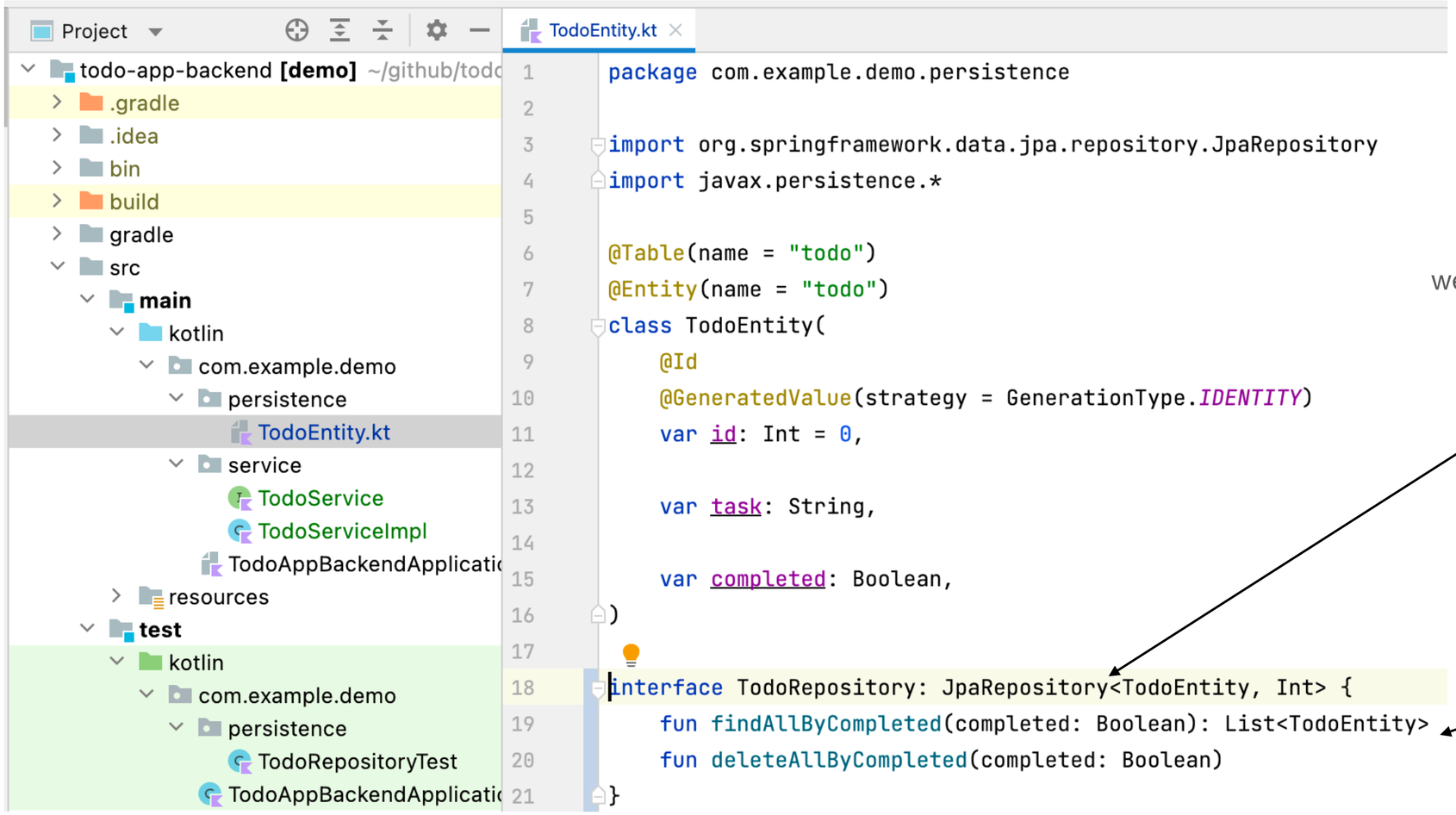
TODO APP Backend

Adding custom Queries to Repository

Exercise4



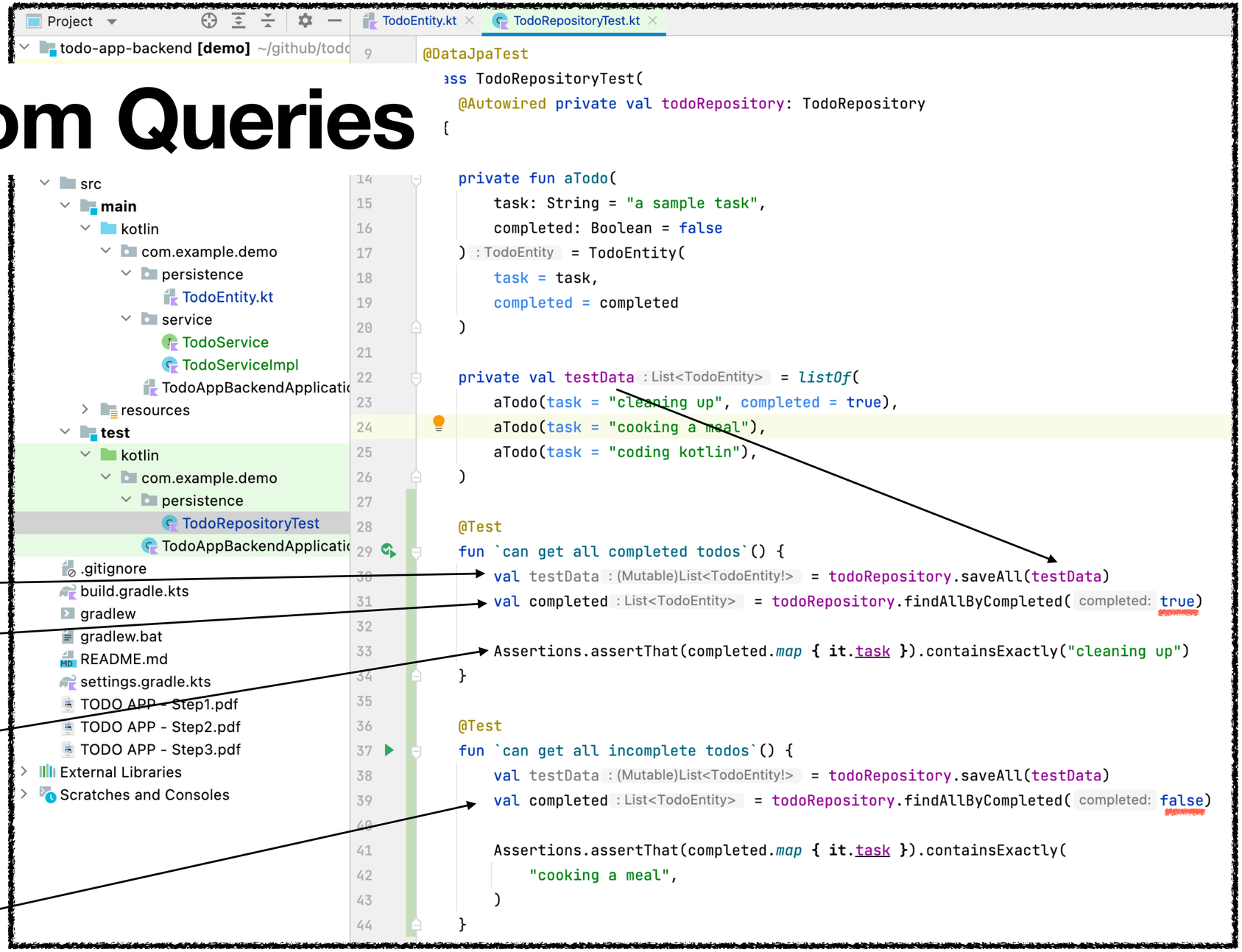
Sometimes it is necessary / more efficient to work with specific queries to archive certain use-cases.
In our example app we want to be able to get all Todos that are completed/incompleted
as well as delete all completed / incomplete.

Remember: Since we implement JpaRepository we automatically get convenient query abstractions like „save“, „findById“, „delete“ etc.
Basically everything we need to archive CRUD

Define custom Queries that fulfill our special needs that can not be archived with pure CRUD

Test the custom Queries

Exercise4



Add test data to the database

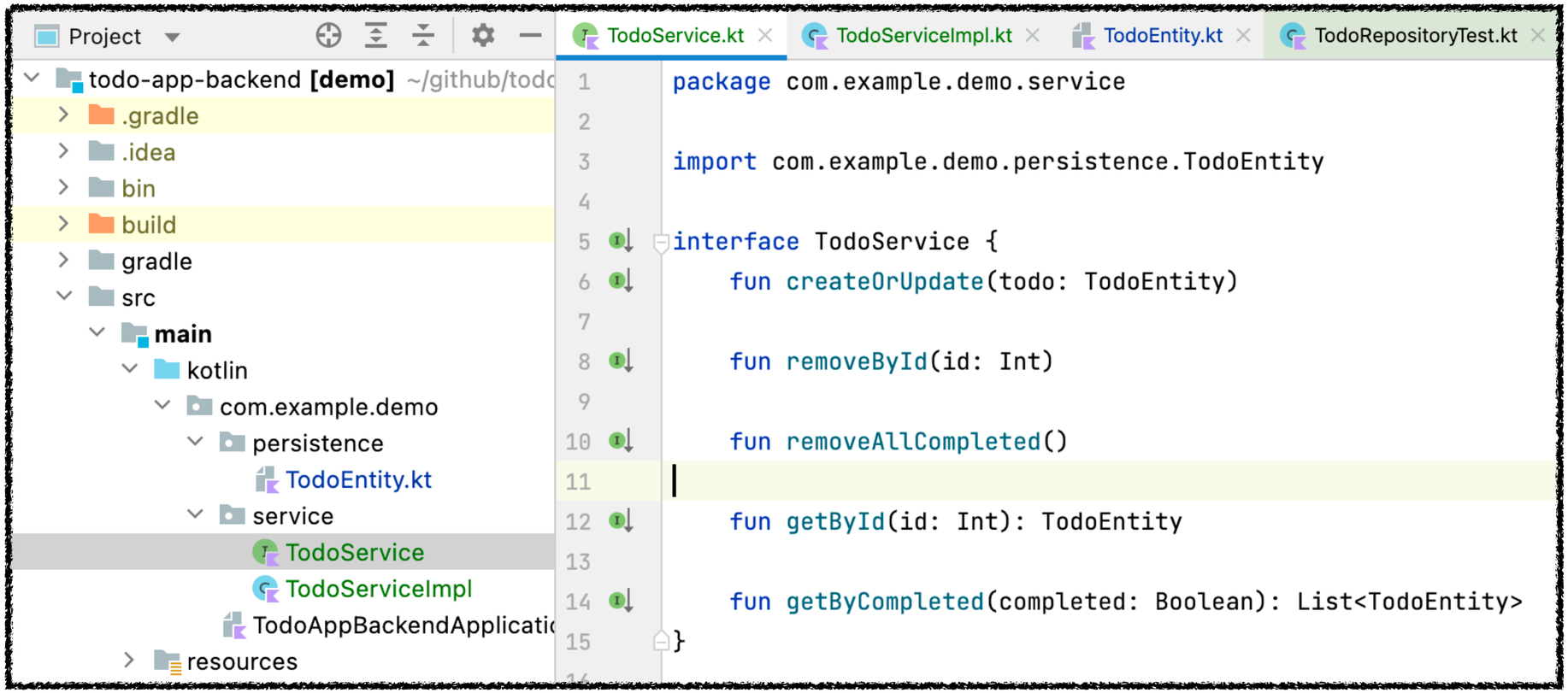
Use our custom query and save the result in variable

Check if our custom query result contains ONLY the one test data todo that is ‚completed‘

Same for completed == false

Adding a Service Interface

Exercise4

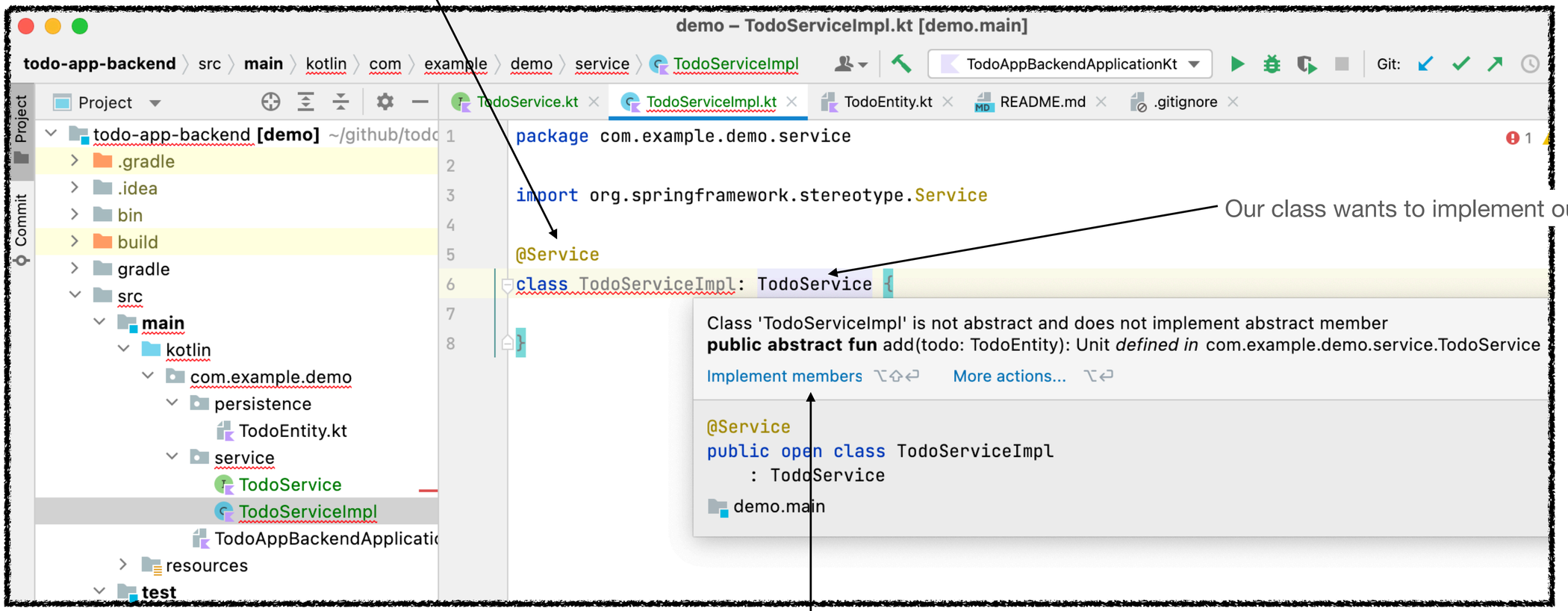


Adding an Interface is not necessarily needed but a great way to describe what our application can do and giving us the possibility to easily change the implementation in the future. Thereby we are adding the possibility to decouple our „technical“ implementation (ServiceImpl) from the business logic (Interface).

Add Class that wants to Implement the Interface

Exercise4

Class is annotate with @Service to let it be picked up as a Bean by Spring.
Also we use this @Service to make clear this class contains a service with business logic.

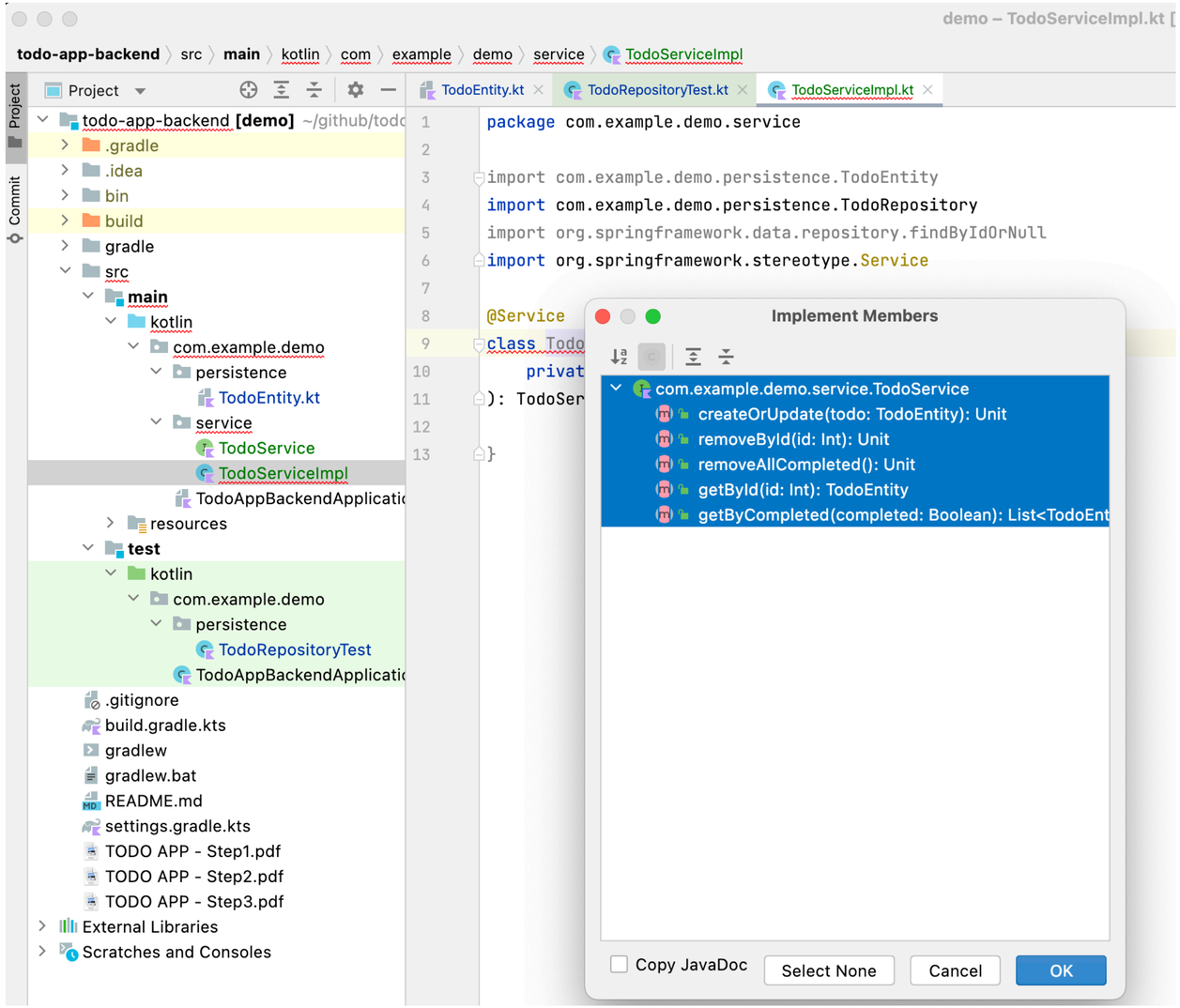


Our class wants to implement our interface

IntelliJ is already complaining about that we should include all the methods
that are required by interface we want to implement :)

Let IntelliJ create all the methods that needs to be implemented

Exercise4



Select all and hit OK

All necessary methods auto-created for us :)

Exercise4

```
@Service
class TodoServiceImpl: TodoService {
    override fun createOrUpdate(todo: TodoEntity) {
        TODO( reason: "Not yet implemented")
    }

    override fun removeById(id: Int) {
        TODO( reason: "Not yet implemented")
    }

    override fun removeAllCompleted() {
        TODO( reason: "Not yet implemented")
    }

    override fun getById(id: Int): TodoEntity {
        TODO( reason: "Not yet implemented")
    }

    override fun getByCompleted(completed: Boolean): List<TodoEntity> {
        TODO( reason: "Not yet implemented")
    }
}
```

Inject the todoRepository Bean into our Service

Exercise4

```
@Service
class TodoServiceImpl(
    private val todoRepository: TodoRepository
): TodoService {
    override fun createOrUpdate(todo: TodoEntity) {
        TODO( reason: "Not yet implemented")
    }

    override fun removeById(id: Int) {
        TODO( reason: "Not yet implemented")
```

We use springs dependency injection mechanism here to be able to use the todoRepository in our service

Implement interface by using todoRepository

Exercise4

```kotlin
@Service
class TodoServiceImpl(
    private val todoRepository: TodoRepository
): TodoService {
    override fun createOrUpdate(todo: TodoEntity) {
        todoRepository.save(todo)
    }

    override fun removeById(id: Int) {
        with(todoRepository.findByIdOrNull(id)) { this: TodoEntity?
            requireNotNull( value: this) { "could not find todo with id '$id'" }
            todoRepository.delete( entity: this)
        }
    }
    override fun getById(id: Int): TodoEntity {
        return todoRepository.findByIdOrNull(id)
            ?: throw IllegalArgumentException("could not find todo with id '$id'")
    }

    override fun removeAllCompleted() {
        todoRepository.deleteAllByCompleted( completed: true)
    }

    override fun getByCompleted(completed: Boolean): List<TodoEntity> {
        return todoRepository.findAllByCompleted(completed)
    }
}
```

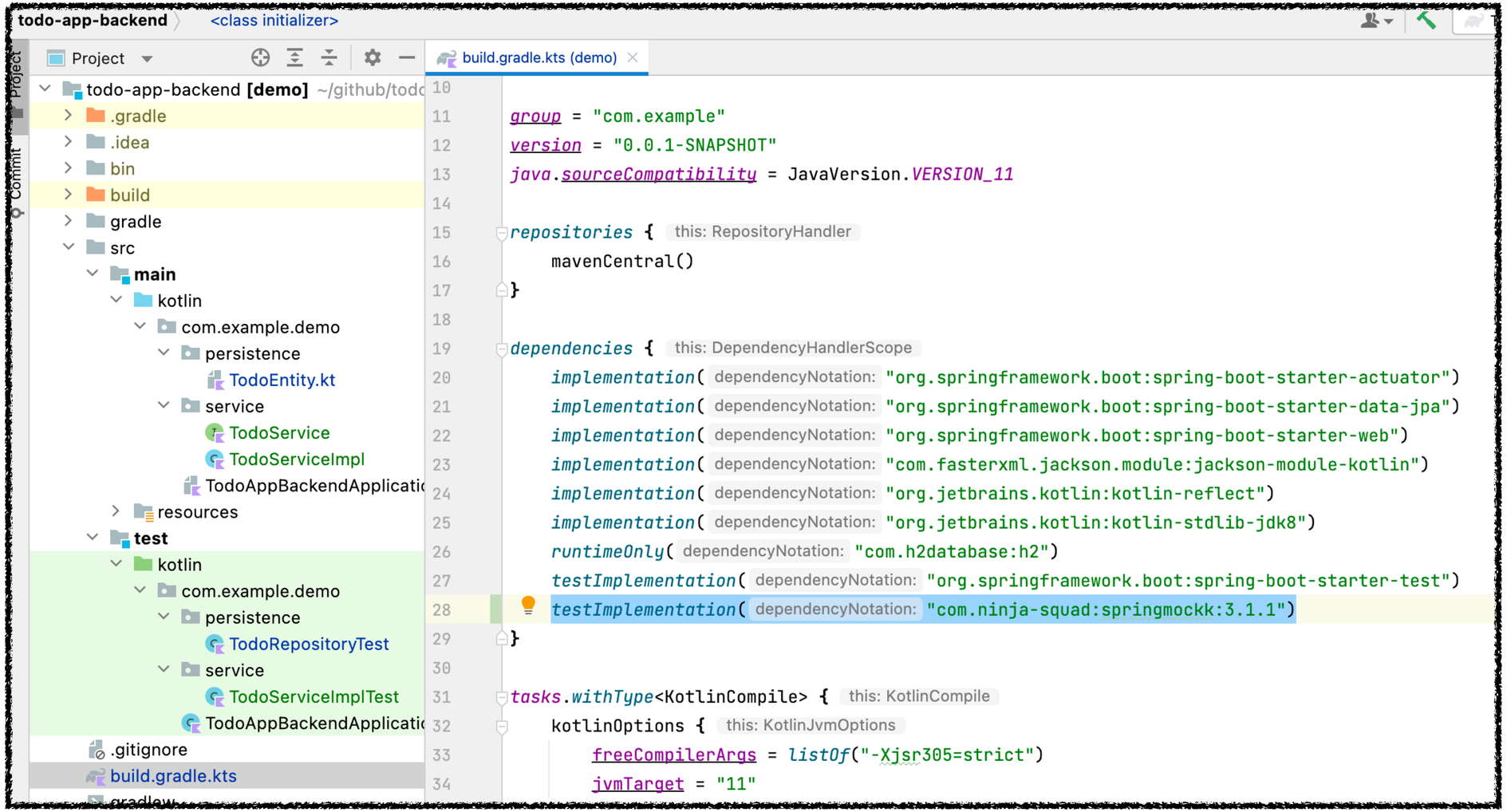
Using ‚with‘ - a so called scoping function (part of the Kotlin standard lib) here

Will throw IllegalArgumentException if todo could not be found in DB

Using the so called ‚Elvis operator‘ to do something as a fallback if the left hand side operation of the Elvis (?:) is `null`. In our case we just throw an IllegalArgumentException on null case.

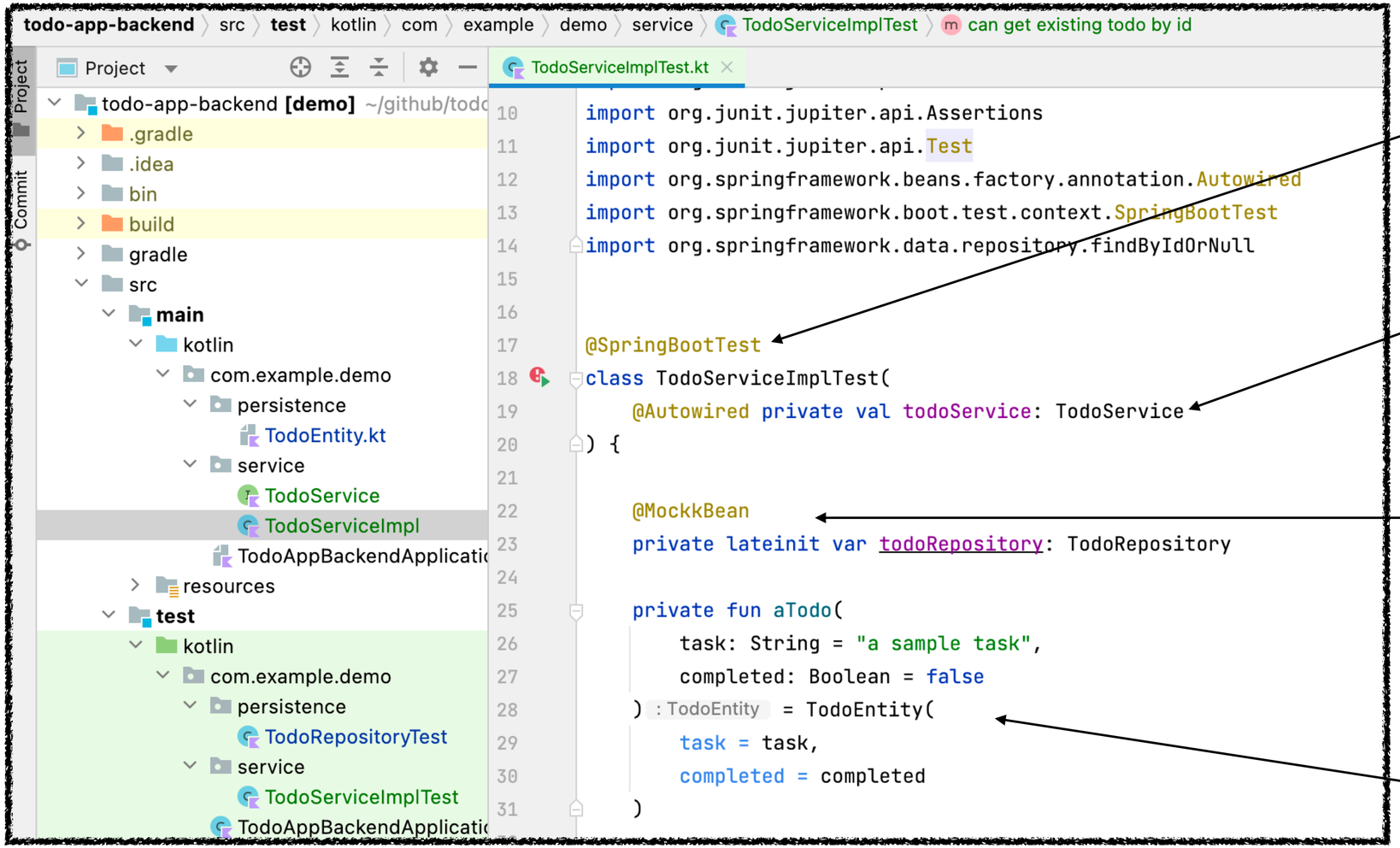
Test the Service / add Spring-Mockk Dependency

Exercise4



Create Test Class for our Service Impl

Exercise4



Annotate the class with @SpringBootTest
-> this will initialize the spring context on test start, which allows us to use our beans with spring DI (dependency injection)

Inject our subject under test

Let's Mock the repository bean.
This will create fake repository that we can control per test to provoke all the behaviors we want to test.
(this is what we added Mockk dependency for)

Again add a little helper function to create todos in a convenient fashion

Add tests

Exercise4

```
33      @Test
34      fun `will create or update by calling repository save`() {
35          val aTestTodo : TodoEntity = aTodo()
36          every { todoRepository.save(aTestTodo) } returns aTestTodo
37
38          todoService.createOrUpdate(aTestTodo)
39          verify { todoRepository.save(aTestTodo) }
40      }
41
42      @Test
43      fun `can get existing todo by id`() {
44          val aTestTodo : TodoEntity = aTodo()
45          every { todoRepository.findByIdOrNull( id: 4711) } returns aTestTodo
46
47          val result : TodoEntity = todoService.getById( id: 4711)
48          assertThat(result).isEqualTo(aTestTodo)
49      }
50
51      @Test
52      fun `will throw if trying to get todo by id that not exists`() {
53          val aTestTodo : TodoEntity = aTodo()
54          every { todoRepository.findByIdOrNull( id: 4711) } returns null
55
56          Assertions.assertThrows(IllegalArgumentException::class.java) {
57              todoService.getById( id: 4711)
58          }
59      }
```

Configure our mocked bean (the repository) to whenever it gets called return a test todo we have just created

Call our service function

Check if our service function has called the expected repository function (save) of our mocked bean and thereby tried to persist our test todo

Configure mocked bean (todoRepository) to return a test todo whenever a given id (4711) gets passed as parameter

Call our service function we want to test and save result

Check if result is exactly our test todo reference

Configure mock to return null on given id

Check if expected exception has been thrown when service function gets called with this given id

Add more tests

Exercise4

Repeat what we just learned

Exercise4

Based on what we learned during this tutorial,
please add a ‚getAll‘ functionality to our service
by extending the interface, the serviceImpl and the tests. 🙂

The getAll should return a List of TodoEntity.
Therefore you need to find out which build-in jpa repository
function this is and also prove this by a test.

Hint: the jpa repository will give us all possibilities we need to
archive the exercise. No need to write custom jpa query this time.