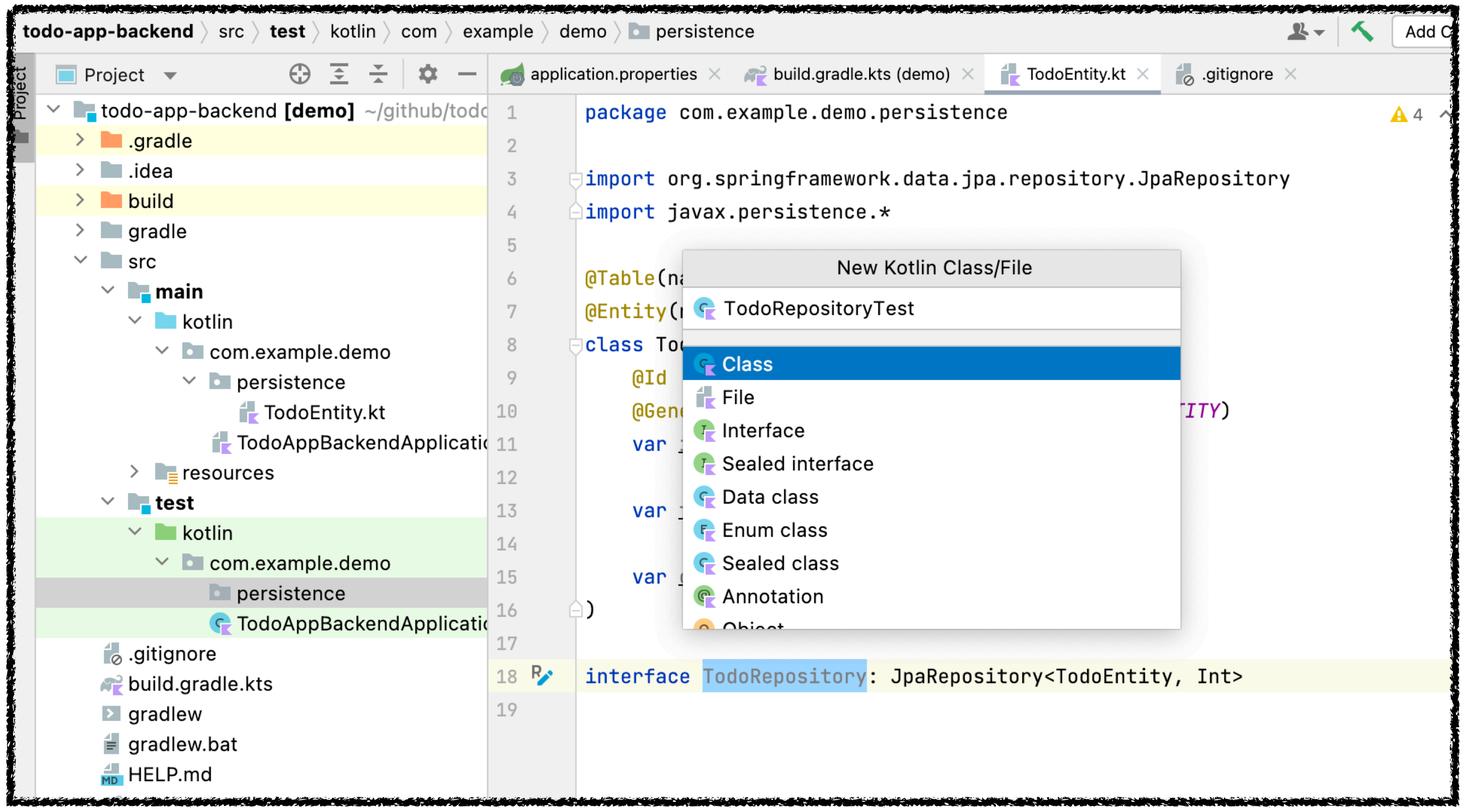
TODO APP Backend

Create JPA Repository Test Class

Exercise3

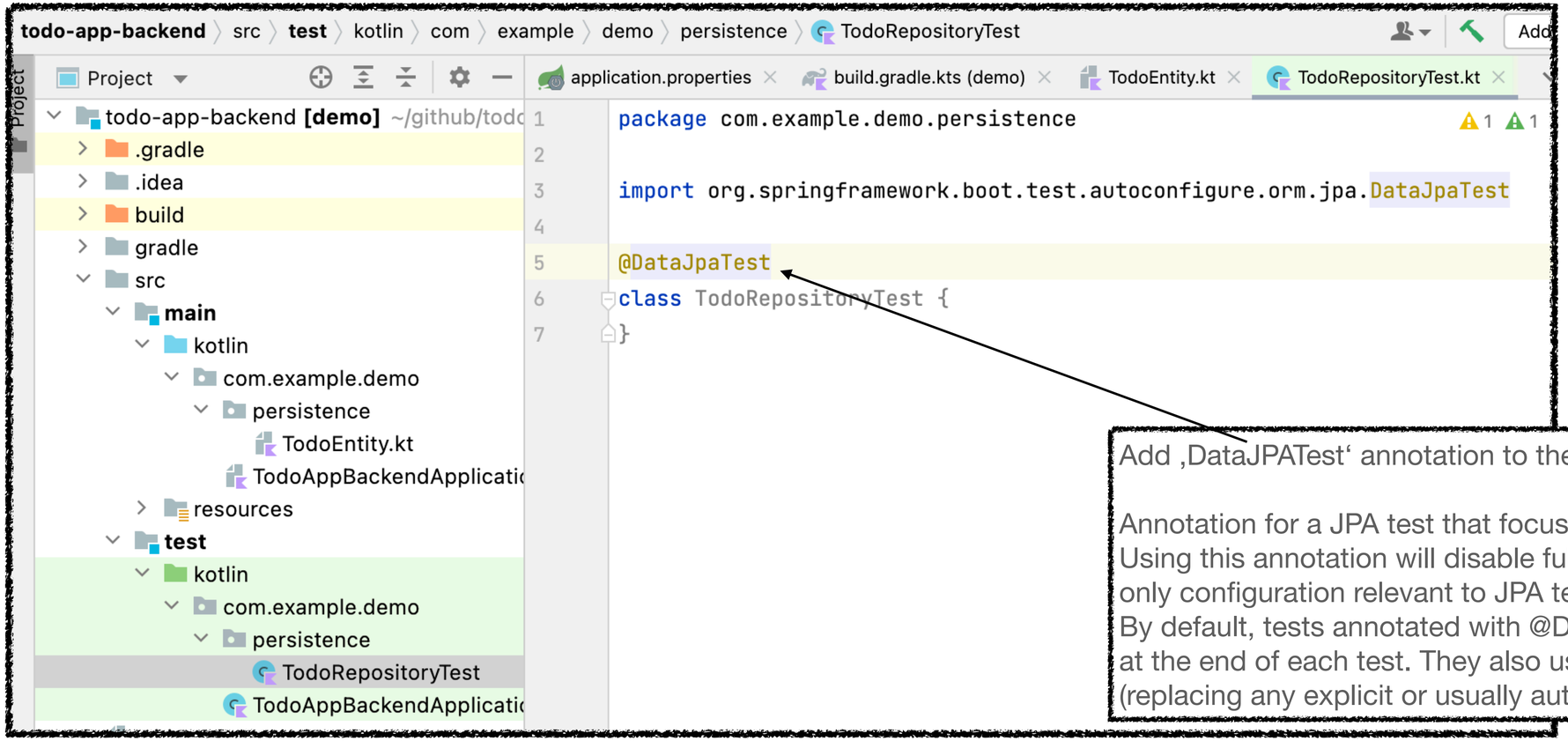


By convention a test should always be placed in a test package with same name as the corresponding class you want to test.

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

Make it a ‚DataJpaTest‘

Exercise3



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 our project)

Create a Test

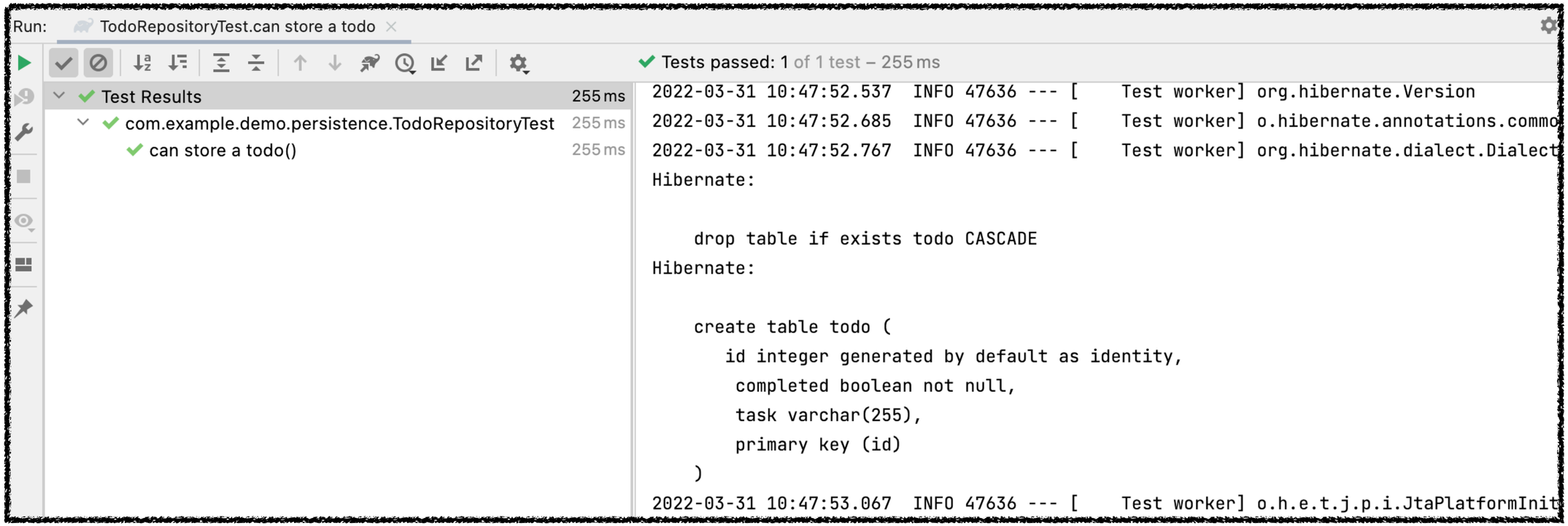
Exercise3

```
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 our project)

Should be able to run test

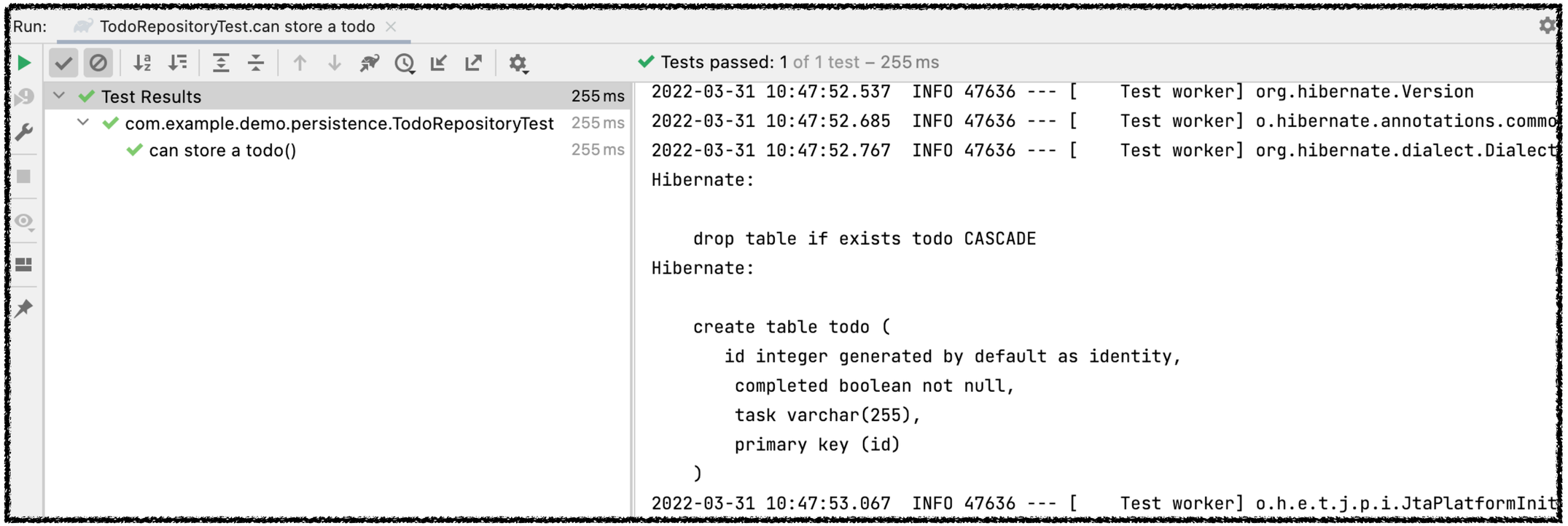
Exercise3



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



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                                    ⚠ 2

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

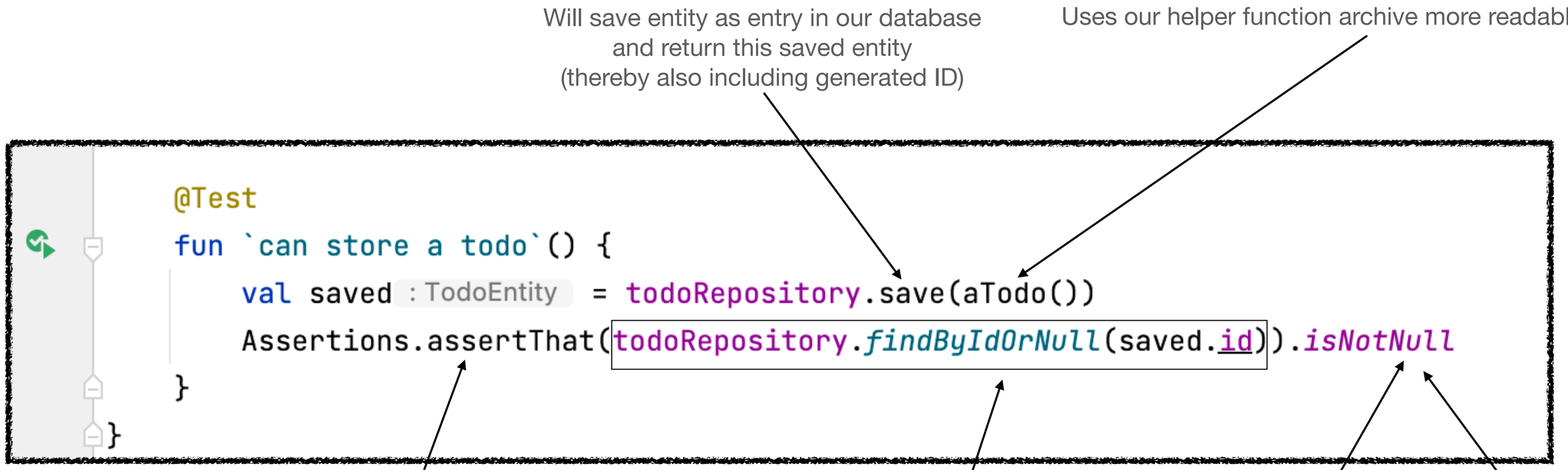
Add little helper function to create Todo with defaults

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"),
)

@Test
fun `can read a todo`() {
    val testData : (Mutable)List<TodoEntity!> = todoRepository.saveAll(testData)

    val testDataTodo : TodoEntity? = todoRepository.findByIdOrNull(testData[0].id)
    Assertions.assertThat(testDataTodo?.task).isEqualTo("cleaning up")
    Assertions.assertThat(testDataTodo?.completed).isTrue
}
```

Introduced a little helper here to get a list of Todo Entities

Bring the test into a state/precondition we want to have

Try to read first stored entity

Check if it has expected values

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)

    todoRepository.delete(testData[1])

    Assertions.assertThat(todoRepository.findAll()).containsExactly(
        testData[0],
        testData[2],
    )
}
```

Create test data / a precondition

Remove an entry from the database

Check if only the not removed entries are still in the database

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))

    todoRepository.save(todoEntity.apply { completed = true })

    val updatedTodo : TodoEntity? = todoRepository.findByIdOrNull(todoEntity.id)
    Assertions.assertThat(updatedTodo?.completed).isTrue
}
```

Create test data

Update the complete field of this entity

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
```