```
class  BookServiceImpl ..... {

    Book addBook(Book book) {

    // calling repo.save();
    }

}
```

```
interface BookRepository..... {

findAll(); ...
save();...
deleteById()...
findById();....


}
```

what type of testing you want to do on service layer methods
what are we expecting while testing service method

Book addBook(bookobj) { }

Controller	Service	Repository	
addBook()	addBook()	save()	insert record/update record
getAllBooks()	getAllBooks()	findAll();	

while testing service method
ex
while testing addBook()

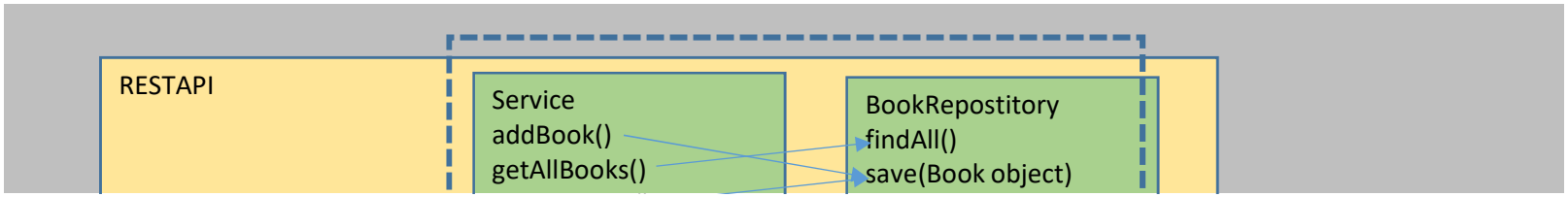
opt 1
check the retured value
check the param
check the process

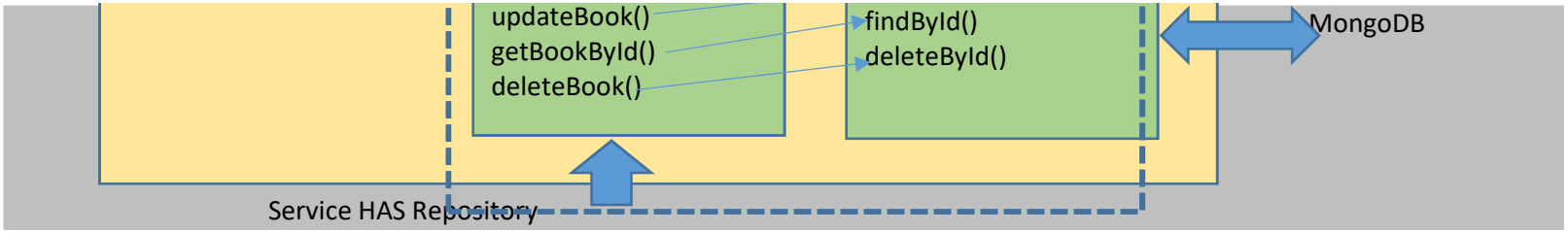
opt2
check which method of repository called

when service method is called

1 we need to check whether service method is calling particular repository method or not
2 no need to check whether repository method is doing its task or not

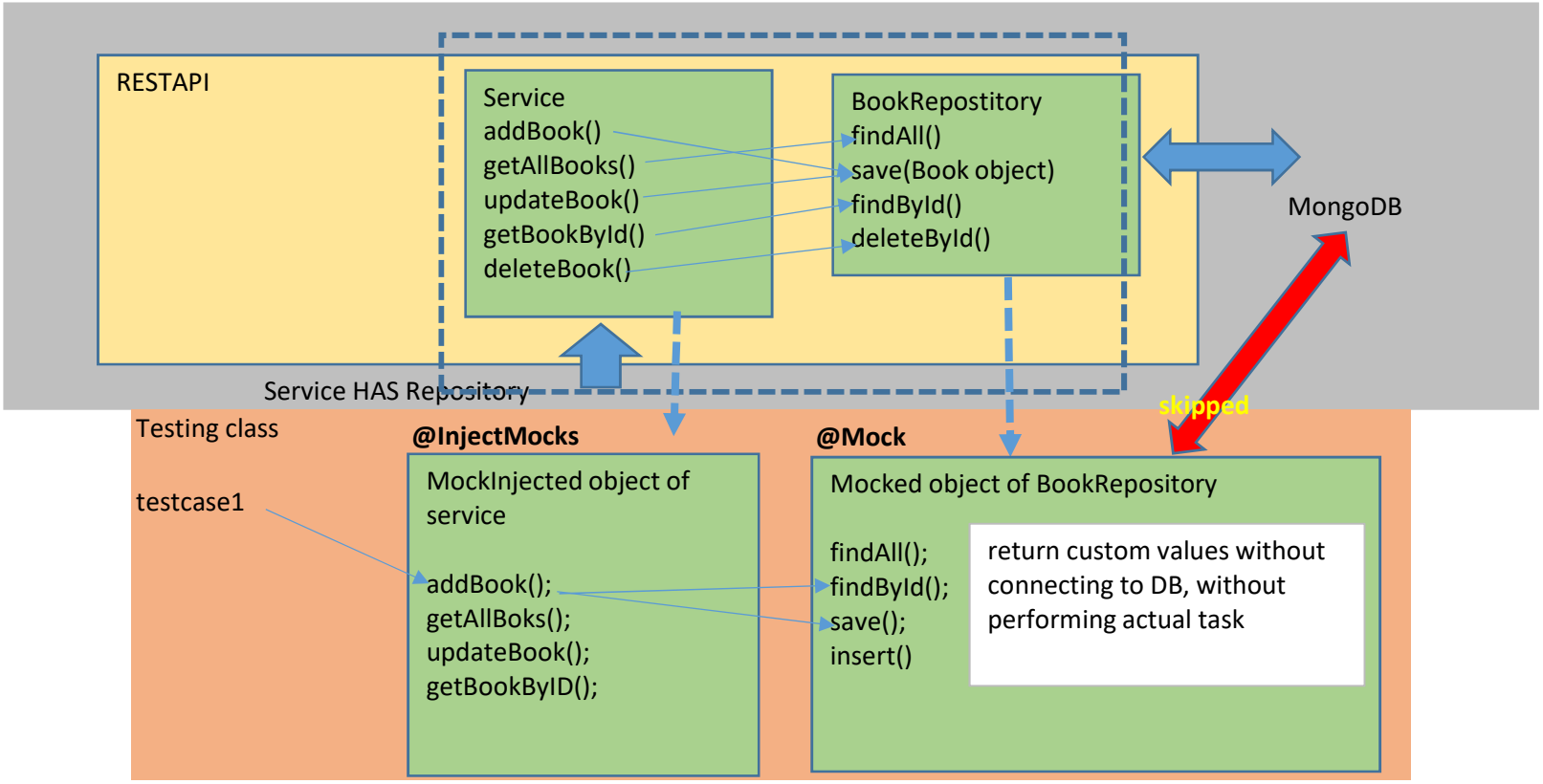
service ----> repo





Service is dependent
Repository is dependency

We can create **mocked object** of dependency
Mocked object methods can skip their actual task
Mocked object methods can return customized values



Mocked object dependency
InjectMock object dependent

service method

```
17      @Override
18  @   public Book addBook(Book book) throws BookAlreadyExistingException {
19          // check whether record found with book.bookId
20          // if not found : save record
21          // if found : throw exception
22          if(bookRepository.findById(book.getBookId()).isEmpty()){
23              return bookRepository.save(book);
24          }
25          else{
26              throw new BookAlreadyExistingException();
27          }
28      }
```

when service.addBook()

success failure

findById findById

save

Writing test cases for testing service layer

Service HAS Repository

Service is Dependent

Repository is Dependency

Step 1 Make sure springboot application is ready with all layers and with all functionalities

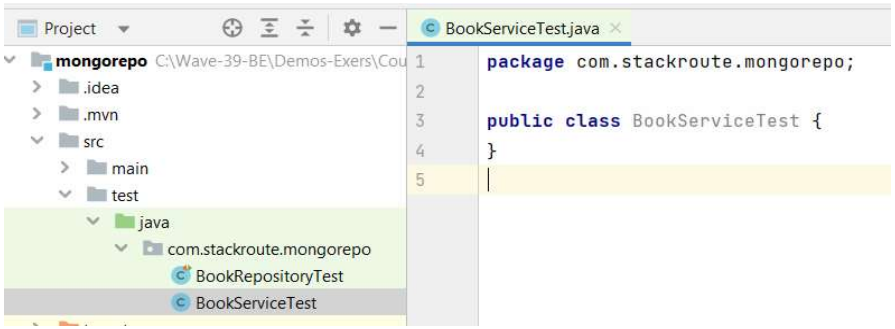
copy sprint3 demo (mongodb-with crud operations)

make sure mongodb is running (optional)

make sure testing dependency added in pom (default)

```
34          <dependency>
35              <groupId>org.springframework.boot</groupId>
36              <artifactId>spring-boot-starter-test</artifactId>
37              <scope>test</scope>
38          </dependency>
```

Step 2 Create testing class under testing folder



Step 3 Add required annotations

```
@ExtendWith(MockitoExtension.class)
public class BookServiceTest {

}
```

Step 4 Create dependeny and dependent objects

```
@ExtendWith(MockitoExtension.class)
public class BookServiceTest {
    // Service HAS Repository
    // Mocked object of Repository
    // InjectMocks object of serviceimpl

    @Mock
    private BookRepository bookRepository;

    @InjectMocks
    private BookServiceImpl bookService;

    private Book book;
    private Author author;
```

Step 5 Define setup/clean

```
    @BeforeEach
    public void setup(){
        author = new Author( authorName: "Bg Sw", address: "Chennai", age: 35);
        book = new Book( bookId: "BK0001", name: "Spirit of C", subject: "C",author, price: 345, stock: 56);
    }

    @AfterEach
    public void clean(){
        author=null;
        book=null;
    }
```

Step 6 Write test cases

Test 1 check whether service.addBook() working fine or not (Success)

```java
@Override
public Book addBook(Book book) throws BookAlreadyExistingException {
    // check whether record found with book.bookId
    // if not found : save record
    // if found : throw exception
    if(bookRepository.findById(book.getBookId()).isEmpty()){
        return bookRepository.save(book);
    }
    else{
        throw new BookAlreadyExistingException();
    }
}
```

if service.addBook() is success: means it has to call repo.findById() and save()
findById() must return no object
save() returns book object

```java
@Test
public void testAddBookSuccess() throws BookAlreadyExistingException {
    // repo.findById(book.bookid) then return empty optional
    // repo.save(book) then return book object
    when(bookRepository.findById(book.getBookId())).thenReturn(Optional.ofNullable( value: null));
    when(bookRepository.save(book)).thenReturn(book);
    assertEquals(book, bookService.addBook(book));
    verify(bookRepository,times( wantedNumberOfInvocations: 1)).findById(book.getBookId());
    verify(bookRepository,times( wantedNumberOfInvocations: 1)).save(book);
    // how many times service.addbook() should call repo.findById() : 1
    // how many times service.addbook() should call repo.save() : 1
    // Note : equals() to be defined in Book Model class
}
```

Note : equals() must be overriden in model classes

Test 2 check whether service.addBook() working fine or not (Failure)

```java
@Override
public Book addBook(Book book) throws BookAlreadyExistingException {
    // check whether record found with book.bookId
    // if not found : save record
    // if found : throw exception
    if(bookRepository.findById(book.getBookId()).isEmpty()){
        return bookRepository.save(book);
    }
    else{
        throw new BookAlreadyExistingException();
    }
}
```

```java
@Test
public void testAddBookFailure(){
    // repo.findById(book.bookid) then return Optional object with book object
    when(bookRepository.findById(book.getBookId())).thenReturn(Optional.of(book));
    assertThrows(BookAlreadyExistingException.class,()->bookService.addBook(book));
    verify(bookRepository,times( wantedNumberOfInvocations: 1)).findById(book.getBookId());
    verify(bookRepository,times( wantedNumberOfInvocations: 0)).save(book);
    // how many times service.addbook() should call repo.findById() : 1
    // how many times service.addbook() should call repo.save() : 0
}
```

Test 3 check service.delete() (Success)

```
36      @Override
37      public boolean deleteBook(String bkid) throws BookNotFoundException {
38          // delete book only if found, else throw exception
39          if( bookRepository.findById(bkid).isPresent() ) {
40              bookRepository.deleteById(bkid);
41              return true;
42          }
43          else{
44              throw new BookNotFoundException();
45          }
46      }
```

```
73      @Test
74      public void testDeleteBookSuccess() throws BookNotFoundException {
75          // repo.findById(book.bookid) then return Optional object with book
76          when(bookRepository.findById(book.getBookId())).thenReturn(Optional.of(book));
77          boolean result=bookService.deleteBook(book.getBookId());
78          assertEquals( expected: true, result);
79          verify(bookRepository,times( wantedNumberOfInvocations: 1)).findById(book.getBookId());
80          verify(bookRepository,times( wantedNumberOfInvocations: 1)).deleteById(book.getBookId());
81      }
```

Test 4 check service.delete() (Failure)

```
83      @Test
84      public void testDeleteBookFailure(){
85          when(bookRepository.findById(book.getBookId())).thenReturn(Optional.ofNullable( value: null));
86          assertThrows(BookNotFoundException.class,()->bookService.deleteBook(book.getBookId()));
87          verify(bookRepository,times( wantedNumberOfInvocations: 1)).findById(book.getBookId());
88          verify(bookRepository,times( wantedNumberOfInvocations: 0)).deleteById(book.getBookId());
89      }
```