

# **TODO APP Backend**

# Create JPA Repository Test Class

## Exercise3

The screenshot shows an IDE interface with the following details:

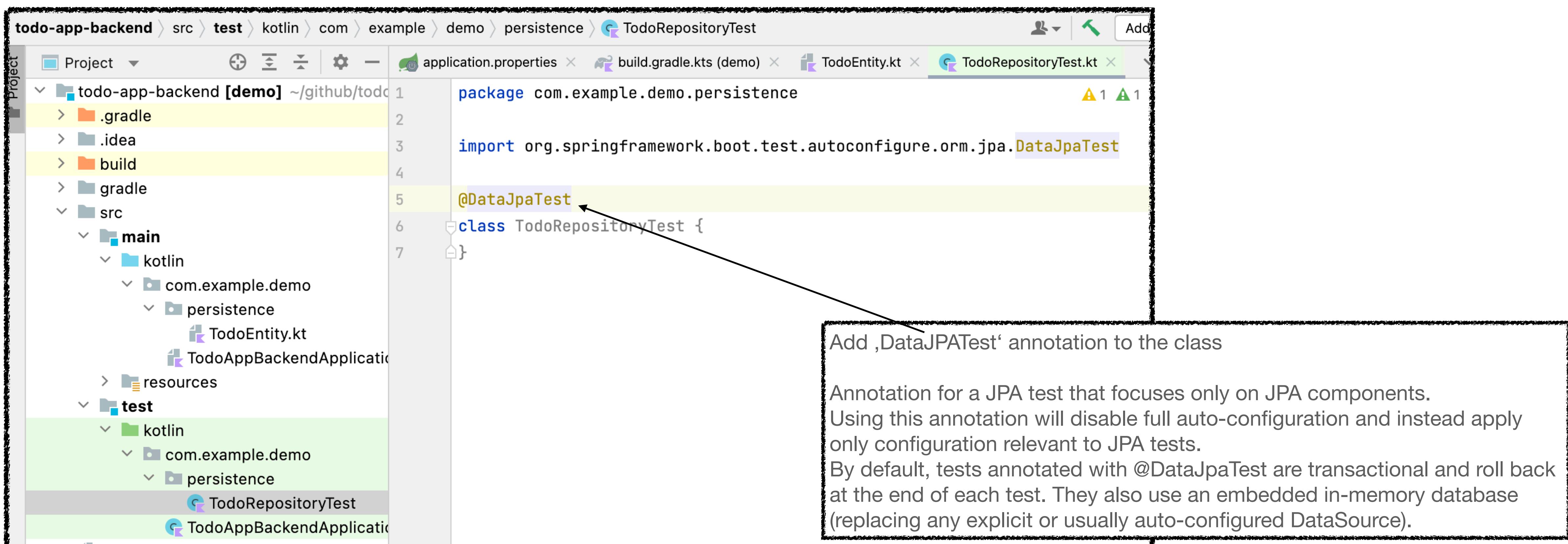
- Project Structure:** The project is named "todo-app-backend". It contains a "src" directory with "main" and "test" sub-directories. "main/kotlin/com.example.demo/persistence" contains "TodoEntity.kt". "test/kotlin/com.example.demo/persistence" contains "TodoAppBackendApplication.kt".
- Code Editor:** The file "TodoEntity.kt" is open. The code defines a class "TodoEntity" with annotations like @Table, @Entity, @Id, and @GeneratedValue. It also defines variables "var id: Long" and "var todo: String".
- Tooltips:** A tooltip titled "New Kotlin Class/File" is displayed, listing options: Class (highlighted), File, Interface, Sealed interface, Data class, Enum class, Sealed class, Annotation, and Object.
- Toolbar:** The toolbar includes icons for Project, Build, Run, and others.
- Status Bar:** The status bar at the bottom shows "18 R TodoRepository: JpaRepository<TodoEntity, Int>".

By convention a test should always be placed in a test package with same name as the corresponding class you want to test.

Also use same name as class you want to test suffixed with 'Test'

# Make it a ,DataJpaTest'

## Exercise3



```
todo-app-backend > src > test > kotlin > com > example > demo > persistence > TodoRepositoryTest
```

Project

```
todo-app-backend [demo] ~/github/todo-app-backend
> .gradle
> .idea
> build
> gradle
src
  main
    kotlin
      com.example.demo
        persistence
          TodoEntity.kt
          TodoAppBackendApplication.kt
  resources
test
  kotlin
    com.example.demo
      persistence
        TodoRepositoryTest.kt
        TodoAppBackendApplication.kt
```

```
application.properties
build.gradle.kts (demo)
TodoEntity.kt
TodoRepositoryTest.kt
```

```
1 package com.example.demo.persistence
2
3 import org.springframework.boot.test.autoconfigure.orm.jpa.DataJpaTest
4
5 @DataJpaTest
6 class TodoRepositoryTest {
7 }
```

Add ,DataJpaTest' annotation to the class

Annotation for a JPA test that focuses only on JPA components. Using this annotation will disable full auto-configuration and instead apply only configuration relevant to JPA tests. By default, tests annotated with @DataJpaTest are transactional and roll back at the end of each test. They also use an embedded in-memory database (replacing any explicit or usually auto-configured DataSource).

# Create a Test

## Exercise3

```
package com.example.demo.persistence

import org.junit.jupiter.api.Test
import org.springframework.boot.test.autoconfigure.orm.jpa.DataJpaTest

@DataJpaTest
class TodoRepositoryTest{

    @Test
    fun `can store a todo`() {
        TODO(reason: "Not yet implemented")
    }
}
```

Add a Junit5 test.  
(junit5 comes as transitive dependency of  
`spring-boot-starter-test` dependency  
and is thereby available in our project)

# Create a Test

## Exercise3

```
package com.example.demo.persistence

import org.junit.jupiter.api.Test
import org.springframework.beans.factory.annotation.Autowired
import org.springframework.boot.test.autoconfigure.orm.jpa.DataJpaTest

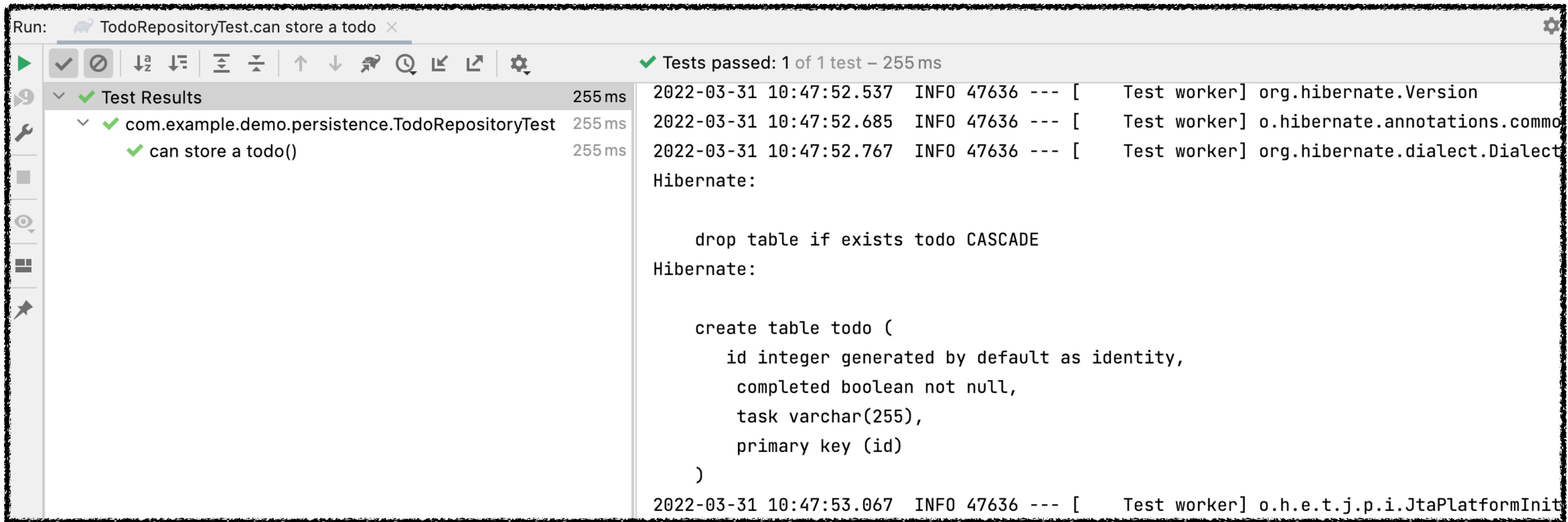
@DataJpaTest
class TodoRepositoryTest {

    @Test
    fun `can store a todo`() {
        // TODO
    }
}
```

Add a JUnit5 test.  
(junit5 comes as transitive dependency of  
`spring-boot-starter-test` dependency  
and is thereby available in our project)

# Should be able to run test

## Exercise3



Run: TodoRepositoryTest.can store a todo

Tests passed: 1 of 1 test – 255 ms

Test Results

com.example.demo.persistence.TodoRepositoryTest

can store a todo()

2022-03-31 10:47:52.537 INFO 47636 --- [ Test worker] org.hibernate.Version

2022-03-31 10:47:52.685 INFO 47636 --- [ Test worker] o.hibernate.annotations.commo

2022-03-31 10:47:52.767 INFO 47636 --- [ Test worker] org.hibernate.dialect.Dialect

Hibernate:

```
drop table if exists todo CASCADE
```

Hibernate:

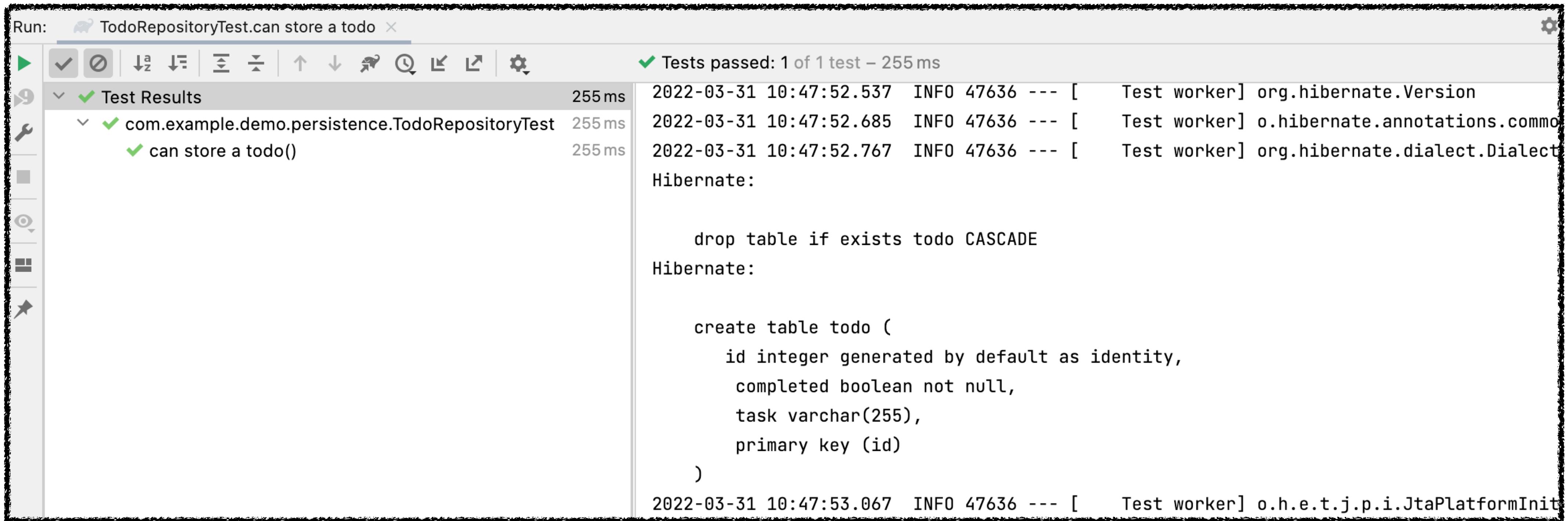
```
create table todo (
    id integer generated by default as identity,
    completed boolean not null,
    task varchar(255),
    primary key (id)
)
```

2022-03-31 10:47:53.067 INFO 47636 --- [ Test worker] o.h.e.t.j.p.i.JtaPlatformInit

As you can see in the log, since we have annotated our test class with `@DataJpaTest` it drops the database table automatically for us to have a clean DB state before every test

# Should be able to run test

## Exercise3



Run: TodoRepositoryTest.can store a todo

Tests passed: 1 of 1 test – 255 ms

Test Results

com.example.demo.persistence.TodoRepositoryTest

can store a todo()

2022-03-31 10:47:52.537 INFO 47636 --- [ Test worker] org.hibernate.Version

2022-03-31 10:47:52.685 INFO 47636 --- [ Test worker] o.hibernate.annotations.commo

2022-03-31 10:47:52.767 INFO 47636 --- [ Test worker] org.hibernate.dialect.Dialect

Hibernate:

```
drop table if exists todo CASCADE
```

Hibernate:

```
create table todo (
    id integer generated by default as identity,
    completed boolean not null,
    task varchar(255),
    primary key (id)
)
```

2022-03-31 10:47:53.067 INFO 47636 --- [ Test worker] o.h.e.t.j.p.i.JtaPlatformInit

As you can see in the log, since we have annotated our test class with `@DataJpaTest` it drops the database table automatically for us to have a clean DB state before every test

# Inject our subject under test

## Exercise3

```
package com.example.demo.persistence

import org.assertj.core.api.Assertions
import org.junit.jupiter.api.Test
import org.springframework.beans.factory.annotation.Autowired
import org.springframework.boot.test.autoconfigure.orm.jpa.DataJpaTest

@DataJpaTest
class TodoRepositoryTest(
    @Autowired private val todoRepository: TodoRepository
) {

    @Test
    fun `can store a todo`() {
        // TODO
    }
}
```

Lets inject our TodoRepository Bean to be able to use it at our tests

# Add little helper function to create Todo with defaults

## Exercise3

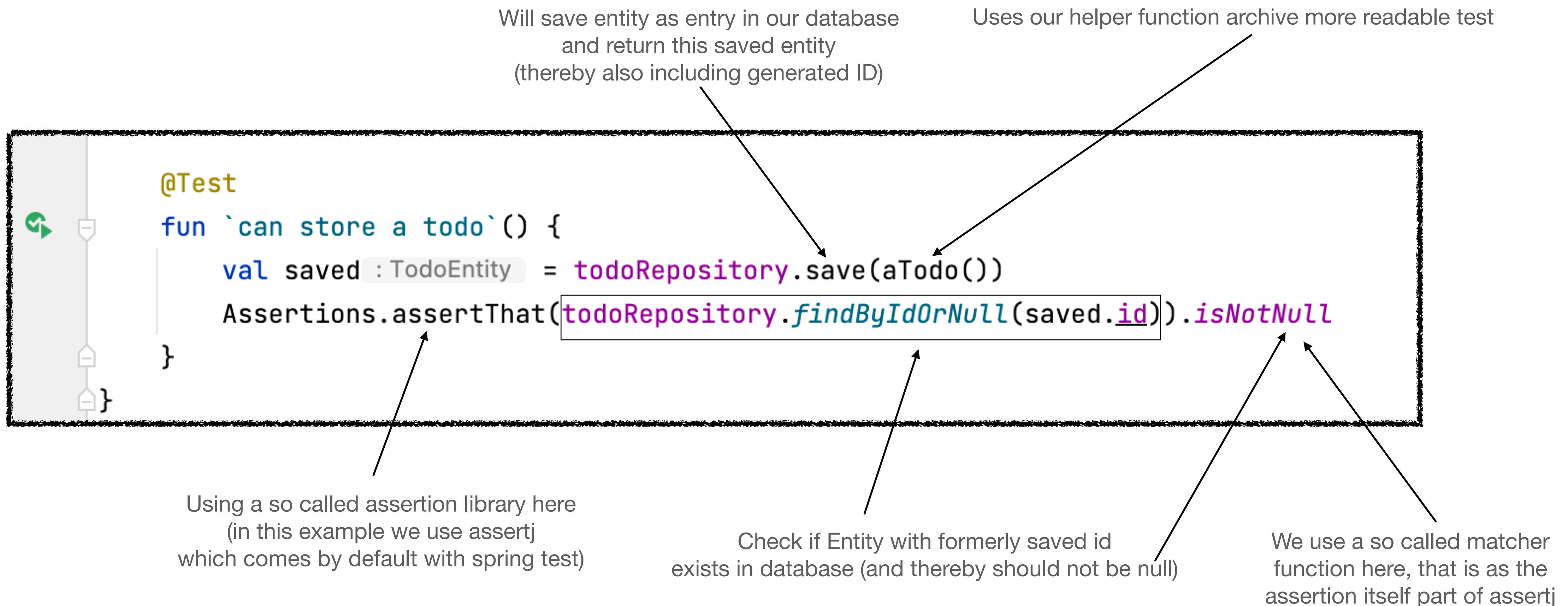
```
@DataJpaTest
class TodoRepositoryTest(
    @Autowired private val todoRepository: TodoRepository
) {

    private fun aTodo(
        task: String = "a sample task",
        completed: Boolean = false
    ): TodoEntity = TodoEntity(
        task = task,
        completed = completed
)

    @Test
    fun `can store a todo`() {
        // TODO
    }
}
```

# Write test to check if we can store something

## Exercise3



# Write test to check if we can read from DB

## Exercise3

```
private val testData : List<TodoEntity> = listOf(  
    aTodo(task = "cleaning up", completed = true),  
    aTodo(task = "cooking a meal"),  
    aTodo(task = "coding kotlin"), ← Introduced a little helper here  
)  
  
@Test  
fun `can read a todo`() {  
    val testData : (Mutable)List<TodoEntity!> = todoRepository.saveAll(testData) ← Bring the test into a  
    state/precondition  
    we want to have  
  
    val testDataTodo : TodoEntity? = todoRepository.findByIdOrNull(testData[0].id) ← Try to read  
    first stored entity  
    Assertions.assertThat(testDataTodo?.task).isEqualTo("cleaning up") ← Check if it has  
    expected values  
    Assertions.assertThat(testDataTodo?.completed).isTrue  
}
```

# Write test to check if we can delete a todo

## Exercise3

```
@Test  
fun `can remove a todo`() {  
    val testData : (Mutable)List<TodoEntity!> = todoRepository.saveAll(testData) ← Create test data / a precondition  
  
    todoRepository.delete(testData[1]) ← Remove an entry from the database  
  
    Assertions.assertThat(todoRepository.findAll()).containsExactly(  
        testData[0],  
        testData[2], ← Check if only the not removed entries  
    )  
}
```

# Write test to check if we can update a todo

## Exercise3

```
@Test  
fun `can update a todo`() {  
    val todoEntity : TodoEntity = todoRepository.save(aTodo(completed = false)) ← Create test data  
  
    todoRepository.save(todoEntity.apply { completed = true }) ← Update the complete field of this entity  
  
    val updatedTodo : TodoEntity? = todoRepository.findByIdOrNull(todoEntity.id)  
    Assertions.assertThat(updatedTodo?.completed).isTrue ← Check if it was update in database by reading it again and expect completed to be true  
}
```

# Run all test together from terminal

## Exercise3

```
./gradlew test
```