Spring Testing Exercises

Exercise 1: Basic Unit Test for a Service Method Task: Write a unit test for a service method that adds two numbers.

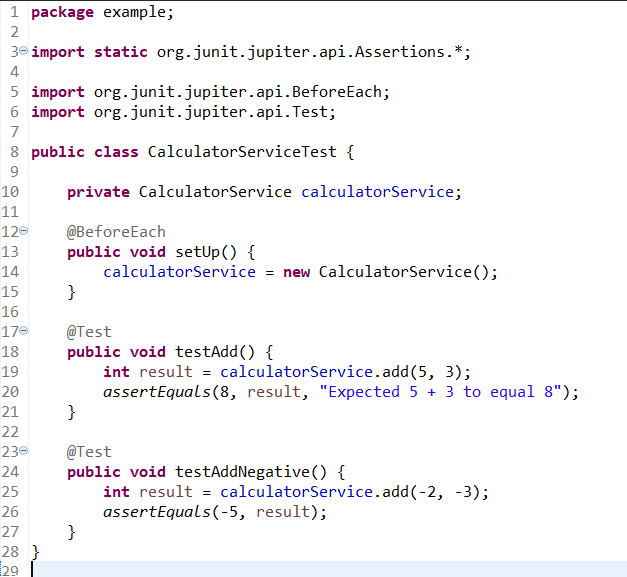
Service:

@Service public class CalculatorService { public int add(int a, int b) { return a + b;

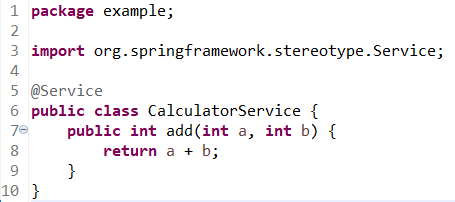
}

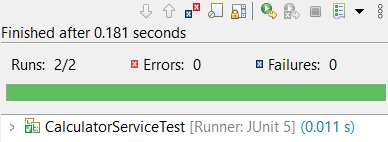
}

**CalculatorServiceTest.java:**



**CalculateService.java:**





Exercise 2: Mocking a Repository in a Service Test Task: Test a service that uses a repository to fetch data.

Entity:

@Entity public class User { @Id private Long id; private String name; // getters and setters

}

Repository:

public interface UserRepository extends JpaRepository<User, Long> {

}

Service:

@Service public class UserService { @Autowired private UserRepository userRepository;

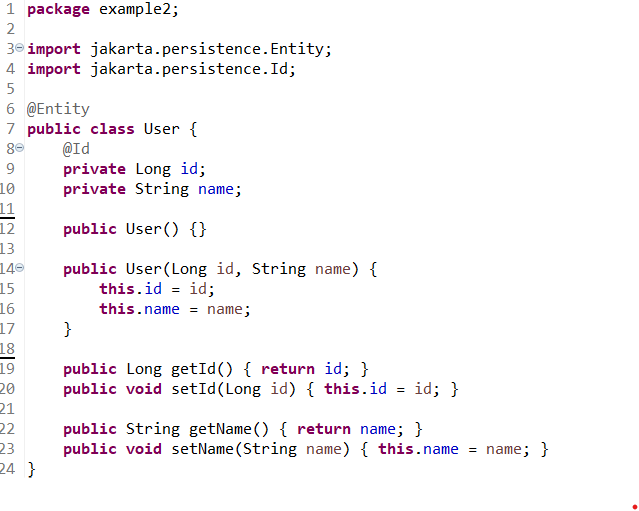
public User getUserById(Long id) {

return userRepository.findById(id).orElse(null);

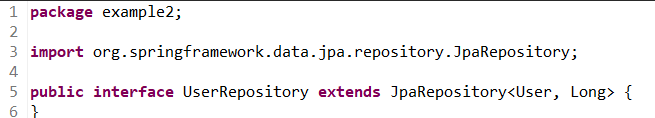
}

}

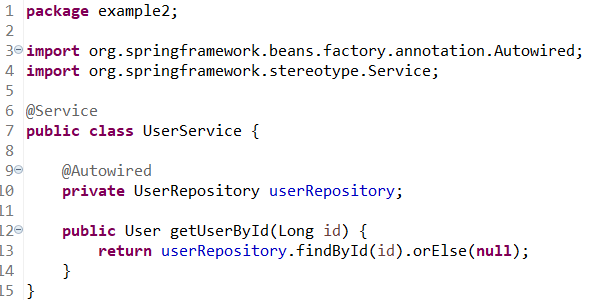
User.java:



**UserRepository.java:**

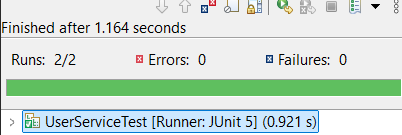


**UserService.java:**



**UserServiceTest.java:**





Exercise 3: Testing a REST Controller with MockMvc Task: Test a controller endpoint that returns a user.

Controller:

@RestController

@RequestMapping("/users") public class UserController {

@Autowired private UserService userService;

@GetMapping("/{id}") public ResponseEntity<User> getUser(@PathVariable Long id) { return ResponseEntity.ok(userService.getUserById(id));

}

}

Test:

Write code for this.

# Exercise 4: Integration Test with Spring Boot

Task: Write an integration test that tests the full flow from controller to database.

Test:

Write code for this.

Exercise 5: Test Controller POST Endpoint Task: Test a POST endpoint that creates a user.

Controller:

@PostMapping public ResponseEntity<User> createUser(@RequestBody User user) { return ResponseEntity.ok(userService.saveUser(user));

}

Test:

Write code for this.

Exercise 6: Test Service Exception Handling Task: Test how a service handles a missing user.

Test:

Write code for this.

Exercise 7: Test Custom Repository Query Task: Add and test a custom query method.

Repository:

public interface UserRepository extends JpaRepository<User, Long> {

List<User> findByName(String name);

}

Test:

Write code for this.

Exercise 8: Test Controller Exception Handling Task: Add and test a @ControllerAdvice for handling exceptions.

Exception Handler:

@ControllerAdvice

public class GlobalExceptionHandler {

@ExceptionHandler(NoSuchElementException.class)

public ResponseEntity<String> handleNotFound(NoSuchElementException ex) { return ResponseEntity.status(HttpStatus.NOT\_FOUND).body("User not found");

}

}

Test:

Write code for this.

Exercise 9: Parameterized Test with JUnit Task: Use @ParameterizedTest to test multiple inputs.

Test:

Write code for this.