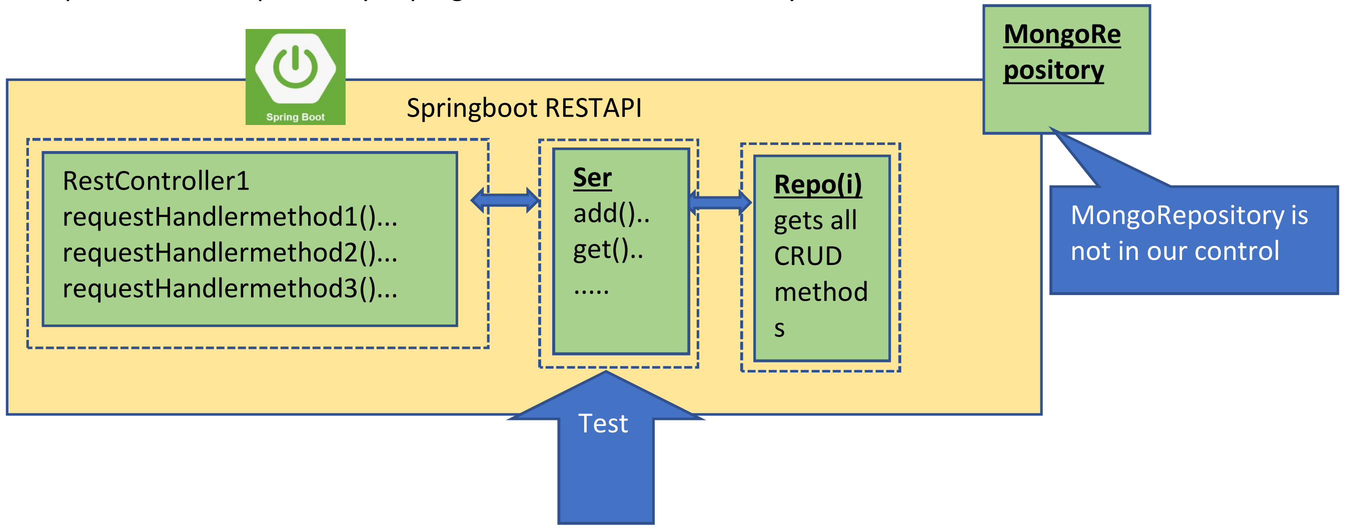
while testing CustomerRepository

Dependent is 'CustomerRepository'
Dependency is 'MongoRepository'

If both Dependent and Dependency in programmers controll, then only mocker environment can be created for test


MongoRepository
Spring Boot
Springboot RESTAPI
RestController1
requestHandlermethod1()...
requestHandlermethod2()...
requestHandlermethod3()...
Ser
add()..
get()..
.....
Repo(i)
gets all CRUD methods
MongoRepository is not in our control
Test

While testing service layer

Dependency is CustomerRepository
Dependent is ServiceImpl class

When both are in programmers control, mocked setup can be created

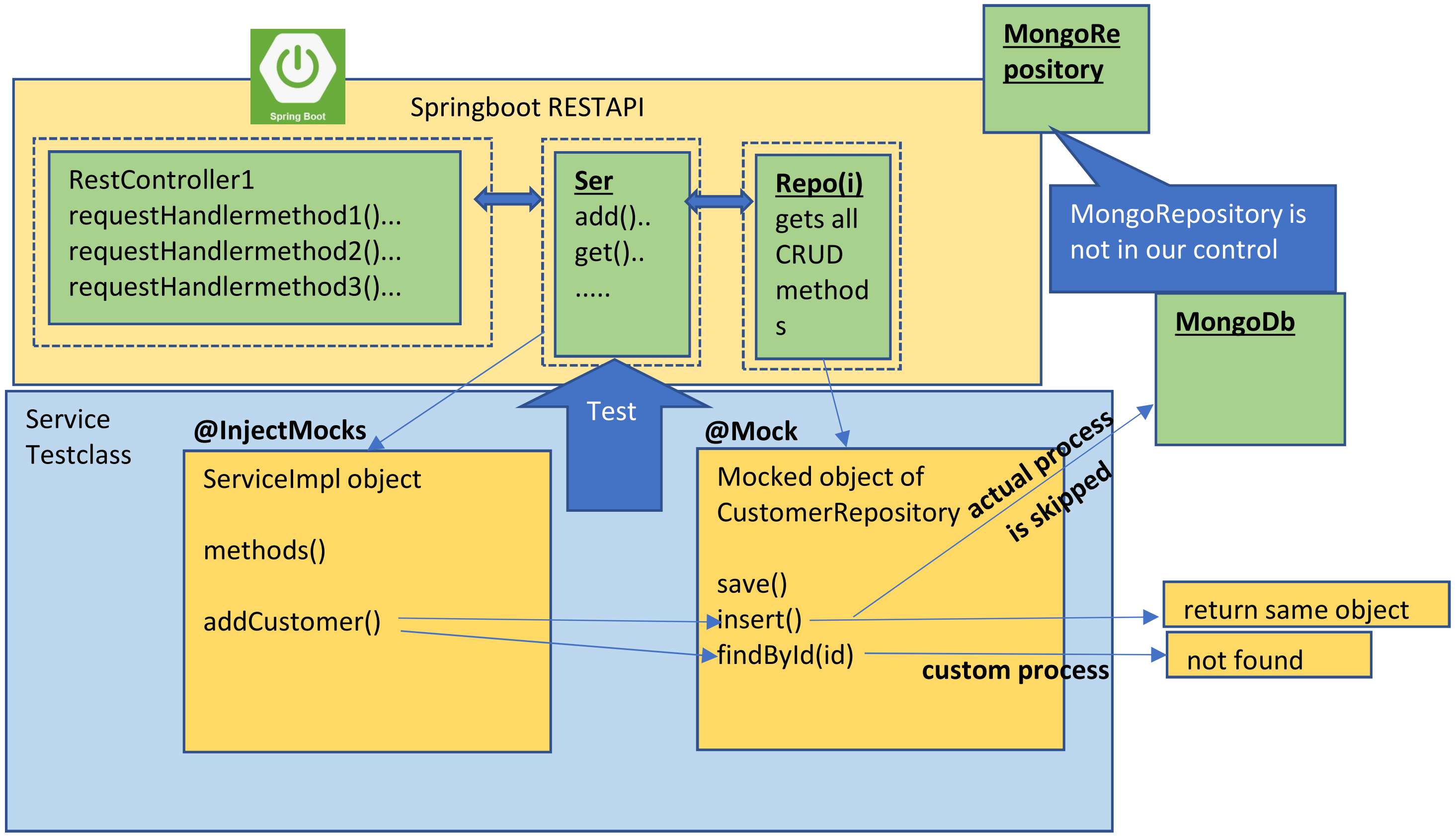
Mocked environment

ServiceImpl class needs CustomerRepository

By making mocked object of CustomerRepository
we can make customerRepository to stop the actual work and performs our customized task

ex.
CustomerRepository.findById(id)
    actual process of this method : finding record in db based on received id

But, mocked object of CustomerRepository.findById(id)
    skips actual process
    by skipping, performs customezed task



@Mock
To be used to create dependency object
CustomerRepository

@InjectMocks
To be used to create dependent object
so, dependent object can use mocked object of dependency
CustomerServiceImpl

example1

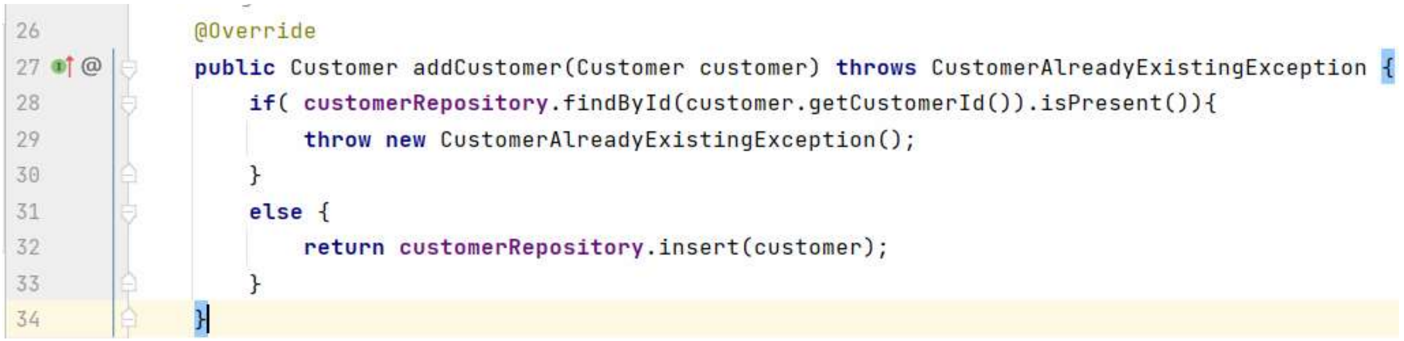
if service layer addCustomer() to be tested with positive result

-->repo.findById() record not found
-->repo.insert() return object back
service.addCustomer()

example2

if service layer addCustomer() to be tested with -ve result

service.addCustomer()
-->repo.findById() record found



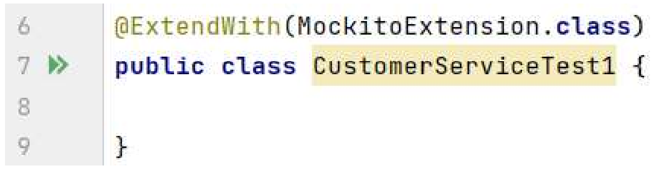
```java
@Override
public Customer addCustomer(Customer customer) throws CustomerAlreadyExistingException {
    if( customerRepository.findById(customer.getCustomerId()).isPresent()){
        throw new CustomerAlreadyExistingException();
    }
    else {
        return customerRepository.insert(customer);
    }
}
```

Steps to write test cases for service layer using mockito

1 Make sure sprintboot application working with all layers

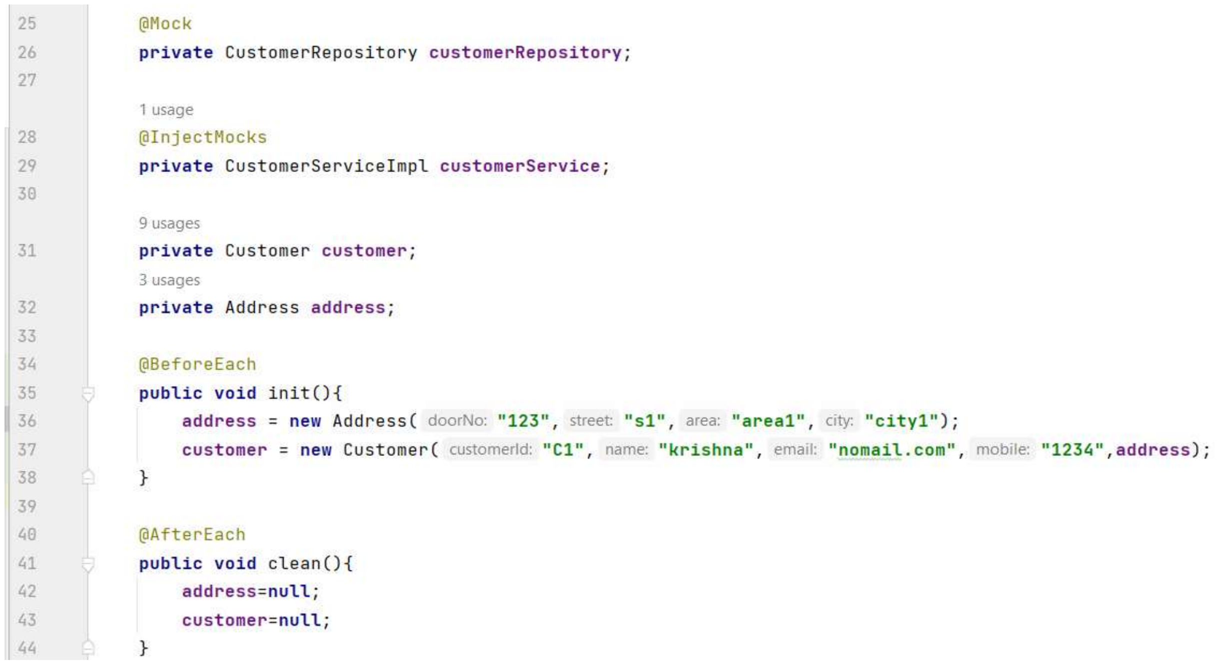
2 Create test class with required annonation

@ExtendWith(MockitoExtension.class)



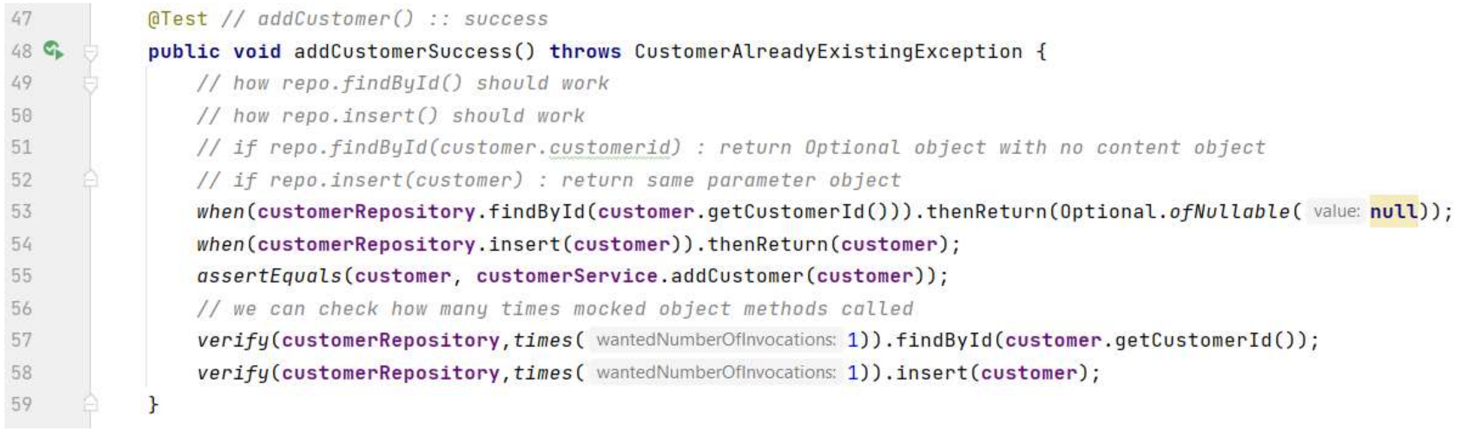
```java
@ExtendWith(MockitoExtension.class)
public class CustomerServiceTest1 {

}
```

3 Create @Mock and @InjectMocks objects as required
and other objects



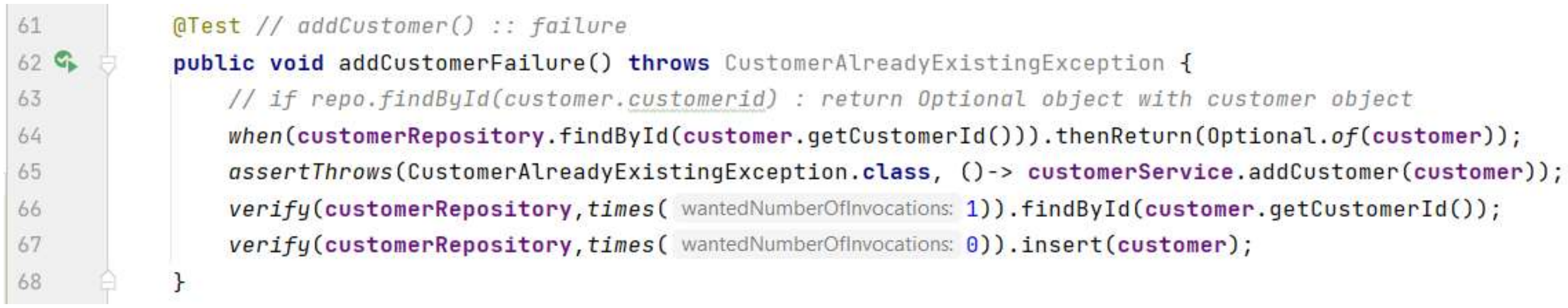
```java
@Mock
private CustomerRepository customerRepository;

1 usage
@InjectMocks
private CustomerServiceImpl customerService;

9 usages
private Customer customer;
3 usages
private Address address;

@BeforeEach
public void init(){
    address = new Address( doorNo: "123", street: "s1", area: "area1", city: "city1");
    customer = new Customer( customerId: "C1", name: "krishna", email: "nomail.com", mobile: "1234",address);
}

@AfterEach
public void clean(){
    address=null;
    customer=null;
}
```

4 write test case

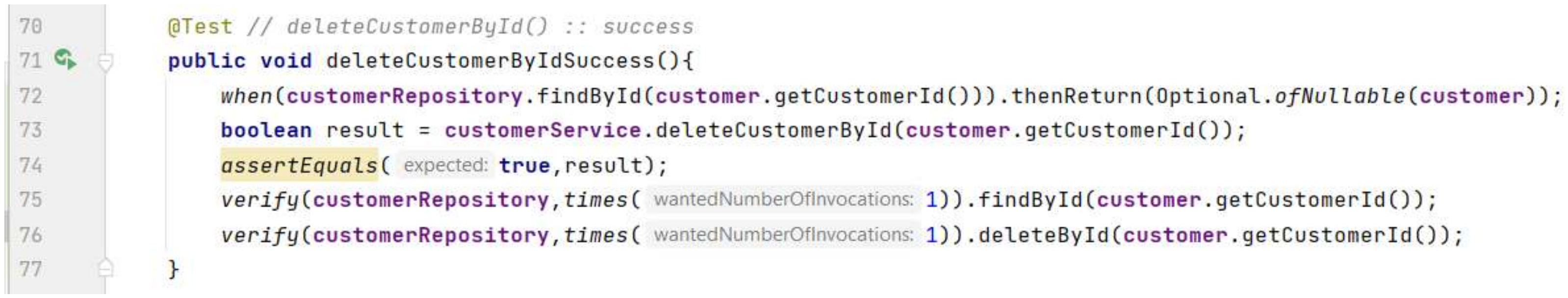


```java
@Test // addCustomer() :: success
public void addCustomerSuccess() throws CustomerAlreadyExistingException {
    // how repo.findById() should work
    // how repo.insert() should work
    // if repo.findById(customer.customerid) : return Optional object with no content object
    // if repo.insert(customer) : return same parameter object
    when(customerRepository.findById(customer.getCustomerId())).thenReturn(Optional.ofNullable( value: null));
    when(customerRepository.insert(customer)).thenReturn(customer);
    assertEquals(customer, customerService.addCustomer(customer));
    // we can check how many times mocked object methods called
    verify(customerRepository,times( wantedNumberOfInvocations: 1)).findById(customer.getCustomerId());
    verify(customerRepository,times( wantedNumberOfInvocations: 1)).insert(customer);
}
```

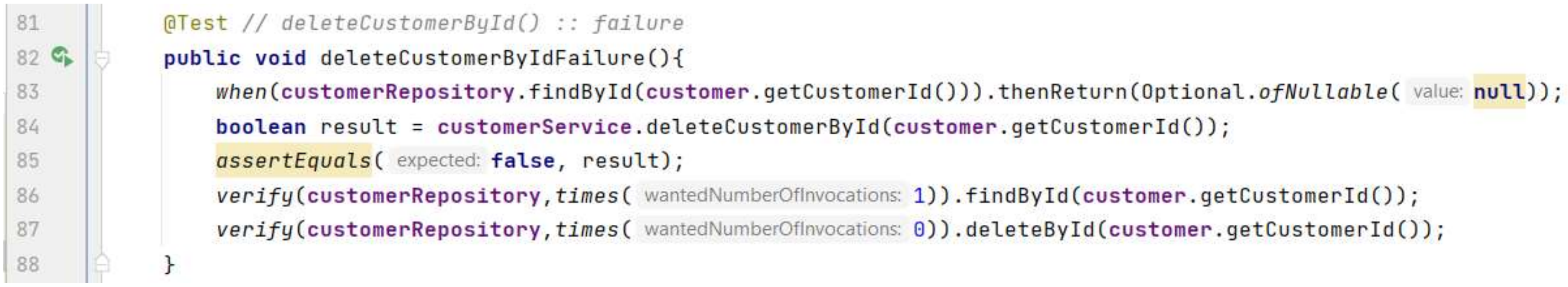
when
used to return particular result when a particular method called



```java
@Test // addCustomer() :: failure
public void addCustomerFailure() throws CustomerAlreadyExistingException {
    // if repo.findById(customer.customerid) : return Optional object with customer object
    when(customerRepository.findById(customer.getCustomerId())).thenReturn(Optional.of(customer));
    assertThrows(CustomerAlreadyExistingException.class, ()-> customerService.addCustomer(customer));
    verify(customerRepository,times( wantedNumberOfInvocations: 1)).findById(customer.getCustomerId());
    verify(customerRepository,times( wantedNumberOfInvocations: 0)).insert(customer);
}
```



```java
@Test // deleteCustomerById() :: success
public void deleteCustomerByIdSuccess(){
    when(customerRepository.findById(customer.getCustomerId())).thenReturn(Optional.ofNullable(customer));
    boolean result = customerService.deleteCustomerById(customer.getCustomerId());
    assertEquals( expected: true,result);
    verify(customerRepository,times( wantedNumberOfInvocations: 1)).findById(customer.getCustomerId());
    verify(customerRepository,times( wantedNumberOfInvocations: 1)).deleteById(customer.getCustomerId());
}
```



```java
@Test // deleteCustomerById() :: failure
public void deleteCustomerByIdFailure(){
    when(customerRepository.findById(customer.getCustomerId())).thenReturn(Optional.ofNullable( value: null));
    boolean result = customerService.deleteCustomerById(customer.getCustomerId());
    assertEquals( expected: false, result);
    verify(customerRepository,times( wantedNumberOfInvocations: 1)).findById(customer.getCustomerId());
    verify(customerRepository,times( wantedNumberOfInvocations: 0)).deleteById(customer.getCustomerId());
}
```

Y
service

z
repo

```
addCustomer()
{

check whether same id existing or not
          by using repo.findById().


          findAll()
          filter by id
          count 0
}
```

boss

| | |
|---|---|
| findById(); | 1 |
| findAll(); | 0 |