if(JSon){
                export.toJSon(id);
            }else{
                export.toXML(id, tipo: "repo");
            }
        }else{
            if(JSon){
                export.toJSon( o: "Hubo un problema al borrar el Repositorio");
            }
            else export.toXML( o: "Hubo un problema al borrar el Repositorio", tipo: "error");
        }
    }else{
        if(JSon){
            export.toJSon( o: "no se ha podido borrar el Repositorio porque no existe");
        }
        else export.toXML( o: "no se ha podido borrar el Repositorio porque no existe", tipo: "error");
    }
}
```

Por último, hacemos un select para listar todos los repositorios que haya en la lista y los saca con formato json o xml.

```
public void selectRepositorios(boolean JSon) throws JAXBException, SQLException {
    repositorio.selectAll();
    if(!repositorio.getRepositoriosList().isEmpty()){
        List<Repositorio> ans = repositorio.getRepositoriosList();

        if (ans!=null){
            if(JSon){
                export.toJSon(ans);
            }
            else export.toXML(ans, tipo: "repo");
        }else{
            if(JSon){
                export.toJSon( o: "Hubo un problema al leer los Repositorios");
            }
            else export.toXML( o: "Hubo un problema al leer los Repositorios", tipo: "error");
        }
    }else{
        if(JSon){
            export.toJSon( o: "no hay Repositorios guardados en la base de datos, puebe a aniadir uno primero");
        }
        else export.toXML( o: "no hay Repositorios guardados en la base de datos, puebe a aniadir uno primero", tipo: "error");
    }
}
```

Clase Export

En esta clase nos creamos los métodos que sacan el xml y el json.

```java
public void toXML(Object o,String tipo) throws JAXBException {
    JAXBContext jaxbContext=null;
    if(!tipo.equals("error")){
        JAXBLists.getInstance().fillLst(List.of(o));
        jaxbContext = JAXBContext.newInstance(JAXBLists.class);
    }
    else{
        ErrorObject.getInstance().initString((String) o);
        jaxbContext = JAXBContext.newInstance(ErrorObject.class);
    }

    Marshaller jaxbMarshaller = jaxbContext.createMarshaller();
    jaxbMarshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
    jaxbMarshaller.marshal(o, System.out);
}

/**
 * returns throughout the terminal in xml style of the object
 * @param o
 */
public void toJSon(Object o){
    final Gson prettyGson = new GsonBuilder().setPrettyPrinting().create();
    System.out.println(prettyGson.toJson(o));
}
```

Test Junit

Clase CommitRepositoryTest

Creamos un commit por defecto para comprobar el funcionamiento de los test. Establecemos el orden en el que se ejecutan.

```java
@DisplayName("CommitRepository tests")
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
public class CommitRepositoryTest {

    private Commit testObject = new Commit( id: "testId", titulo: "test", mensaje: "mensajeTest", fecha: "2222-22-22", id_repositorio: "idRepo",
            id_proyecto: "idProyect", id_autor: "idAutor", id_issue: "idIssue");
    private CommitRepository repositoryTest = CommitRepository.getInstance();
```

```java
@Test
@Order(1)
public void insertCommit(){
    try {
        Commit ans = repositoryTest.insert(testObject.getId(),testObject.getTitulo(),testObject.getMensaje(),testObject.getFecha(),testObject.getId_repositorio(),
                testObject.getId_proyecto(),testObject.getId_autor(), testObject.getId_issue());
        Assertions.assertEquals(testObject,ans);
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

@Test
@Order(2)
public void selectCommit(){
    try{
        List<Commit> ans = repositoryTest.selectAll();
        Assertions.assertEquals( expected: 1,ans.size());
    } catch (SQLException throwables) {
        throwables.printStackTrace();
    }
}
```

```java
@Test
@Order(3)
public void updateCommit(){

    Commit alter = new Commit( id: "testId", titulo: "test", mensaje: "mensajeTestAunMasLargo", fecha: "2222-22-22", id_repositorio: "idRepo",
            id_proyecto: "idProyect", id_autor: "idAutor", id_issue: "idIssue");
    try{
        String ans = repositoryTest.update(alter);
        Assertions.assertEquals(alter.getId(),ans);
    } catch (SQLException throwables) {
        throwables.printStackTrace();
    }
}

@Test
@Order(4)
public void deleteCommit(){
    try{
        String ans = repositoryTest.delete(testObject);
        Assertions.assertEquals(testObject.getId(),ans);
    } catch (SQLException throwables) {
        throwables.printStackTrace();
    }
}
```

Clase DepartamentoRepositoryTest

Creamos un departamento por defecto para comprobar el funcionamiento de los test. Establecemos el orden en el que se ejecutan.

```java
@DisplayName("DepartamentoRepository test")
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
public class DepartamentoRepositoryTest {

    private Departamento testObject = new Departamento( id: "testId", nombre: "test", id_jefe: "testIdJefe", presupuesto: 445.6);
    private DepartamentoRepository repositoryTest = DepartamentoRepository.getInstance();
```

```java
@Test
@Order(1)
public void insertDepartamento(){
    try {
        Departamento ans = repositoryTest.insert(testObject.getId(),testObject.getNombre(),testObject.getId_jefe(), testObject.getPresupuesto());
        Assertions.assertEquals(testObject,ans);
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

@Test
@Order(2)
public void selectDepartamento(){
    try{
        List<Departamento> ans = repositoryTest.selectAll();
        Assertions.assertEquals( expected: 1,ans.size());
    } catch (SQLException throwables) {
        throwables.printStackTrace();
    }
}
```

```java
@Test
@Order(3)
public void updateDepartamento(){

    Departamento alter = new Departamento( id: "testId", nombre: "test", id_jefe: "testIdJefe", presupuesto: 78815.6);
    try{
        String ans = repositoryTest.update(alter);
        Assertions.assertEquals(alter.getId(),ans);
    } catch (SQLException throwables) {
        throwables.printStackTrace();
    }
}

@Test
@Order(4)
public void deleteDepartamento(){
    try{
        String ans = repositoryTest.delete(testObject);
        Assertions.assertEquals(testObject.getId(),ans);
    } catch (SQLException throwables) {
        throwables.printStackTrace();
    }
}
```

Clase IssueRepositoryTest

Creamos un issue por defecto para comprobar el funcionamiento de los test. Establecemos el orden en el que se ejecutan.

```java
@DisplayName("IssueRepository test")
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
public class IssueRepositoryTest {

    private Issue testObject = new Issue( id: "testId", titulo: "test", texto: "testText", fecha: "2222-22-22", id_proyecto: "testProyId", id_repositorio: "testRepoId", solucionado: true);
    private IssueRepository repositoryTest = IssueRepository.getInstance();
```

```java
@Test
@Order(1)
public void insertIssue(){
    try {
        Issue ans = repositoryTest.insert(testObject.getId(),testObject.getTitulo(),testObject.getTexto(),testObject.getFecha(),testObject.getId_proyecto(),
                testObject.getId_repositorio(),testObject.isSolucionado());
        Assertions.assertEquals(testObject,ans);
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

@Test
@Order(2)
public void selectIssue(){
    try{
        List<Issue> ans = repositoryTest.selectAll();
        Assertions.assertEquals( expected: 1,ans.size());
    } catch (SQLException throwables) {
        throwables.printStackTrace();
    }
}
```

```java
@Test
@Order(3)
public void updateIssue(){

    Issue alter = new Issue( id: "testId", titulo: "test", texto: "testTextAunMasLargo", fecha: "2222-22-22", id_proyecto: "testProyId", id_repositorio: "testRepoId", solucionado: false);
    try{
        String ans = repositoryTest.update(alter);
        Assertions.assertEquals(alter.getId(),ans);
    } catch (SQLException throwables) {
        throwables.printStackTrace();
    }
}

@Test
@Order(4)
public void deleteIssue(){
    try{
        String ans = repositoryTest.delete(testObject);
        Assertions.assertEquals(testObject.getId(),ans);
    } catch (SQLException throwables) {
        throwables.printStackTrace();
    }
```

Clase ProgramadorRepositoryTest

Creamos un programador por defecto para comprobar el funcionamiento de los test. Establecemos el orden en el que se ejecutan.

```java
@DisplayName("ProgramadorRepository test")
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
public class ProgramadorRepositoryTest {

    private Programador testObject = new Programador( id: "testId", nombre: "juanito", alta: "20202-22-22", salario: 234.5);
    private ProgramadorRepository repositoryTest = ProgramadorRepository.getInstance();
```

```java
@Test
@Order(1)
public void insertProgramador(){
    try {
        Programador ans = repositoryTest.insert(testObject.getId(),testObject.getNombre(),testObject.getAlta(), testObject.getSalario());
        Assertions.assertEquals(testObject,ans);
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

@Test
@Order(2)
public void selectProgramador(){
    try{
        List<Programador> ans = repositoryTest.selectAll();
        Assertions.assertEquals( expected: 1,ans.size());
    } catch (SQLException throwables) {
        throwables.printStackTrace();
    }
}
```

```java
@Test
@Order(3)
public void updateProgramador(){

    Programador alter = new Programador( id: "testId", nombre: "jorge", alta: "20202-22-22", salario: 778.7);
    try{
        String ans = repositoryTest.update(alter);
        Assertions.assertEquals(alter.getId(),ans);
    } catch (SQLException throwables) {
        throwables.printStackTrace();
    }
}

@Test
@Order(4)
public void deleteProgramador(){
    try{
        String ans = repositoryTest.delete(testObject);
        Assertions.assertEquals(testObject.getId(),ans);
    } catch (SQLException throwables) {
        throwables.printStackTrace();
    }
}
```

Clase ProyectoRepositoryTest

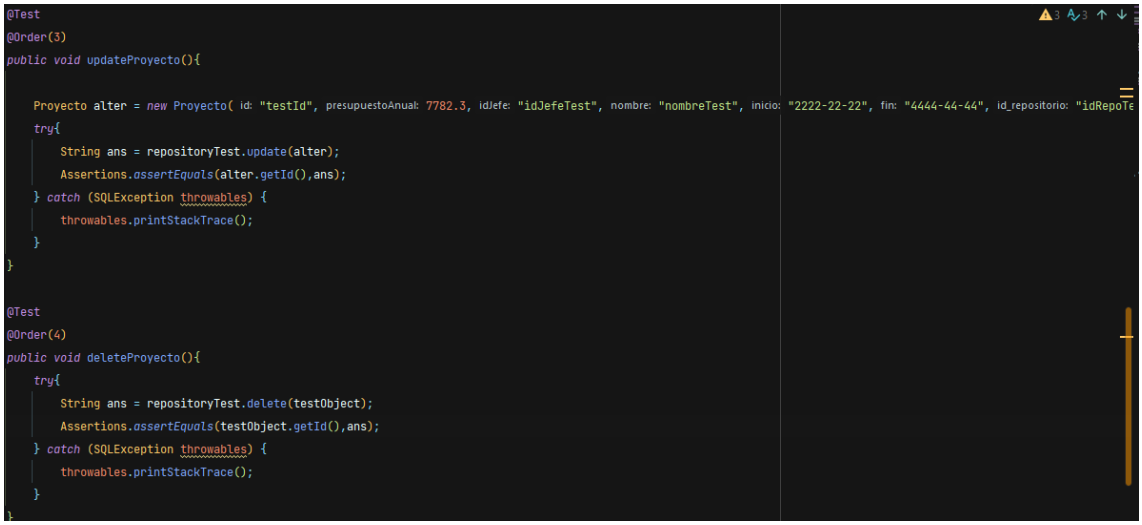
Creamos un proyecto por defecto para comprobar el funcionamiento de los test. Establecemos el orden en el que se ejecutan.

```java
@DisplayName("ProyectoRepository test")
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
public class ProyectoRepositoryTest {

    private Proyecto testObject = new Proyecto( id: "testId", presupuestoAnual: 4458.6, idJefe: "idJefeTest", nombre: "nombreTest", inicio: "2222-22-22", fin: "2222-22-22", id_repositorio
    private ProyectoRepository repositoryTest = ProyectoRepository.getInstance();
```

```java
@Test
@Order(1)
public void insertProyecto(){
    try {
        Proyecto ans = repositoryTest.insert(testObject.getId(),testObject.getPresupuestoAnual(),testObject.getId_jefe(),testObject.getNombre(), testObject.getInicio(),
                testObject.getFin(),testObject.getId_repositorio(),testObject.isFinalizado());
        Assertions.assertEquals(testObject,ans);
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

@Test
@Order(2)
public void selectProyecto(){
    try{
        List<Proyecto> ans = repositoryTest.selectAll();
        Assertions.assertEquals( expected: 1,ans.size());
    } catch (SQLException throwables) {
        throwables.printStackTrace();
    }
}
```



```java
@Test
@Order(3)
public void updateProyecto(){

    Proyecto alter = new Proyecto( id: "testId", presupuestoAnual: 7782.3, idJefe: "idJefeTest", nombre: "nombreTest", inicio: "2222-22-22", fin: "4444-44-44", id_repositorio: "idRepoTe
    try{
        String ans = repositoryTest.update(alter);
        Assertions.assertEquals(alter.getId(),ans);
    } catch (SQLException throwables) {
        throwables.printStackTrace();
    }
}

@Test
@Order(4)
public void deleteProyecto(){
    try{
        String ans = repositoryTest.delete(testObject);
        Assertions.assertEquals(testObject.getId(),ans);
    } catch (SQLException throwables) {
        throwables.printStackTrace();
    }
}
```

Clase RepositorioRepositoryTest

Creamos un repositorio por defecto para comprobar el funcionamiento de los test.
Establecemos el orden en el que se ejecutan.

```java
@DisplayName("RepositorioRepository test")
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
public class RepositorioRepositoryTest {

    private Repositorio testObject = new Repositorio( id: "testId", nombre: "testName", fecha: "2222-22-22", id_proyecto: "idProjectTest");
    private RepositorioRepository repositoryTest = RepositorioRepository.getInstance();
```

```java
@Test
@Order(1)
public void insertRepositorio(){
    try {
        Repositorio ans = repositoryTest.insert(testObject.getId(),testObject.getNombre(),testObject.getFecha(),testObject.getId_proyecto());
        Assertions.assertEquals(testObject,ans);
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

@Test
@Order(2)
public void selectRepositorio(){
    try{
        List<Repositorio> ans = repositoryTest.selectAll();
        Assertions.assertEquals( expected: 1,ans.size());
    } catch (SQLException throwables) {
        throwables.printStackTrace();
    }
}
```

```java
@Test
@Order(3)
public void updateRepositorio(){

    Repositorio alter = new Repositorio( id: "testId", nombre: "testNameAunMasLargo", fecha: "2222-22-22", id_proyecto: "idProjectTest");;
    try{
        String ans = repositoryTest.update(alter);
        Assertions.assertEquals(alter.getId(),ans);
    } catch (SQLException throwables) {
        throwables.printStackTrace();
    }
}

@Test
@Order(4)
public void deleteRepositorio(){
    try{
        String ans = repositoryTest.delete(testObject);
        Assertions.assertEquals(testObject.getId(),ans);
    } catch (SQLException throwables) {
        throwables.printStackTrace();
    }
}
```

JPA

Paquete Model

Todas las clases Pojo

Nos creamos nuestras clases pojo, encima de cada clase ponemos el @Entity y @Table para generar nuestra tabla.

```java
@Entity
@Table (name = "Commit")
public class Commit {
    private String id;
    private String titulo;
    private String mensaje;
    private String fecha;
    private String id_repositorio;
    private String id_proyecto;
    private String id_autor;
    private String id_issue;

    public Commit(String id, String titulo, String mensaje, String fecha, String id_repositorio, String id_proyecto, Stri
        this.id = id;
        this.titulo = titulo;
        this.mensaje = mensaje;
        this.fecha = fecha;
        this.id_repositorio = id_repositorio;
        this.id_proyecto = id_proyecto;
        this.id_autor = id_autor;
        this.id_issue = id_issue;
    }

    public Commit() {
    }
```

Luego, en cada getter añadimos el @Column para crear nuestras columnas dentro de la tabla y además ponemos el @Id en nuestro campo id para establecerle el identificador.

```
@Id
@Column (nullable = false)
public String getId() { return id; }

public void setId(String id) { this.id = id; }

@Column (nullable = false)
public String getTitulo() { return titulo; }

public void setTitulo(String titulo) { this.titulo = titulo; }
@Column (nullable = false)
public String getMensaje() { return mensaje; }

public void setMensaje(String mensaje) { this.mensaje = mensaje; }
@Column (nullable = false)
public String getFecha() { return fecha; }

public void setFecha(String fecha) { this.fecha = fecha; }
@Column (nullable = false)
public String getId_repositorio() { return id_repositorio; }

public void setId_repositorio(String id_repositorio) { this.id_repositorio = id_repositorio; }
@Column (nullable = false)
public String getId_proyecto() { return id_proyecto; }

public void setId_proyecto(String id_proyecto) { this.id_proyecto = id_proyecto; }
@Column (nullable = false)
public String getId_autor() { return id_autor; }

public void setId_autor(String id_autor) { this.id_autor = id_autor; }
@Column (nullable = false)
public String getId_issue() { return id_issue; }

public void setId_issue(String id_issue) { this.id_issue = id_issue; }
```

Paquete Repository

Todas las clases Repository

Nos creamos el entityManagerFactory, EntityManager y EntityTransaction, en createEntityManagerFactory metemos el mismo nombre que le asignamos en nuestro fichero persistence.

Creamos el metodo find all que simplemente nos saca la lista con todos los commits que tenemos mediante .createQuery y diciéndole de que clase queremos.

El metodo insertar simplemente mediante el metodo persist nos inserta el objeto en la tabla.

```java
public class CommitRepository {
    EntityManagerFactory entityManagerFactory = Persistence.createEntityManagerFactory( persistenceUnitName: "practica");
    EntityManager manager = entityManagerFactory.createEntityManager();
    EntityTransaction transaction = manager.getTransaction();

    /**
     * Creation of the method that search all the commits in the database
     * @return null and print the commit list
     */
    public List<Commit> findAll() {
        System.out.println("Listado de todos los Commits: ");
        List<Commit> commits = (List<Commit>) manager.createQuery( s: "FROM Commit ").getResultList();
        for (Commit e : commits) {
            System.out.println(e.toString());
        }
        return null;
    }

    /**
     * Creation of the method that insert the commit into the database
     * @param c
     * @return commit
     */
    public Commit insertar(Commit c) {
            manager.getTransaction().begin();
            manager.persist(c);
            manager.getTransaction().commit();
            return c;
    }
```

El metodo actualizar simplemente mediante el merge y luego diciéndole que campo queremos modificar nos actualiza el objeto que le hayamos metido.

El metodo borrar simplemente elimina el objeto mediante remove buscándolo por la id.

```
public Commit actualizar(Commit c){
        manager.getTransaction().begin();
        manager.merge(c);
        c.setMensaje("Mensaje Actualizado");
        manager.getTransaction().commit();
        System.out.println("Elemento Actualizado: "+c.toString());
        return c;
}

/**
 * Creation of the method that delete the commit of the database
 * @param c
 * @return commit
 */
public Commit borrar(Commit c){
        manager.getTransaction().begin();
        c = manager.find(Commit.class, c.getId());
        manager.remove(c);
        manager.getTransaction().commit();
        System.out.println("Elemento Borrado: "+ c.toString());
        return c;
}
```

Fichero Persistence.xml

Le establecemos el nombre y nos aseguramos de que sea el mismo que el de créateEntityManagerFactory, luego le agregamos el driver, la url de conexión y el dialecto. En nuestro caso hemos puesto créate-drop para que se borre toda la base de datos una vez finalizamos la ejecución.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence http://xmlns.jcp.org/xml/ns/persistence/persister
             version="2.2">
    <persistence-unit name="practica">
        <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
        <properties>
            <property name="hibernate.connection.url" value="jdbc:h2:~/test2"/>
            <property name="hibernate.connection.driver_class" value="org.h2.Driver"/>
            <property name="hibernate.dialect" value="org.hibernate.dialect.H2Dialect"/>
            <property name="hibernate.connection.user" value="sa"/>
            <property name="hibernate.connection.password" value=""/>
            <property name="hibernate.hbm2ddl.auto" value="create-drop"/>
            <property name="show_sql" value="true"/>
            <property name="hibernate.temp.use_jdbc_metadata_defaults" value="false"/>
        </properties>
    </persistence-unit>
</persistence>
```