**Spring MVC unit test using Mockito + MockMVC - Controller Layer Testing**

Application - **spring-mockito-example**

Dependencies – **Dev Tool, Lombok, Spring Data JPA, Spring Web, H2**

In this tutorial, we are going to see how to write unit testing for our controller using mockito framework. This is the standard max industry following it.

we'll write two methods. One is Post and another one is Get. So, for these two methods, we'll write the unit test and we'll run it. So, let's start a small development on that.

Here am not going to write the service layer and directly inject the repository in my controller.

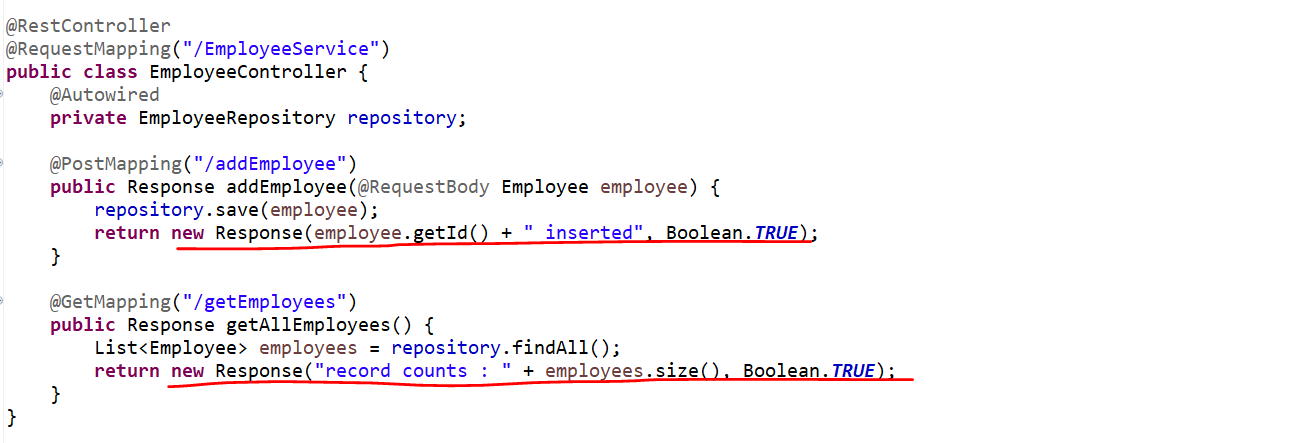
create a response class or custom response class with name make it name **Response** with few fields, string message and boolean status whether it is failing or we are getting the success result so for that we'll mention the true and false

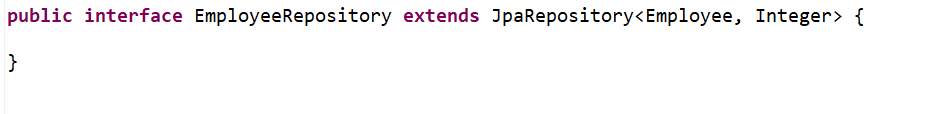


-------------------------------------------------------------------------------------------------------------------------



-------------------------------------------------------------------------------------------------------------------------





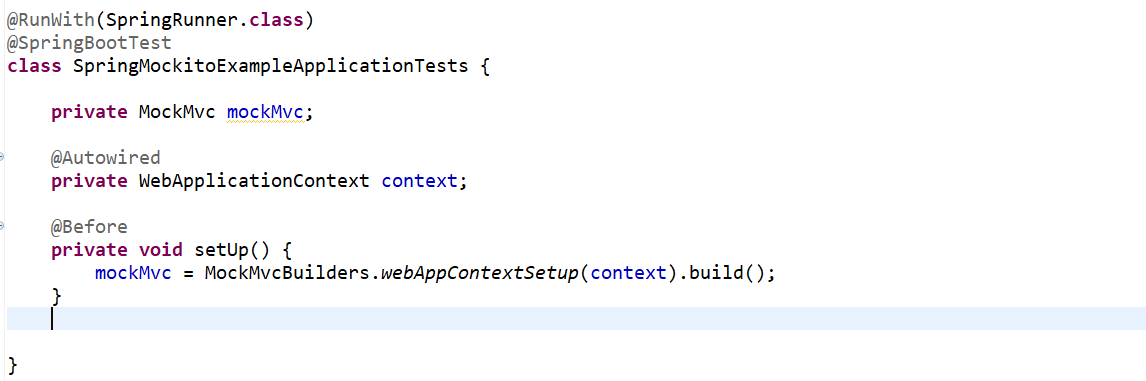
Now we have the controller, model and persistence layer which is the DAO let’s write the test cases.

Now let’s write the Unit Test Case using the **MockMVC** and **Mockito**.

**Now let’s Inject the MockMVC.**

**Now we need to Inject the WebApplicationContext so that It will Interact to our Controller.**

Now write a initialize method where we can initialize our MockMVC. So, I want to initialize this MockMVC before loading my class.

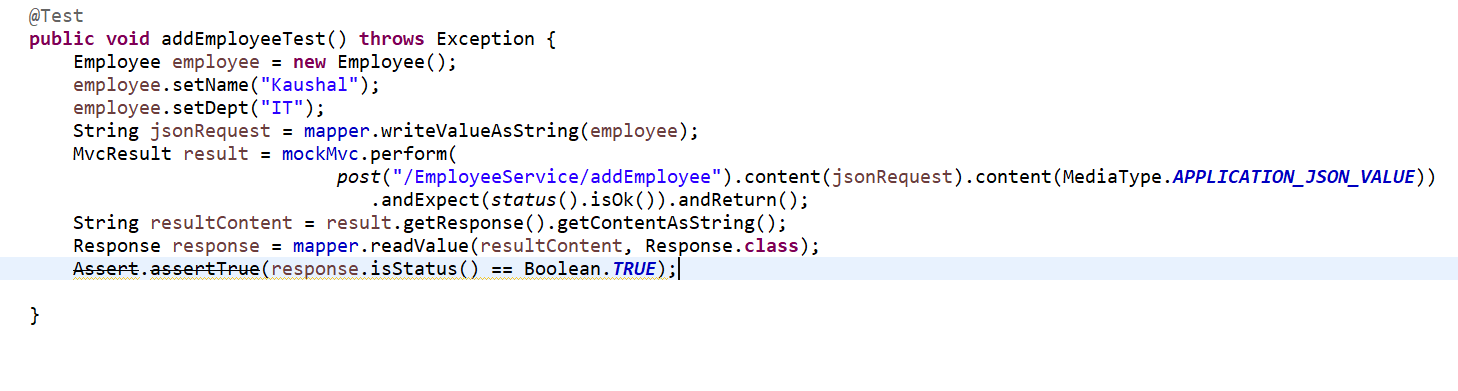


Now using this MockMVC we can call the perform method that is **POST/PUT** what we need.

Now let’s write the test method for our **POST** method that is the **addEmployee**().

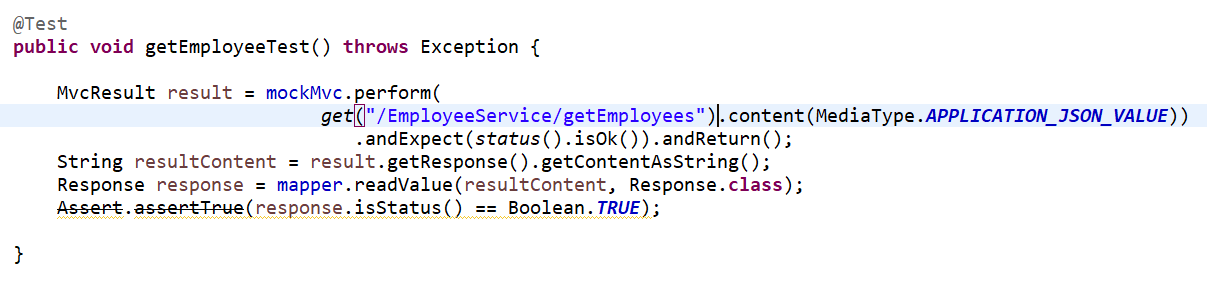
Now create manually one Employee object and let’s convert it to the String JSON using **ObjectMaper**.

Now we need the one parameter **MvcResult** it will hit the controllerand it will get the Response in format of MvcResult. For **AssertTrue** if condition is true it will be a valid test case. If the expected value==controller returned value it will pass the test.



This is for the POST method.

Now let’s write a Test case for our GET method.



So, we have written the test case for these methods.

Now let’s run our application if it will work then we can run our test case…

**POST** [**http://localhost:8080/EmployeeService/addEmployee**](http://localhost:8080/EmployeeService/addEmployee)

Request:

{

        "name": "Kaushal",

        "dept": "It Software"

}

Response:

{

    "message": "1 inserted",

    "status": **true**

}

**GET** [**http://localhost:8080/EmployeeService/getEmployees**](http://localhost:8080/EmployeeService/getEmployees)

Response

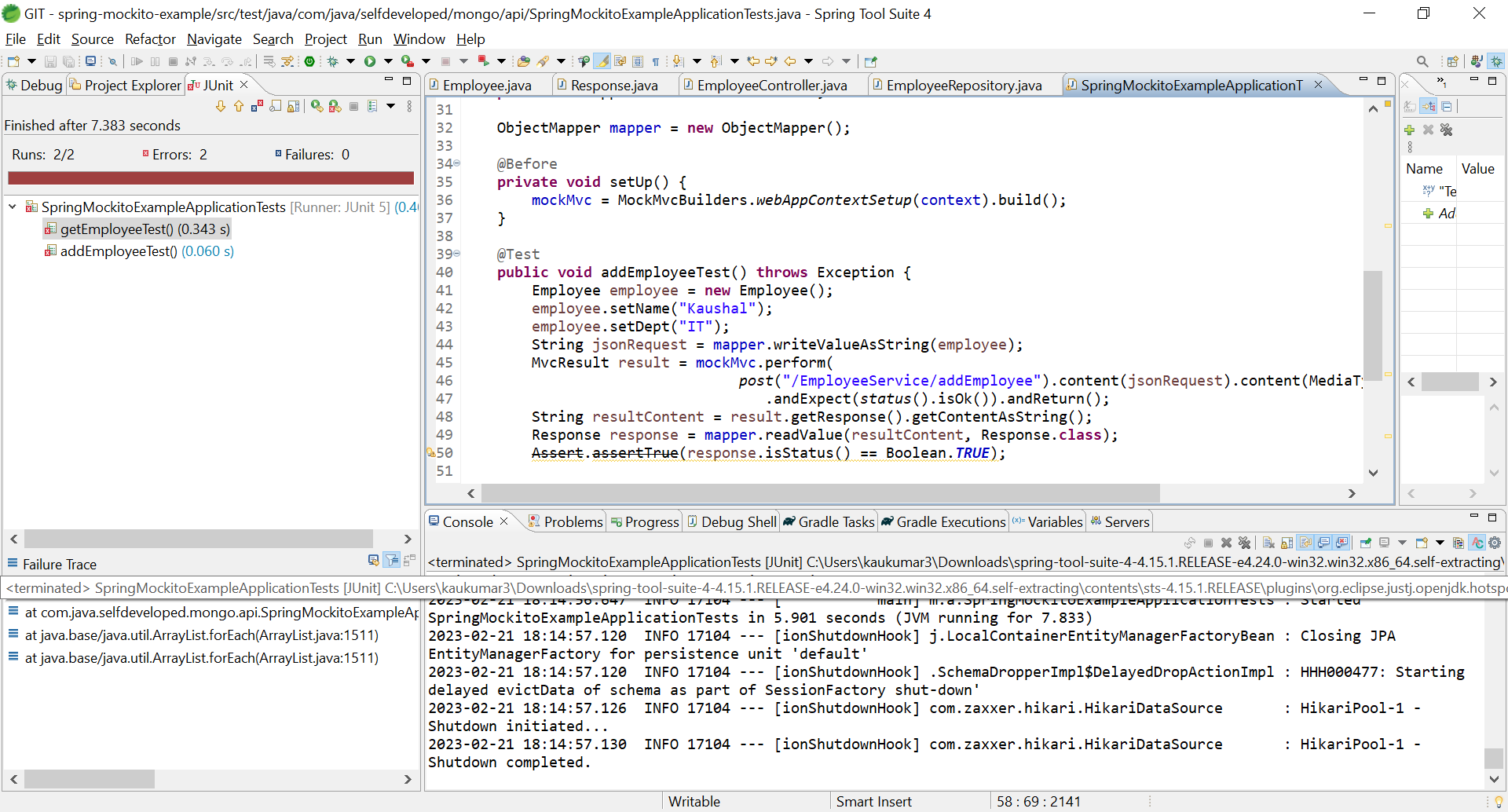
{

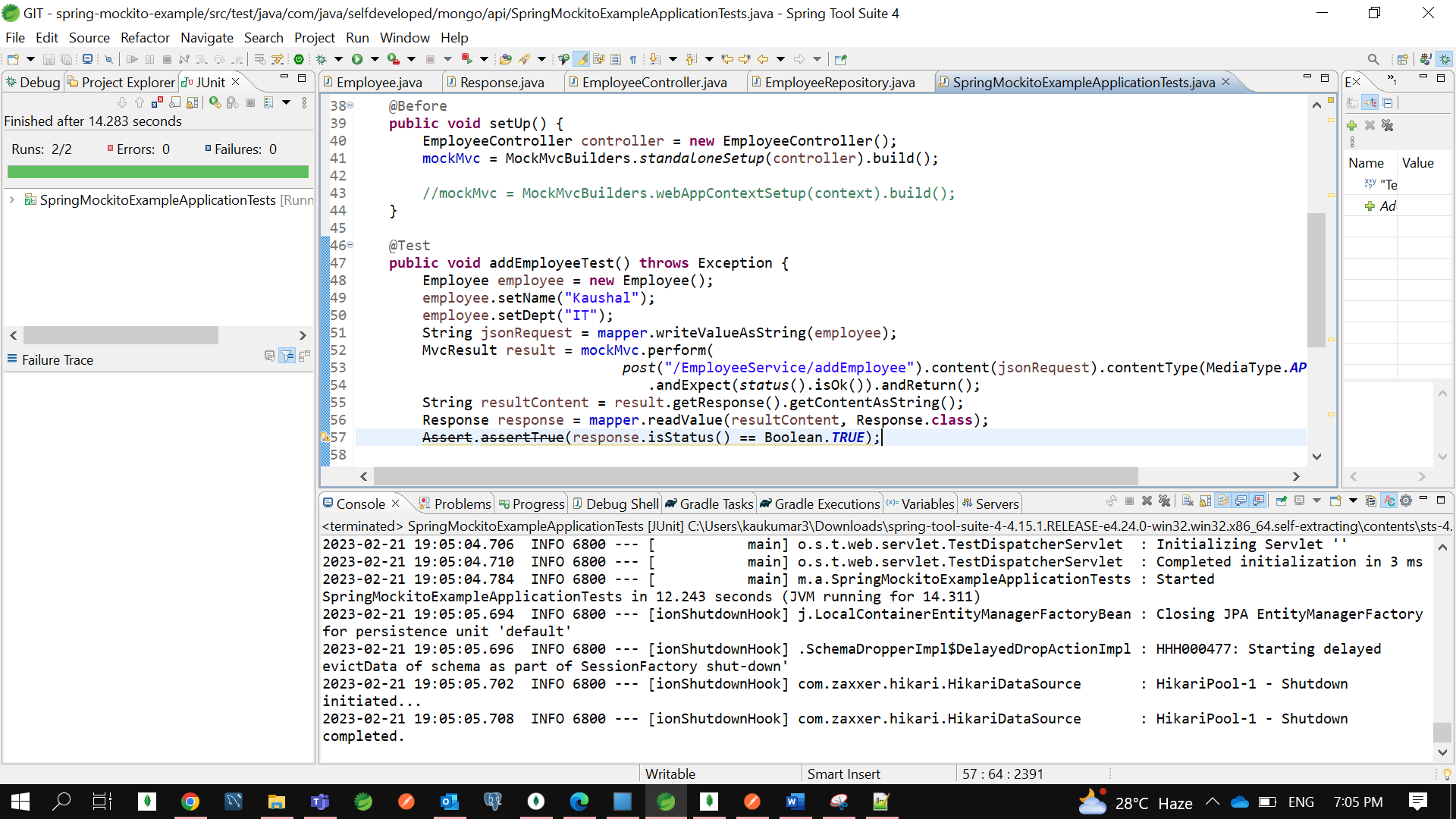
    "message": "record counts : 2",

    "status": **true**

}

Now let’s run our Junit test-case…





**Test Class-**

@RunWith(SpringRunner.**class**)

@SpringBootTest

@AutoConfigureMockMvc

**class** SpringMockitoExampleApplicationTests {

@Autowired

**private** MockMvc mockMvc;

@Autowired

**private** WebApplicationContext context;

ObjectMapper mapper = **new** ObjectMapper();

@Before

**public** **void** setUp() {

EmployeeController controller = **new** EmployeeController();

mockMvc = MockMvcBuilders.*standaloneSetup*(controller).build();

//mockMvc = MockMvcBuilders.webAppContextSetup(context).build();

}

@Test

**public** **void** addEmployeeTest() **throws** Exception {

Employee employee = **new** Employee();

employee.setName("Kaushal");

employee.setDept("IT");

String jsonRequest = mapper.writeValueAsString(employee);

MvcResult result = mockMvc.perform(

*post*("/EmployeeService/addEmployee").content(jsonRequest).contentType(MediaType.***APPLICATION\_JSON\_VALUE***))

.andExpect(*status*().isOk()).andReturn();

String resultContent = result.getResponse().getContentAsString();

Response response = mapper.readValue(resultContent, Response.**class**);

~~Assert~~.~~assertTrue~~(response.isStatus() == Boolean.***TRUE***);

}

@Test

**public** **void** getEmployeeTest() **throws** Exception {

MvcResult result = mockMvc.perform(

*get*("/EmployeeService/getEmployees").content(MediaType.***APPLICATION\_JSON\_VALUE***))

.andExpect(*status*().isOk()).andReturn();

String resultContent = result.getResponse().getContentAsString();

Response response = mapper.readValue(resultContent, Response.**class**);

~~Assert~~.~~assertTrue~~(response.isStatus() == Boolean.***TRUE***);

}

}

**Controller Class---**

@RestController

@RequestMapping("/EmployeeService")

**public** **class** EmployeeController {

@Autowired

**private** EmployeeRepository repository;

@PostMapping("/addEmployee")

**public** Response addEmployee(@RequestBody Employee employee) {

repository.save(employee);

**return** **new** Response(employee.getId() + " inserted", Boolean.***TRUE***);

}

@GetMapping("/getEmployees")

**public** Response getAllEmployees() {

List<Employee> employees = repository.findAll();

**return** **new** Response("record counts : " + employees.size(), Boolean.***TRUE***);

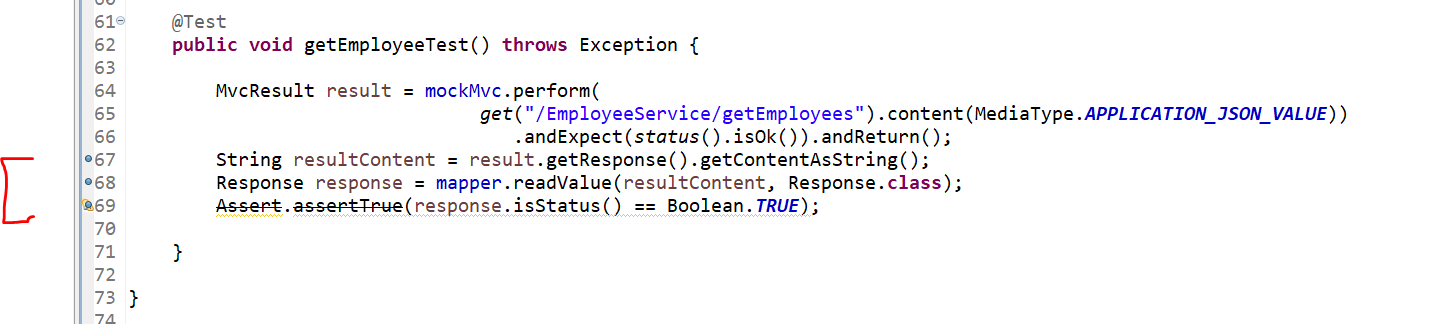
}

}

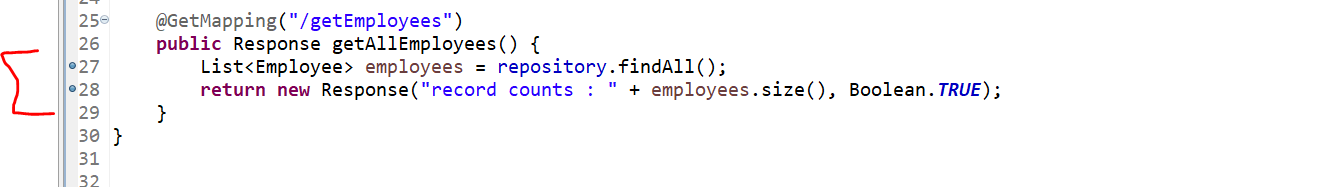
Actually, there are many other tools which we can use for testcase but here we are using MockMVC with Mockito.

Let’s Debug once ----

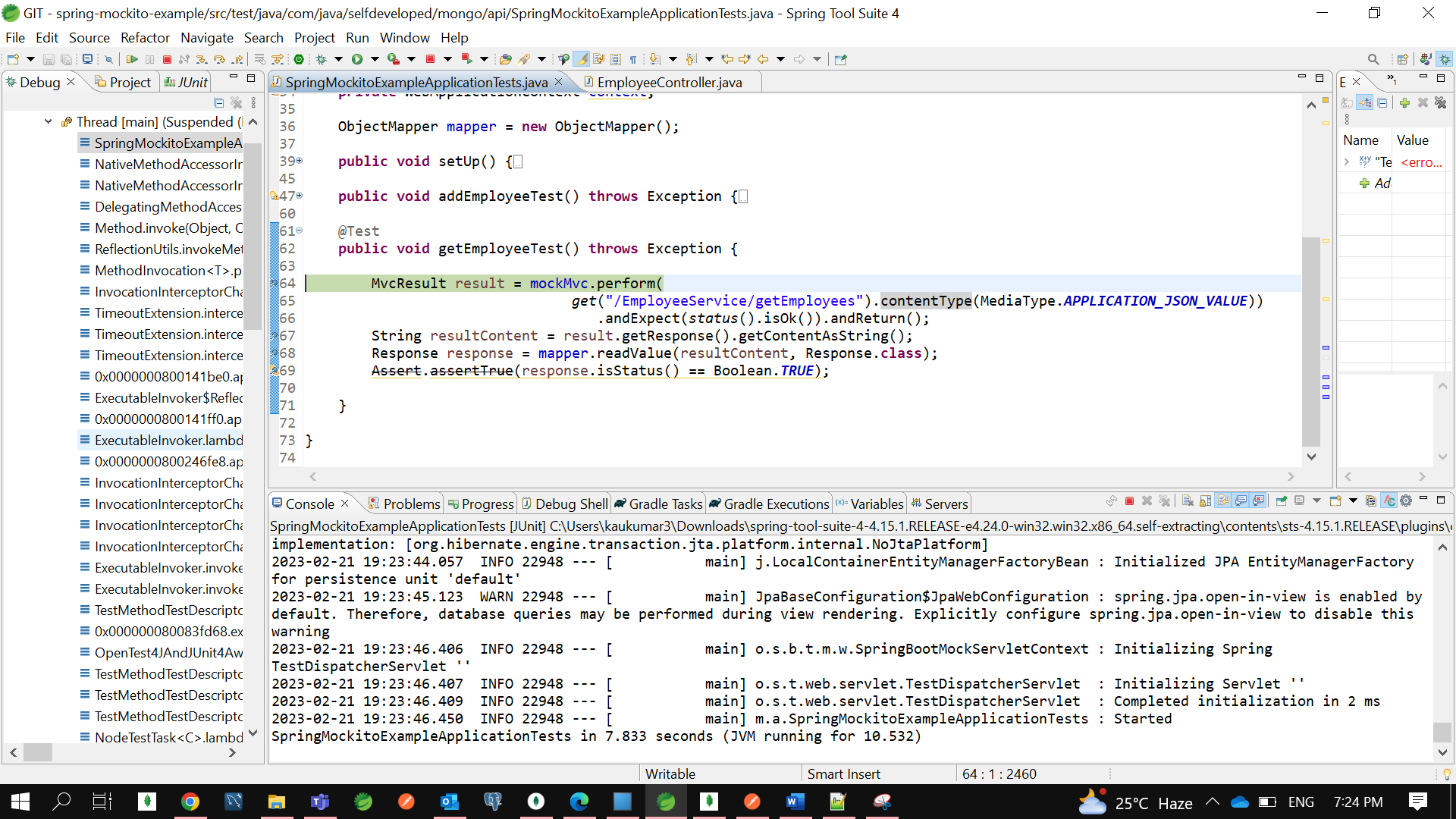
We have putted 3 break points in our Test Controller class for **getEmployee**() method…

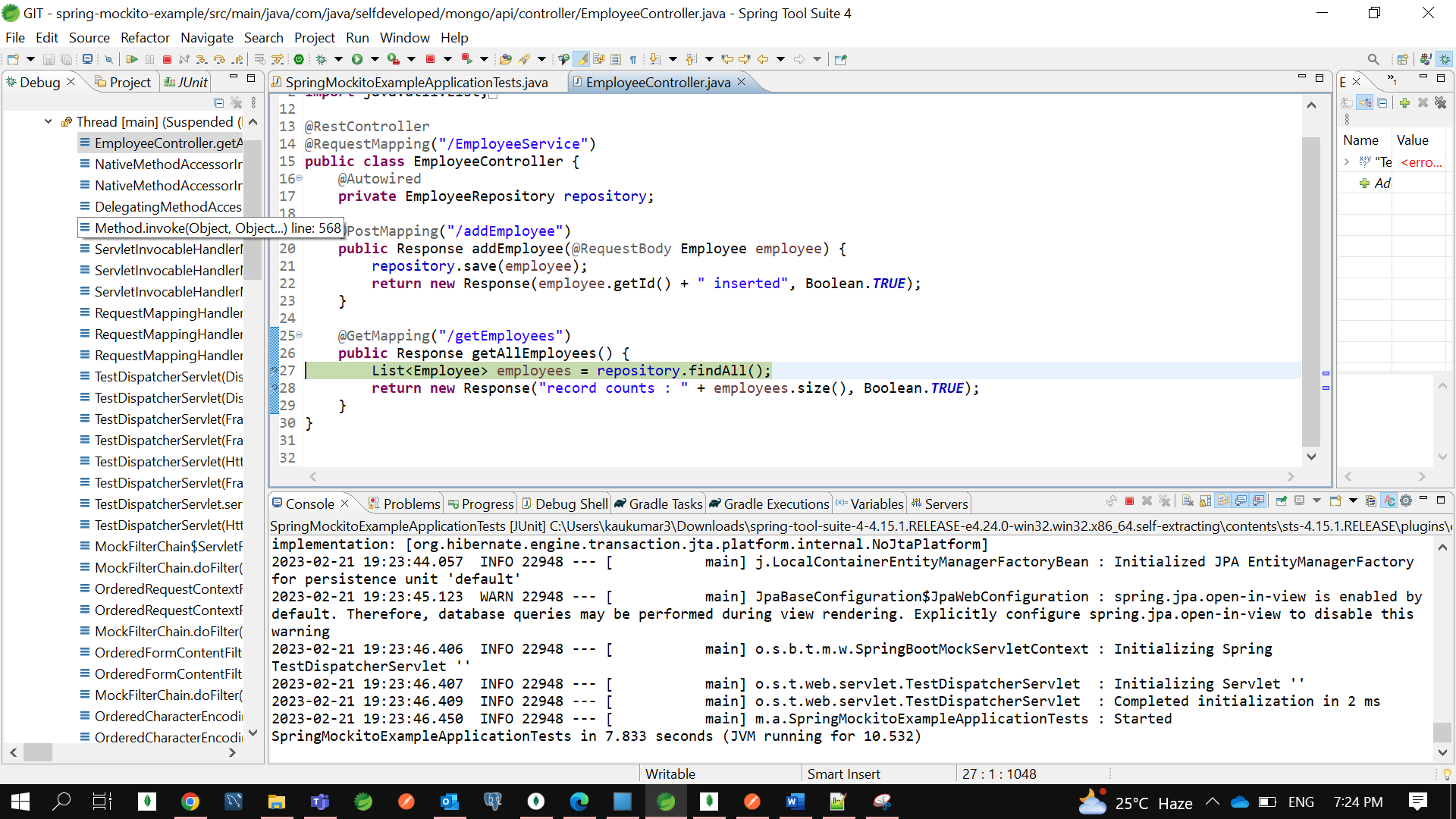


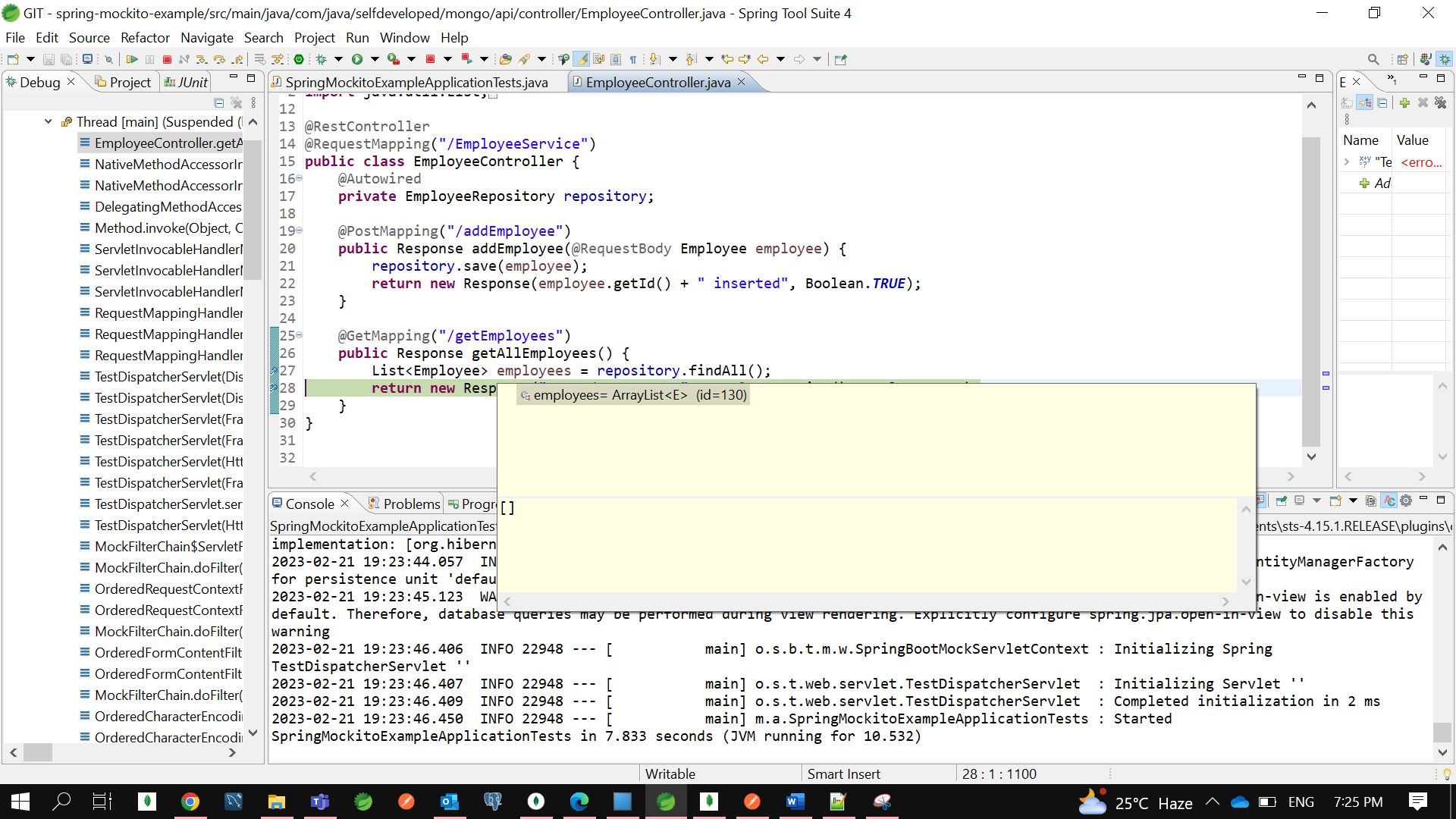
We have putted 2 break points in our Actual Controller for **getEmployee**() method…

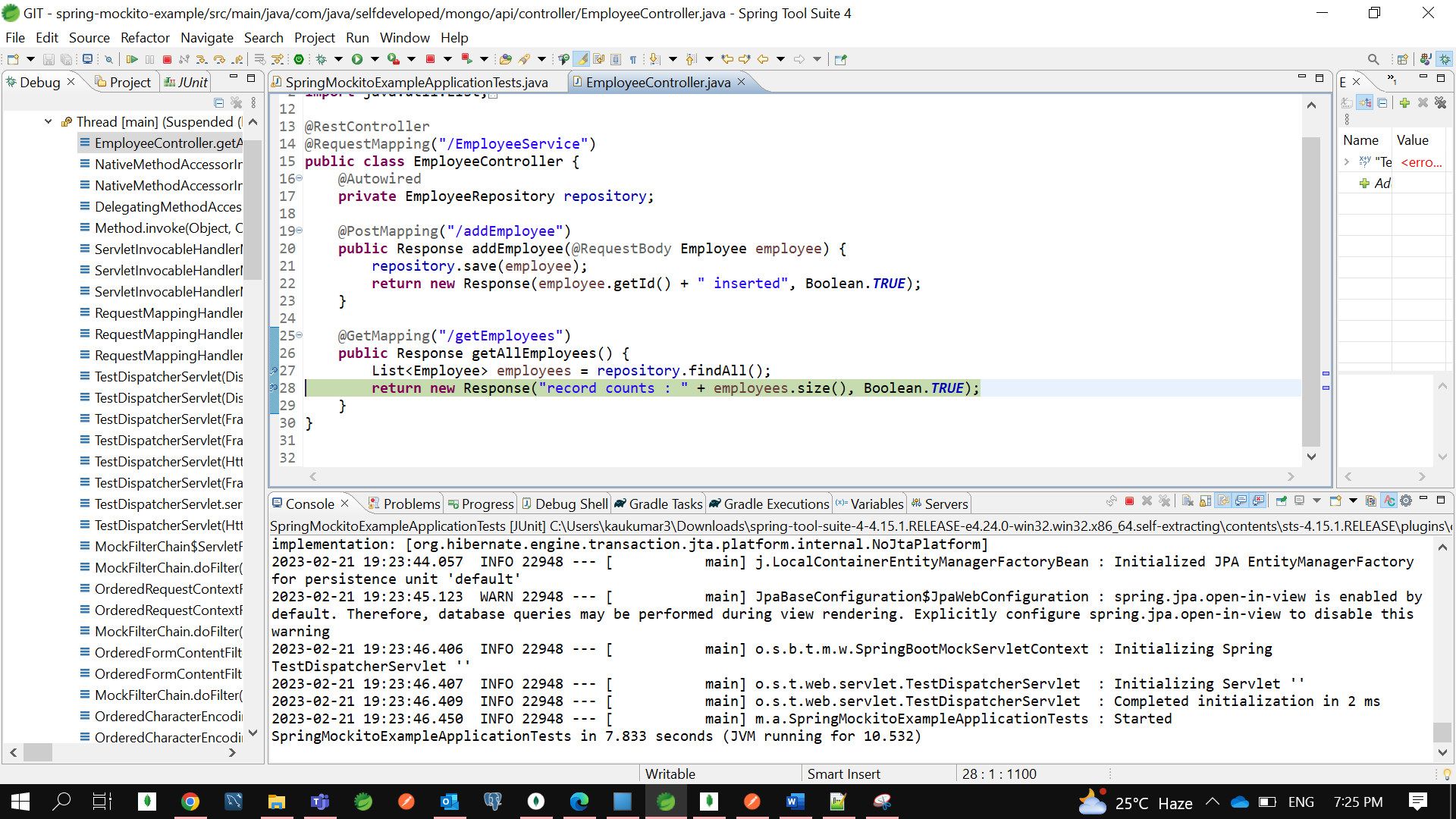


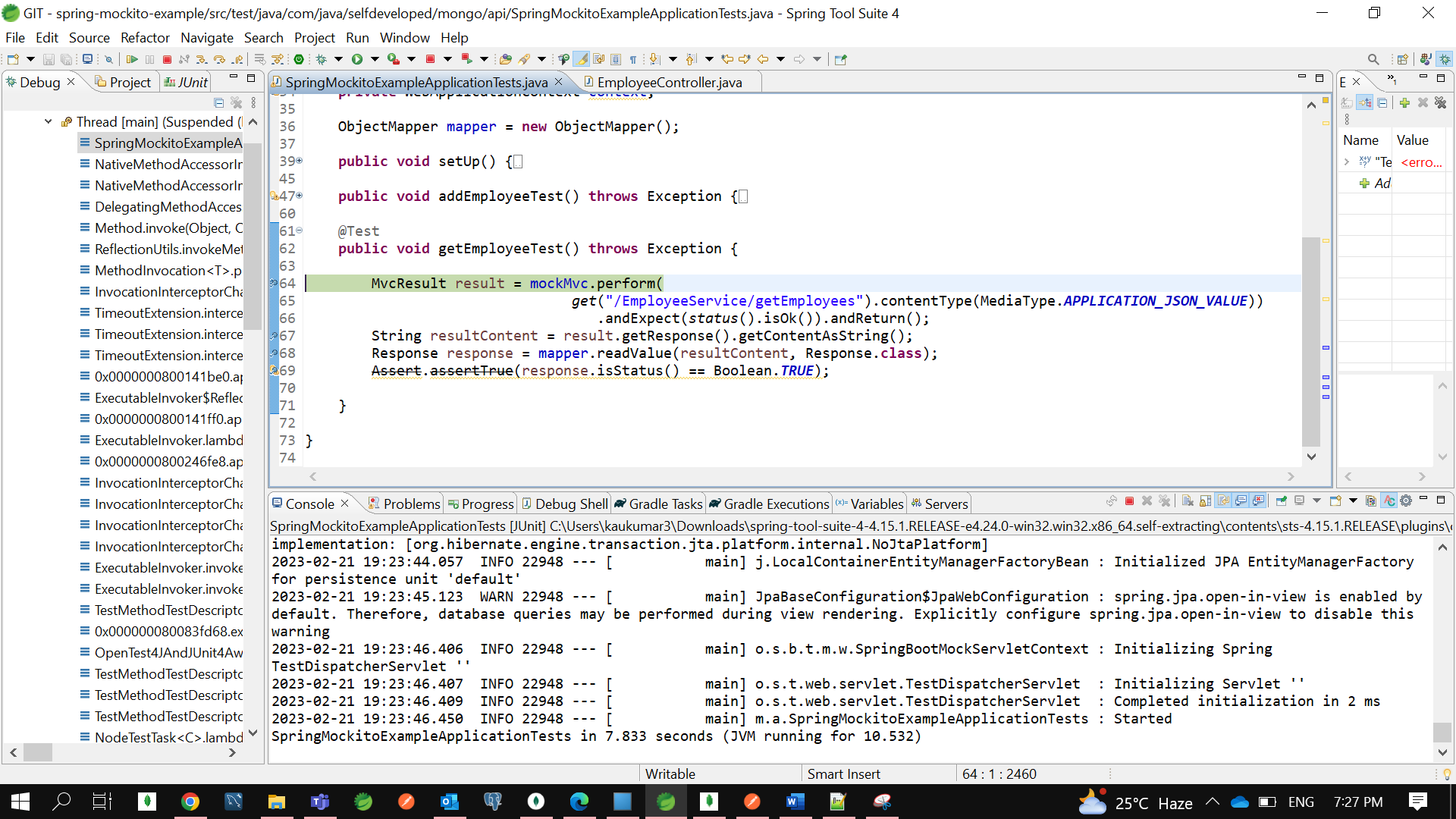
Let’s Run our Test class in a Debug Mode and trace it the flow….

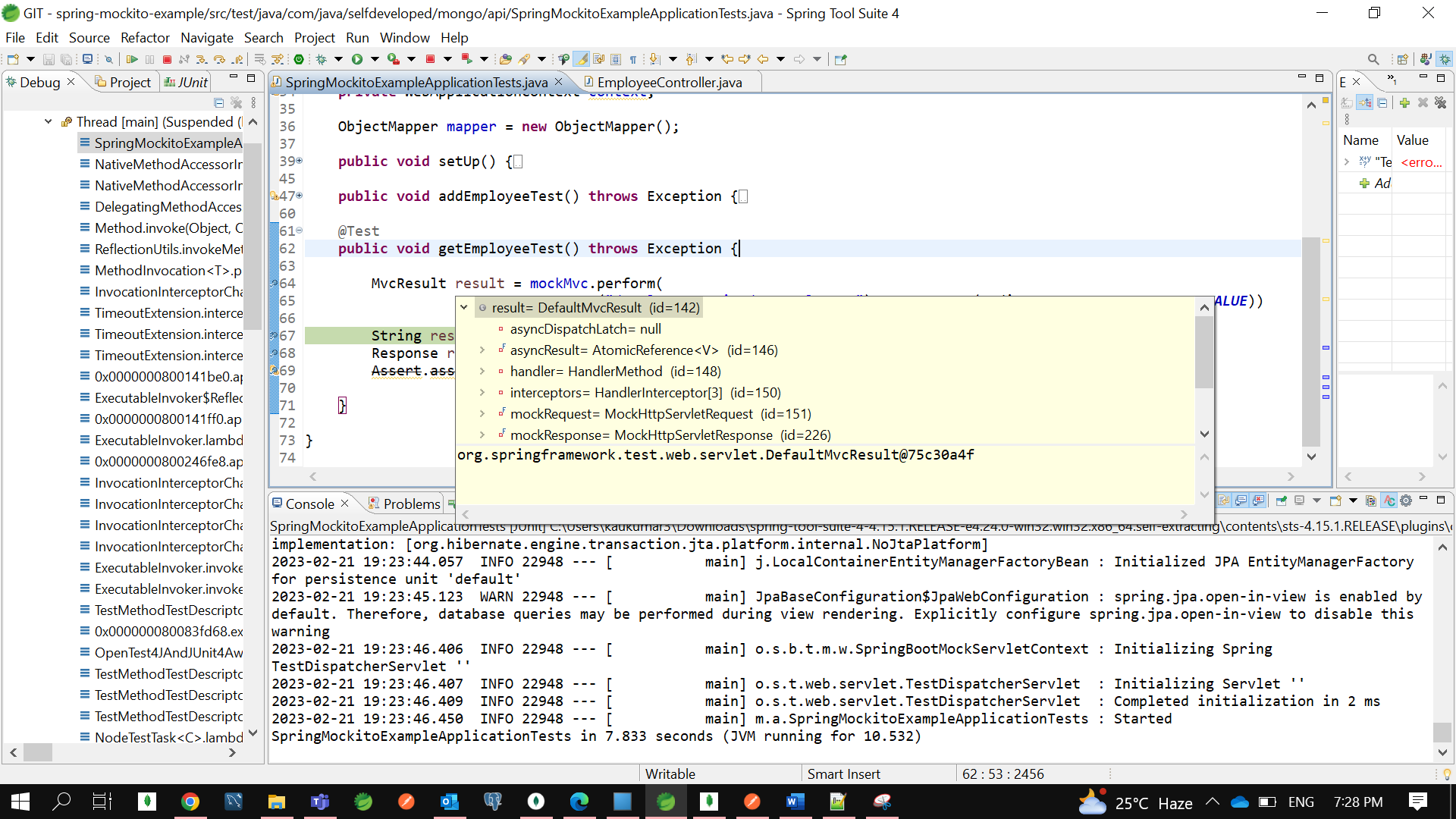


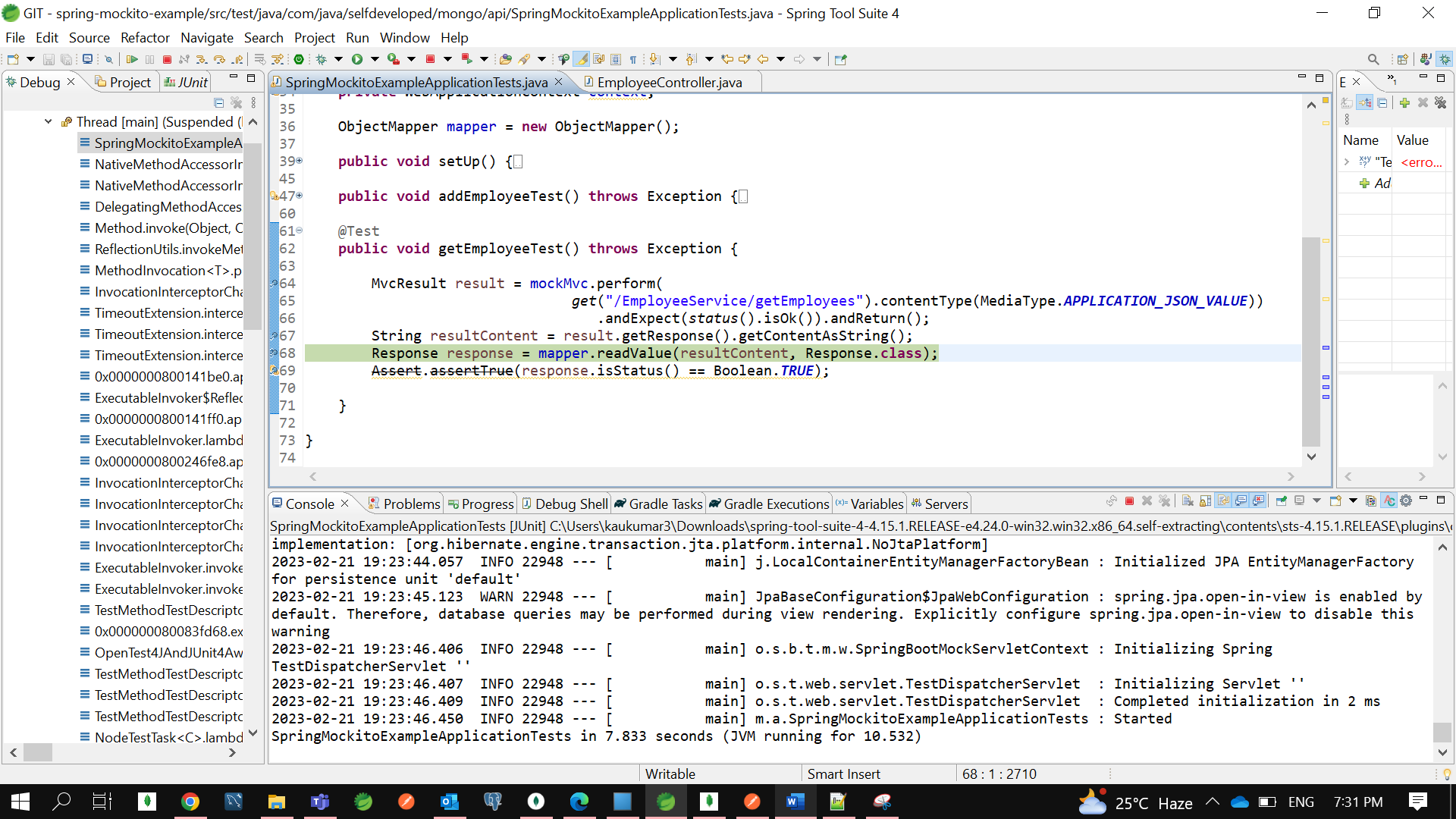


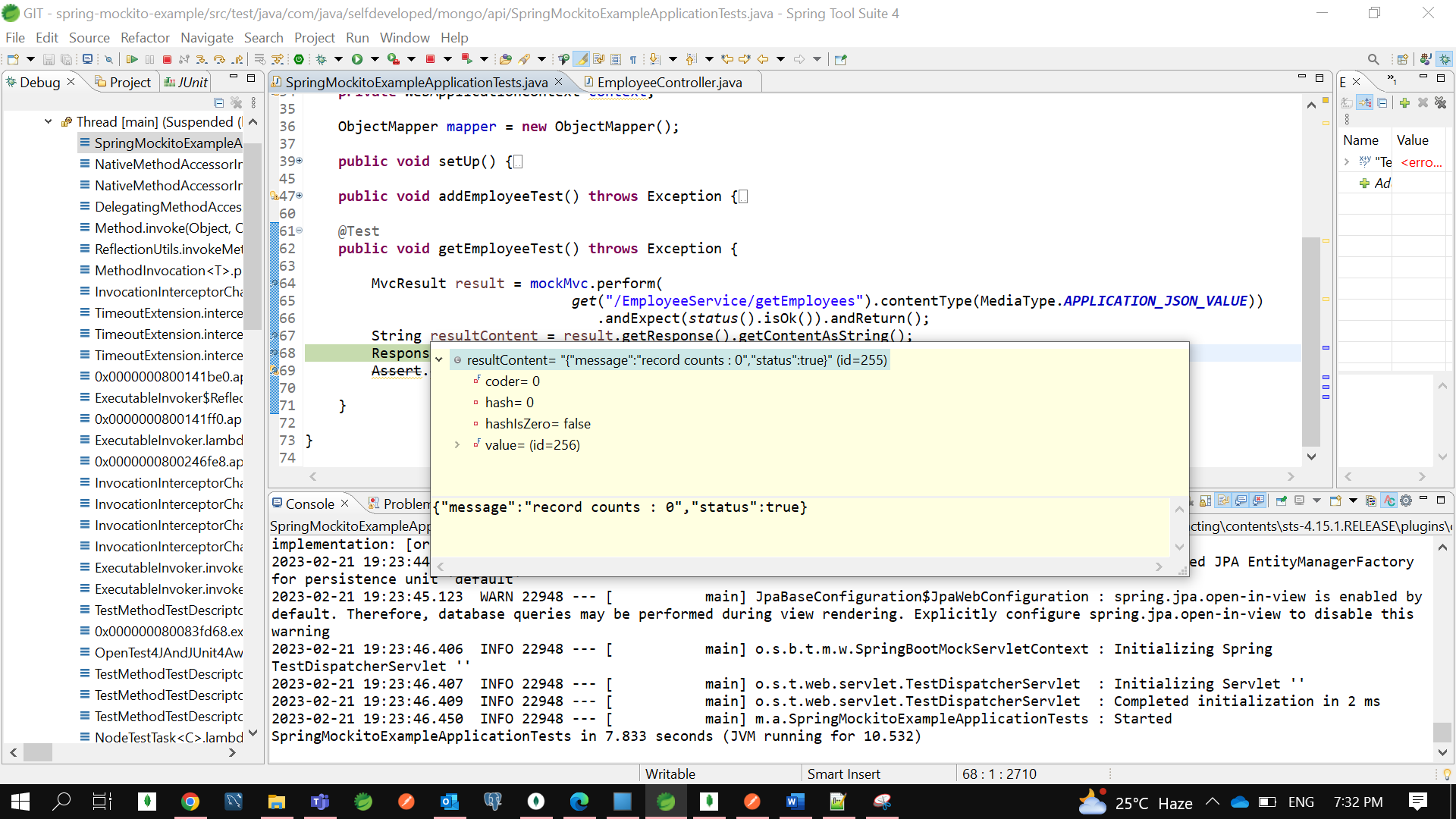


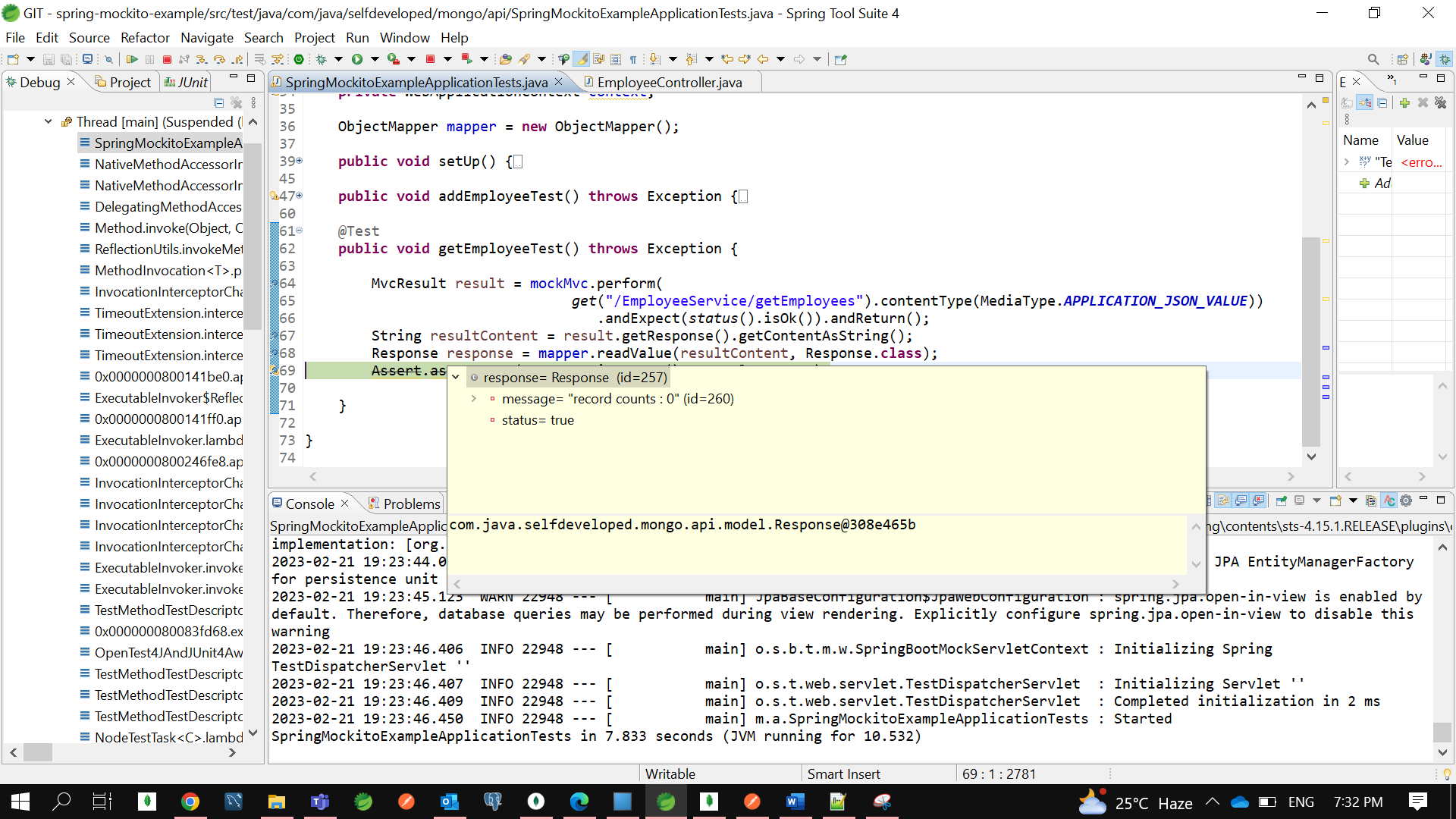


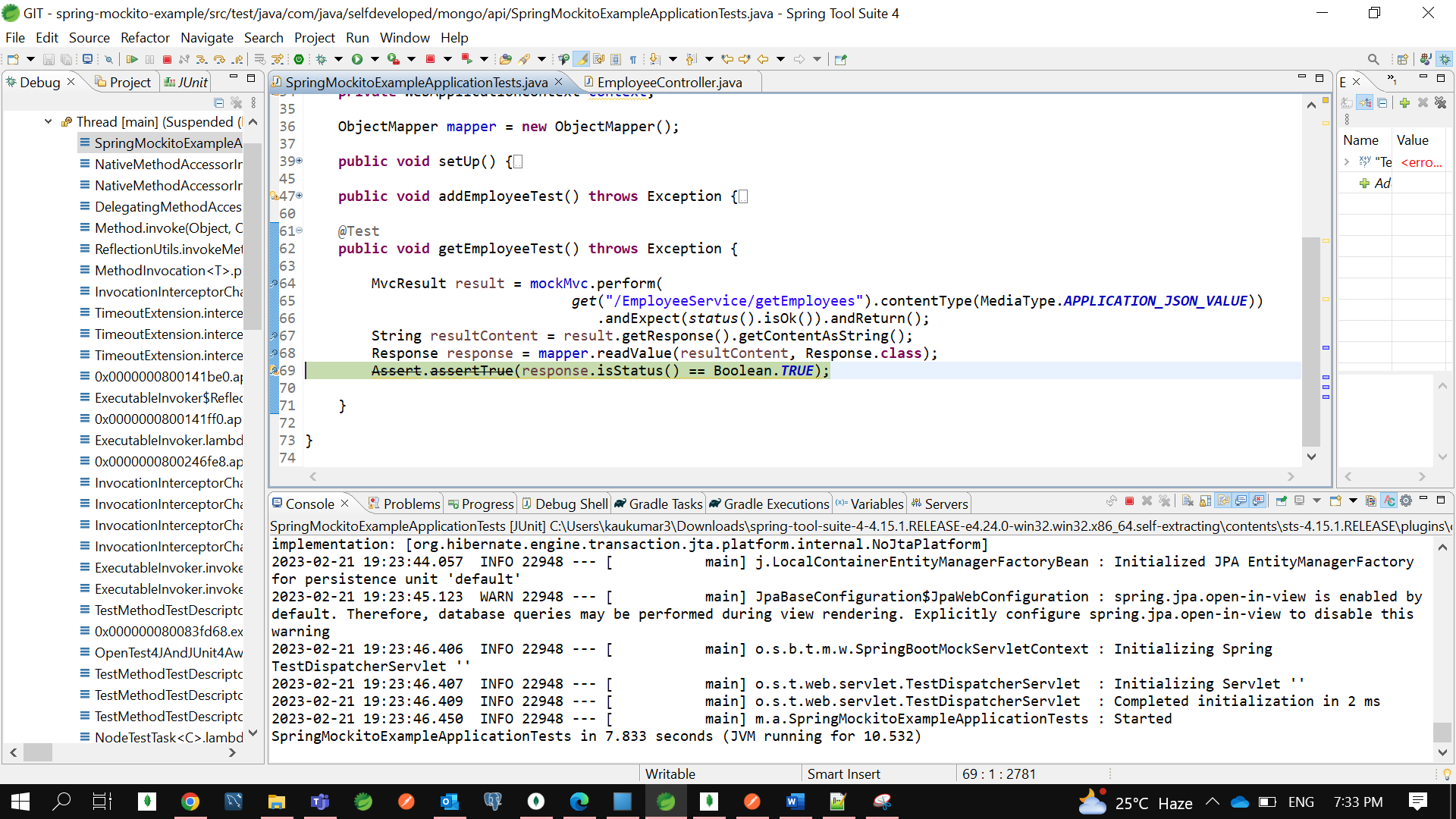


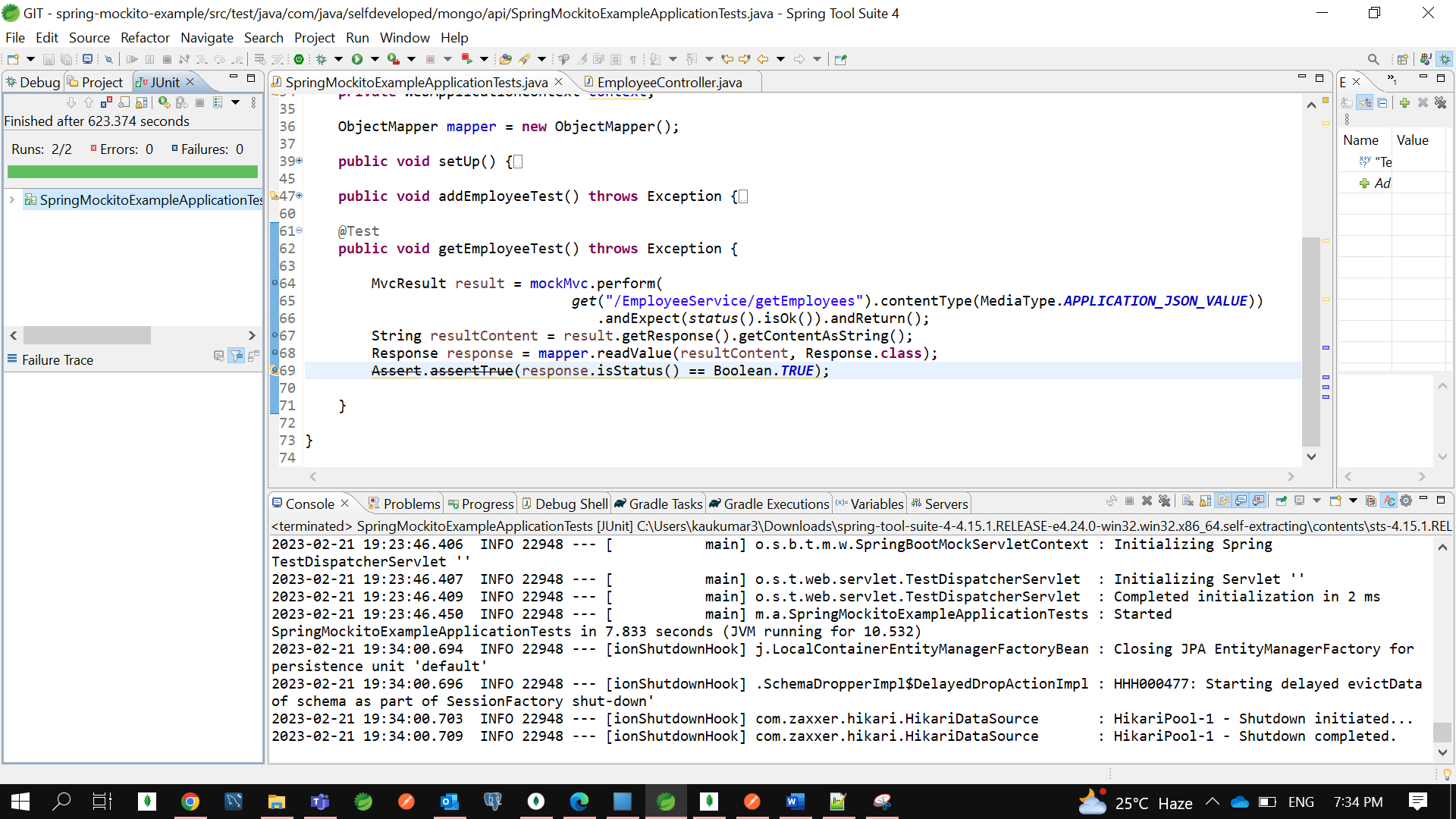






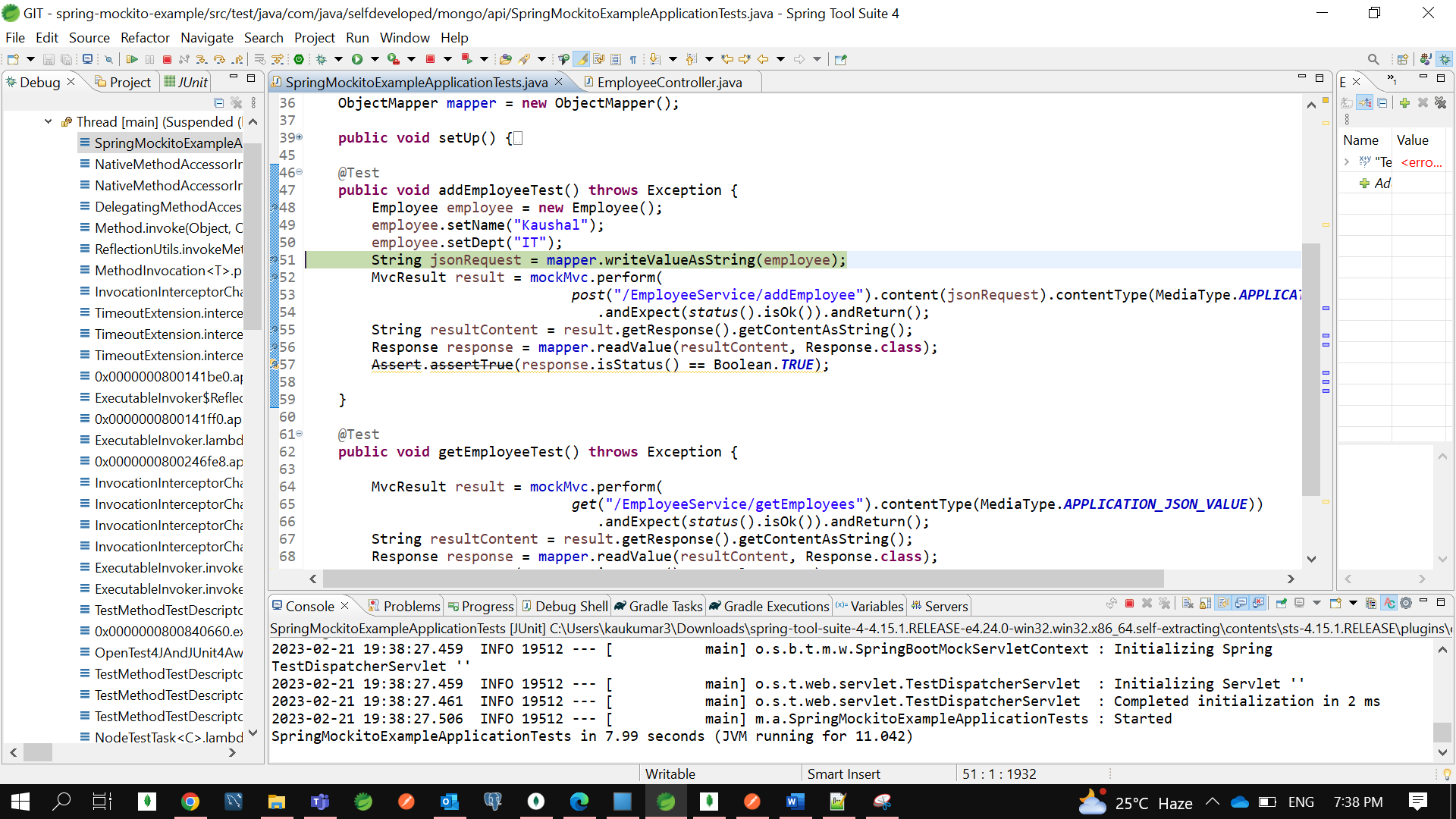


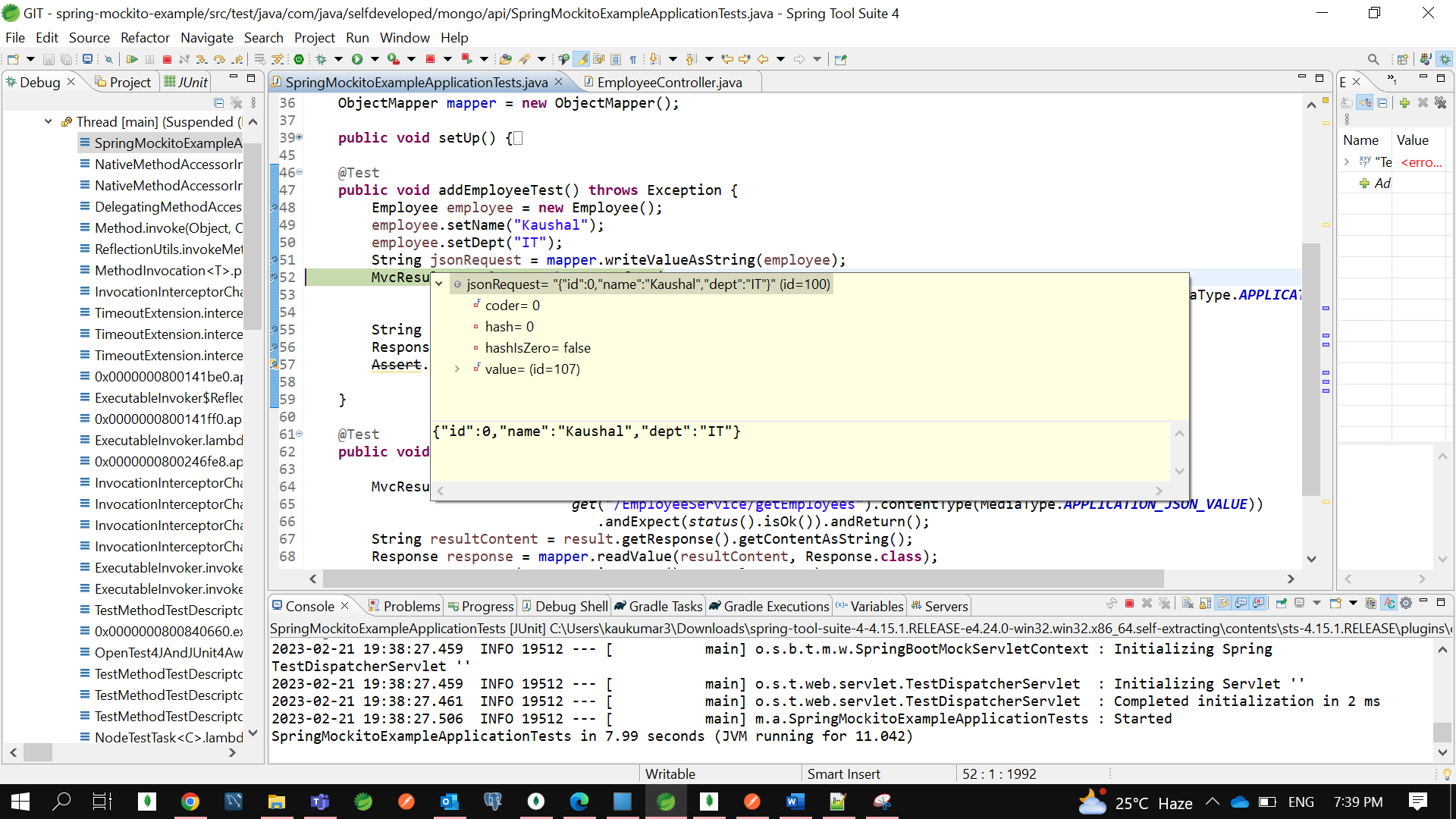


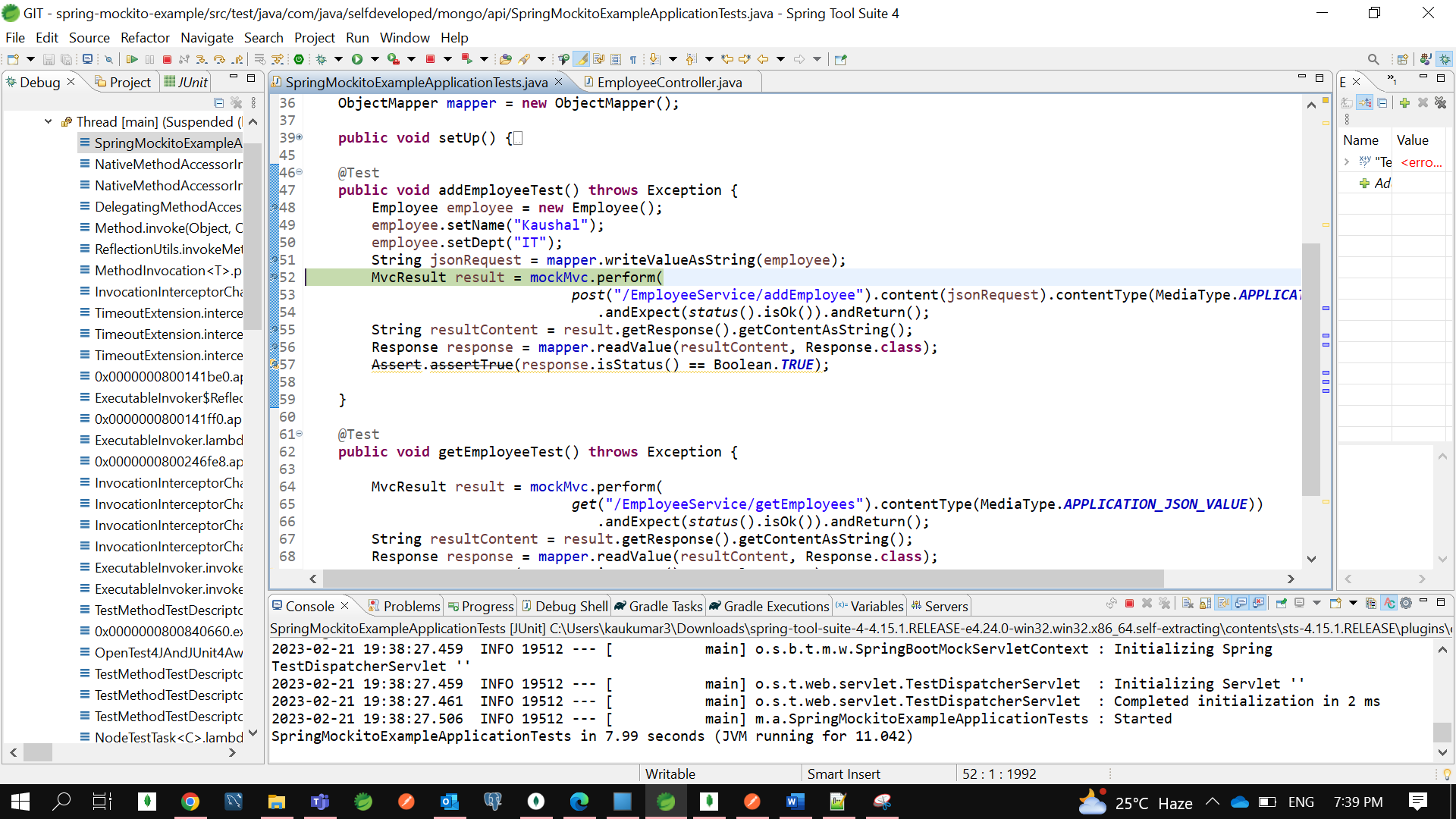


Now Let’s Trace it for **addEmployeeTest** class….

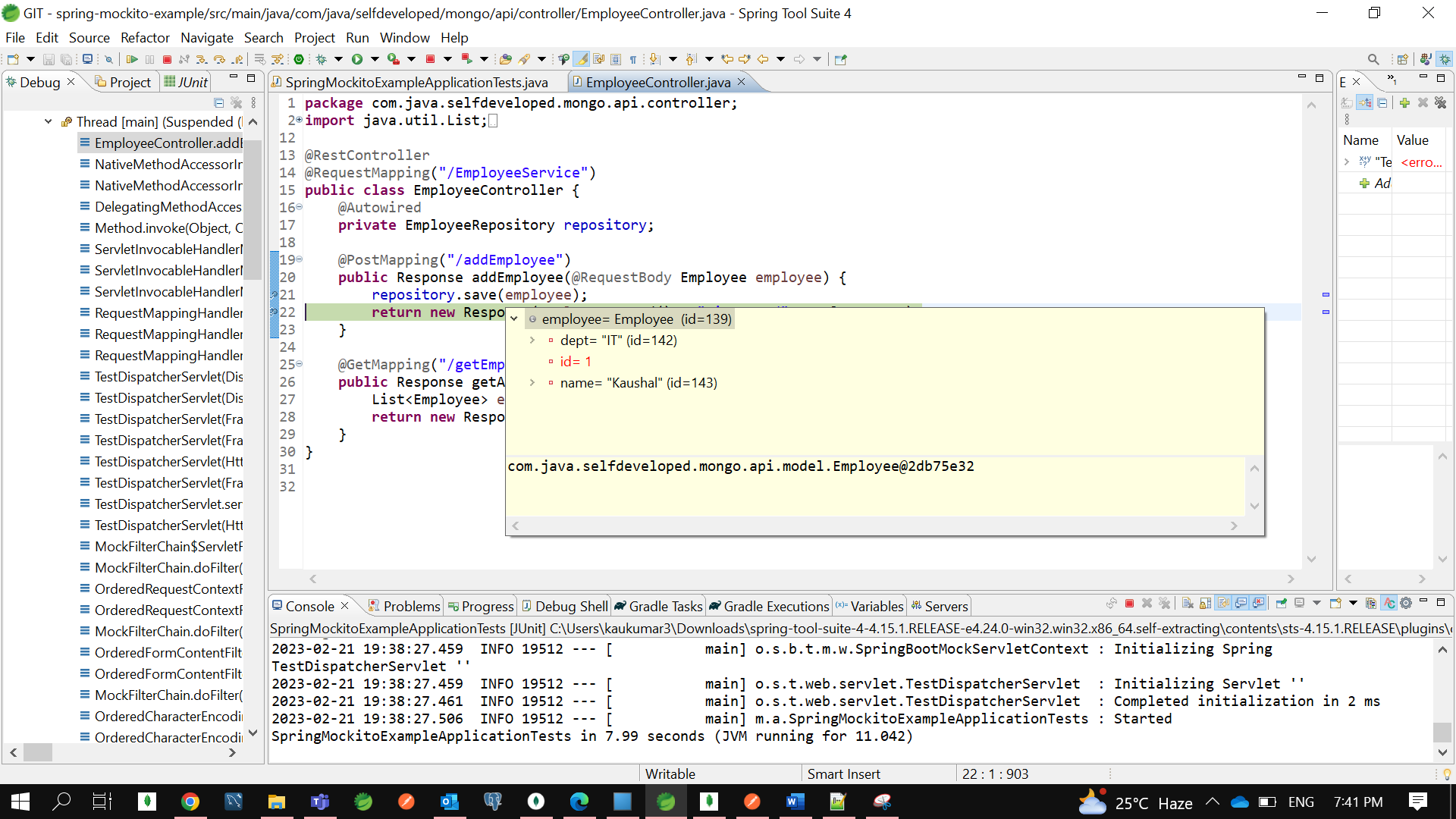
Let’s Debug it as Junit….

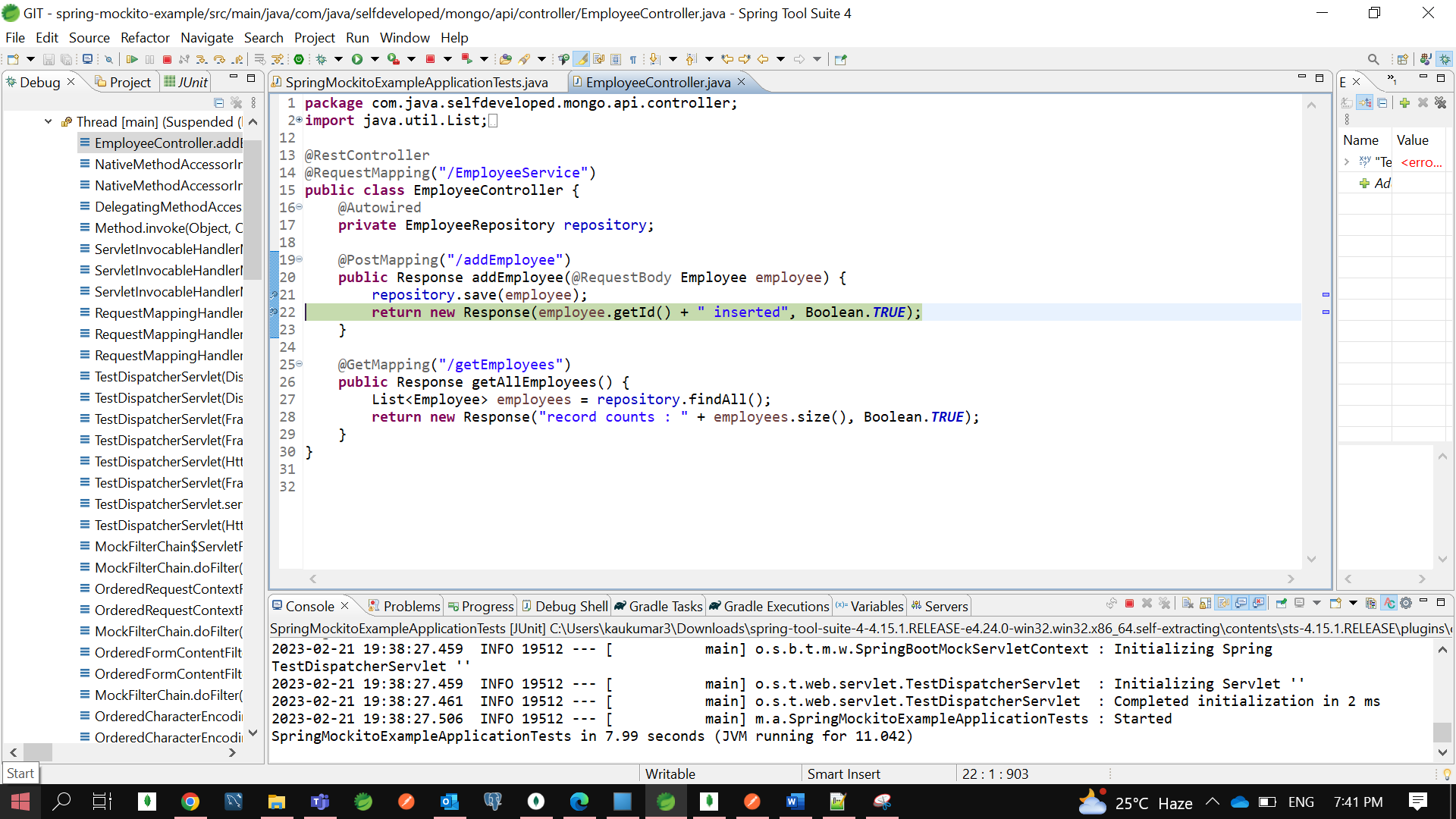


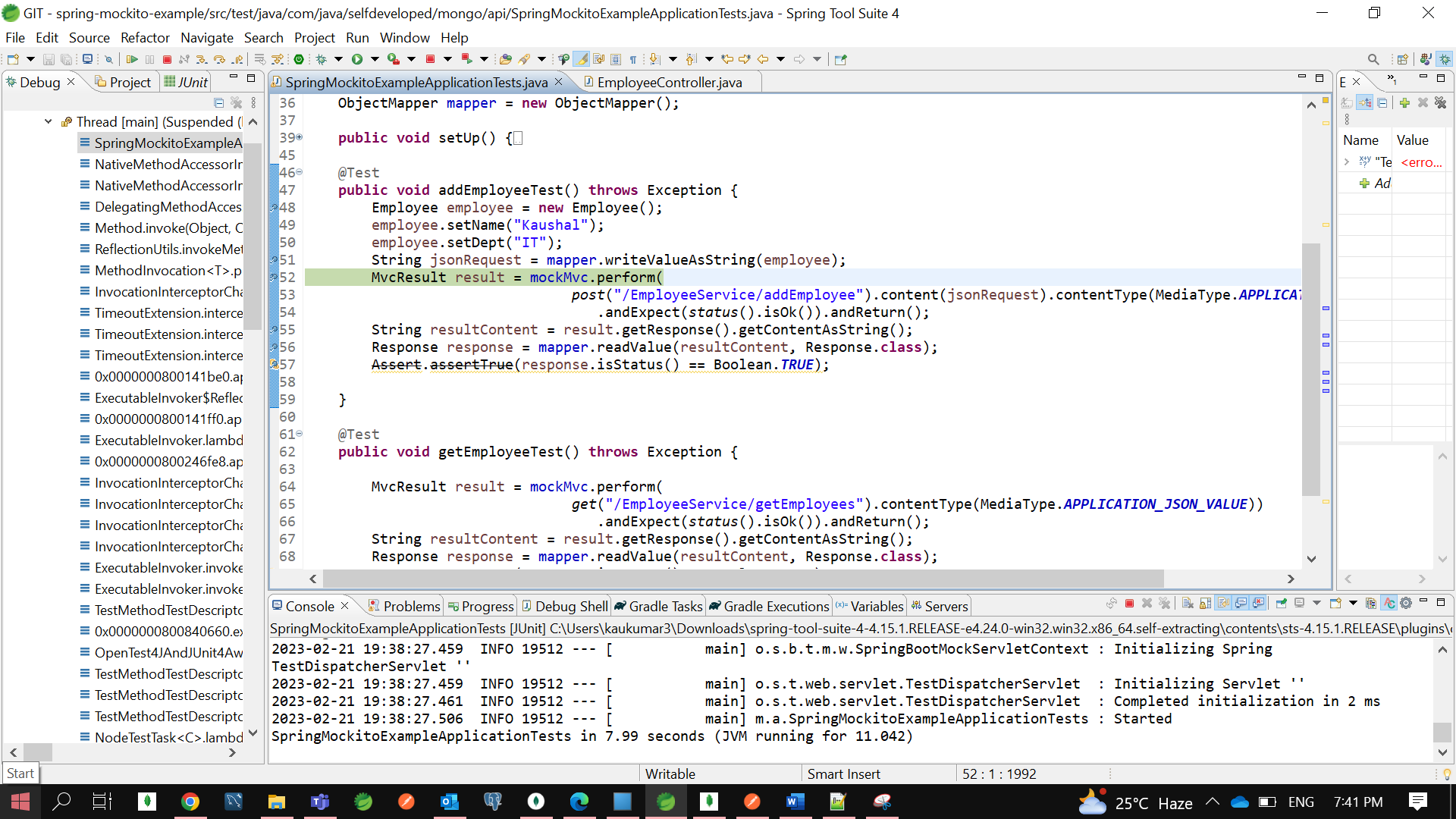


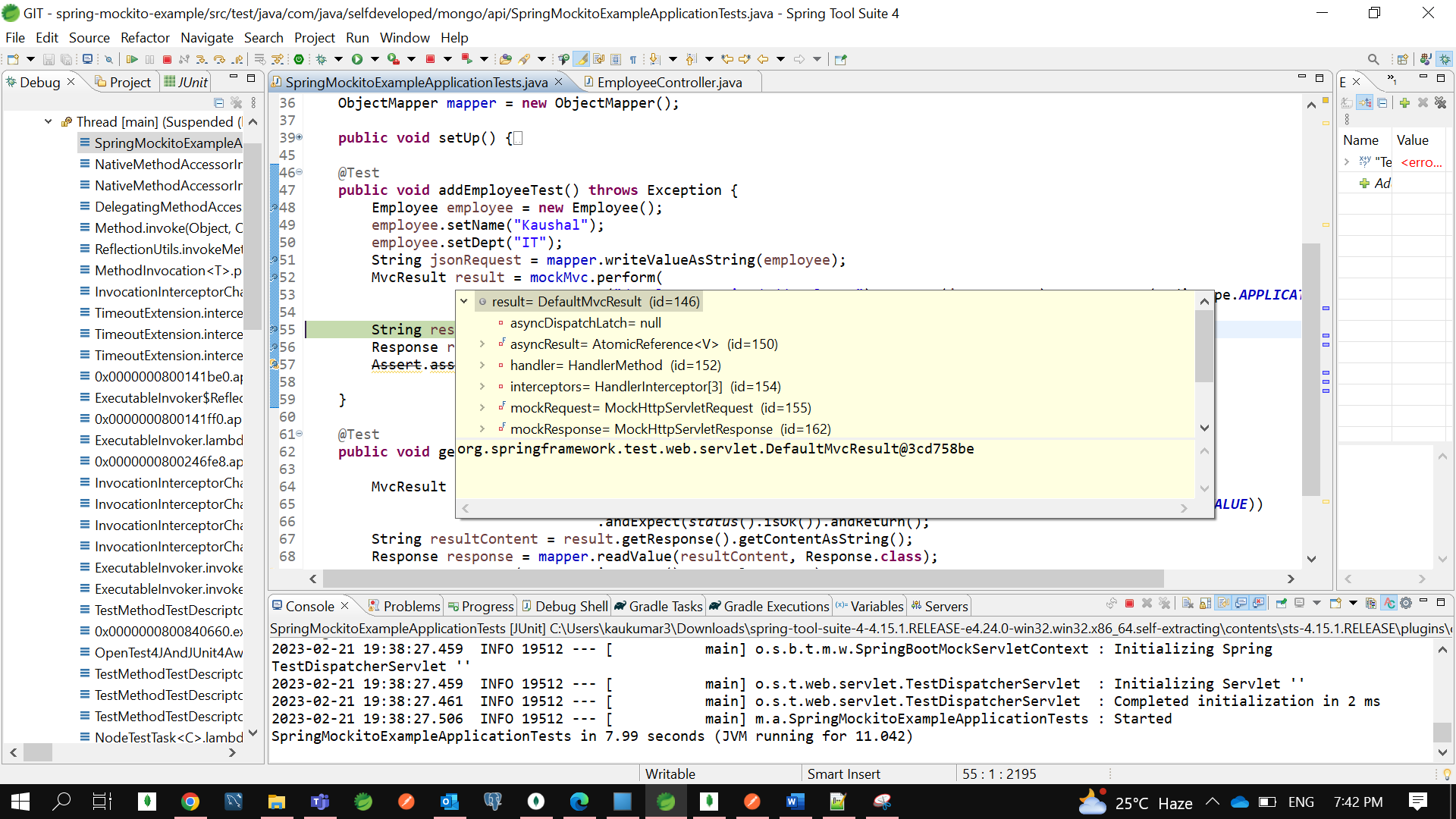


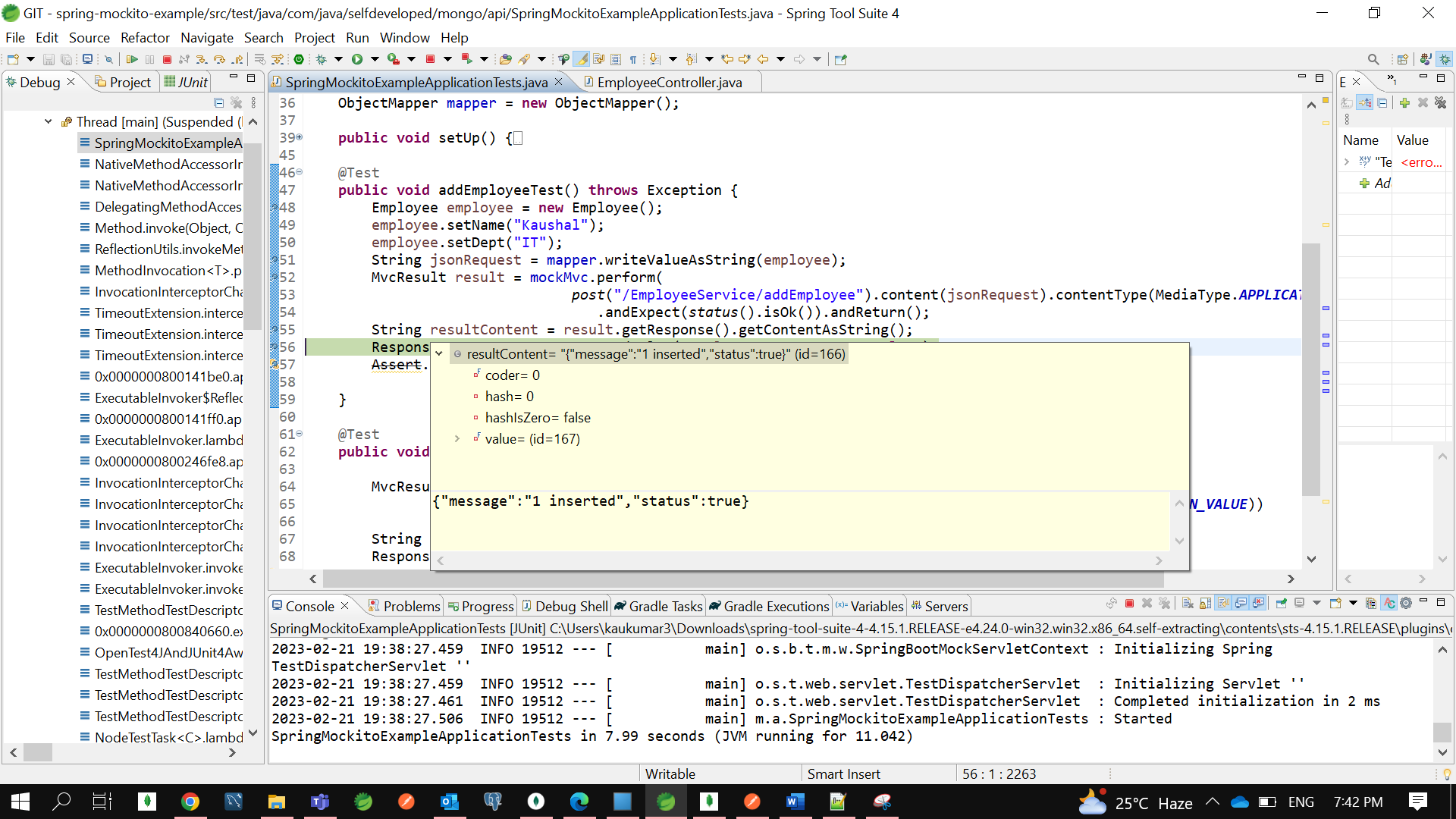


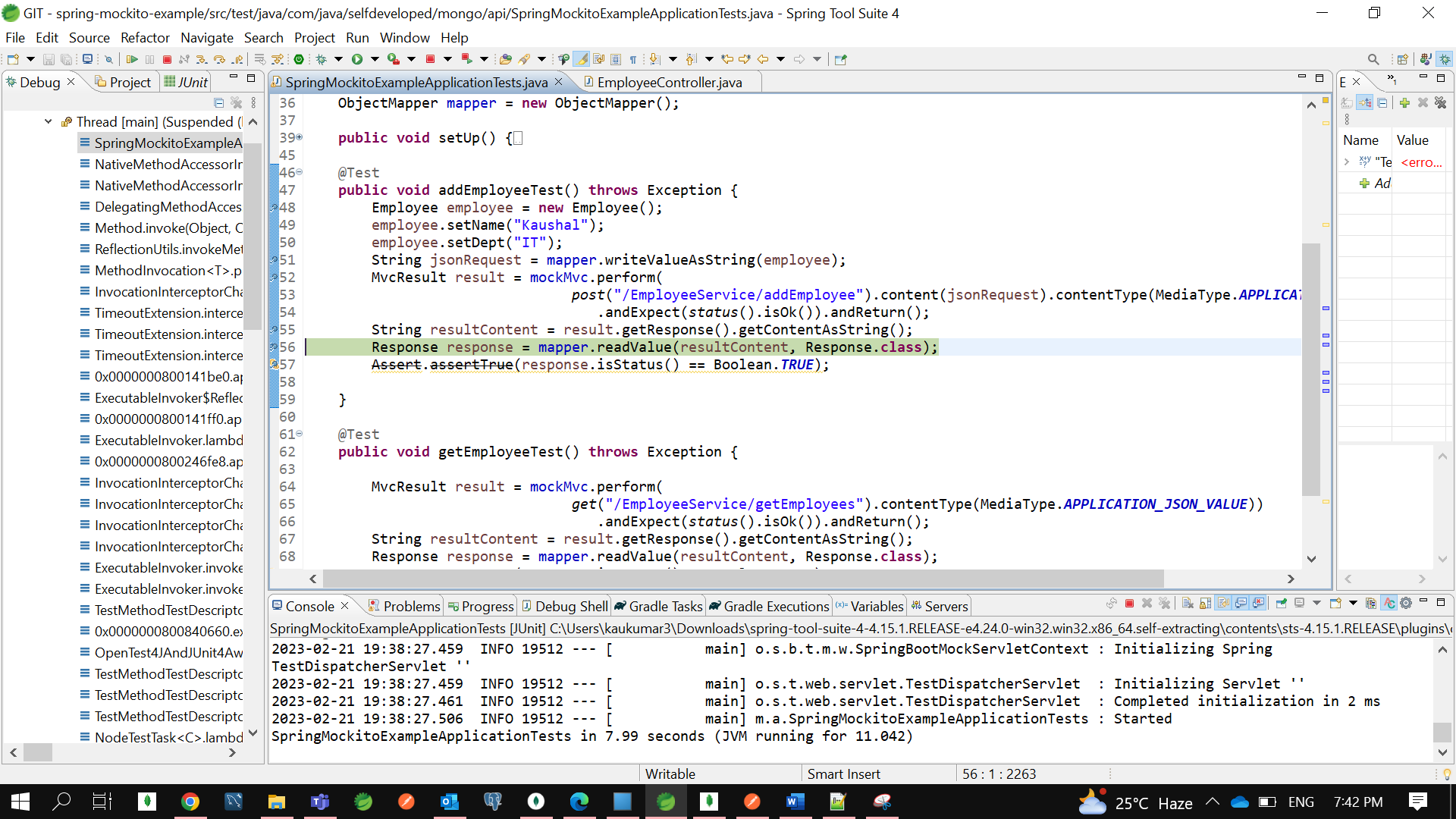


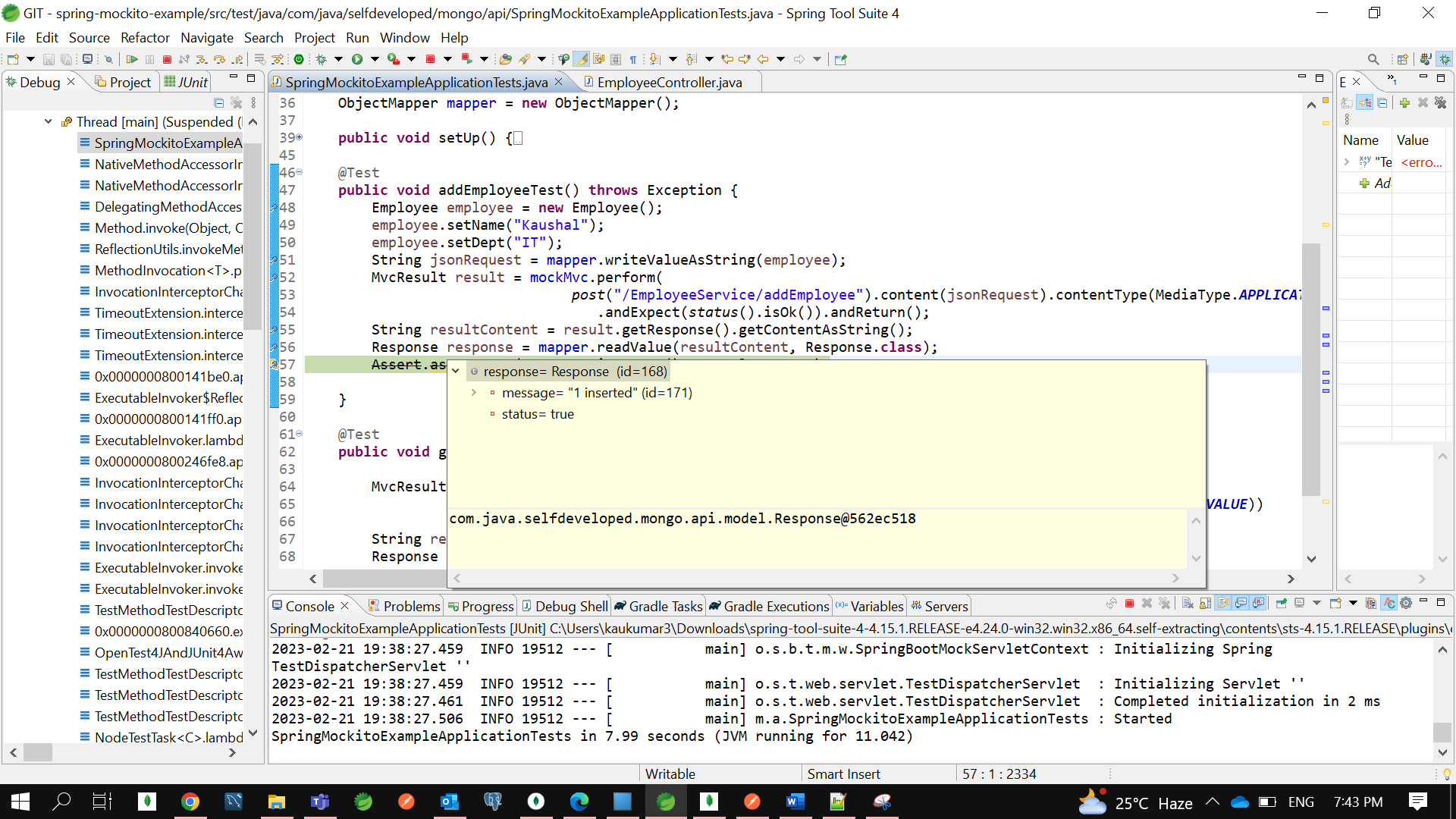


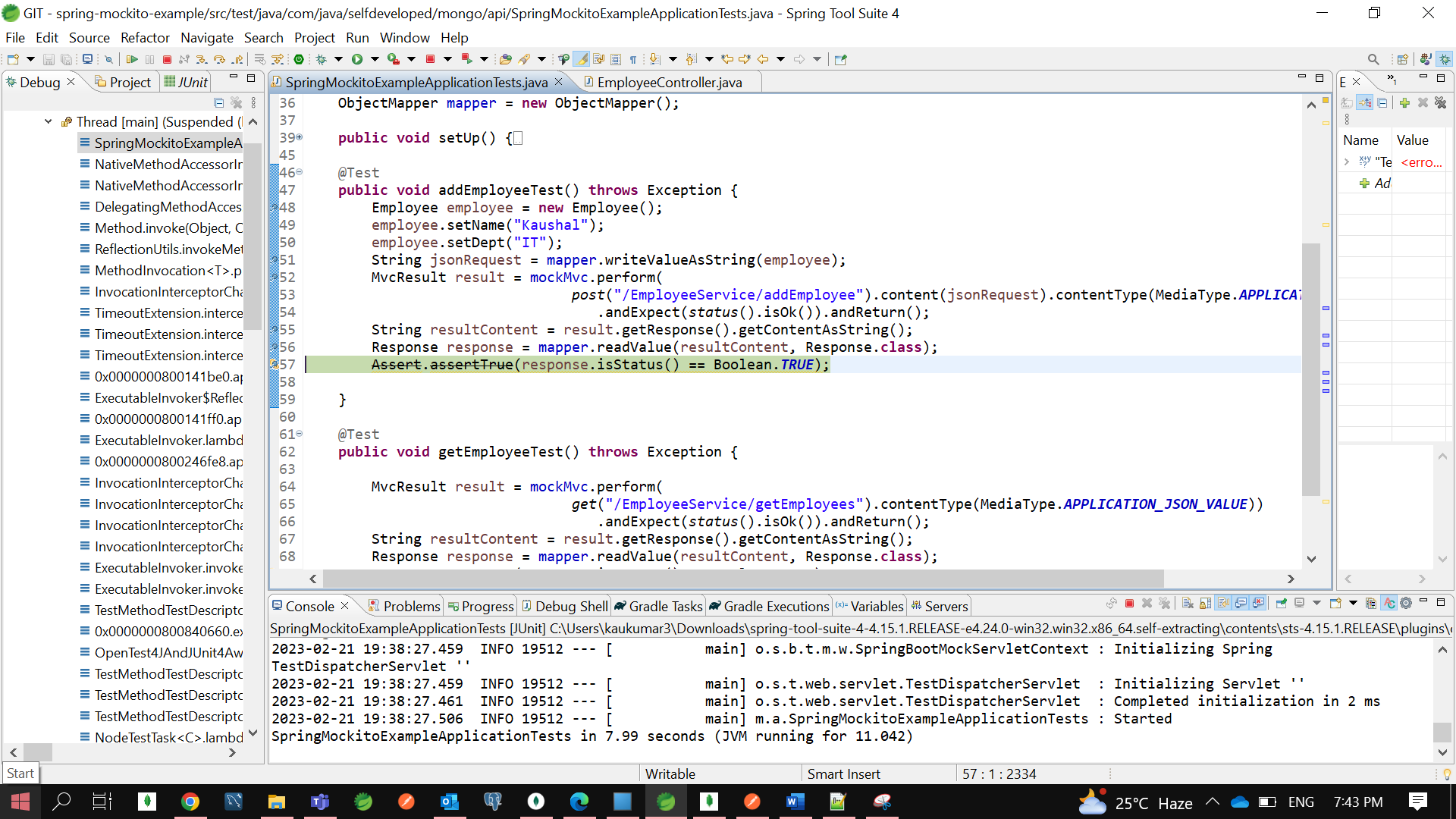


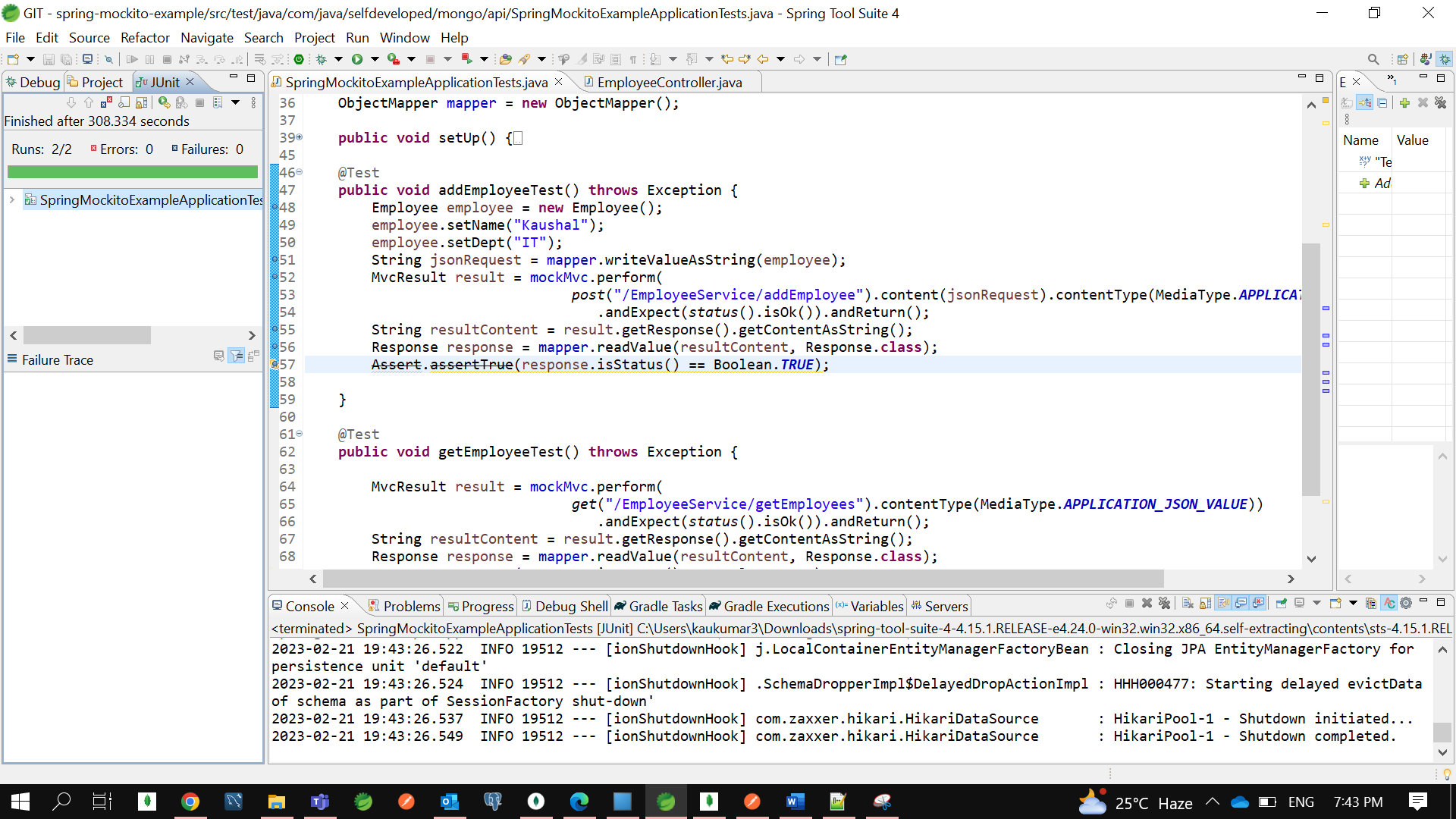












Let’s run together …

