Unit Testing / Integration testing

Unit testing and Integration testing in Spring Boot

Rajeev Gupta
Java Trainer & Consultant
MTech. CS

Spring boot testing support

- One of the major advantage of DI is that it is easier to test our code
- You can mock objects rather then real objects
- We need both unit testing and integration testing (spring applicationcontext involved in this process). It is useful to able to perform integration testing without requiring deployment of application or needed to connect other infrastrcuture
- We can use spring boot started test dependency to manage all testing requirment in our projects

Spring boot testing support

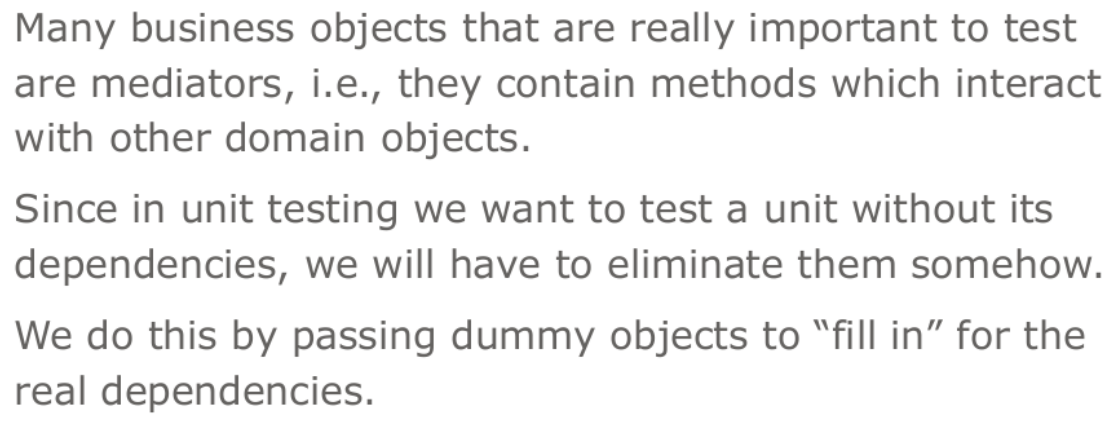
- Spring-boot-test-startar provide numbers of utilities and annotation to help testing the real world applications
- Test support provided by two modules
  - Spring-boot-test:contain core testing support
  - Spring-boot-test-autoconfiguration: support auto configuration for test
- Developer just need to use Spring-boot-test-startar ‘starter’ download
- all required testing modules such as junit, AssertJ, hamcrest and number of other libarires.

Spring boot:Test scope dependencies

- If you use the spring-boot-starter-test 'starter', you will find following provided libraries:
  - **Junit**: standared for unit testing in java world
  - **Spring test & spring boot Test**: utilities and integration test support for spring boot applications
  - **AssertJ**: A fluent assertion library
  - **Hamcrest**: A libary of matchers objects (also known as constraints or predicates)
  - **Mockito**: A java mocking framework
  - **JSONassert**: An asssertion libray specially for JSON data
  - **JsonPath**: XPath for JSON

Need of mocking

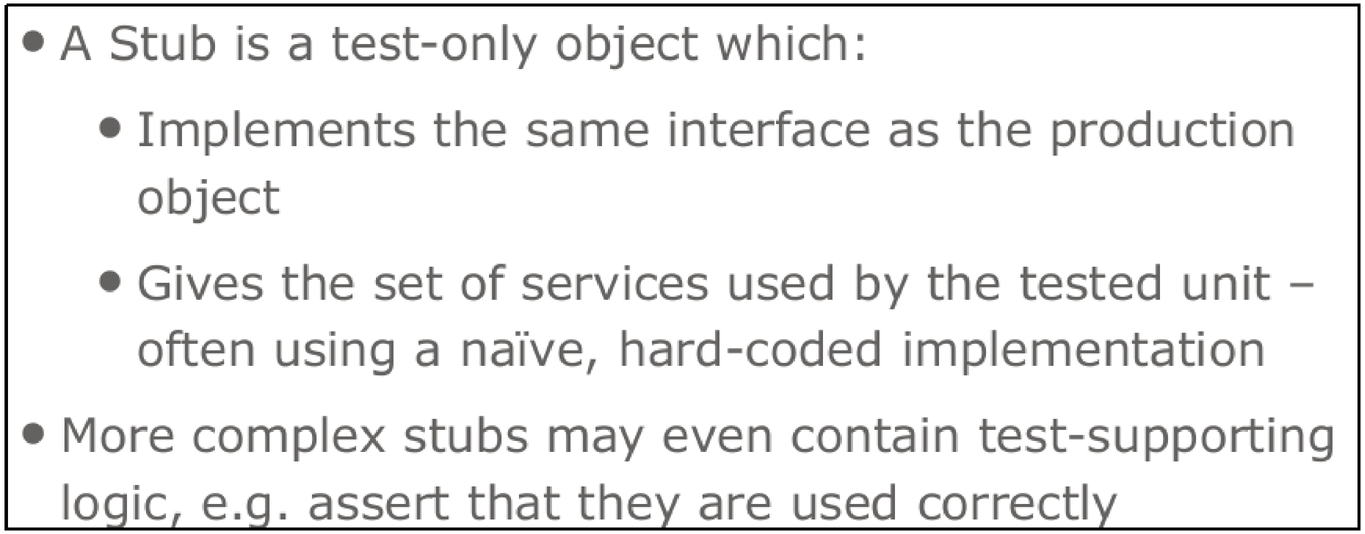
- The JUnit testing framework, makes the testing of an independent, non-void method, simple.
- But... how do we test an method with no return value?
- How do we test a method that interacts with other domain objects?
- Dependencies – In order for a function to run it oftenrequires Files, DB, JNDI or generally – other units
    - How can we test a single unit, without being
    - affected by other units – their errors, set-up
    - complexities and performance issues?

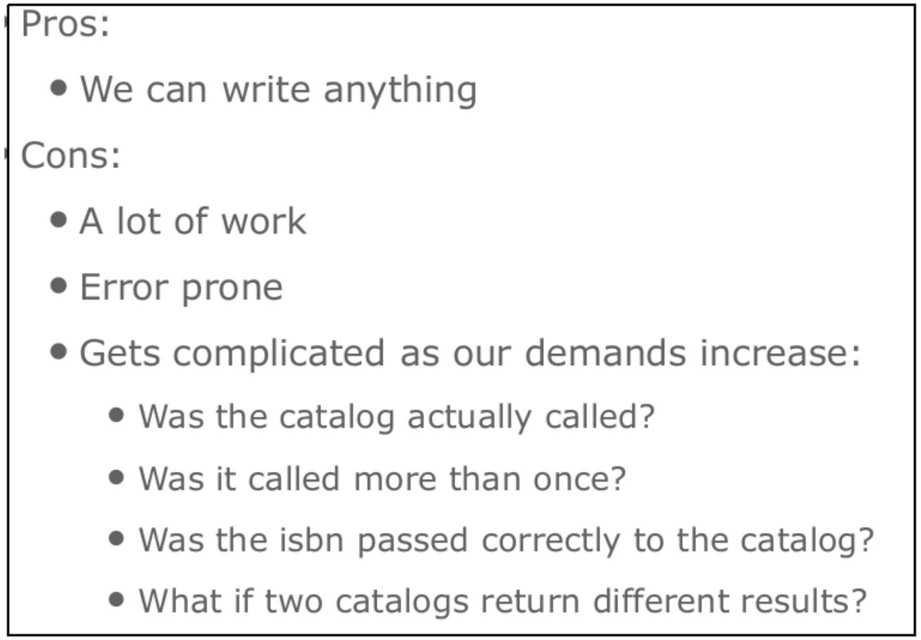

Many business objects that are really important to test are mediators, i.e., they contain methods which interact with other domain objects.

Since in unit testing we want to test a unit without its dependencies, we will have to eliminate them somehow.

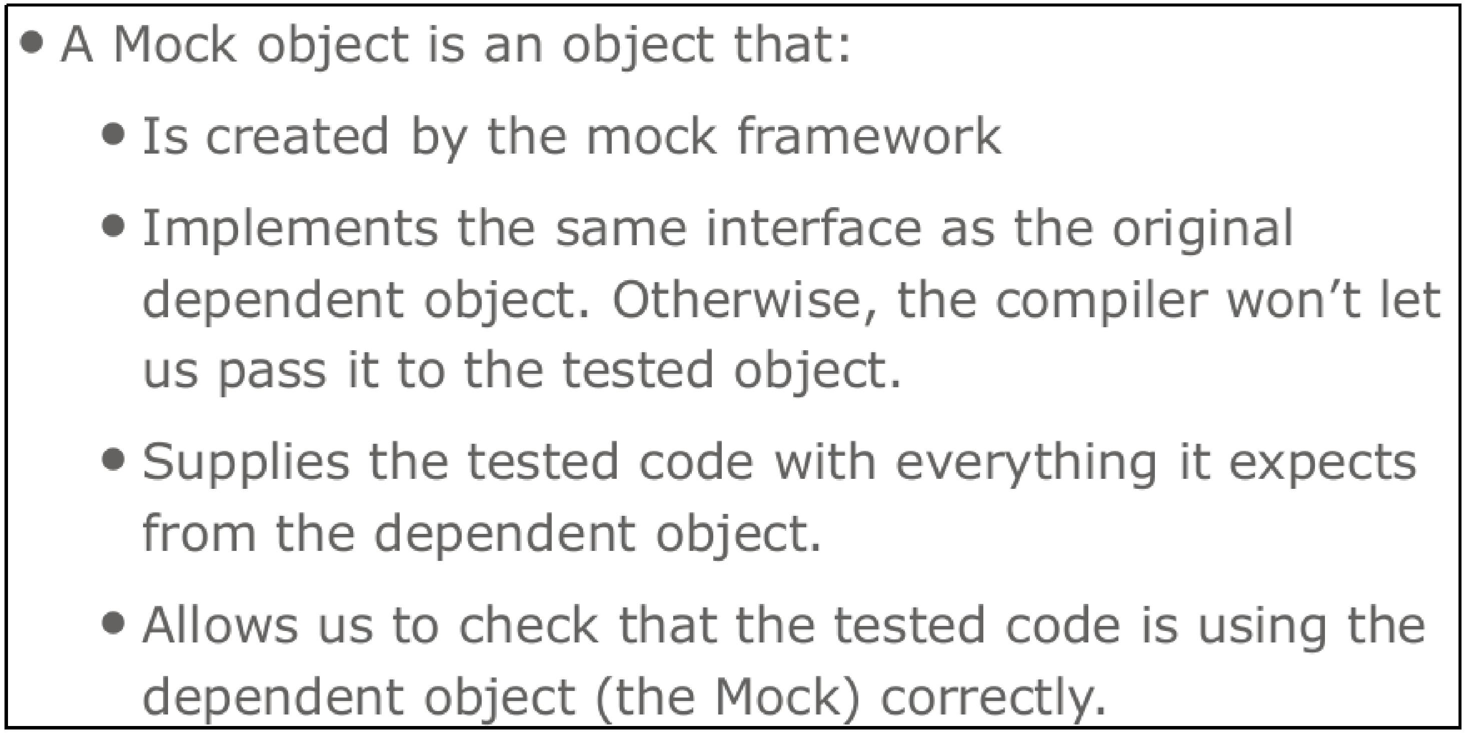
We do this by passing dummy objects to “fill in” for the real dependencies.

What is stub?


- A Stub is a test-only object which:
  - Implements the same interface as the production object
  - Gives the set of services used by the tested unit – often using a naïve, hard-coded implementation
- More complex stubs may even contain test-supporting logic, e.g. assert that they are used correctly


- Pros:
  - We can write anything
- Cons:
  - A lot of work
  - Error prone
  - Gets complicated as our demands increase:
    - Was the catalog actually called?
    - Was it called more than once?
    - Was the isbn passed correctly to the catalog?
    - What if two catalogs return different results?

Mock objects?



- A Mock object is an object that:
  - Is created by the mock framework
  - Implements the same interface as the original dependent object. Otherwise, the compiler won't let us pass it to the tested object.
  - Supplies the tested code with everything it expects from the dependent object.
  - Allows us to check that the tested code is using the dependent object (the Mock) correctly.

To use a Mock object we should:

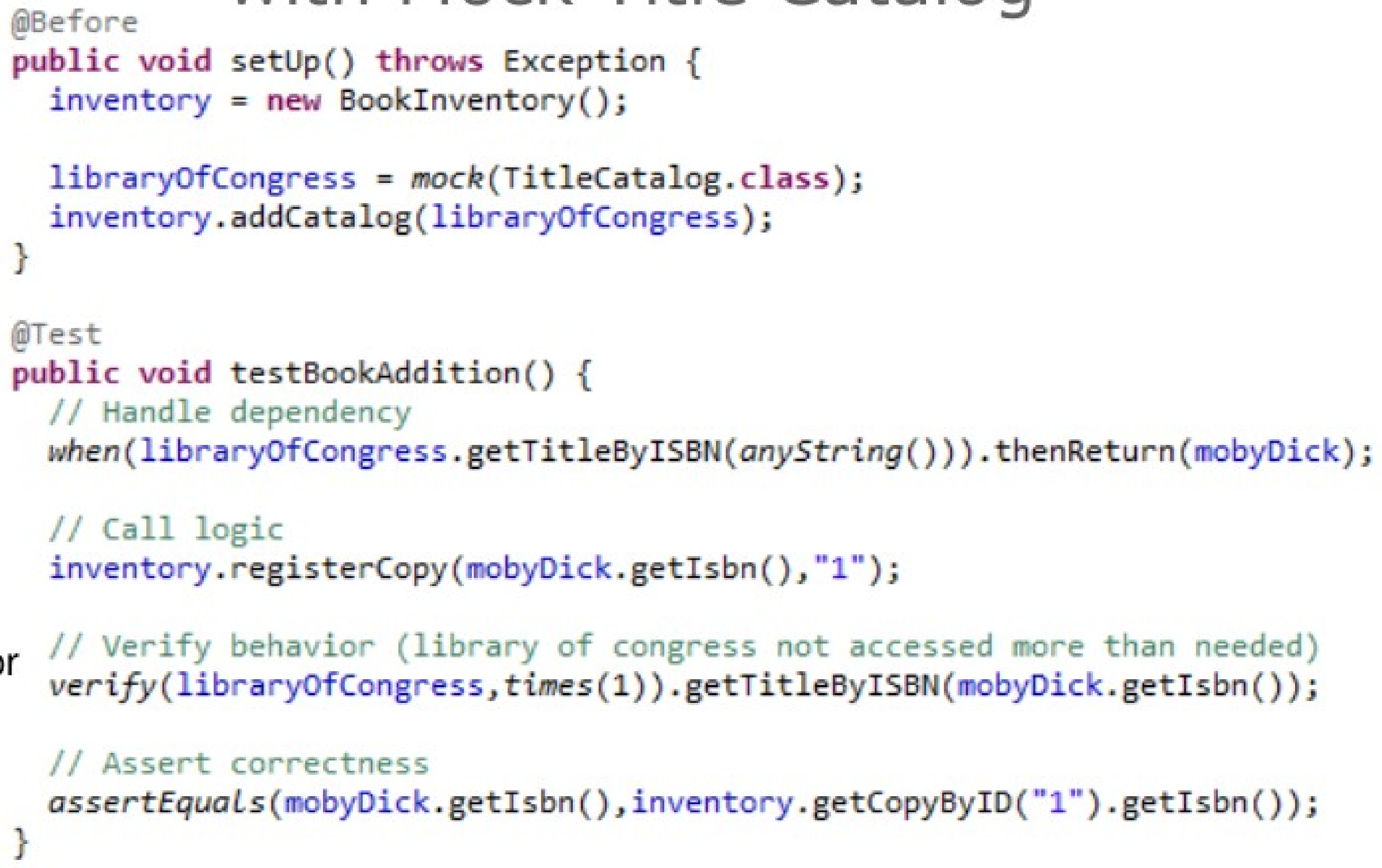
Create instance of the Mock.

Define bahavior during test.

Invoke the tested code while passing the Mock as parameter.

Verify the Mock has been used correctly.

Example: Testing book addition with Mock Title Catalog



Construct

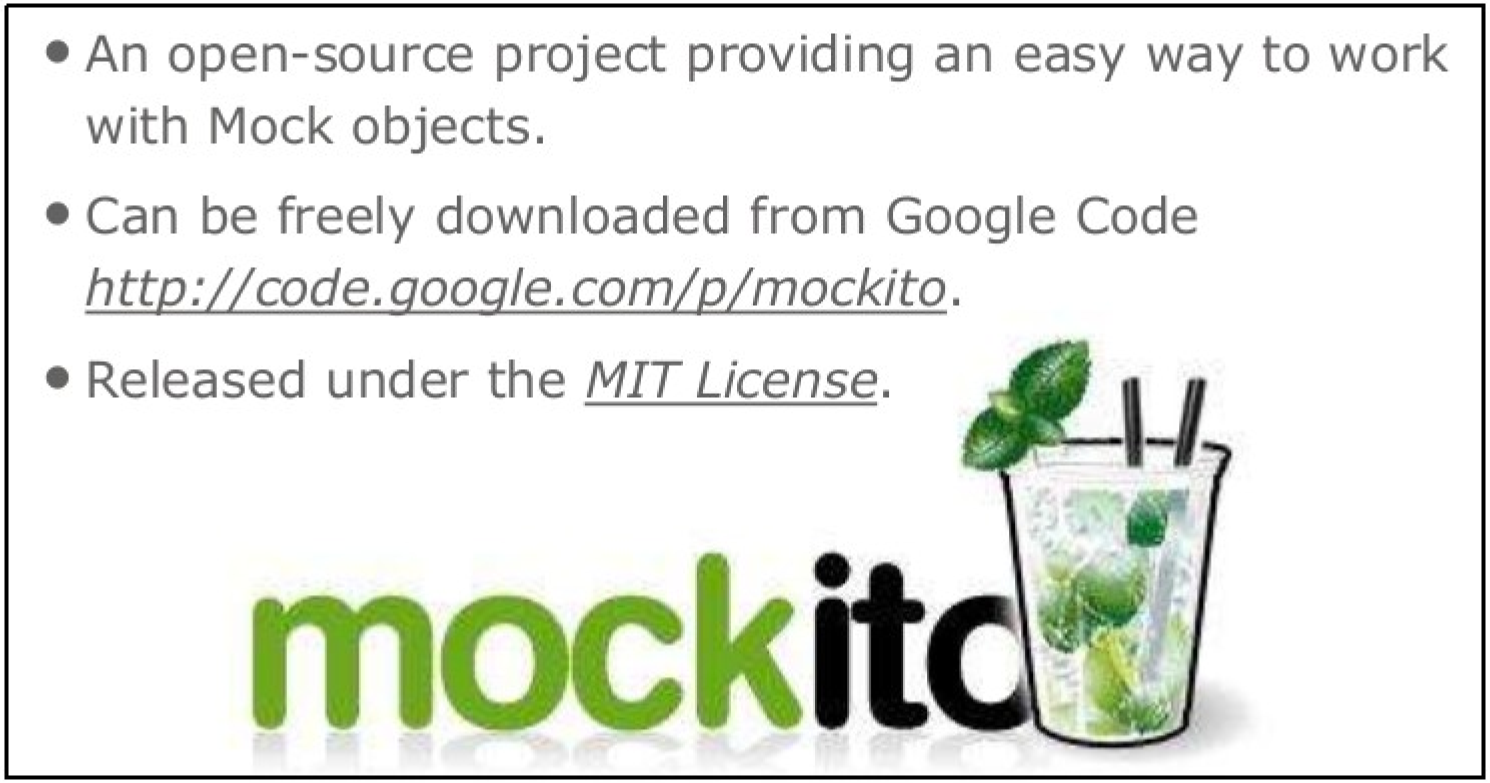
```
@Before
public void setUp() throws Exception {
  inventory = new BookInventory();

  libraryOfCongress = mock(TitleCatalog.class);
  inventory.addCatalog(libraryOfCongress);
}
```

Set behavior

Verify behavior

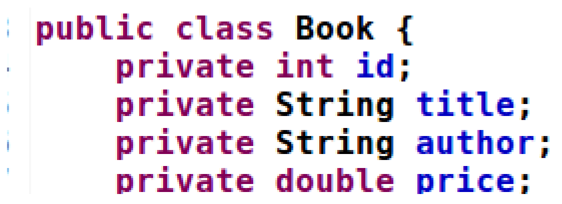
```
@Test
public void testBookAddition() {
  // Handle dependency
  when(libraryOfCongress.getTitleByISBN(anyString())).thenReturn(mobyDick);

  // Call logic
  inventory.registerCopy(mobyDick.getIsbn(),"1");

  // Verify behavior (library of congress not accessed more than needed)
  verify(libraryOfCongress,times(1)).getTitleByISBN(mobyDick.getIsbn());

  // Assert correctness
  assertEquals(mobyDick.getIsbn(),inventory.getCopyByID("1").getIsbn());
}
```

mockito

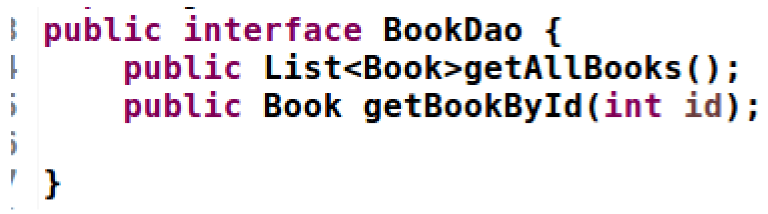


- An open-source project providing an easy way to work with Mock objects.
- Can be freely downloaded from Google Code *http://code.google.com/p/mockito*.
- Released under the *MIT License*.

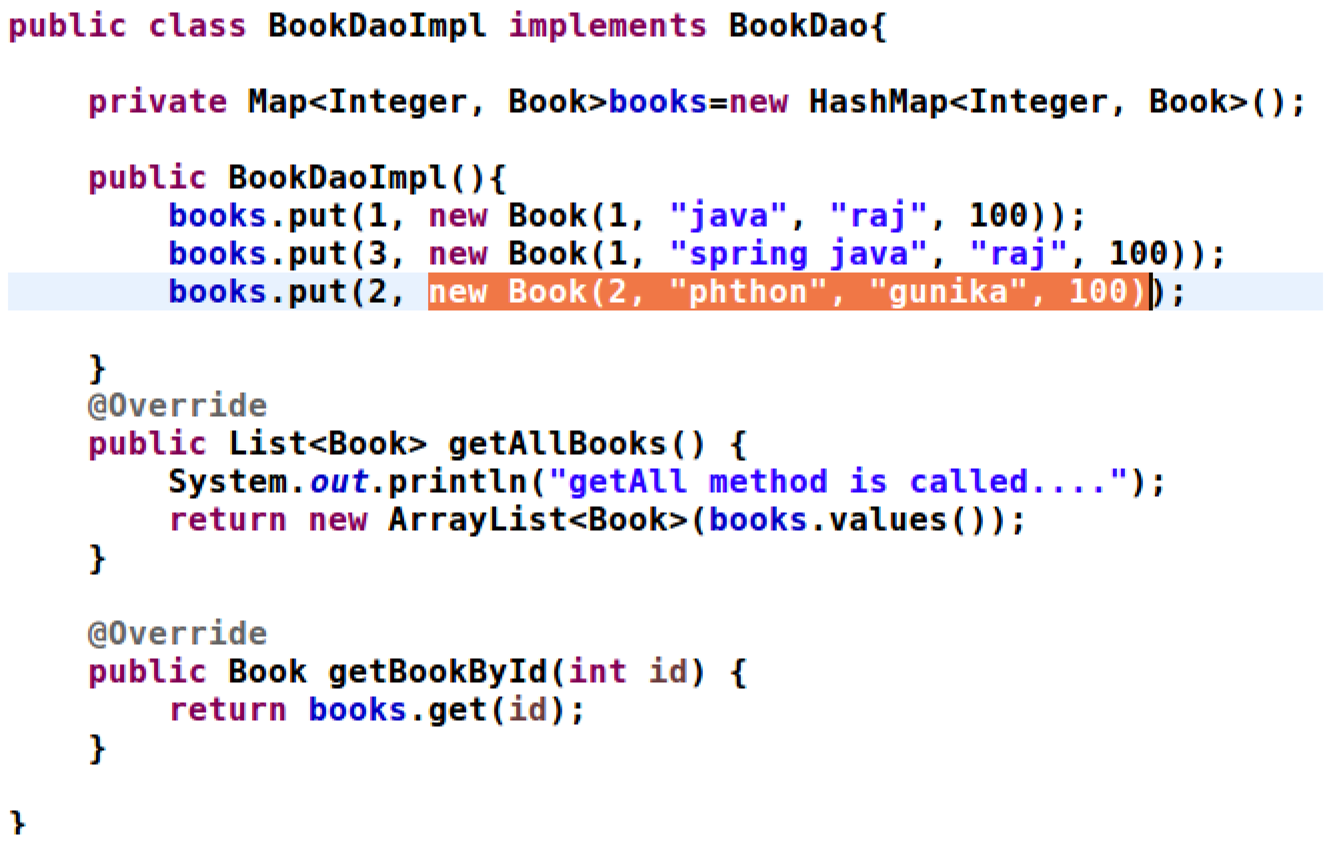
Designing Dao layer



```
public class Book {
    private int id;
    private String title;
    private String author;
    private double price;
```

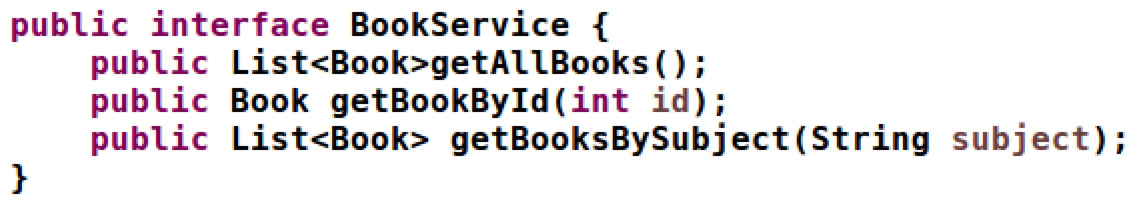


```
public interface BookDao {
    public List<Book>getAllBooks();
    public Book getBookById(int id);

}
```

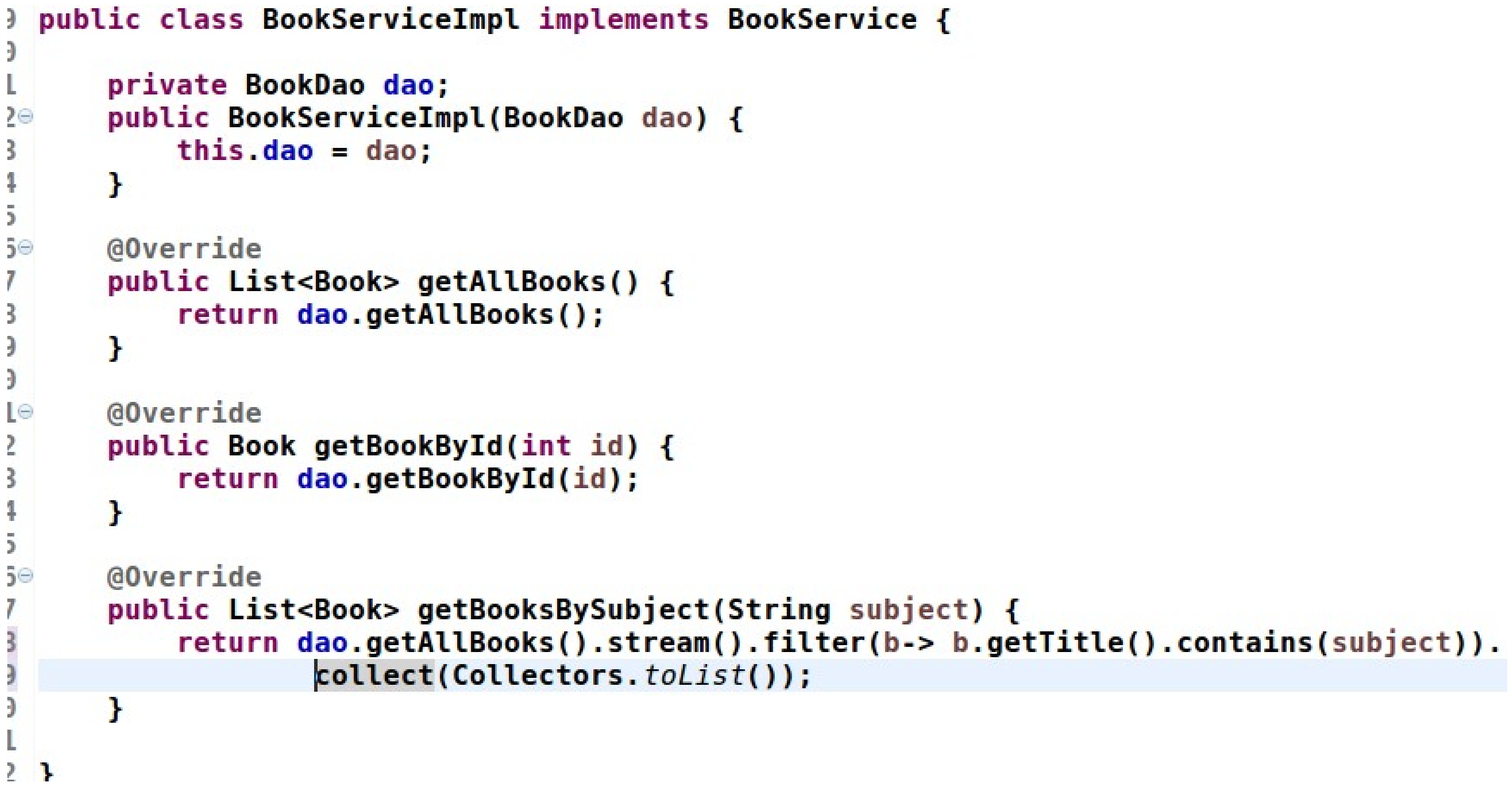


```
public class BookDaoImpl implements BookDao{

    private Map<Integer, Book>books=new HashMap<Integer, Book>();

    public BookDaoImpl(){
        books.put(1, new Book(1, "java", "raj", 100));
        books.put(3, new Book(1, "spring java", "raj", 100));
        books.put(2, new Book(2, "phthon", "gunika", 100));

    }
    @Override
    public List<Book> getAllBooks() {
        System.out.println("getAll method is called....");
        return new ArrayList<Book>(books.values());
    }

    @Override
    public Book getBookById(int id) {
        return books.get(id);
    }

}
```

Designing Service layer

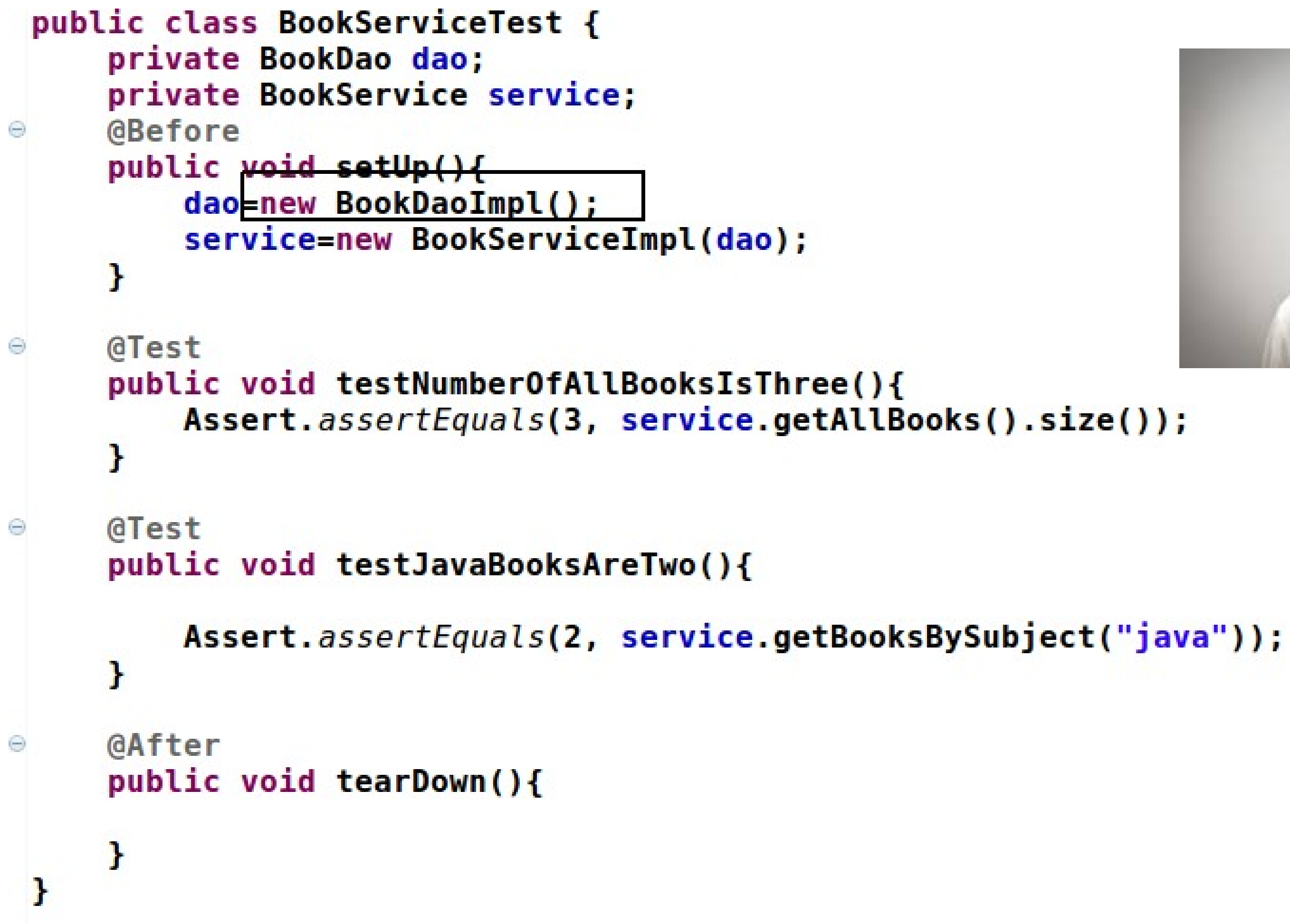


```java
public interface BookService {
    public List<Book>getAllBooks();
    public Book getBookById(int id);
    public List<Book> getBooksBySubject(String subject);
}
```



```java
public class BookServiceImpl implements BookService {

    private BookDao dao;
    public BookServiceImpl(BookDao dao) {
        this.dao = dao;
    }

    @Override
    public List<Book> getAllBooks() {
        return dao.getAllBooks();
    }

    @Override
    public Book getBookById(int id) {
        return dao.getBookById(id);
    }

    @Override
    public List<Book> getBooksBySubject(String subject) {
        return dao.getAllBooks().stream().filter(b-> b.getTitle().contains(subject)).
                collect(Collectors.toList());
    }

}
```

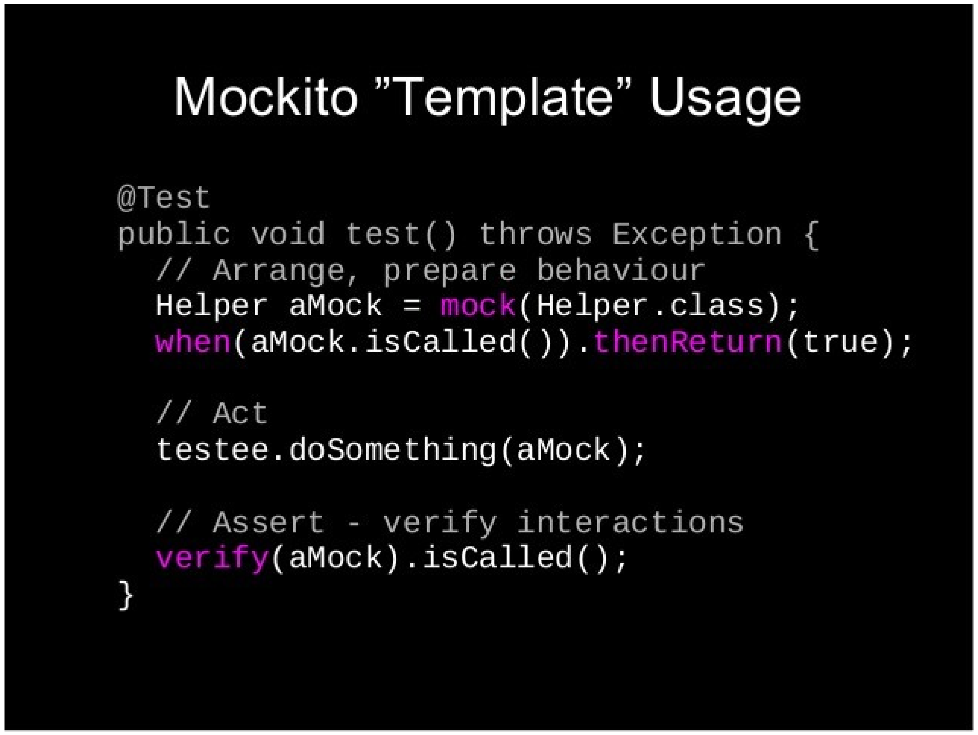
Service layer testing with stub



```
public class BookServiceTest {
    private BookDao dao;
    private BookService service;
    @Before
    public void setUp(){
        dao=new BookDaoImpl();
        service=new BookServiceImpl(dao);
    }

    @Test
    public void testNumberOfAllBooksIsThree(){
        Assert.assertEquals(3, service.getAllBooks().size());
    }

    @Test
    public void testJavaBooksAreTwo(){

        Assert.assertEquals(2, service.getBooksBySubject("java"));
    }

    @After
    public void tearDown(){

    }
}
```

What is mocking?

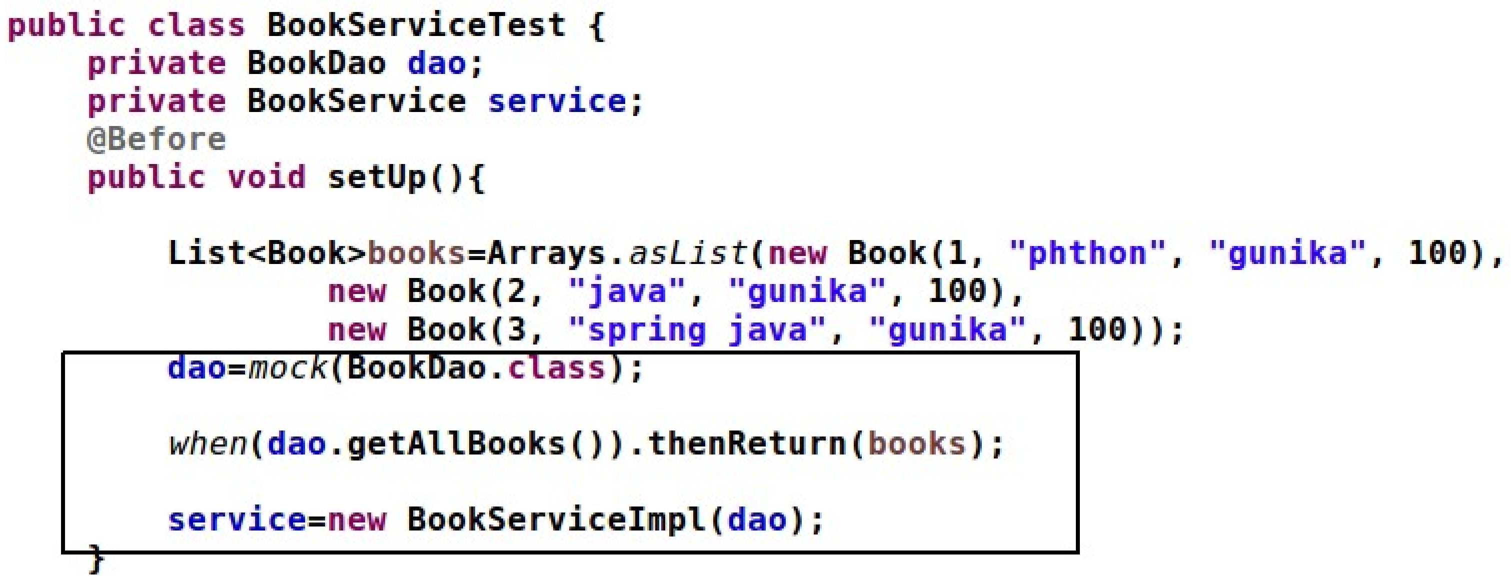
- mocking is creating objects that simulates the behaviour of real objects
- Unlike stub mock can dynamically created from code- at runtime
- mock offer more functionality then stubbing
- You can verify method call and a lot of other things
- Mock ==> stub ==> verify



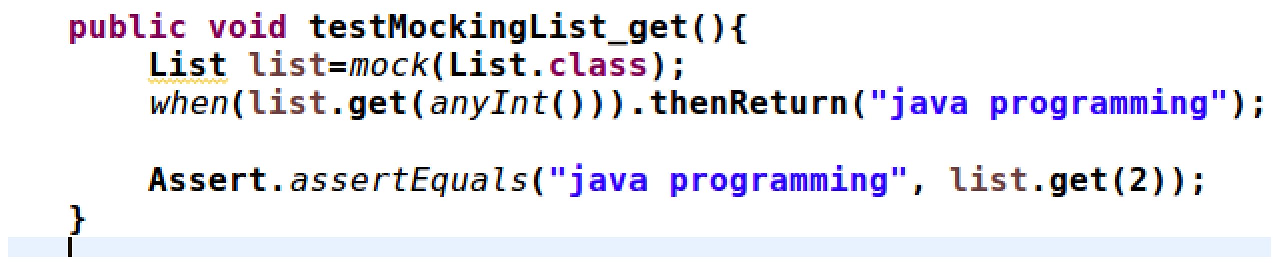
Mockito ”Template” Usage

```
@Test
public void test() throws Exception {
  // Arrange, prepare behaviour
  Helper aMock = mock(Helper.class);
  when(aMock.isCalled()).thenReturn(true);

  // Act
  testee.doSomething(aMock);

  // Assert - verify interactions
  verify(aMock).isCalled();
}
```

Service layer testing with mock

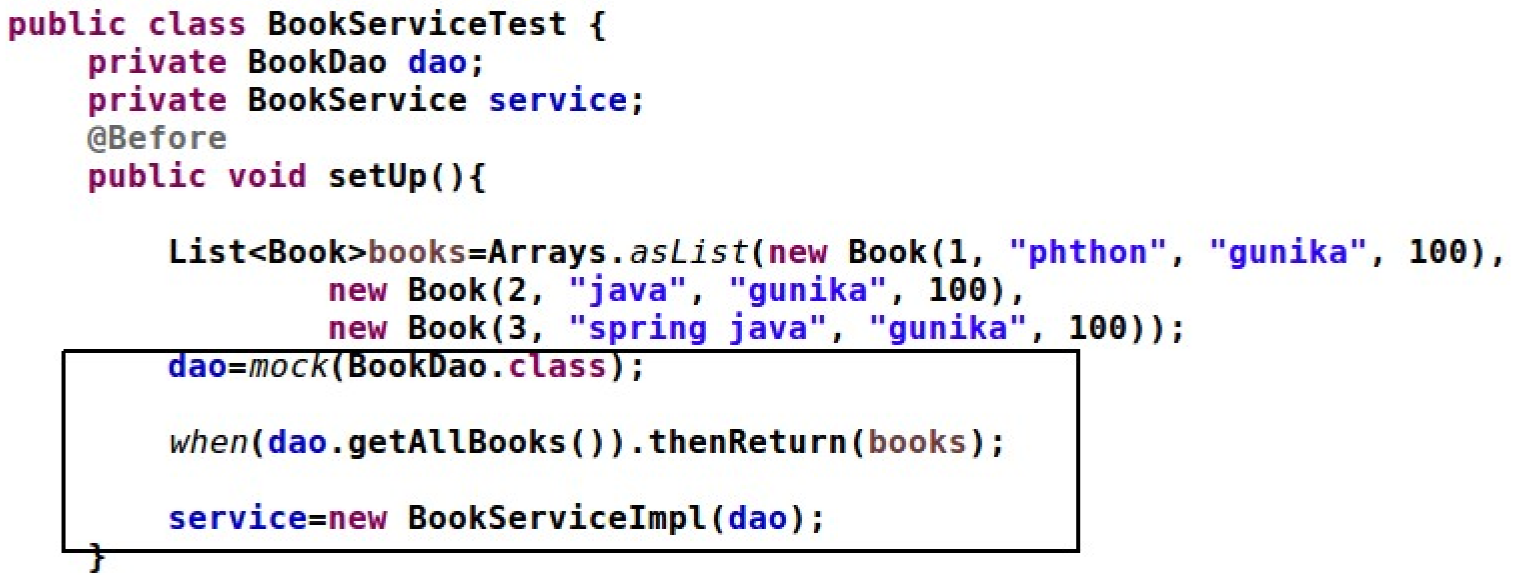


```
public class BookServiceTest {
    private BookDao dao;
    private BookService service;
    @Before
    public void setUp(){

        List<Book>books=Arrays.asList(new Book(1, "phthon", "gunika", 100),
                new Book(2, "java", "gunika", 100),
                new Book(3, "spring java", "gunika", 100));
        dao=mock(BookDao.class);

        when(dao.getAllBooks()).thenReturn(books);

        service=new BookServiceImpl(dao);
    }
```



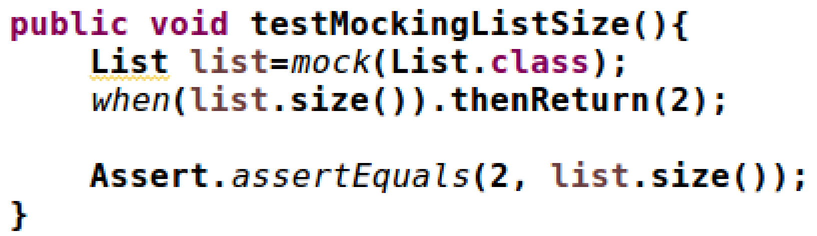
```
    public void testMockingList_get(){
        List list=mock(List.class);
        when(list.get(anyInt())).thenReturn("java programming");

        Assert.assertEquals("java programming", list.get(2));
    }
```

Service layer testing with mock

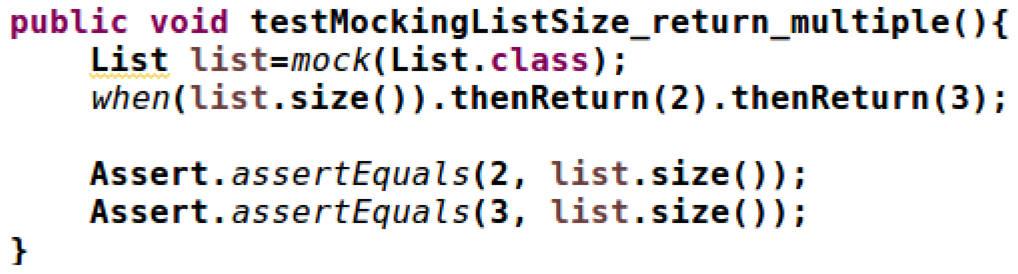


```
public class BookServiceTest {
    private BookDao dao;
    private BookService service;
    @Before
    public void setUp(){

        List<Book>books=Arrays.asList(new Book(1, "phthon", "gunika", 100),
                new Book(2, "java", "gunika", 100),
                new Book(3, "spring java", "gunika", 100));
        dao=mock(BookDao.class);

        when(dao.getAllBooks()).thenReturn(books);

        service=new BookServiceImpl(dao);
    }
```

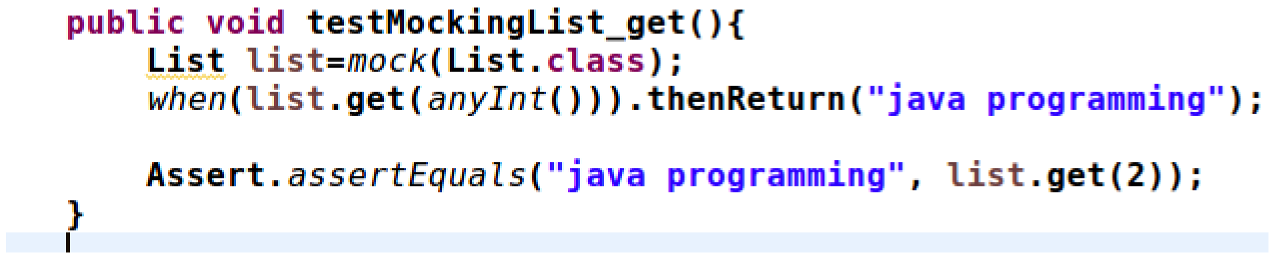
Mockito with list



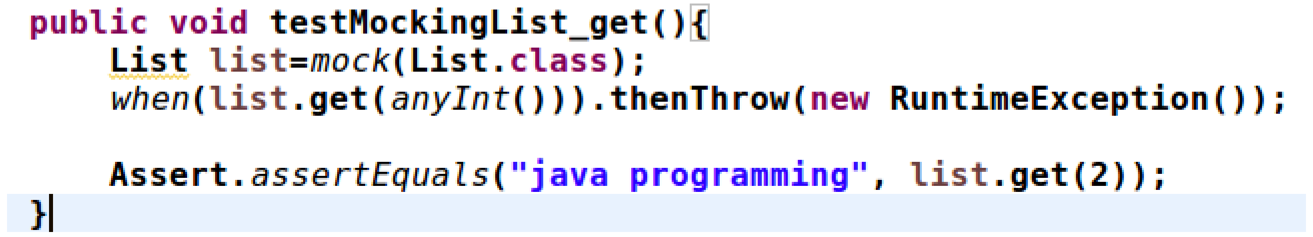
```
public void testMockingListSize(){
    List list=mock(List.class);
    when(list.size()).thenReturn(2);

    Assert.assertEquals(2, list.size());
}
```



```
public void testMockingListSize_return_multiple(){
    List list=mock(List.class);
    when(list.size()).thenReturn(2).thenReturn(3);

    Assert.assertEquals(2, list.size());
    Assert.assertEquals(3, list.size());
}
```



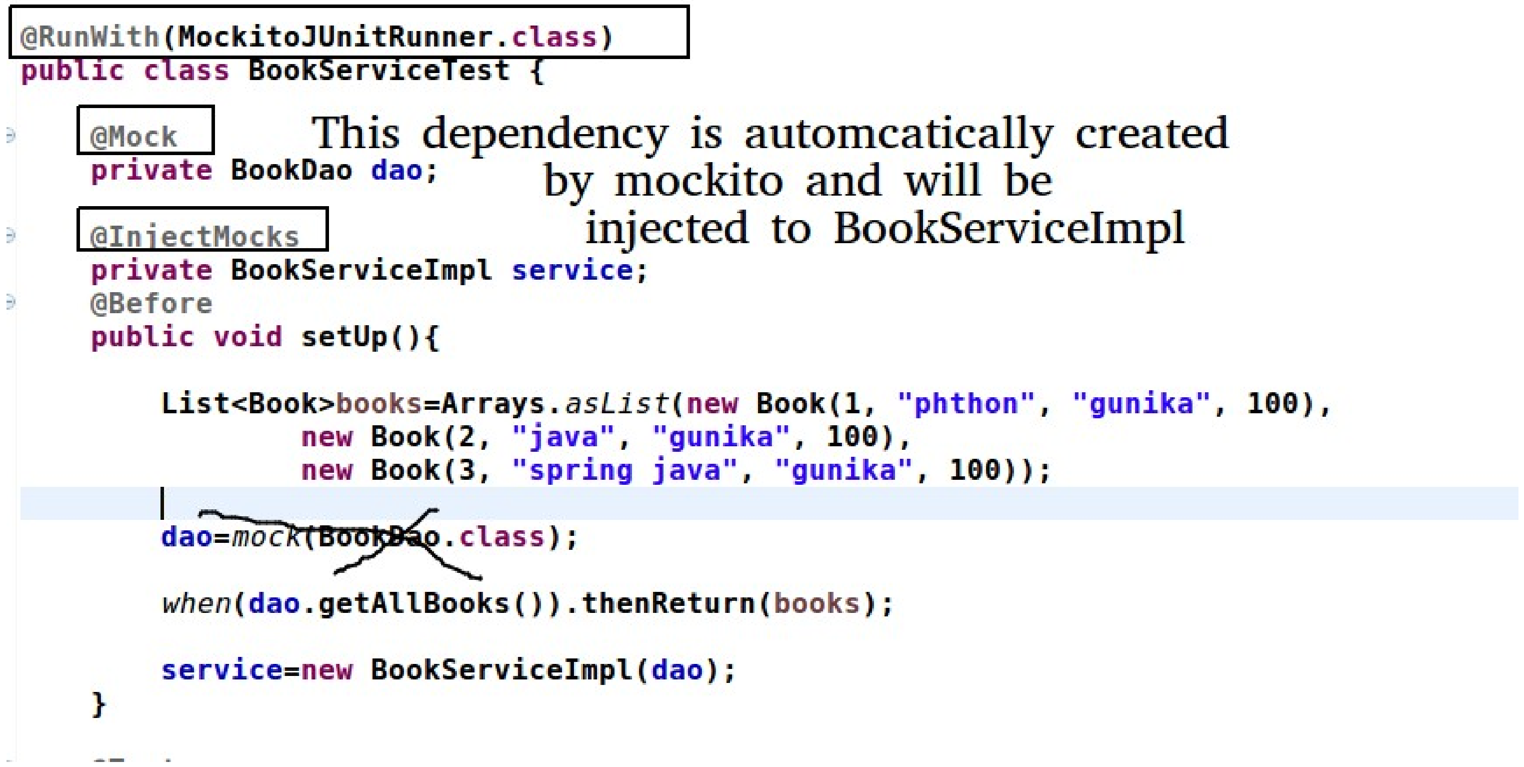
```
public void testMockingList_get(){
    List list=mock(List.class);
    when(list.get(anyInt())).thenReturn("java programming");

    Assert.assertEquals("java programming", list.get(2));
}
```



```
public void testMockingList_get(){
    List list=mock(List.class);
    when(list.get(anyInt())).thenThrow(new RuntimeException());

    Assert.assertEquals("java programming", list.get(2));
}
```

Using mockito annotations

- @RunWith(MockitoJUnitRunner.class)
- @Mock
- @InjectMocks

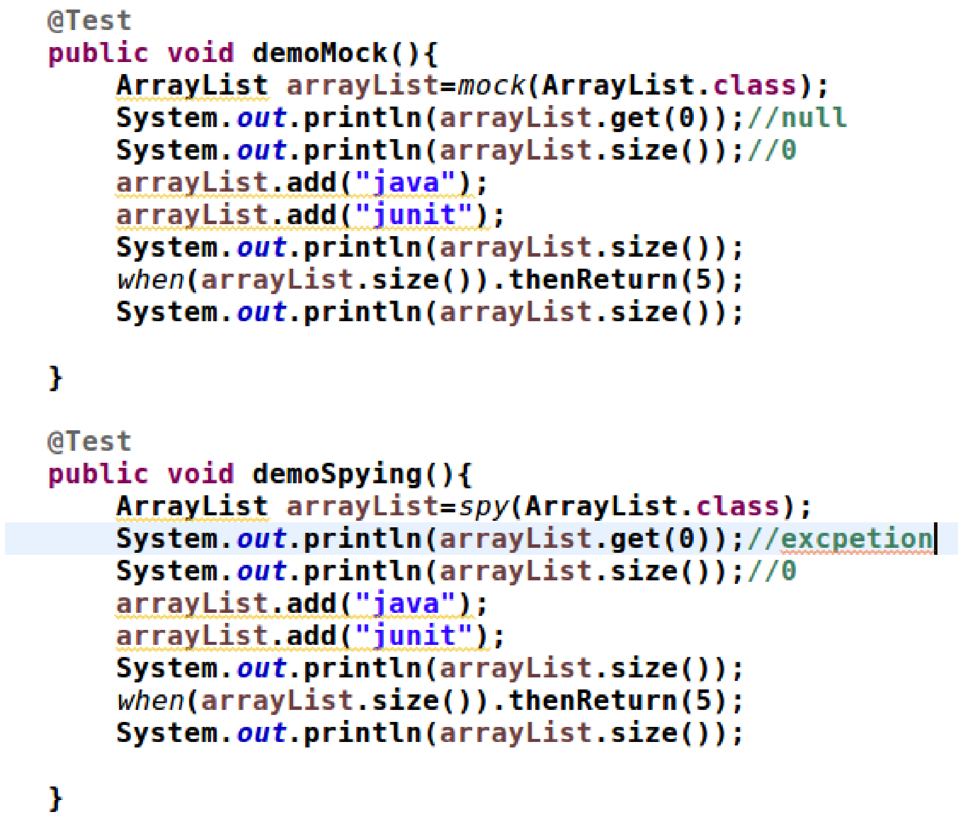


```
@RunWith(MockitoJUnitRunner.class)
public class BookServiceTest {

    @Mock
    private BookDao dao;

    @InjectMocks
    private BookServiceImpl service;
    @Before
    public void setUp(){

        List<Book>books=Arrays.asList(new Book(1, "phthon", "gunika", 100),
                new Book(2, "java", "gunika", 100),
                new Book(3, "spring java", "gunika", 100));

        dao=mock(BookDao.class);

        when(dao.getAllBooks()).thenReturn(books);

        service=new BookServiceImpl(dao);
    }
```

This dependency is automcatically created by mockito and will be injected to BookServiceImpl

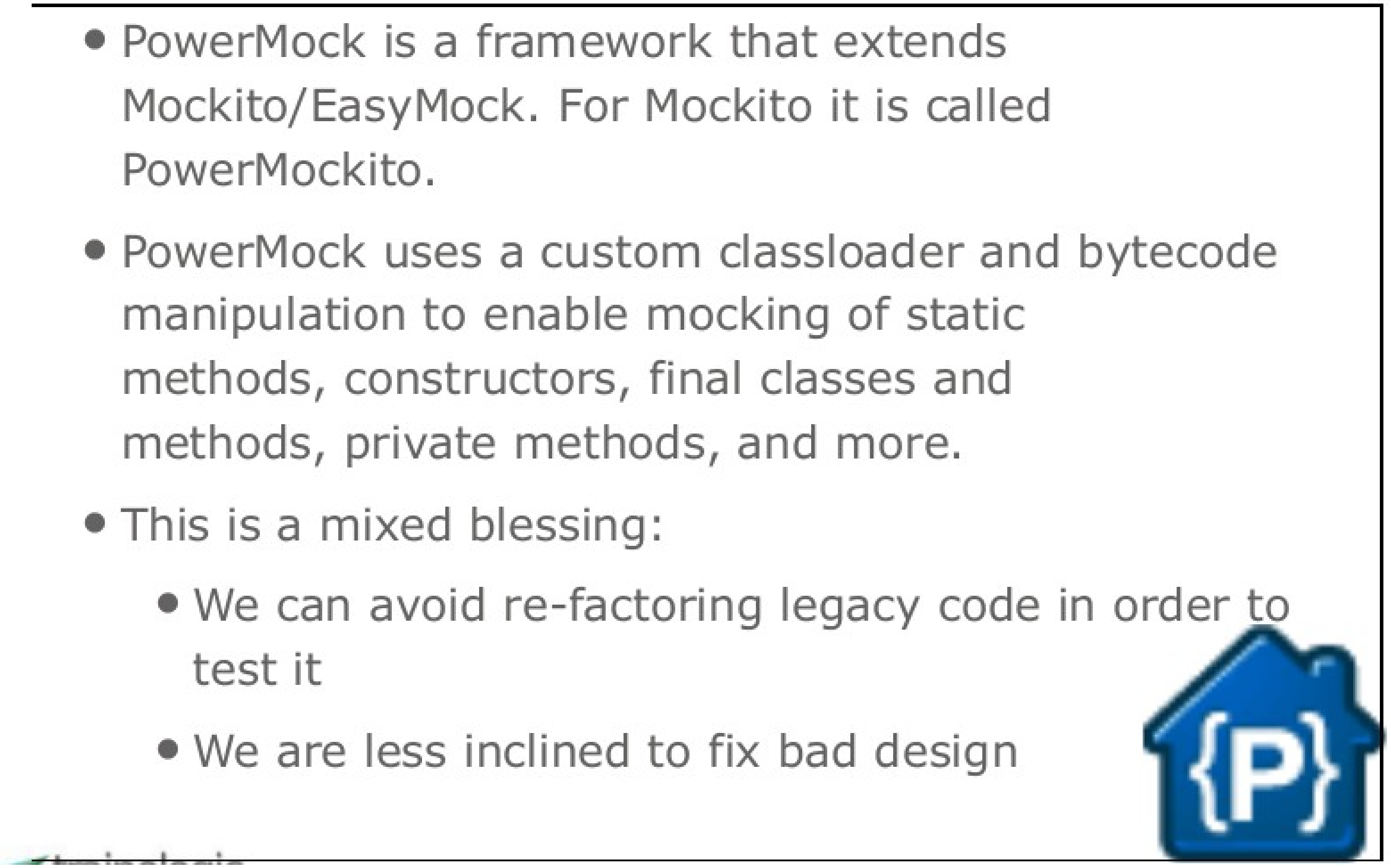
Mockito spying

- A Mockito mock allows us to stub a method call.
- That means we can stub a method to return a specific object. ...
- A Mockito spy is a partial mock. We can mock a part of the object by stubbing few methods, while real method invocations will be used for the other.



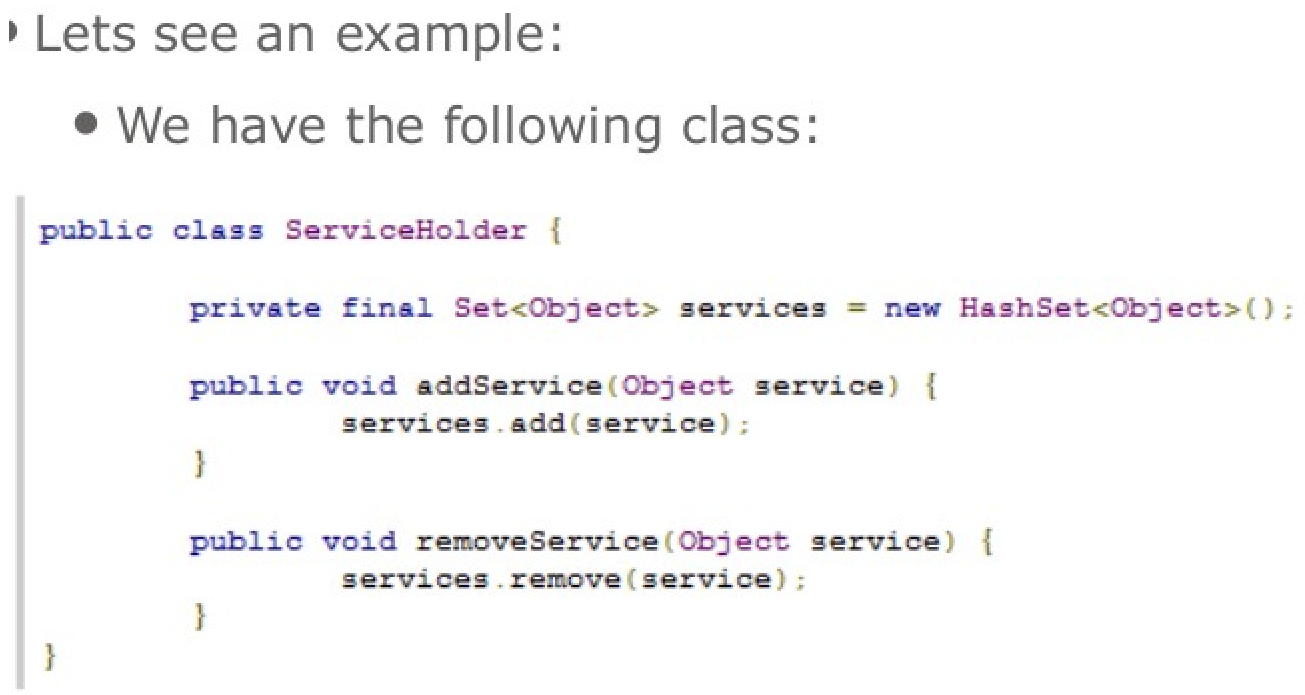
```
@Test
public void demoMock(){
    ArrayList arrayList=mock(ArrayList.class);
    System.out.println(arrayList.get(0));//null
    System.out.println(arrayList.size());//0
    arrayList.add("java");
    arrayList.add("junit");
    System.out.println(arrayList.size());
    when(arrayList.size()).thenReturn(5);
    System.out.println(arrayList.size());

}

@Test
public void demoSpying(){
    ArrayList arrayList=spy(ArrayList.class);
    System.out.println(arrayList.get(0));//excpetion
    System.out.println(arrayList.size());//0
    arrayList.add("java");
    arrayList.add("junit");
    System.out.println(arrayList.size());
    when(arrayList.size()).thenReturn(5);
    System.out.println(arrayList.size());

}
```

PowerMock



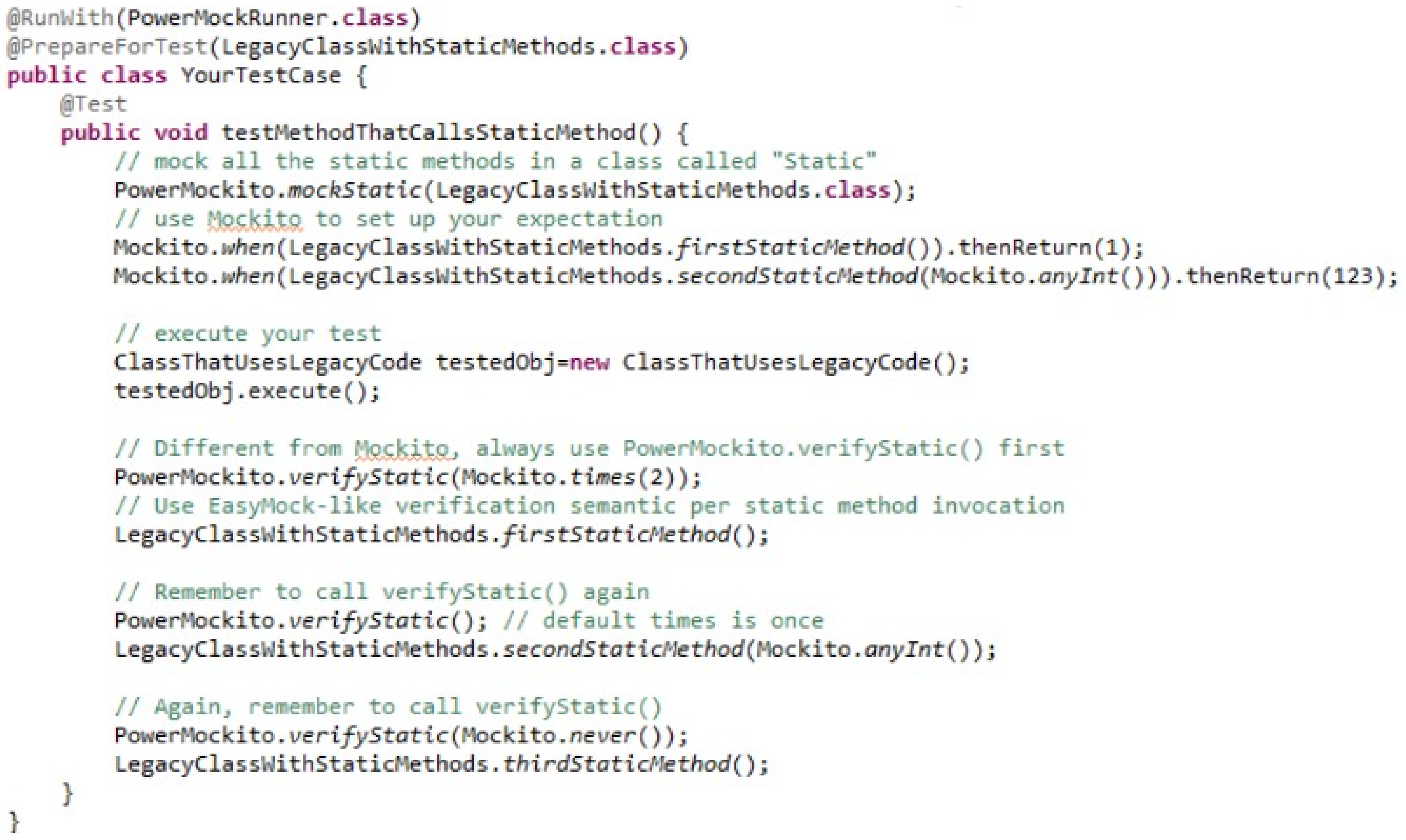
- PowerMock is a framework that extends Mockito/EasyMock. For Mockito it is called PowerMockito.
- PowerMock uses a custom classloader and bytecode manipulation to enable mocking of static methods, constructors, final classes and methods, private methods, and more.
- This is a mixed blessing:
  - We can avoid re-factoring legacy code in order to test it
  - We are less inclined to fix bad design

PowerMock Example



- Lets see an example:
  - We have the following class:

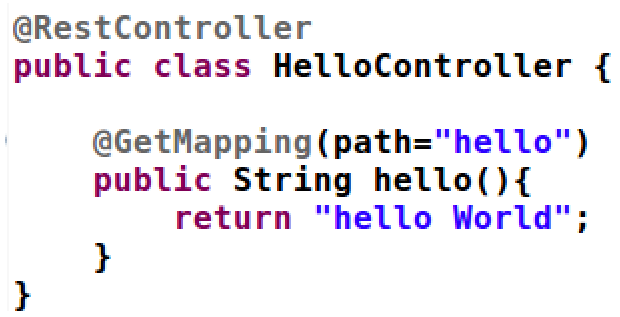
```java
public class ServiceHolder {

        private final Set<Object> services = new HashSet<Object>();

        public void addService(Object service) {
                services.add(service);
        }

        public void removeService(Object service) {
                services.remove(service);
        }
}
```



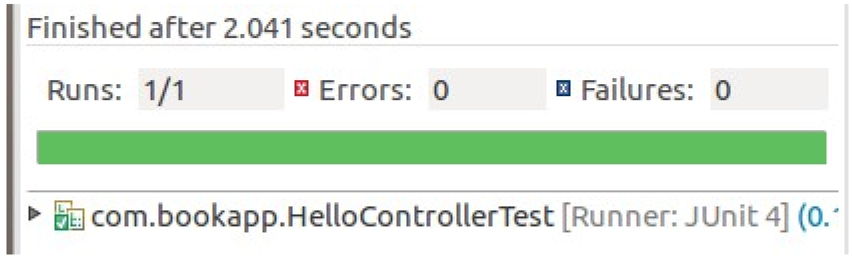
```
@RunWith(PowerMockRunner.class)
@PrepareForTest(LegacyClassWithStaticMethods.class)
public class YourTestCase {
    @Test
    public void testMethodThatCallsStaticMethod() {
        // mock all the static methods in a class called "Static"
        PowerMockito.mockStatic(LegacyClassWithStaticMethods.class);
        // use Mockito to set up your expectation
        Mockito.when(LegacyClassWithStaticMethods.firstStaticMethod()).thenReturn(1);
        Mockito.when(LegacyClassWithStaticMethods.secondStaticMethod(Mockito.anyInt())).thenReturn(123);

        // execute your test
        ClassThatUsesLegacyCode testedObj=new ClassThatUsesLegacyCode();
        testedObj.execute();

        // Different from Mockito, always use PowerMockito.verifyStatic() first
        PowerMockito.verifyStatic(Mockito.times(2));
        // Use EasyMock-like verification semantic per static method invocation
        LegacyClassWithStaticMethods.firstStaticMethod();

        // Remember to call verifyStatic() again
        PowerMockito.verifyStatic(); // default times is once
        LegacyClassWithStaticMethods.secondStaticMethod(Mockito.anyInt());

        // Again, remember to call verifyStatic()
        PowerMockito.verifyStatic(Mockito.never());
        LegacyClassWithStaticMethods.thirdStaticMethod();
    }
}
```

Hello world REST controller test

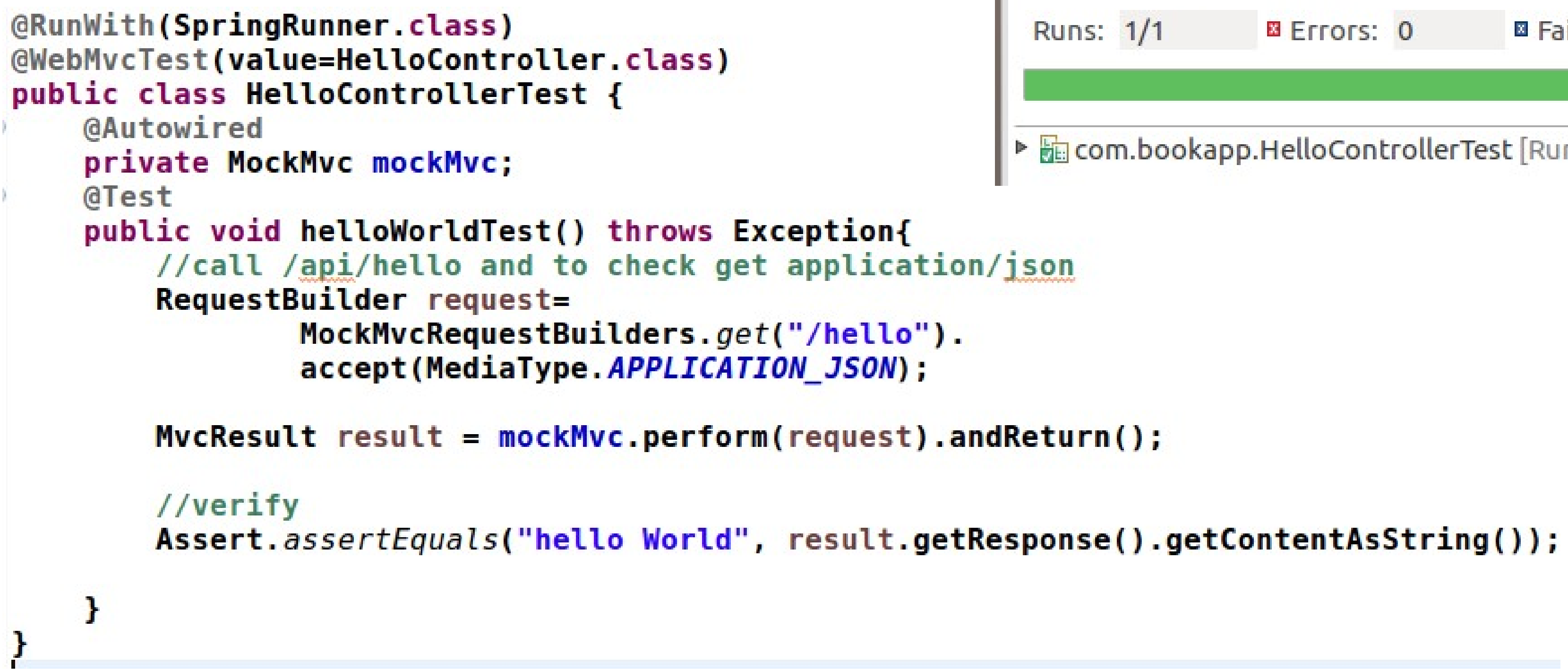
- We want to unit test,an controller without launching entire application
- SpringMockMvc framework help to solve this problem.



```
@RestController
public class HelloController {

    @GetMapping(path="hello")
    public String hello(){
        return "hello World";
    }
}
```



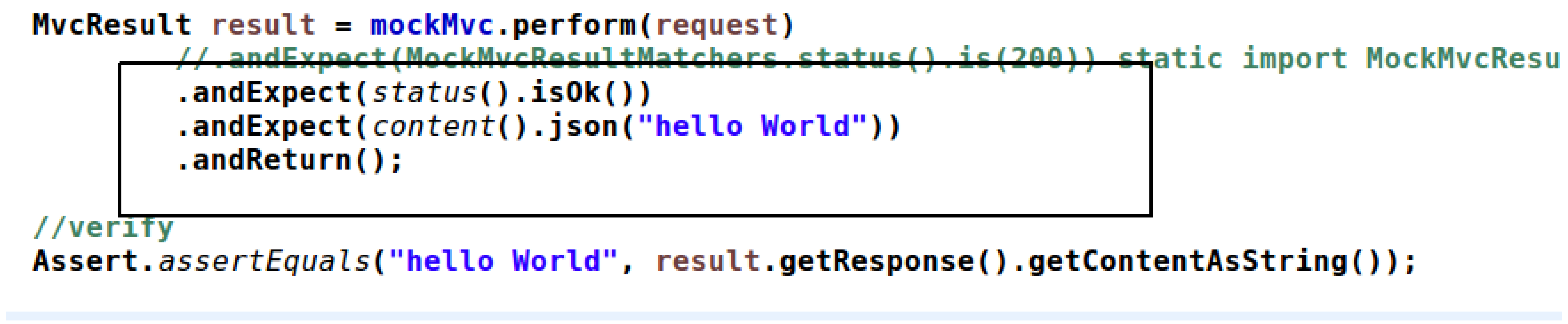
```
Finished after 2.041 seconds

Runs: 1/1    Errors: 0    Failures: 0

com.bookapp.HelloControllerTest [Runner: JUnit 4] (0.
```



```
@RunWith(SpringRunner.class)
@WebMvcTest(value=HelloController.class)
public class HelloControllerTest {
    @Autowired
    private MockMvc mockMvc;
    @Test
    public void helloWorldTest() throws Exception{
        //call /api/hello and to check get application/json
        RequestBuilder request=
                MockMvcRequestBuilders.get("/hello").
                accept(MediaType.APPLICATION_JSON);

        MvcResult result = mockMvc.perform(request).andReturn();

        //verify
        Assert.assertEquals("hello World", result.getResponse().getContentAsString());

    }
}
```

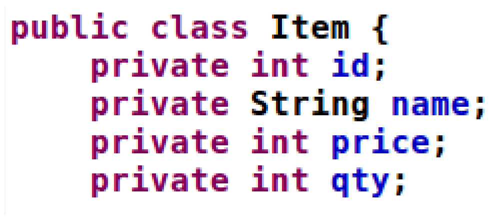
Hello world REST controller test

- Using response matchers to check contect and status

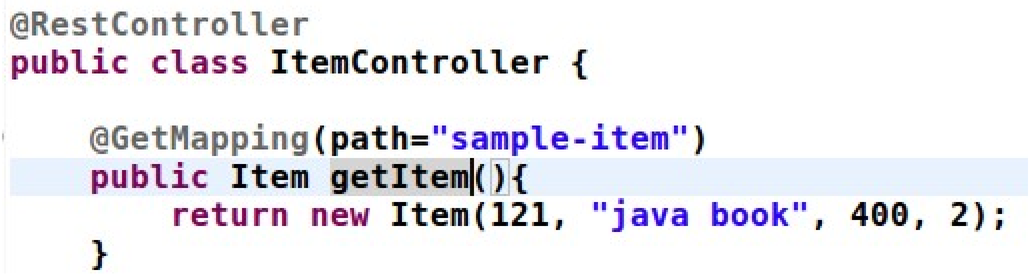


```
MvcResult result = mockMvc.perform(request)
        //.andExpect(MockMvcResultMatchers.status().is(200)) static import MockMvcResu
        .andExpect(status().isOk())
        .andExpect(content().json("hello World"))
        .andReturn();

//verify
Assert.assertEquals("hello World", result.getResponse().getContentAsString());
```

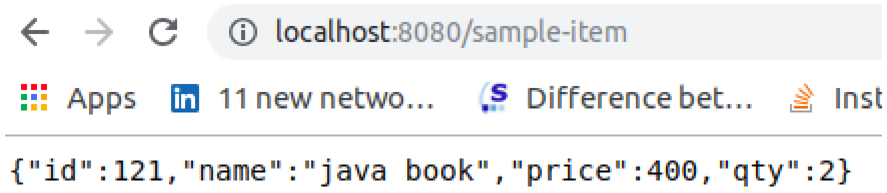
Unit testing Item controller



```java
public class Item {
    private int id;
    private String name;
    private int price;
    private int qty;
```



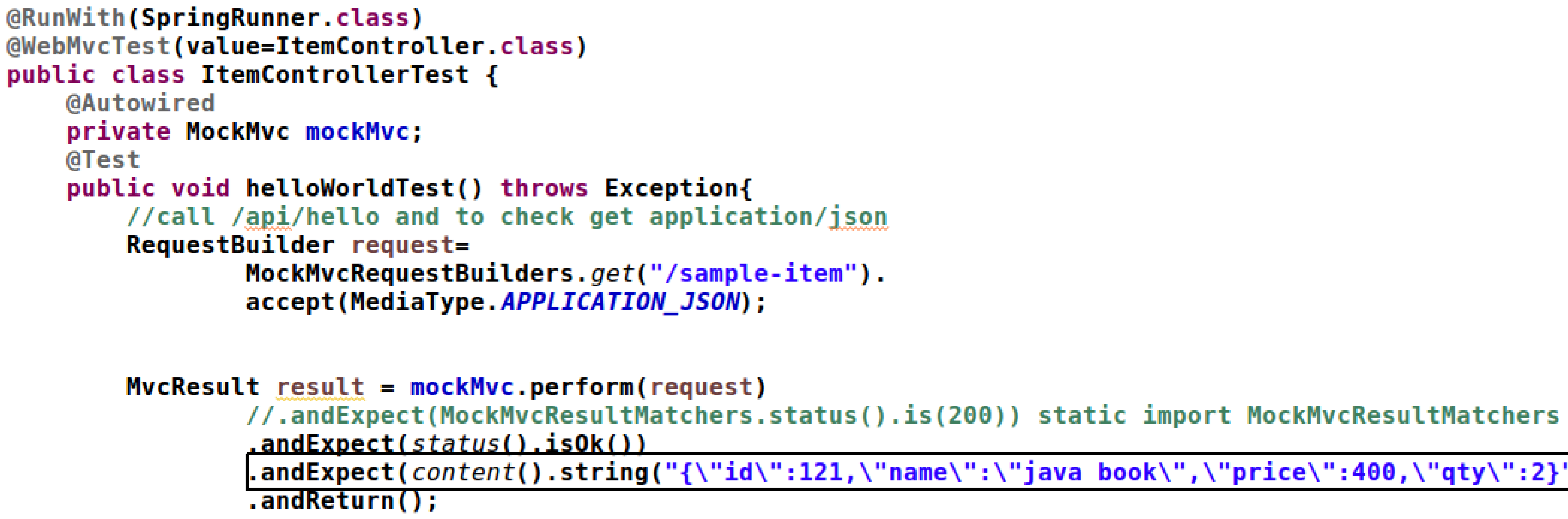
```java
@RestController
public class ItemController {

    @GetMapping(path="sample-item")
    public Item getItem(){
        return new Item(121, "java book", 400, 2);
    }
```



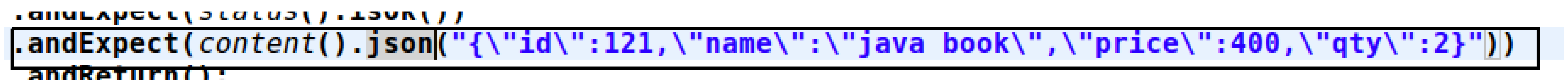
localhost:8080/sample-item

Apps 11 new netwo... Difference bet... Inst

```
{"id":121,"name":"java book","price":400,"qty":2}
```



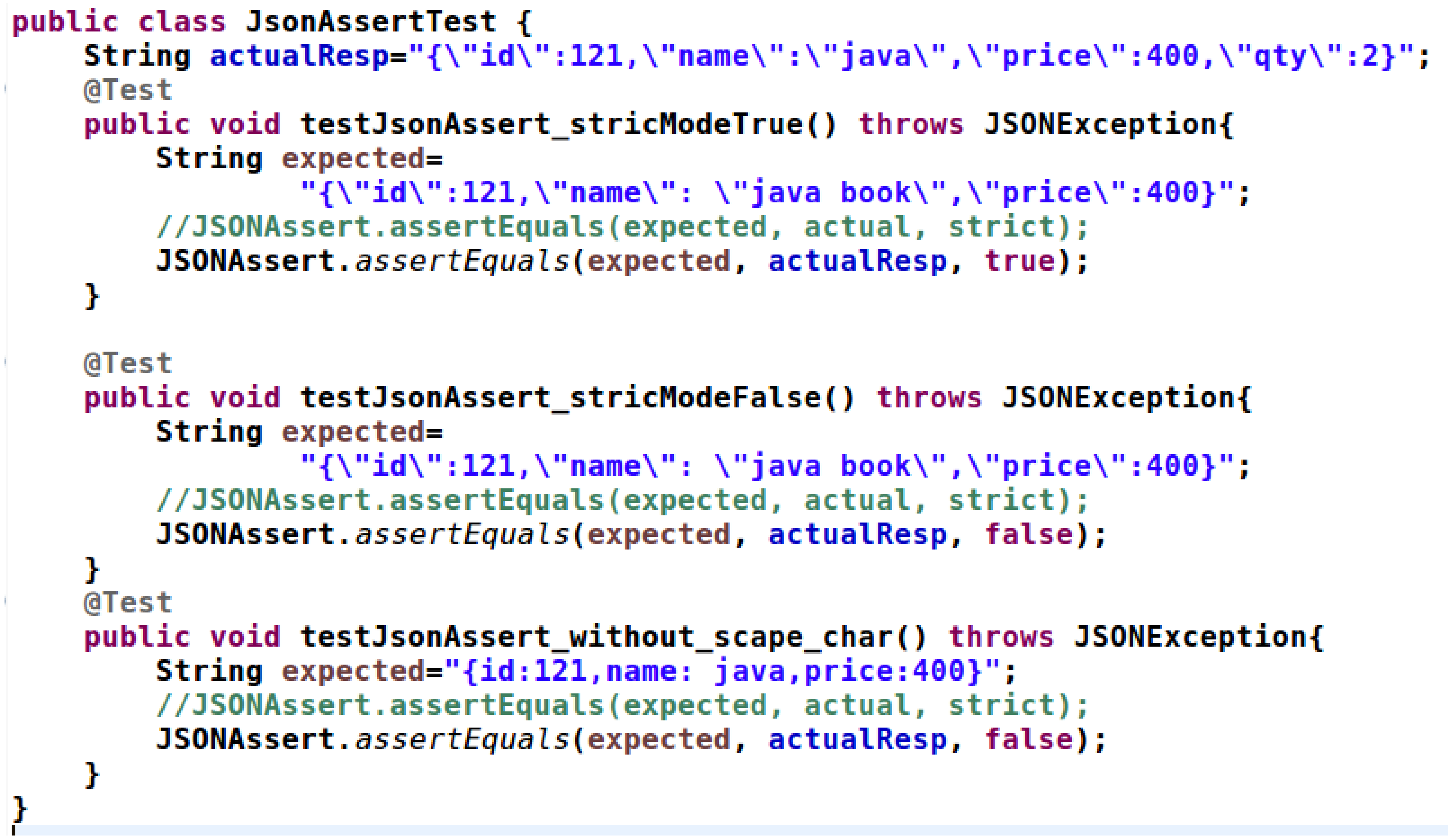
```java
@RunWith(SpringRunner.class)
@WebMvcTest(value=ItemController.class)
public class ItemControllerTest {
    @Autowired
    private MockMvc mockMvc;
    @Test
    public void helloWorldTest() throws Exception{
        //call /api/hello and to check get application/json
        RequestBuilder request=
                MockMvcRequestBuilders.get("/sample-item").
                accept(MediaType.APPLICATION_JSON);


        MvcResult result = mockMvc.perform(request)
                //.andExpect(MockMvcResultMatchers.status().is(200)) static import MockMvcResultMatchers
                .andExpect(status().isOk())
                .andExpect(content().string("{\"id\":121,\"name\":\"java book\",\"price\":400,\"qty\":2}"))
                .andReturn();
```



```java
.andExpect(status().isOk())
.andExpect(content().json("{\"id\":121,\"name\":\"java book\",\"price\":400,\"qty\":2}"))
```

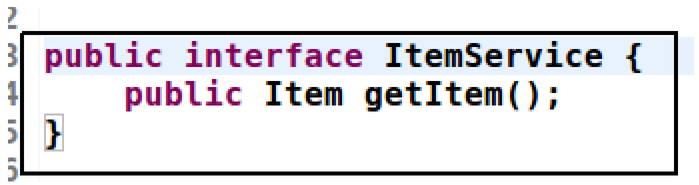
Json Assert method

JSON assert ignore missing space/extra spaces/missing data
Strick mode: true/false

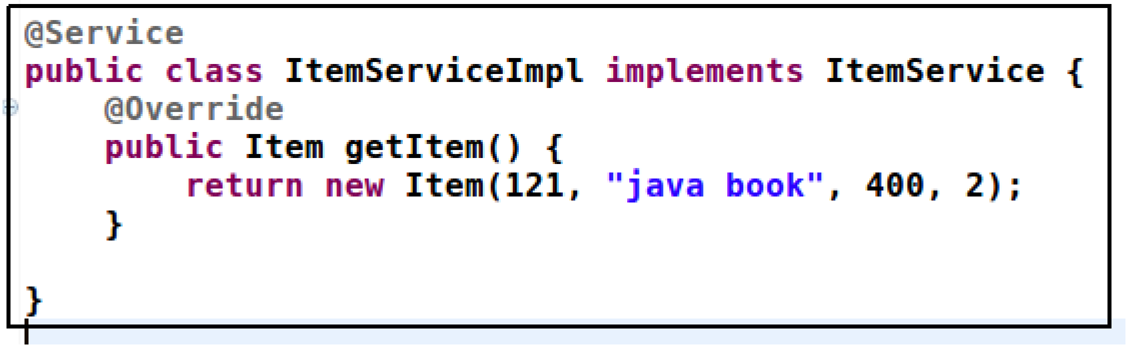


```java
public class JsonAssertTest {
    String actualResp="{\"id\":121,\"name\":\"java\",\"price\":400,\"qty\":2}";
    @Test
    public void testJsonAssert_stricModeTrue() throws JSONException{
        String expected=
                "{\"id\":121,\"name\": \"java book\",\"price\":400}";
        //JSONAssert.assertEquals(expected, actual, strict);
        JSONAssert.assertEquals(expected, actualResp, true);
    }

    @Test
    public void testJsonAssert_stricModeFalse() throws JSONException{
        String expected=
                "{\"id\":121,\"name\": \"java book\",\"price\":400}";
        //JSONAssert.assertEquals(expected, actual, strict);
        JSONAssert.assertEquals(expected, actualResp, false);
    }
    @Test
    public void testJsonAssert_without_scape_char() throws JSONException{
        String expected="{id:121,name: java,price:400}";
        //JSONAssert.assertEquals(expected, actual, strict);
        JSONAssert.assertEquals(expected, actualResp, false);
    }
}
```

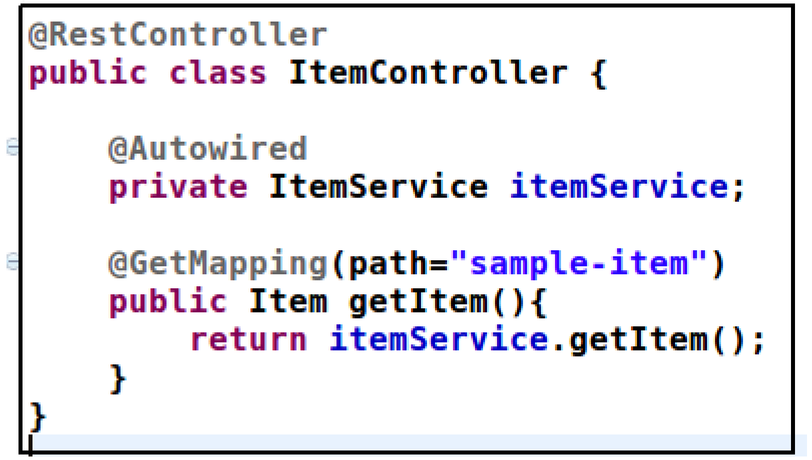
Now lets introduce service layer



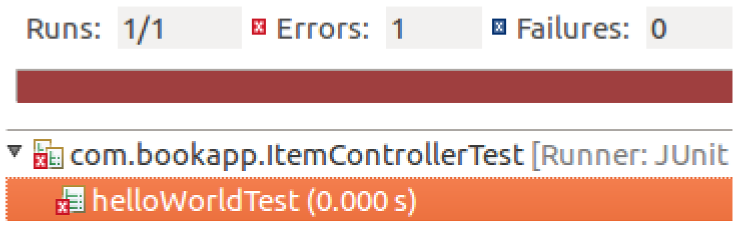
```java
public interface ItemService {
    public Item getItem();
}
```



```java
@Service
public class ItemServiceImpl implements ItemService {
    @Override
    public Item getItem() {
        return new Item(121, "java book", 400, 2);
    }

}
```



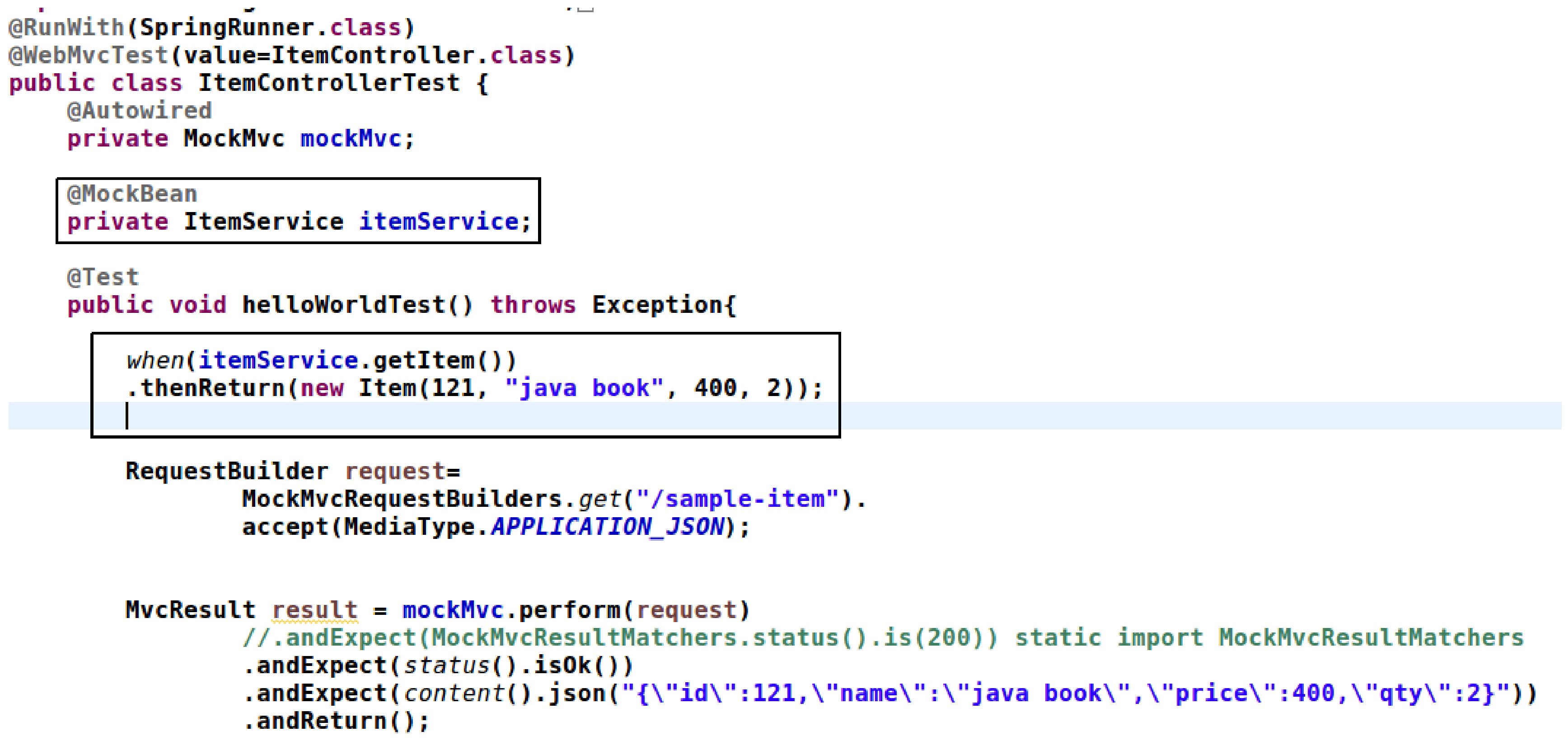
```java
@RestController
public class ItemController {

    @Autowired
    private ItemService itemService;

    @GetMapping(path="sample-item")
    public Item getItem(){
        return itemService.getItem();
    }
}
```



```
Runs: 1/1    Errors: 1    Failures: 0

▾ com.bookapp.ItemControllerTest [Runner: JUnit
    helloWorldTest (0.000 s)
```

Mocking service layer

- Business service need to be mocked as we are doing unit testing for Rest endpoint



```java
@RunWith(SpringRunner.class)
@WebMvcTest(value=ItemController.class)
public class ItemControllerTest {
    @Autowired
    private MockMvc mockMvc;

    @MockBean
    private ItemService itemService;

    @Test
    public void helloWorldTest() throws Exception{

        when(itemService.getItem())
        .thenReturn(new Item(121, "java book", 400, 2));


        RequestBuilder request=
                MockMvcRequestBuilders.get("/sample-item").
                accept(MediaType.APPLICATION_JSON);


        MvcResult result = mockMvc.perform(request)
                //.andExpect(MockMvcResultMatchers.status().is(200)) static import MockMvcResultMatchers
                .andExpect(status().isOk())
                .andExpect(content().json("{\"id\":121,\"name\":\"java book\",\"price\":400,\"qty\":2}"))
                .andReturn();
```

Mocking service layer

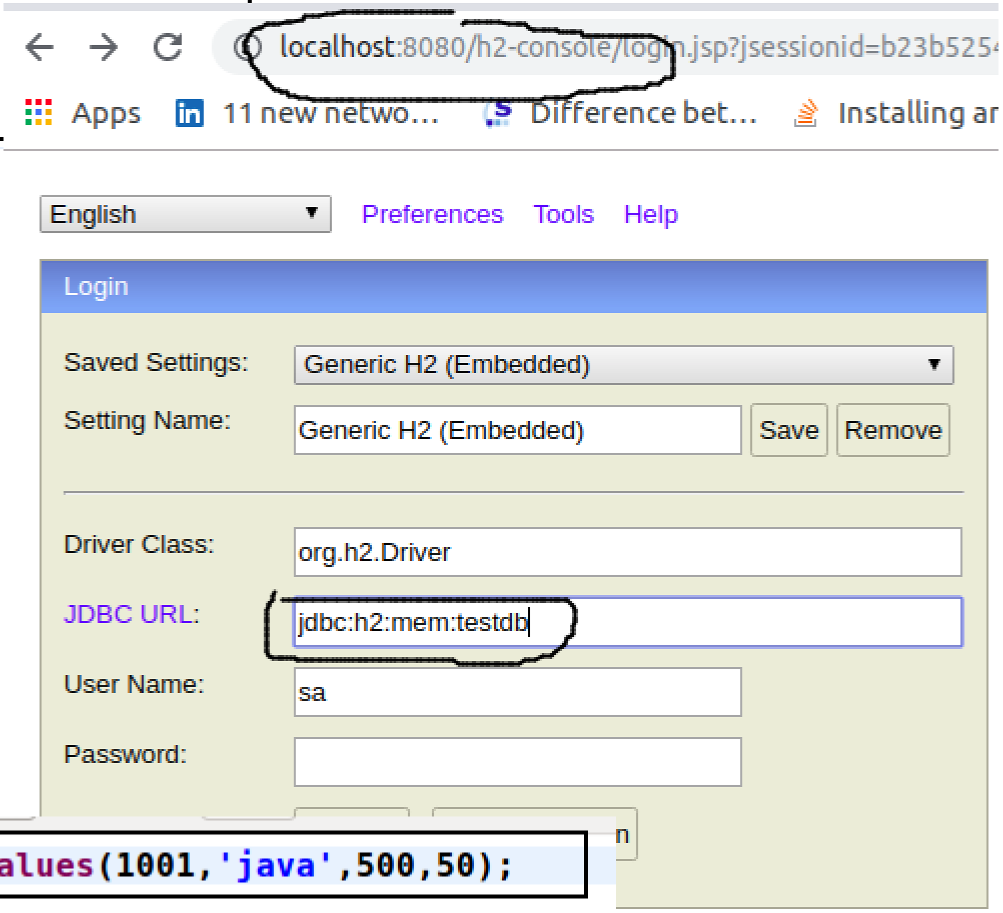
- Without mocking service layer, it will be an integration test, and service layer will talk all the way to dao and database which is not required
- Now we are totally isolated from service layer and true unit test

Integrating with data layer H2

```xml
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
</dependency>
```

```java
@Entity
@Table(name="item_table")
public class Item {
    @Id
    private int id;
    private String name;
    private int price;
    private int qty;

    @Transient
    private int value;
```

```properties
spring.jpa.show-sql=true
spring.h2.console.enabled=true
```



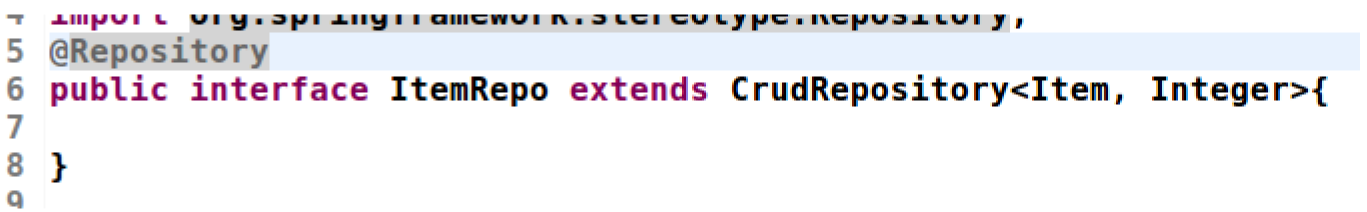
localhost:8080/h2-console/login.jsp?jsessionid=b23b525

Saved Settings:	Generic H2 (Embedded)
Setting Name:	Generic H2 (Embedded)
Driver Class:	org.h2.Driver
JDBC URL:	jdbc:h2:mem:testdb
User Name:	sa
Password:	

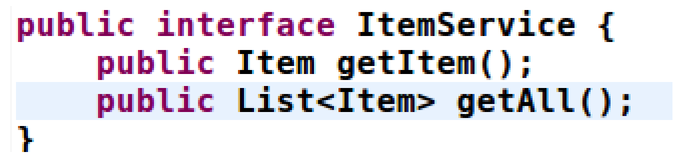
```sql
insert into item_table(id, name,price,qty) values(1001,'java',500,50);
```

put data in data.sql keep it into resources folder

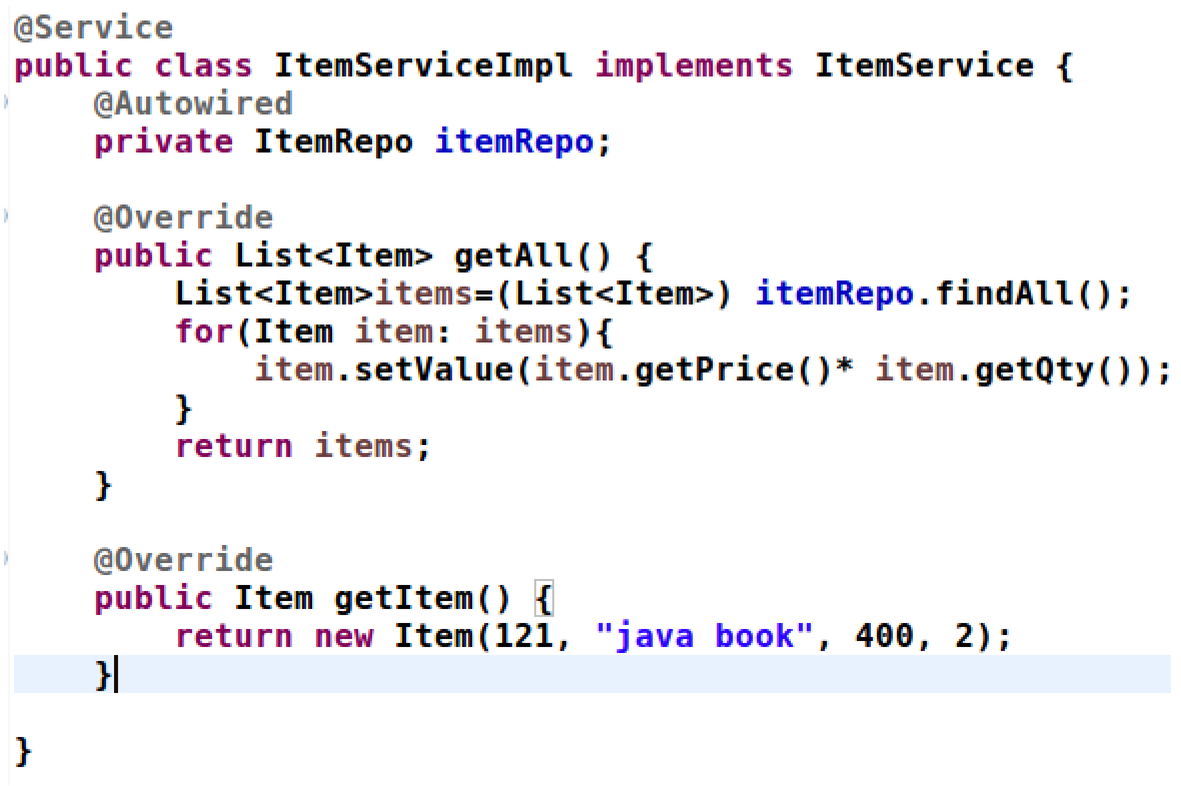
Creating dao and service layer



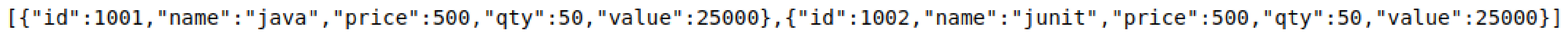
```java
import org.springframework.stereotype.Repository;
@Repository
public interface ItemRepo extends CrudRepository<Item, Integer>{

}
```



```java
public interface ItemService {
    public Item getItem();
    public List<Item> getAll();
}
```

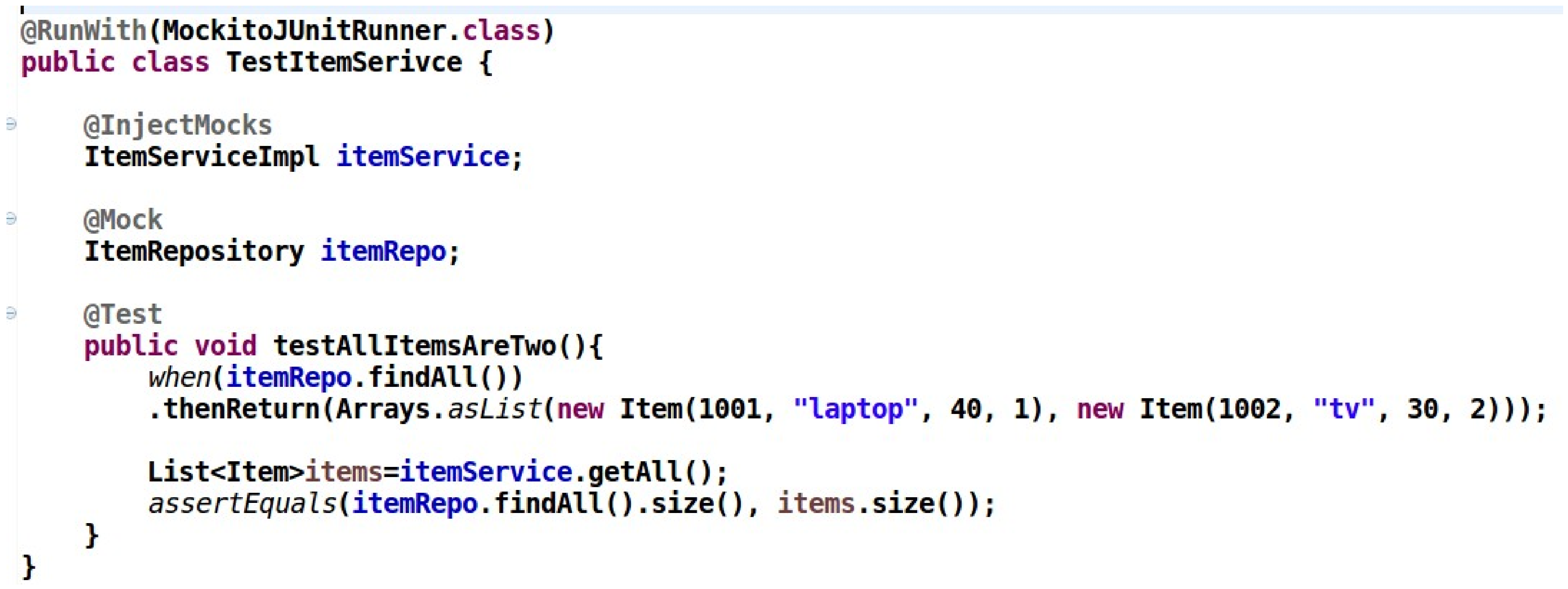


```java
@Service
public class ItemServiceImpl implements ItemService {
    @Autowired
    private ItemRepo itemRepo;

    @Override
    public List<Item> getAll() {
        List<Item>items=(List<Item>) itemRepo.findAll();
        for(Item item: items){
            item.setValue(item.getPrice()* item.getQty());
        }
        return items;
    }

    @Override
    public Item getItem() {
        return new Item(121, "java book", 400, 2);
    }

}
```



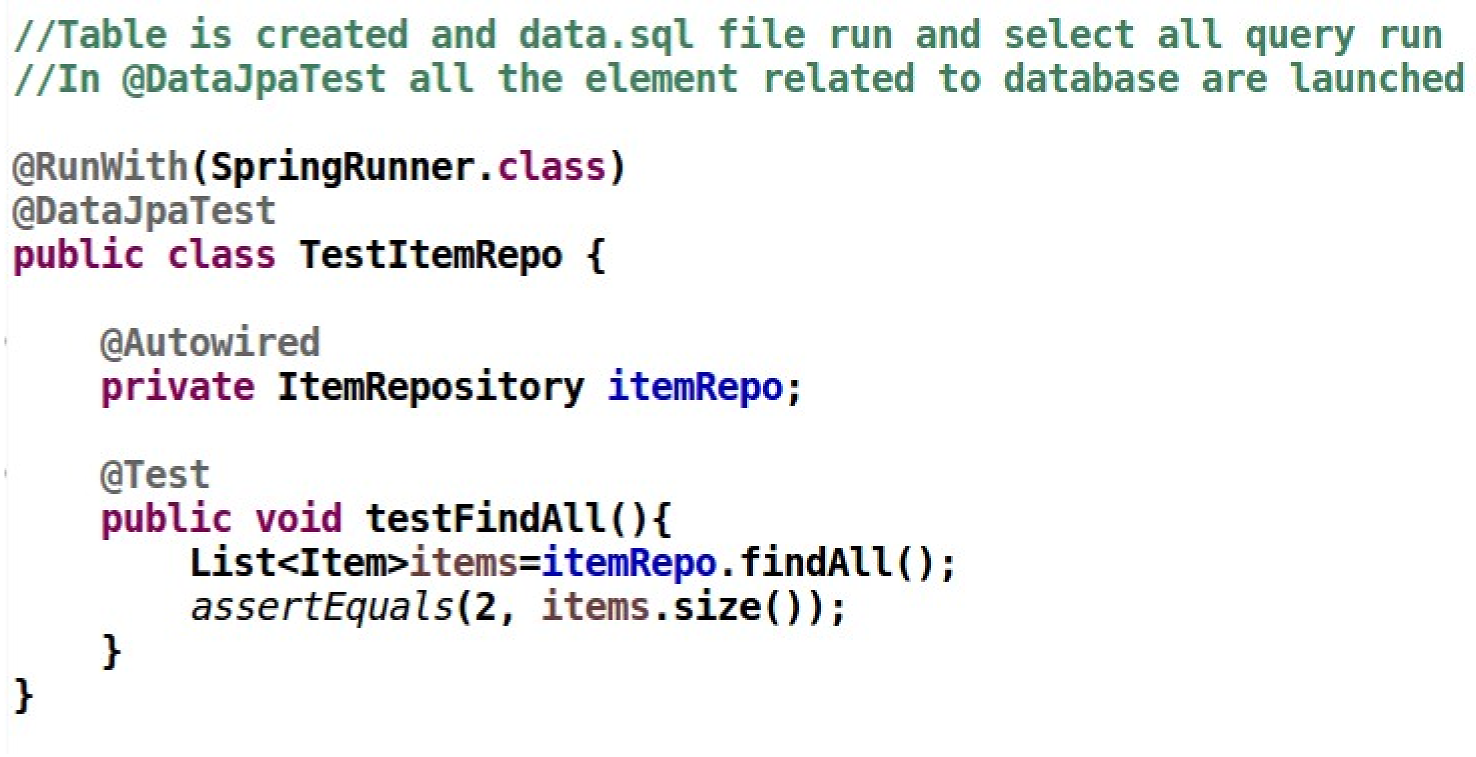
```
[{"id":1001,"name":"java","price":500,"qty":50,"value":25000},{"id":1002,"name":"junit","price":500,"qty":50,"value":25000}]
```

Testing service layer

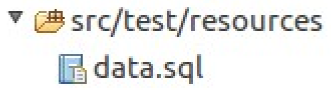


```java
@RunWith(MockitoJUnitRunner.class)
public class TestItemSerivce {

    @InjectMocks
    ItemServiceImpl itemService;

    @Mock
    ItemRepository itemRepo;

    @Test
    public void testAllItemsAreTwo(){
        when(itemRepo.findAll())
        .thenReturn(Arrays.asList(new Item(1001, "laptop", 40, 1), new Item(1002, "tv", 30, 2)));

        List<Item>items=itemService.getAll();
        assertEquals(itemRepo.findAll().size(), items.size());
    }
}
```

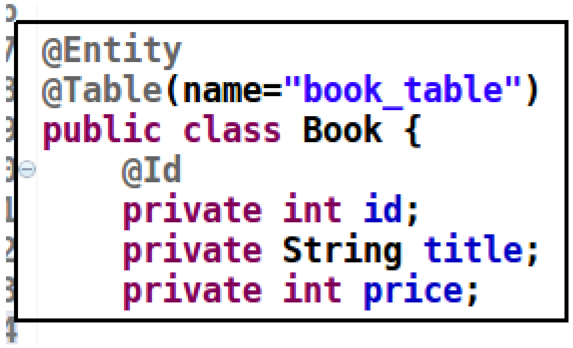
Testing dao layer



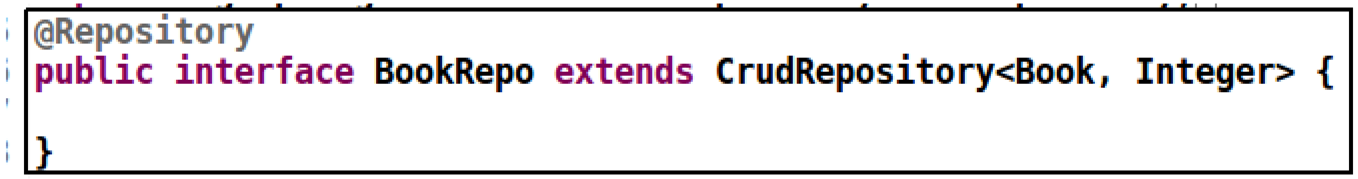
```java
//Table is created and data.sql file run and select all query run
//In @DataJpaTest all the element related to database are launched

@RunWith(SpringRunner.class)
@DataJpaTest
public class TestItemRepo {

    @Autowired
    private ItemRepository itemRepo;

    @Test
    public void testFindAll(){
        List<Item>items=itemRepo.findAll();
        assertEquals(2, items.size());
    }
}
```



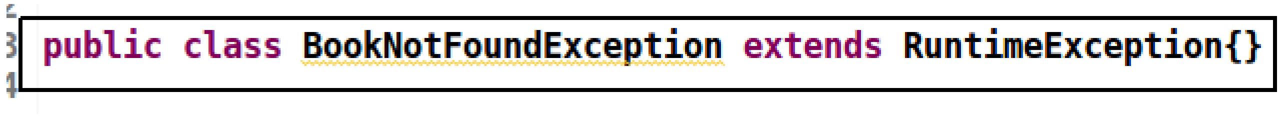
Example Book Application



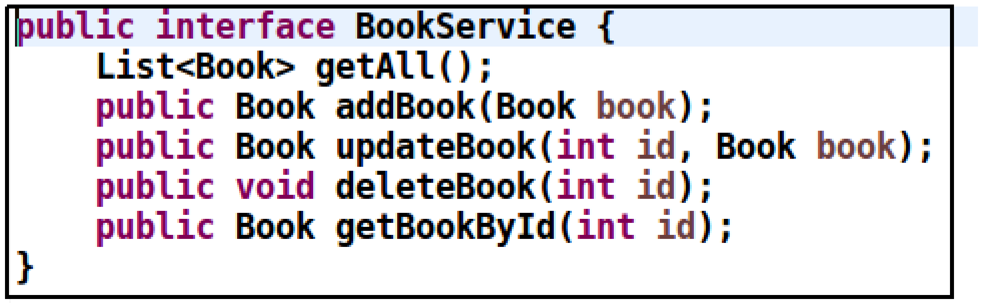
```java
@Entity
@Table(name="book_table")
public class Book {
    @Id
    private int id;
    private String title;
    private int price;
```



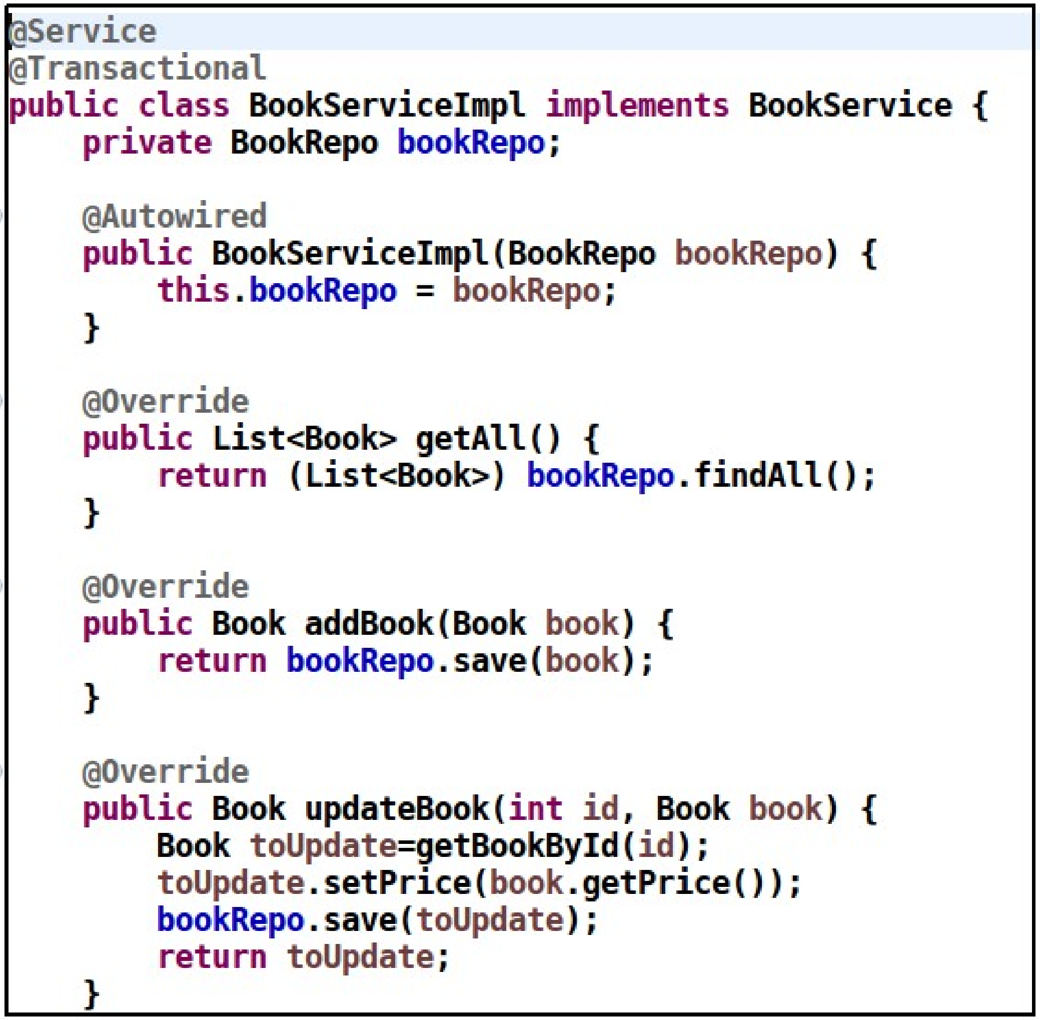
```java
@Repository
public interface BookRepo extends CrudRepository<Book, Integer> {

}
```



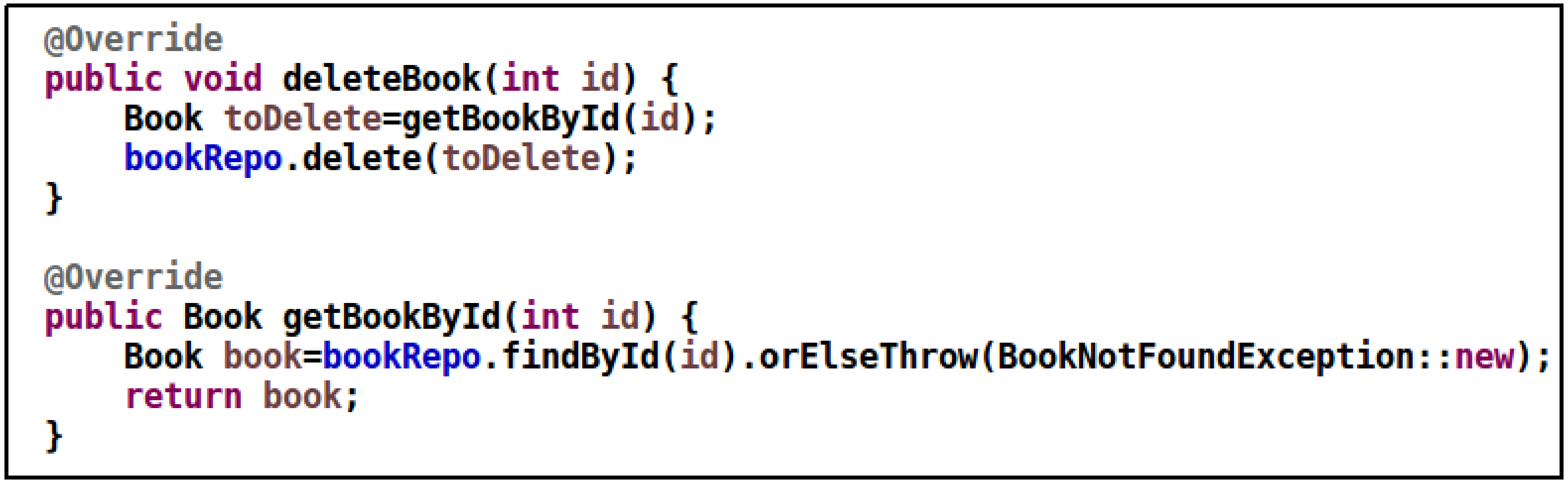
```java
public class BookNotFoundException extends RuntimeException{}
```



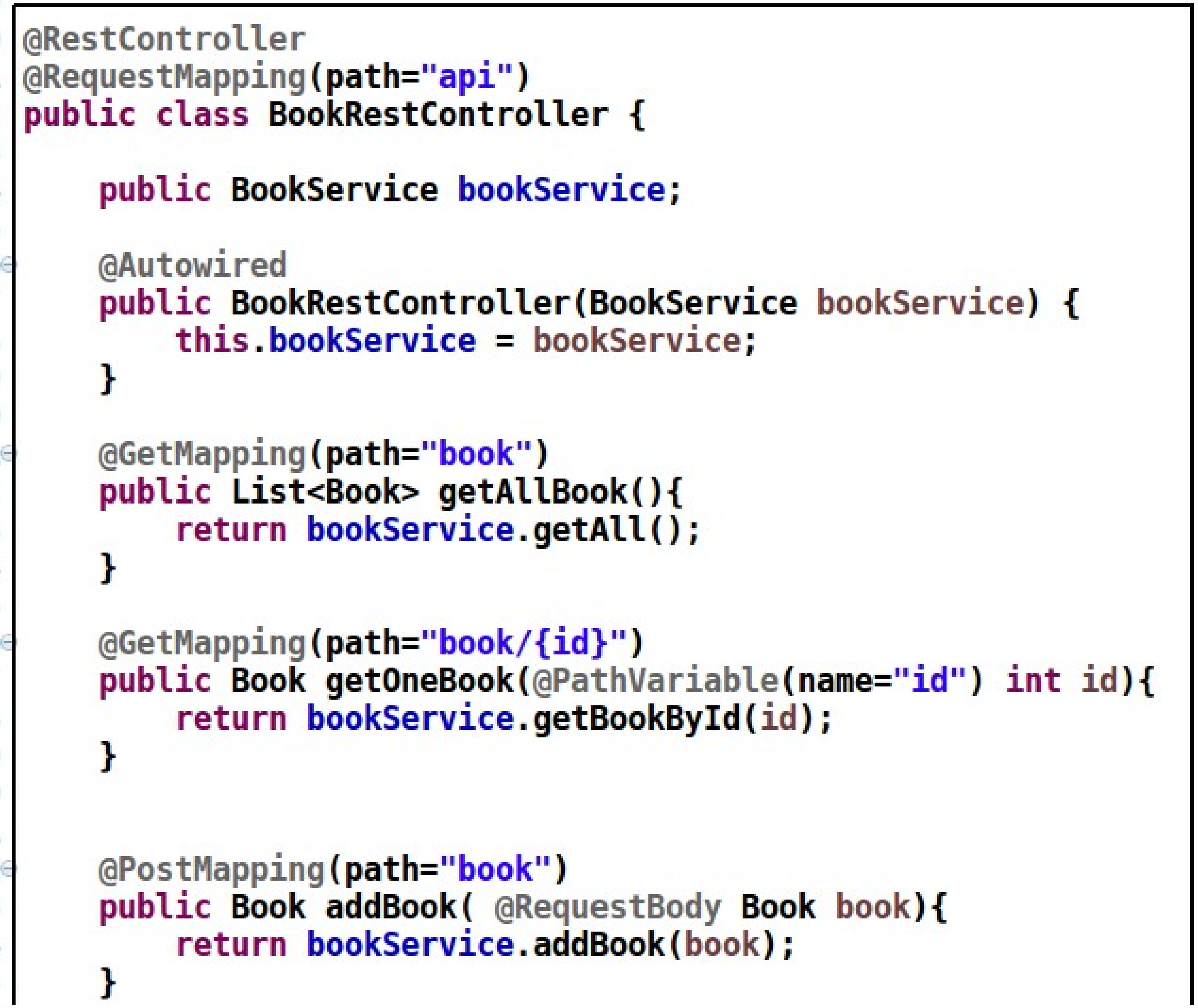
```java
public interface BookService {
    List<Book> getAll();
    public Book addBook(Book book);
    public Book updateBook(int id, Book book);
    public void deleteBook(int id);
    public Book getBookById(int id);
}
```



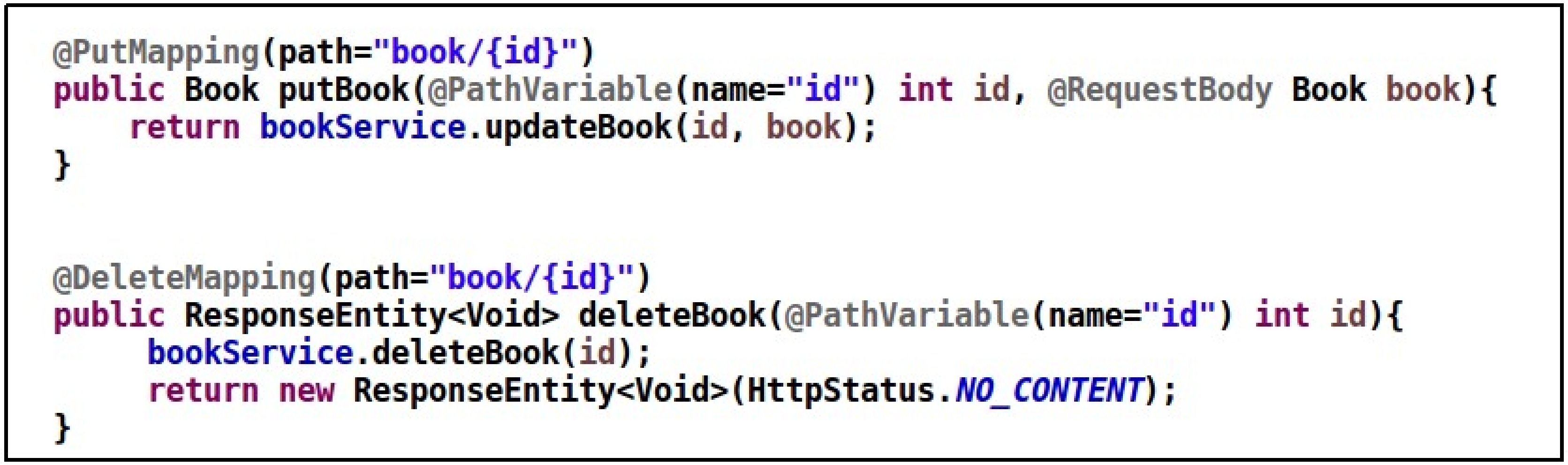
```java
@Service
@Transactional
public class BookServiceImpl implements BookService {
    private BookRepo bookRepo;

    @Autowired
    public BookServiceImpl(BookRepo bookRepo) {
        this.bookRepo = bookRepo;
    }

    @Override
    public List<Book> getAll() {
        return (List<Book>) bookRepo.findAll();
    }

    @Override
    public Book addBook(Book book) {
        return bookRepo.save(book);
    }

    @Override
    public Book updateBook(int id, Book book) {
        Book toUpdate=getBookById(id);
        toUpdate.setPrice(book.getPrice());
        bookRepo.save(toUpdate);
        return toUpdate;
    }
```



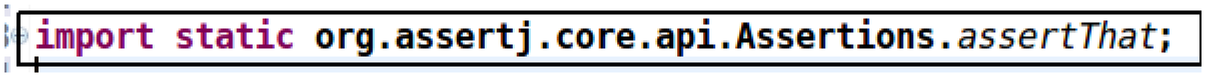
```java
@Override
public void deleteBook(int id) {
    Book toDelete=getBookById(id);
    bookRepo.delete(toDelete);
}

@Override
public Book getBookById(int id) {
    Book book=bookRepo.findById(id).orElseThrow(BookNotFoundException::new);
    return book;
}
```



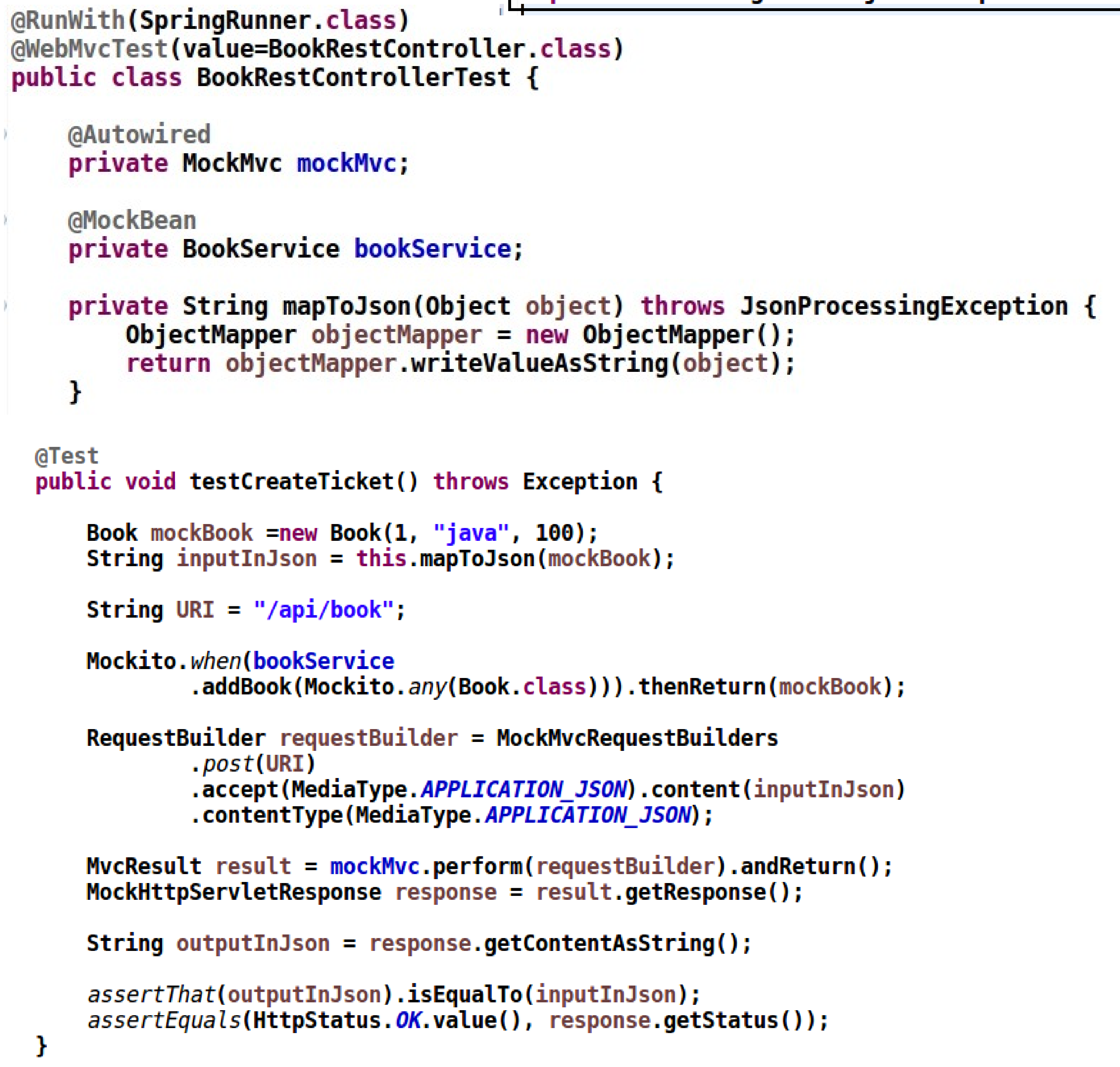
```java
@RestController
@RequestMapping(path="api")
public class BookRestController {

    public BookService bookService;

    @Autowired
    public BookRestController(BookService bookService) {
        this.bookService = bookService;
    }

    @GetMapping(path="book")
    public List<Book> getAllBook(){
        return bookService.getAll();
    }

    @GetMapping(path="book/{id}")
    public Book getOneBook(@PathVariable(name="id") int id){
        return bookService.getBookById(id);
    }


    @PostMapping(path="book")
    public Book addBook( @RequestBody Book book){
        return bookService.addBook(book);
    }
```



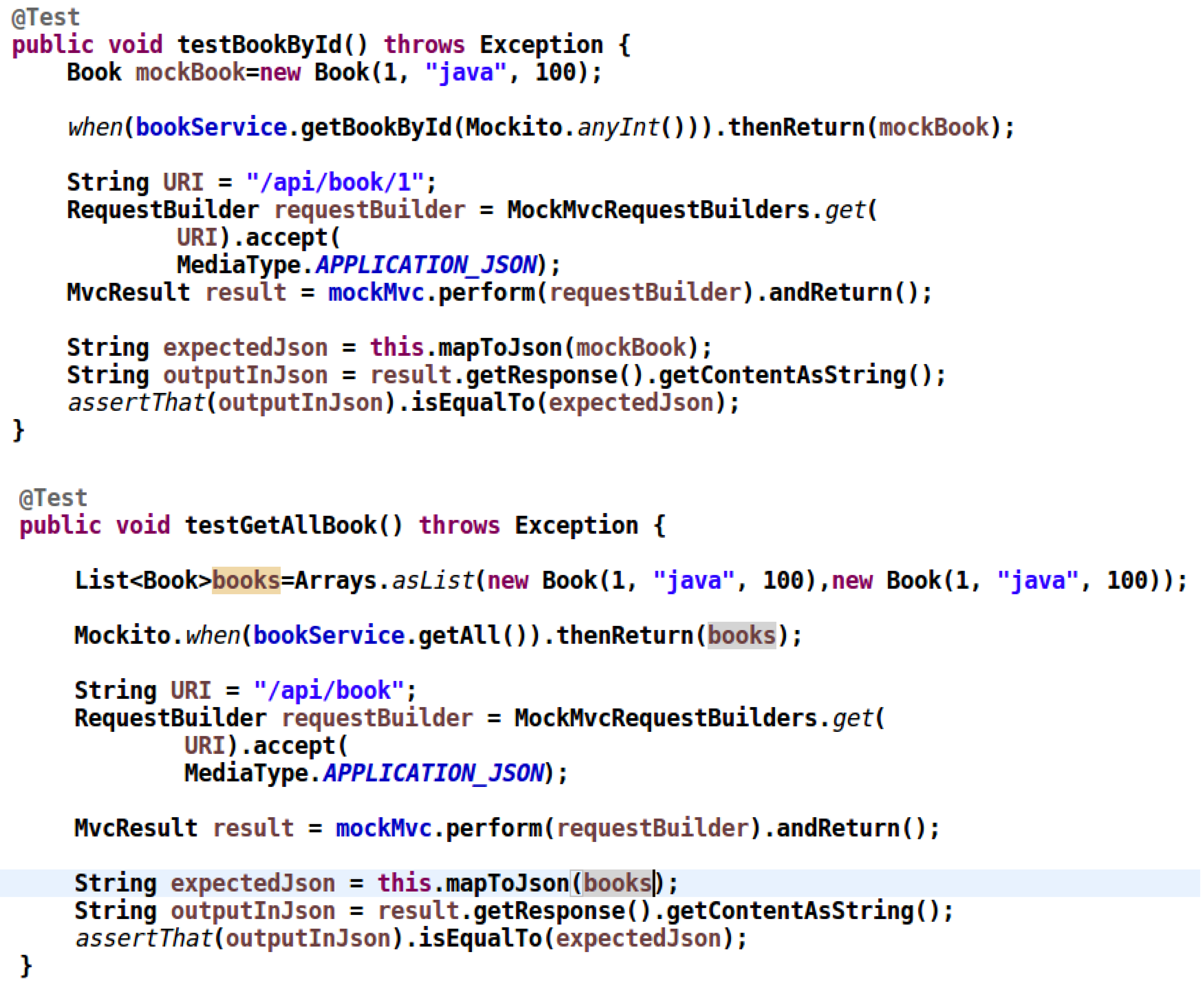
```java
  @PutMapping(path="book/{id}")
  public Book putBook(@PathVariable(name="id") int id, @RequestBody Book book){
      return bookService.updateBook(id, book);
  }


  @DeleteMapping(path="book/{id}")
  public ResponseEntity<Void> deleteBook(@PathVariable(name="id") int id){
       bookService.deleteBook(id);
       return new ResponseEntity<Void>(HttpStatus.NO_CONTENT);
  }
```



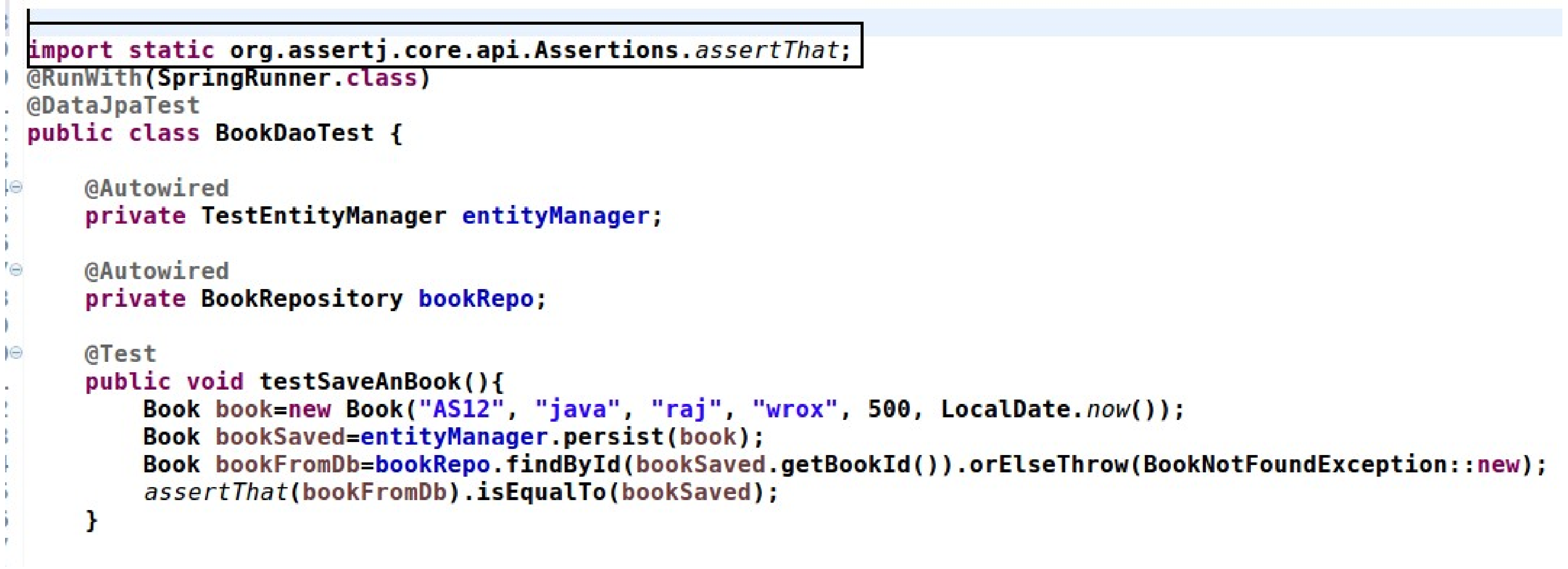
```java
import static org.assertj.core.api.Assertions.assertThat;
```



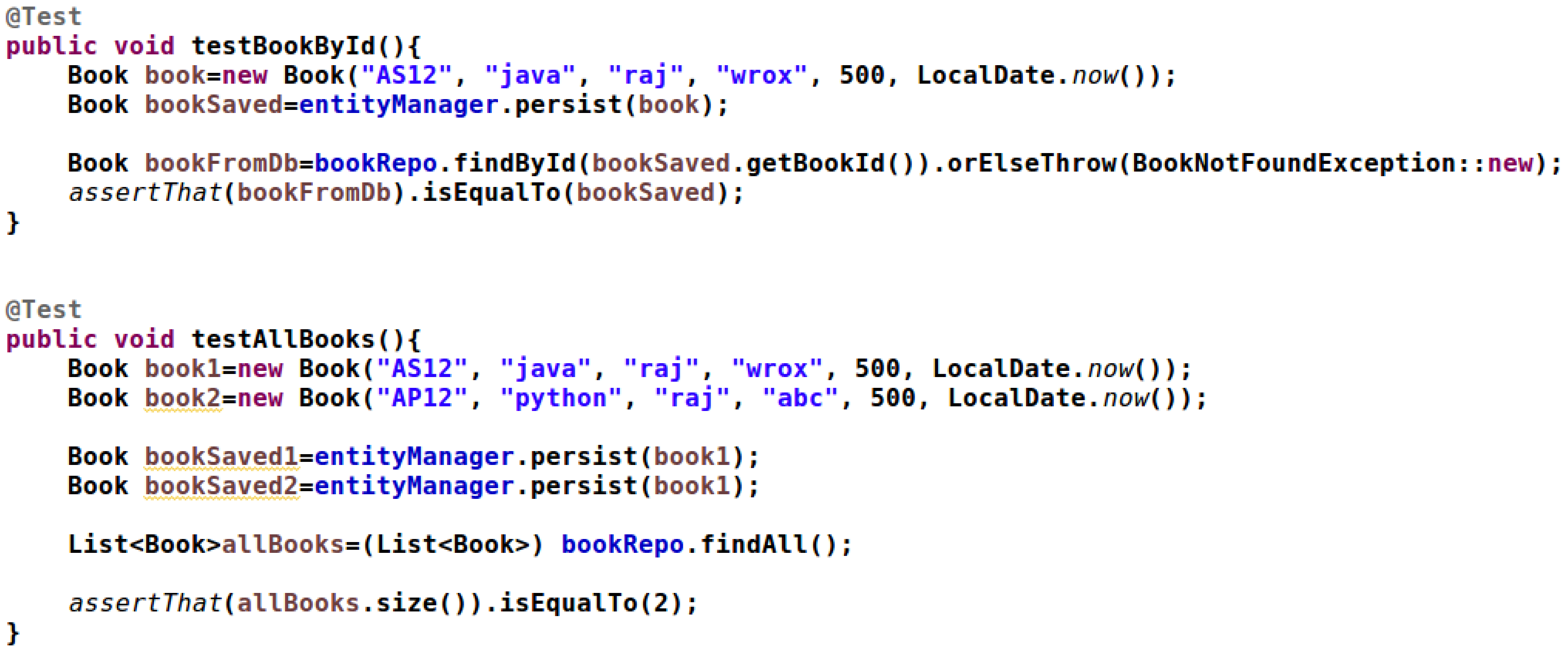
```java
@RunWith(SpringRunner.class)
@WebMvcTest(value=BookRestController.class)
public class BookRestControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @MockBean
    private BookService bookService;

    private String mapToJson(Object object) throws JsonProcessingException {
        ObjectMapper objectMapper = new ObjectMapper();
        return objectMapper.writeValueAsString(object);
    }

  @Test
  public void testCreateTicket() throws Exception {

      Book mockBook =new Book(1, "java", 100);
      String inputInJson = this.mapToJson(mockBook);

      String URI = "/api/book";

      Mockito.when(bookService
              .addBook(Mockito.any(Book.class))).thenReturn(mockBook);

      RequestBuilder requestBuilder = MockMvcRequestBuilders
              .post(URI)
              .accept(MediaType.APPLICATION_JSON).content(inputInJson)
              .contentType(MediaType.APPLICATION_JSON);

      MvcResult result = mockMvc.perform(requestBuilder).andReturn();
      MockHttpServletResponse response = result.getResponse();

      String outputInJson = response.getContentAsString();

      assertThat(outputInJson).isEqualTo(inputInJson);
      assertEquals(HttpStatus.OK.value(), response.getStatus());
  }
```



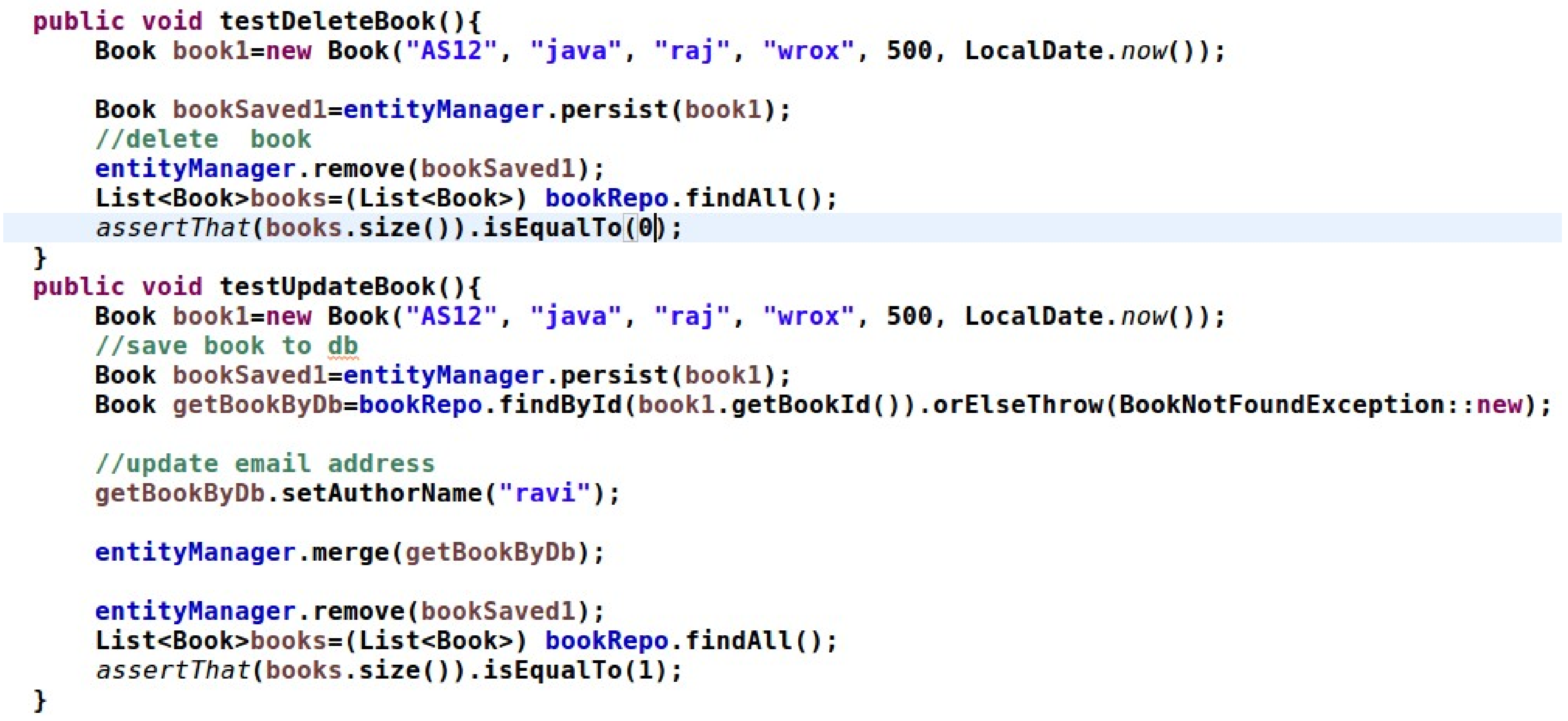
```java
@Test
public void testBookById() throws Exception {
    Book mockBook=new Book(1, "java", 100);

    when(bookService.getBookById(Mockito.anyInt())).thenReturn(mockBook);

    String URI = "/api/book/1";
    RequestBuilder requestBuilder = MockMvcRequestBuilders.get(
            URI).accept(
            MediaType.APPLICATION_JSON);
    MvcResult result = mockMvc.perform(requestBuilder).andReturn();

    String expectedJson = this.mapToJson(mockBook);
    String outputInJson = result.getResponse().getContentAsString();
    assertThat(outputInJson).isEqualTo(expectedJson);
}


@Test
public void testGetAllBook() throws Exception {

    List<Book>books=Arrays.asList(new Book(1, "java", 100),new Book(1, "java", 100));

    Mockito.when(bookService.getAll()).thenReturn(books);

    String URI = "/api/book";
    RequestBuilder requestBuilder = MockMvcRequestBuilders.get(
            URI).accept(
            MediaType.APPLICATION_JSON);

    MvcResult result = mockMvc.perform(requestBuilder).andReturn();

    String expectedJson = this.mapToJson(books);
    String outputInJson = result.getResponse().getContentAsString();
    assertThat(outputInJson).isEqualTo(expectedJson);
}
```



```java
import static org.assertj.core.api.Assertions.assertThat;
@RunWith(SpringRunner.class)
@DataJpaTest
public class BookDaoTest {

    @Autowired
    private TestEntityManager entityManager;

    @Autowired
    private BookRepository bookRepo;

    @Test
    public void testSaveAnBook(){
        Book book=new Book("AS12", "java", "raj", "wrox", 500, LocalDate.now());
        Book bookSaved=entityManager.persist(book);
        Book bookFromDb=bookRepo.findById(bookSaved.getBookId()).orElseThrow(BookNotFoundException::new);
        assertThat(bookFromDb).isEqualTo(bookSaved);
    }
```



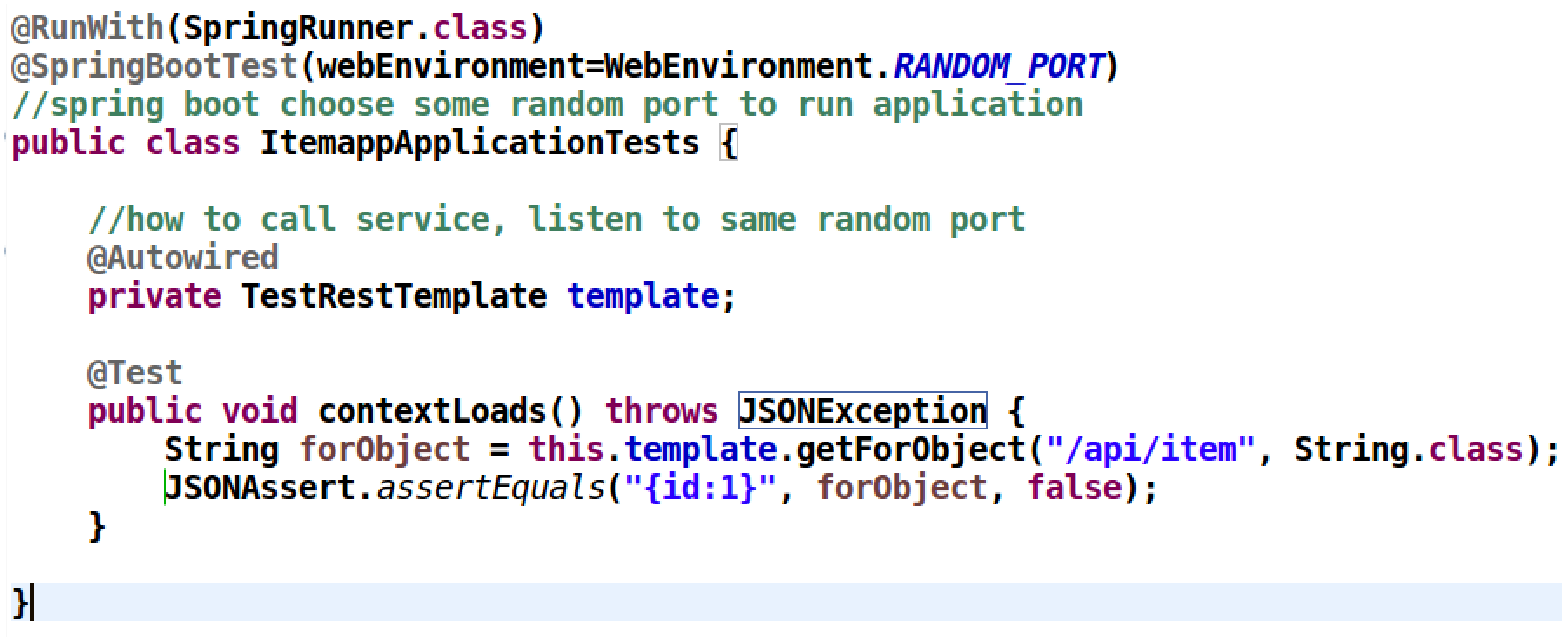
```java
@Test
public void testBookById(){
    Book book=new Book("AS12", "java", "raj", "wrox", 500, LocalDate.now());
    Book bookSaved=entityManager.persist(book);

    Book bookFromDb=bookRepo.findById(bookSaved.getBookId()).orElseThrow(BookNotFoundException::new);
    assertThat(bookFromDb).isEqualTo(bookSaved);
}


@Test
public void testAllBooks(){
    Book book1=new Book("AS12", "java", "raj", "wrox", 500, LocalDate.now());
    Book book2=new Book("AP12", "python", "raj", "abc", 500, LocalDate.now());

    Book bookSaved1=entityManager.persist(book1);
    Book bookSaved2=entityManager.persist(book1);

    List<Book>allBooks=(List<Book>) bookRepo.findAll();

    assertThat(allBooks.size()).isEqualTo(2);
}
```



```java
public void testDeleteBook(){
    Book book1=new Book("AS12", "java", "raj", "wrox", 500, LocalDate.now());

    Book bookSaved1=entityManager.persist(book1);
    //delete  book
    entityManager.remove(bookSaved1);
    List<Book>books=(List<Book>) bookRepo.findAll();
    assertThat(books.size()).isEqualTo(0);
}
public void testUpdateBook(){
    Book book1=new Book("AS12", "java", "raj", "wrox", 500, LocalDate.now());
    //save book to db
    Book bookSaved1=entityManager.persist(book1);
    Book getBookByDb=bookRepo.findById(book1.getBookId()).orElseThrow(BookNotFoundException::new);

    //update email address
    getBookByDb.setAuthorName("ravi");

    entityManager.merge(getBookByDb);

    entityManager.remove(bookSaved1);
    List<Book>books=(List<Book>) bookRepo.findAll();
    assertThat(books.size()).isEqualTo(1);
}
```

Integration testing configuration

- @SpringBootTest Launch complete application
- @SpringBootTest scan to all package till it find a class
- annotated with @SpringBootApplication and launch spring application context
- It launch in memory database and we can test resouce data.sql



```
@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment=WebEnvironment.RANDOM_PORT)
//spring boot choose some random port to run application
public class ItemappApplicationTests {

    //how to call service, listen to same random port
    @Autowired
    private TestRestTemplate template;

    @Test
    public void contextLoads() throws JSONException {
        String forObject = this.template.getForObject("/api/item", String.class);
        JSONAssert.assertEquals("{id:1}", forObject, false);
    }

}
```

Any questions?