To store a ZIP file as a BLOB in an Oracle database using JPA with Java 21, you'll need the following:

1. **A JPA Entity:** Represents your TESTRESULTBLOB table.
2. **A Repository/DAO:** For interacting with the database.
3. **A Service (optional but good practice):** To encapsulate the business logic.
4. **A Main Class:** To demonstrate the process.
5. **A ZIP File:** To be stored.

Here's a sample program, assuming you have a basic Spring Boot setup (as it simplifies JPA configuration). If you're not using Spring Boot, the JPA configuration part will be more verbose.

1. Database Table (Oracle SQL)

First, let's define a simple table for TESTRESULTBLOB:

```
CREATE TABLE TESTRESULTBLOB (
    ID NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY,
    FILE_NAME VARCHAR2(255),
    FILE_CONTENT BLOB,
    UPLOAD_DATE TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (ID)
);
```

2. Maven Dependencies (pom.xml)

Add the necessary dependencies to your pom.xml.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.2.5</version> <relativePath/> </parent>
    <groupId>com.example</groupId>
    <artifactId>zip-blob-storage</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>zip-blob-storage</name>
    <description>Demo project for storing ZIP as BLOB</description>

    <properties>
        <java.version>21</java.version>
    </properties>

    <dependencies>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
        <version>3.2.5</version>
        <scope>compile</scope>
</dependency>
        <dependency>
```

```
            <groupId>com.oracle.database.jdbc</groupId>
            <artifactId>ojdbc11</artifactId>
            <version>23.4.0.24.05</version> <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <version>3.2.5</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

3. application.properties (src/main/resources)
Configure your Oracle database connection.

```
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:XE
spring.datasource.username=your_username
spring.datasource.password=your_password
spring.datasource.driver-class-name=oracle.jdbc.OracleDriver
spring.jpa.hibernate.ddl-auto=update # Use 'update' for development,
'none' for production
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.OracleDi
alect
```

Important: Replace localhost:1521:XE, your_username, and your_password with your actual Oracle database connection details.

4. JPA Entity (TestResultBlob.java)

```
package com.example.zipblobstorage.entity;

import jakarta.persistence.*;
import java.sql.Blob;
import java.time.LocalDateTime;

@Entity
@Table(name = "TESTRESULTBLOB")
public class TestResultBlob {
```

```
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY) // For Oracle,
if using Identity column
    private Long id;

    @Column(name = "FILE_NAME")
    private String fileName;

    @Lob // Specifies that the field should be persisted as a large
object (BLOB in this case)
    @Column(name = "FILE_CONTENT")
    private byte[] fileContent; // Use byte[] for BLOB

    @Column(name = "UPLOAD_DATE")
    private LocalDateTime uploadDate;

    // Constructors
    public TestResultBlob() {
    }

    public TestResultBlob(String fileName, byte[] fileContent) {
        this.fileName = fileName;
        this.fileContent = fileContent;
        this.uploadDate = LocalDateTime.now();
    }

    // Getters and Setters
    public Long getId() {
        return id;
    }

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

    public String getFileName() {
        return fileName;
    }

    public void setFileName(String fileName) {
        this.fileName = fileName;
    }

    public byte[] getFileContent() {
        return fileContent;
    }

    public void setFileContent(byte[] fileContent) {
```

```
        this.fileContent = fileContent;
    }

    public LocalDateTime getUploadDate() {
        return uploadDate;
    }

    public void setUploadDate(LocalDateTime uploadDate) {
        this.uploadDate = uploadDate;
    }

    @Override
    public String toString() {
        return "TestResultBlob{" +
               "id=" + id +
               ", fileName='" + fileName + '\'' +
               ", uploadDate=" + uploadDate +
               ", fileSize=" + (fileContent != null ?
fileContent.length : 0) + " bytes" +
               '}';
    }
}
```

5. JPA Repository (TestResultBlobRepository.java)

```
package com.example.zipblobstorage.repository;

import com.example.zipblobstorage.entity.TestResultBlob;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface TestResultBlobRepository extends
JpaRepository<TestResultBlob, Long> {
}
```

6. Service Class (FileStorageService.java)

```
package com.example.zipblobstorage.service;

import com.example.zipblobstorage.entity.TestResultBlob;
import com.example.zipblobstorage.repository.TestResultBlobRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
```

```
import java.util.Optional;

@Service
public class FileStorageService {

    @Autowired
    private TestResultBlobRepository testResultBlobRepository;

    @Transactional
    public TestResultBlob storeZipFile(String filePathString) throws
IOException {
        Path filePath = Paths.get(filePathString);
        if (!Files.exists(filePath) || !Files.isReadable(filePath)) {
            throw new IOException("File not found or not readable: " +
filePathString);
        }

        byte[] fileContent = Files.readAllBytes(filePath);
        String fileName = filePath.getFileName().toString();

        TestResultBlob testResultBlob = new TestResultBlob(fileName,
fileContent);
        return testResultBlobRepository.save(testResultBlob);
    }

    @Transactional(readOnly = true)
    public Optional<TestResultBlob> getFileById(Long id) {
        return testResultBlobRepository.findById(id);
    }

    @Transactional(readOnly = true)
    public byte[] getFileContentById(Long id) throws IOException {
        return testResultBlobRepository.findById(id)
                .map(TestResultBlob::getFileContent)
                .orElseThrow(() -> new IOException("File content not
found for ID: " + id));
    }
}
```

7. Main Application Class (ZipBlobStorageApplication.java)

```
package com.example.zipblobstorage;

import com.example.zipblobstorage.entity.TestResultBlob;
import com.example.zipblobstorage.service.FileStorageService;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
```

```
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;

@SpringBootApplication
public class ZipBlobStorageApplication {

    public static void main(String[] args) {
        SpringApplication.run(ZipBlobStorageApplication.class, args);
    }

    @Bean
    public CommandLineRunner run(FileStorageService
fileStorageService) {
        return args -> {
            // 1. Create a dummy ZIP file for demonstration
            Path dummyZipPath = Paths.get("dummy.zip");
            try {
                createDummyZipFile(dummyZipPath);
                System.out.println("Created dummy ZIP file: " +
dummyZipPath.toAbsolutePath());

                // 2. Store the ZIP file in the database
                System.out.println("Storing ZIP file to database...");
                TestResultBlob storedBlob =
fileStorageService.storeZipFile(dummyZipPath.toString());
                System.out.println("ZIP file stored successfully! " +
storedBlob);

                // 3. Retrieve the ZIP file from the database and
verify
                System.out.println("\nRetrieving ZIP file from
database...");

fileStorageService.getFileById(storedBlob.getId()).ifPresent(retrieved
Blob -> {
                    System.out.println("Retrieved Blob: " +
retrievedBlob.getFileName() +
                                       ", Size: " +
retrievedBlob.getFileContent().length + " bytes");
                    try {
                        // Optionally, write the retrieved content to
a new file to verify integrity
                        Path retrievedZipPath = Paths.get("retrieved_"
+ retrievedBlob.getFileName());
                        Files.write(retrievedZipPath,
```

```
retrievedBlob.getFileContent());
                        System.out.println("Retrieved content written
to: " + retrievedZipPath.toAbsolutePath());
                    } catch (IOException e) {
                        System.err.println("Error writing retrieved
file: " + e.getMessage());
                    }
                });

                // 4. Clean up the dummy ZIP file
                Files.deleteIfExists(dummyZipPath);
                System.out.println("Cleaned up dummy ZIP file.");

            } catch (IOException e) {
                System.err.println("Error during ZIP file operations:
" + e.getMessage());
            }
        };
    }

    private void createDummyZipFile(Path zipFilePath) throws
IOException {
        // This is a minimal example to create a valid, empty ZIP
file.
        // For a real scenario, you would use ZipOutputStream to add
entries.
        byte[] emptyZipBytes = {
            0x50, 0x4B, 0x05, 0x06, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
            0x00, 0x00, 0x00, 0x00, 0x00, 0x00
        };
        Files.write(zipFilePath, emptyZipBytes);
    }
}
```

How to Run:

1. **Ensure Oracle Database is running:** Make sure your Oracle database is accessible at the specified URL (e.g., localhost:1521:XE).
2. **Create the TESTRESULTBLOB table:** Execute the SQL script provided in step 1.
3. **Update application.properties:** Set your actual Oracle database username and password.
4. **Build the project:**
   `mvn clean install`
5. **Run the Spring Boot application:**
   `mvn spring-boot:run`

Explanation:

- **TestResultBlob Entity:**
    - @Entity and @Table(name = "TESTRESULTBLOB") map the class to your database table.
    - @Id and @GeneratedValue(strategy = GenerationType.IDENTITY) configure the primary key generation. GenerationType.IDENTITY is suitable for Oracle's GENERATED BY DEFAULT ON NULL AS IDENTITY columns.
    - @Lob: This is crucial. It tells JPA to store the byte[] field as a large object type in the database. For Oracle, this will map to a BLOB column.
    - private byte[] fileContent;: Using byte[] is the standard and most straightforward way to handle BLOB data with JPA. JPA will automatically convert the byte[] to and from the BLOB type in the database.
- **TestResultBlobRepository:** Extends JpaRepository, providing standard CRUD operations without needing to write boilerplate code.
- **FileStorageService:**
    - storeZipFile(String filePathString): This method reads the content of the ZIP file from the file system into a byte[], creates a TestResultBlob entity, and then saves it to the database using the repository.
    - @Transactional: Ensures that the database operations are performed within a transaction.
    - get...FileById methods: Demonstrate how to retrieve the stored BLOB data.
- **ZipBlobStorageApplication (Main Class):**
    - Uses CommandLineRunner to execute code right after the Spring application starts.
    - createDummyZipFile: A helper method to create a simple, empty ZIP file for testing purposes. In a real application, you would be providing an actual ZIP file.
    - It demonstrates the full flow: creating a dummy ZIP, storing it, and then retrieving it.

This example provides a complete, runnable solution for storing ZIP files as BLOBs in an Oracle database using Java 21, JPA, and Spring Boot.