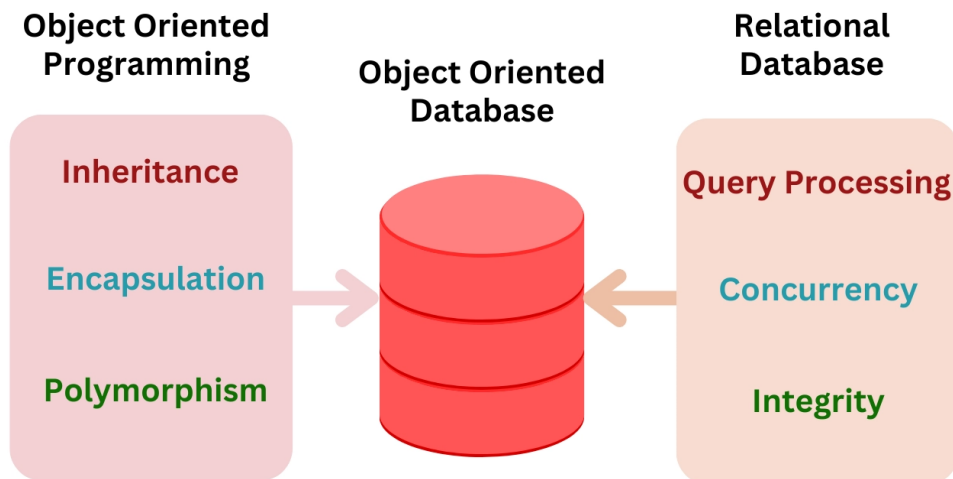


Taller UD11_1: Añadir ObjectDB a un proyecto IntelliJ (Maven)



1. **Que es ObjectDB**
 1. 1. Características clave de la base de datos **ObjectDB**
2. **Añadir ObjectDB mediante Maven**
3. **Tu primer proyecto con ObjectDB**
4. **Actividades**
5. **Fuentes de información**

1. Que es ObjectDB

ObjectDB es un potente sistema de gestión de bases de datos orientado a objetos (ODBMS). Es compacto, fiable, fácil de usar y extremadamente rápido. **ObjectDB** proporciona todos los servicios estándar de administración de bases de datos (almacenamiento y recuperación, transacciones, administración de bloqueos, procesamiento de consultas, etc.), pero de una manera que facilita el desarrollo y acelera las aplicaciones.

1.1. Características clave de la base de datos ObjectDB

- Sistema de gestión de bases de datos orientado a objetos (ODBMS) 100% puro Java.
- Sin API propietaria - administrado únicamente por API de Java estándar (JPA 2 / JDO 2).
- Extremadamente rápido: más rápido que cualquier otro producto JPA/JDO.
- Adecuado para archivos de bases de datos que van desde kilobytes hasta terabytes.
- Admite tanto el modo Cliente-Servidor como el modo Integrado.
- JAR único sin dependencias externas.
- La base de datos se almacena como un único archivo.
- Capacidades avanzadas de consulta e indexación.
- Efectivo en entornos multiusuario con mucha carga.
- Puede integrarse fácilmente en aplicaciones de cualquier tipo y tamaño.
- Probado con Tomcat, Jetty, GlassFish, JBoss y Spring.

ObjectDB se puede descargar y utilizar **sin coste (incluso comercialmente)** con la restricción de un máximo de **10 clases de entidad** y **un millón de objetos de entidad** por archivo de base de datos. Esto podría ser útil para proyectos pequeños, tareas académicas, evaluación y aprendizaje. **ObjectDB** es un software comercial y su uso sin estas restricciones [requiere la compra de una licencia](#).

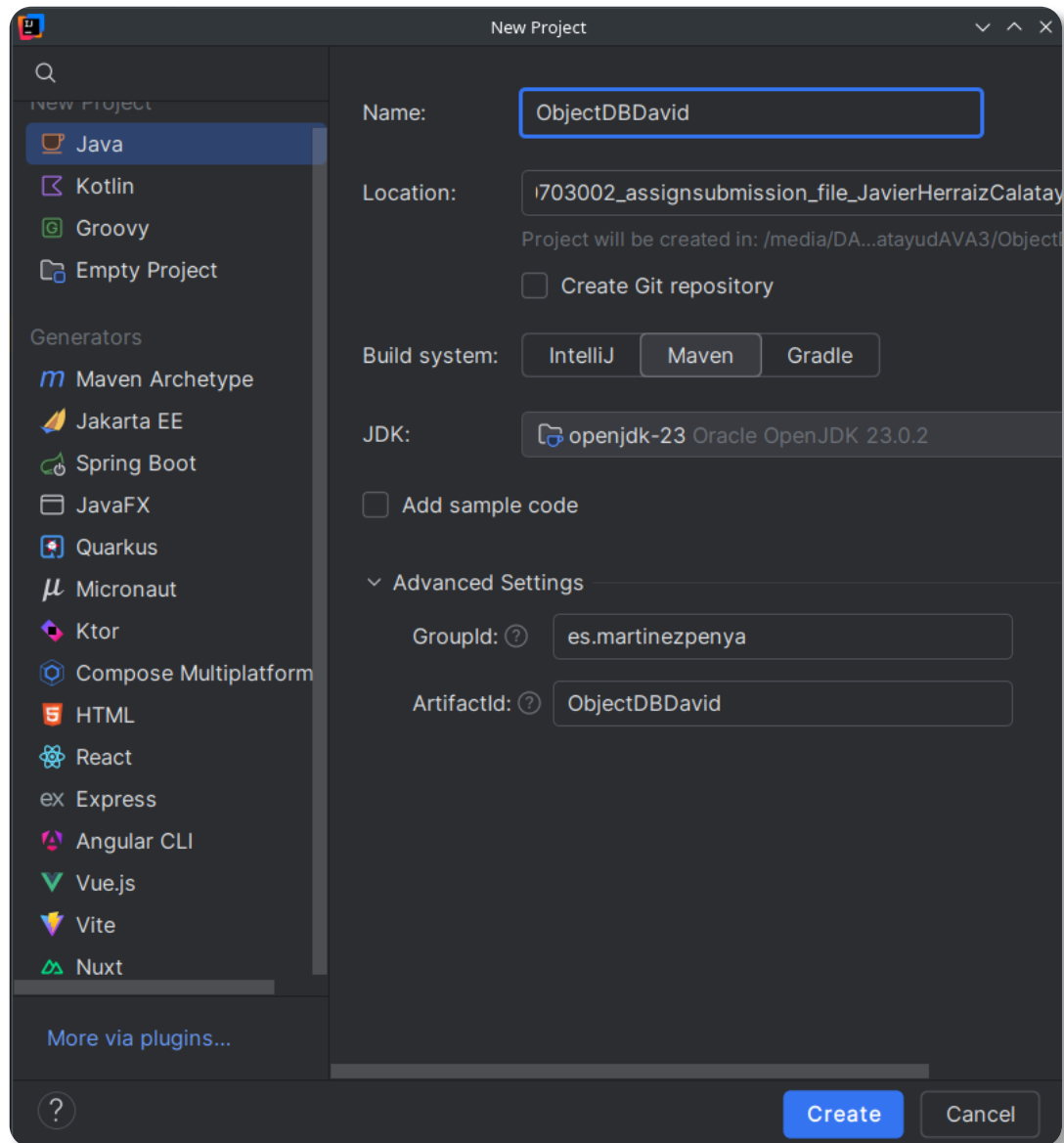
Se recomienda probar **ObjectDB** antes de comprar una licencia.

Más información sobre tipos de licencias: <https://www.objectdb.com/database/purchase>

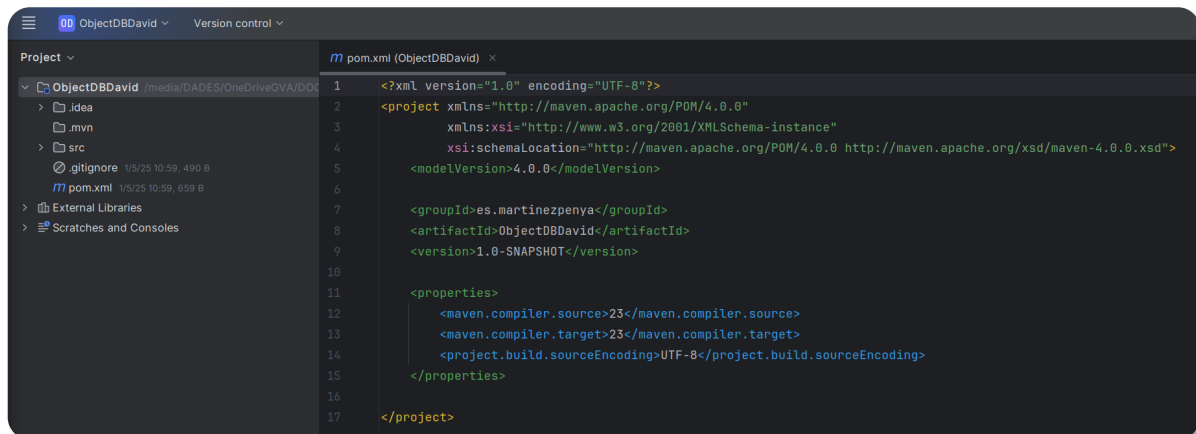
2. Añadir ObjectDB mediante Maven

1. Crear un Nuevo Proyecto con Maven:

- Al crear un nuevo proyecto en IntelliJ, selecciona "Maven" como sistema de construcción.

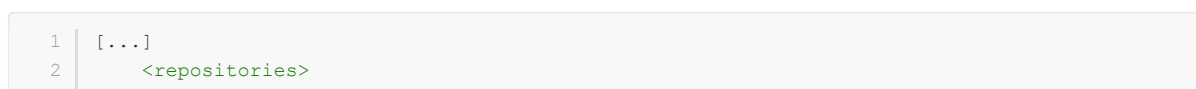


- IntelliJ generará la estructura del proyecto y el archivo `pom.xml`.



2. Agregar Dependencias:

- Abre el archivo `pom.xml` y agrega el repositorio y las dependencias necesarias.



```
3      <repository>
4          <id>objectdb</id>
5          <name>ObjectDB Repository</name>
6          <url>https://m2.objectdb.com</url>
7      </repository>
8  </repositories>
9
10     <dependencies>
11         <dependency>
12             <groupId>com.objectdb</groupId>
13             <artifactId>objectdb</artifactId>
14             <version>2.8.1</version>
15         </dependency>
16     </dependencies>
17     [...]
```

3. Actualizar Dependencias:

- Si cambias el archivo `pom.xml`, IntelliJ mostrará un botón para cargar los cambios:



3. Tu primer proyecto con ObjectDB

En la web del proyecto ObjectDB puedes descargar un tutorial con un proyecto ya creado para ver como funciona, pero nosotros lo vamos a crear desde cero.

1. Código fuente de la clase `Punto.java`

Dentro de la carpeta `src/main/java` crea una nueva clase llamada `Punto.java` con el siguiente contenido:

```

1  import javax.persistence.Entity;
2  import javax.persistence.GeneratedValue;
3  import javax.persistence.Id;
4
5  @Entity
6  public class Punto {
7      @Id
8      @GeneratedValue
9      private long id;
10
11     private int x;
12     private int y;
13
14     public Punto() {
15     }
16
17     Punto(int x, int y) {
18         this.x = x;
19         this.y = y;
20     }
21
22     public Long getId() {
23         return id;
24     }
25     public int getX() {
26         return x;
27     }
28     public int getY() {
29         return y;
30     }
31     public void setId(long id) {
32         this.id = id;
33     }
34     public void setX(int x) {
35         this.x = x;
36     }
37     public void setY(int y) {
38         this.y = y;
39     }
40
41     @Override
42     public String toString() {
43         return String.format("(%d, %d)", this.x, this.y);
44     }
45 }

```

2. En la misma ruta, crea la clase `TestPunto.java`:

```

1  import javax.persistence.*;
2  import java.util.List;
3
4  public class TestPunto {
5      public static void main(String[] args) {
6          // Abre una conexión a la base de datos
7          // Se crea un EntityManagerFactory con la configuración especificada en
8          // el archivo de persistencia
9          EntityManagerFactory emf =
10 Persistence.createEntityManagerFactory("src/main/java/DB/puntos.odb");
11          // Se crea un EntityManager para interactuar con la base de datos
12          EntityManager em = emf.createEntityManager();

```

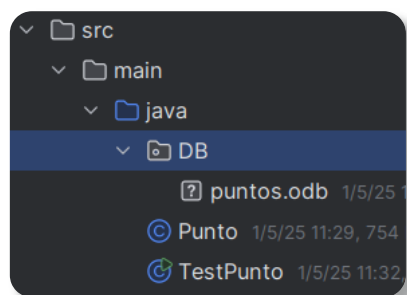
```

11
12         // Almacena 1000 objetos Punto en la base de datos
13         // Inicia una transacción
14         em.getTransaction().begin();
15         for (int i = 0; i < 1000; i++) {
16             // Crea un nuevo objeto Punto con coordenadas aleatorias entre 0 y 99
17             Punto p = new Punto((int) (Math.random() * 100), (int) (Math.random()
18 * 100));
19             // Persiste el objeto Punto en la base de datos
20             em.persist(p);
21         }
22         // Confirma la transacción para guardar los cambios
23         em.getTransaction().commit();
24
25         // Consulta para contar los objetos Punto
26         Query q1 = em.createQuery("SELECT COUNT(p) FROM Punto p");
27         // Imprime el resultado de la consulta
28         System.out.println("Número total de Puntos: " + q1.getSingleResult());
29
30         // Consulta para calcular el promedio de X
31         Query q2 = em.createQuery("SELECT AVG(p.x) FROM Punto p");
32         // Imprime el resultado de la consulta
33         System.out.println("Media de los puntos X: " + q2.getSingleResult());
34
35         // Consulta para obtener todos los objetos Punto
36         TypedQuery<Punto> query = em.createQuery("SELECT p FROM Punto p",
37 Punto.class);
38         // Obtiene la lista de resultados
39         List<Punto> results = query.getResultList();
40         System.out.println("Resultados de la consulta query:");
41         for (Punto p : results) {
42             // Imprime cada objeto Punto
43             System.out.println(p);
44         }
45
46         // Cierra la conexión a la base de datos
47         // Cierra el EntityManager
48         em.close();
49         // Cierra el EntityManagerFactory
50         emf.close();
51     }
52 }

```

3. Crea la BDOO:

Una vez creado todo el proyecto, y comprobado que no tenga errores de compilación, ejecuta el método `main` de la clase `TestPunto` y veras aparecer una carpeta llamada `DB` en la ruta que contiene un único fichero `puntos.odb`.



4. El resultado en pantalla debe ser similar a este:

```

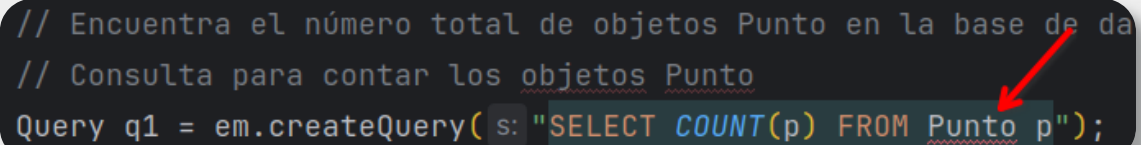
1  Número total de Puntos: 1000
2  Media de los puntos X: 49.05433333333333
3  (96, 19)
4  (40, 65)
5  (87, 97)
6  (24, 26)
7  (46, 19)
8  (28, 79)
9  (32, 18)
10 (51, 18)
11 (64, 22)
12 ...

```

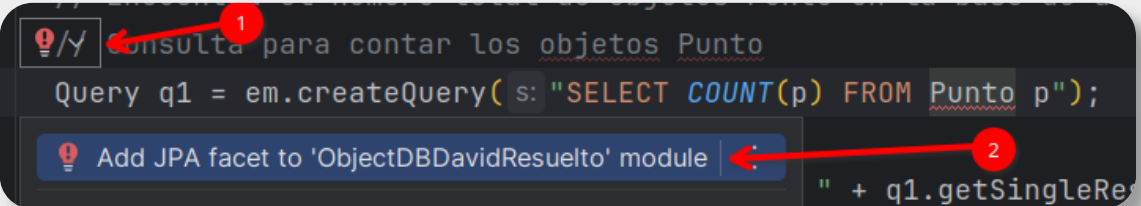
Importante

Puede que hayas observado que la palabra `Punto` dentro de las consultas SQL aparece subrayado en rojo. Eso es porque IntelliJ no detecta la clase `Punto` como una entidad JPA:

```
// Encuentra el número total de objetos Punto en la base de da
// Consulta para contar los objetos Punto
Query q1 = em.createQuery(s: "SELECT COUNT(p) FROM Punto p");
```



La solución es tan simple como hacer clic en la bombilla roja que aparece, y elegir la opción de que IntelliJ nos añada el complemento necesario de JPA para "entender" la persistencia de datos.



```
// Consulta para contar los objetos Punto
Query q1 = em.createQuery(s: "SELECT COUNT(p) FROM Punto p");
" + q1.getSingleRes
```

Ten en cuenta que cada vez que ejecutas el código se añaden 1000 puntos más a la BDOO, así que si lo ejecutas un par de veces más tendrás 3000 puntos en la base de datos.

Si quieres volver a empezar solo debes borrar el fichero `*.odb` y al volver a ejecutar el código se generará de nuevo el fichero con 1000 puntos.

4. Actividades

Ahora genera un documento `pdf` con el siguiente contenido:

1. Explica para que sirven las etiquetas (`@Entity`, `@Id`, `@GeneratedValue` y `@Override`) que aparecen en la clase `Punto`.
2. Modifica la clase `TestPunto.java` para que la base de datos se cree en la ruta `bdoo` en lugar de `DB` y el fichero `.odb` tenga tu nombre (en mi caso `david.odb`).
3. Modifica el método `main` y añade dos consultas más `q3`, `q4` y `q5` que muestren:
 1. `q3`: La media de los puntos Y.
 2. `q4`: consulte los puntos en los que la X o la Y sean superiores a 50.
 3. `q5`: consulta los puntos ordenando primero descendientemente por la X, y en caso de empate ascendientemente por la Y.

Envía tu fichero `pdf` con explicaciones detalladas sobre los cambios requeridos, adjuntando fragmentos del código añadido (con capturas de pantalla o recortes de código fuente) a la tarea de Aules.

5. Fuentes de información

- [Wikipedia](#)
- [Programación \(Grado Superior\) - Juan Carlos Moreno Pérez \(Ed. Ra-ma\)](#)
- Apuntes IES Henri Matisse (Javi García Jimenez?)
- Apuntes AulaCampus
- [Apuntes José Luis Comesaña](#)
- [Apuntes IOC Programació bàsica \(Joan Arnedo Moreno\)](#)
- [Apuntes IOC Programació Orientada a Objectes \(Joan Arnedo Moreno\)](#)
- [FXDocs](#)
- <https://openjfx.io/openjfx-docs/>
- <https://arturoblasco.github.io/pr>