Spring Professional Exam Tutorial v5.0

Question 05

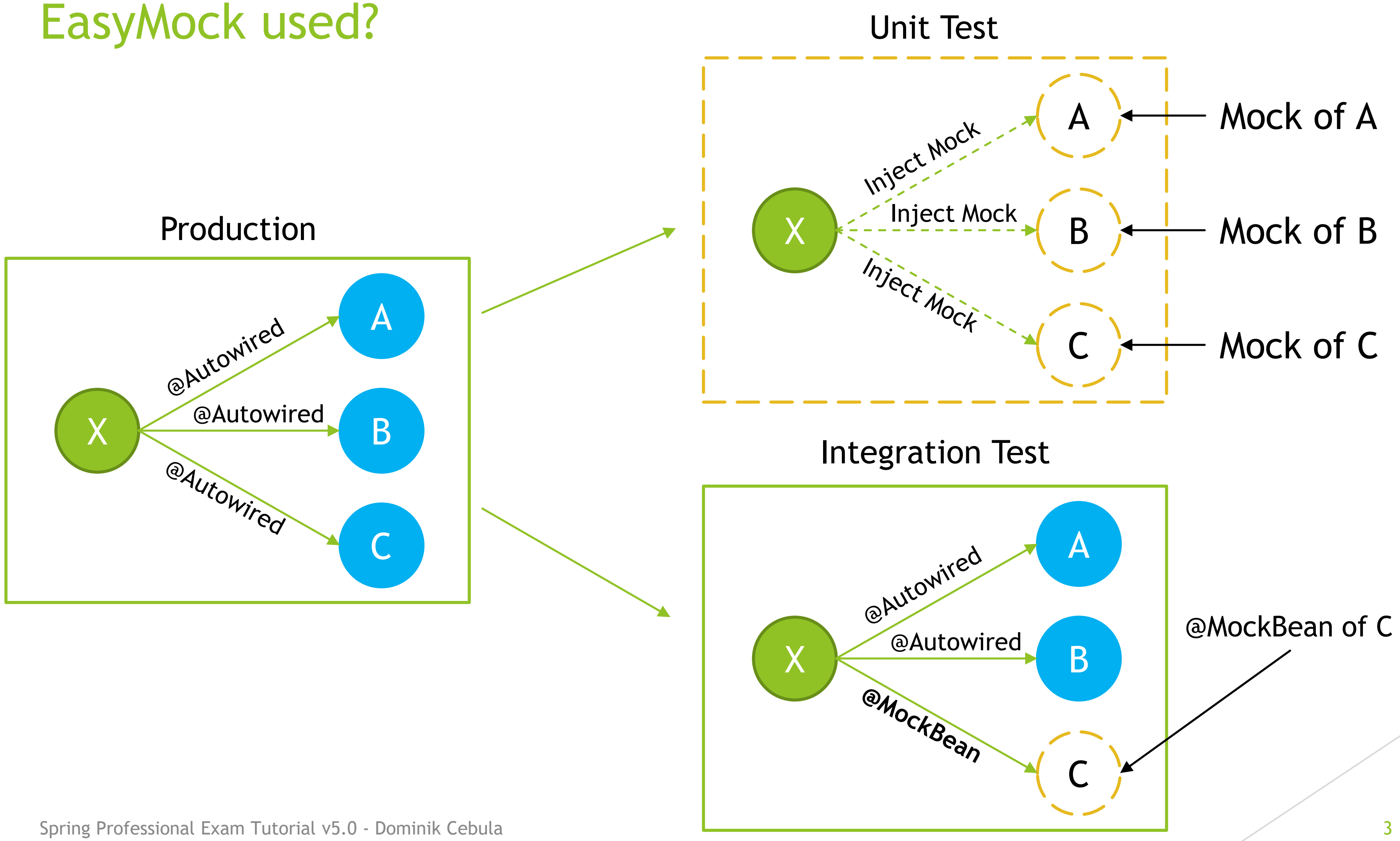
Question 05 - How are mock frameworks such as Mockito or EasyMock used?

Mock Frameworks like Mockito or EasyMock are used mainly during Unit Testing to mock collaborators of classes under test. Mockito or EasyMock can be also used during Integration Testing when goal is to check cooperation between different objects, while still mocking part of the system.

Mock created with Mockito or EasyMock is a dynamic object, which can "pretend" real object and return predefined results when invoking method on it. Additionally Mock allows you to verify if expected method were indeed called with expected arguments.

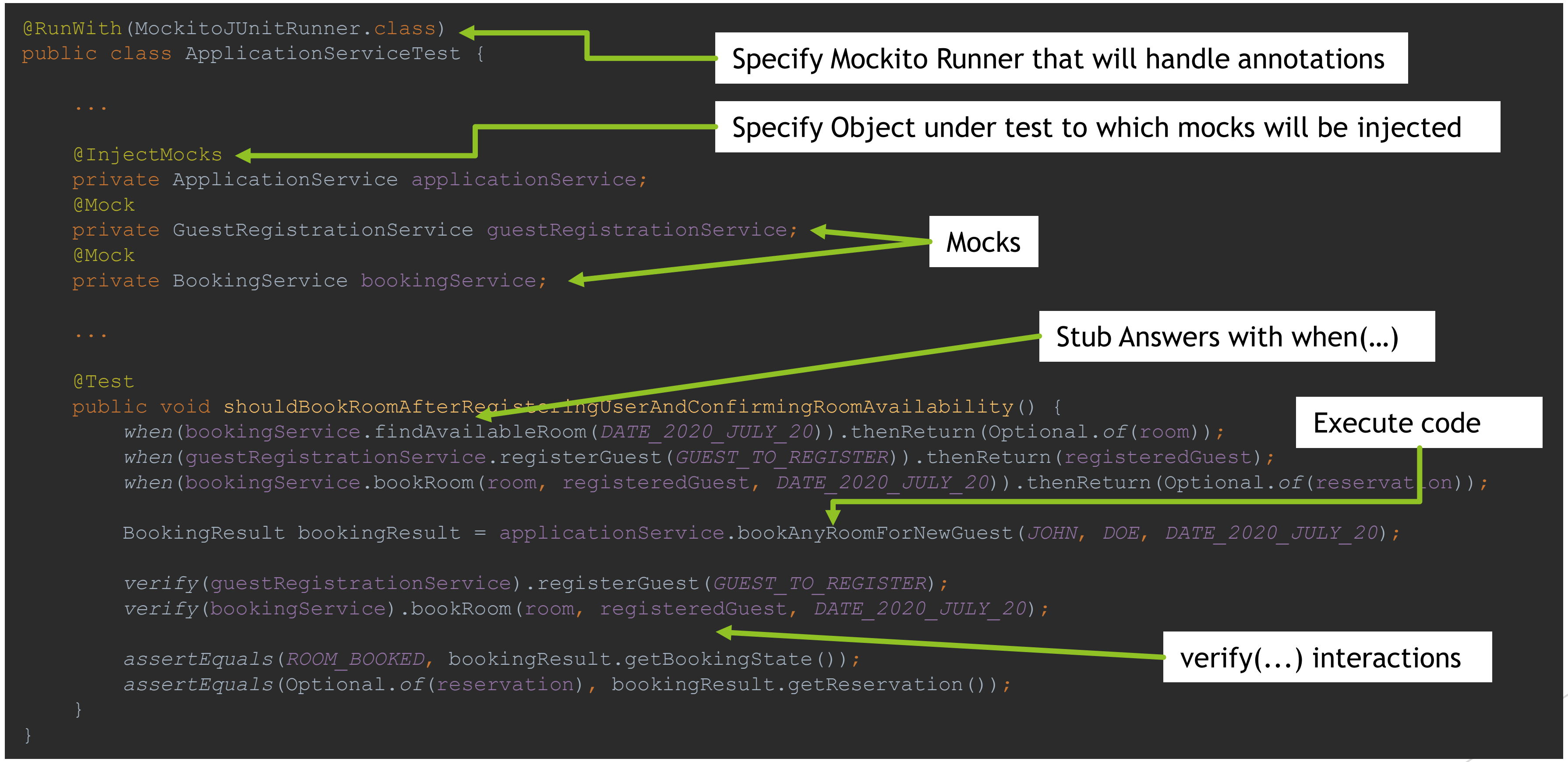
Above frameworks also allows you to inject mocks to classes under test in convenient way, with usage of annotations, with style similar to IoC/DI without having to run within container at all, which is one of the reason why unit tests are so fast and lightweight.

Question 05 - How are mock frameworks such as Mockito or EasyMock used?



Question 05 - How are mock frameworks such as Mockito or EasyMock used?

Mockito usage in Unit Test – for full documentation go to https://site.mockito.org/



Specify Mockito Runner that will handle annotations
Specify Object under test to which mocks will be injected
Mocks
Stub Answers with when(…)
Execute code
verify(…) interactions

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

    ...

    @InjectMocks
    private ApplicationService applicationService;
    @Mock
    private GuestRegistrationService guestRegistrationService;
    @Mock
    private BookingService bookingService;

    ...

    @Test
    public void shouldBookRoomAfterRegisteringUserAndConfirmingRoomAvailability() {
        when(bookingService.findAvailableRoom(DATE_2020_JULY_20)).thenReturn(Optional.of(room));
        when(guestRegistrationService.registerGuest(GUEST_TO_REGISTER)).thenReturn(registeredGuest);
        when(bookingService.bookRoom(room, registeredGuest, DATE_2020_JULY_20)).thenReturn(Optional.of(reservation));

        BookingResult bookingResult = applicationService.bookAnyRoomForNewGuest(JOHN, DOE, DATE_2020_JULY_20);

        verify(guestRegistrationService).registerGuest(GUEST_TO_REGISTER);
        verify(bookingService).bookRoom(room, registeredGuest, DATE_2020_JULY_20);

        assertEquals(ROOM_BOOKED, bookingResult.getBookingState());
        assertEquals(Optional.of(reservation), bookingResult.getReservation());
    }
}
```

Question 05 - How are mock frameworks such as Mockito or EasyMock used?

Mockito usage in Integration Test – prefer usage of `@MockBean` is using Spring Boot

```java
@RunWith(SpringRunner.class)
@ContextConfiguration(classes = ApplicationConfiguration.class)
public class ApplicationServiceIntegrationTest {

    @Autowired
    private ApplicationService applicationService;
    @Autowired
    private RoomRepository roomRepository;
    @Autowired
    private ReservationRepository reservationRepository;

    @Autowired
    private GuestSharableDataService guestSharableDataServiceMock;

    @Test
    public void shouldFetchGuestSharableData() {
        when(guestSharableDataServiceMock.getGuestSharableData()).thenReturn(SHARABLE_DATA);

        String guestSharableData = applicationService.getGuestSharableData();

        verify(guestSharableDataServiceMock).getGuestSharableData();

        assertEquals(SHARABLE_DATA, guestSharableData);
    }

    @Configuration
    public static class ApplicationServiceIntegrationTestMockConfiguration {

        @Bean
        public GuestSharableDataService guestSharableDataService() {
            return mock(GuestSharableDataService.class);
        }
    }
}
```

Use Spring IoC/DI for Integration Test

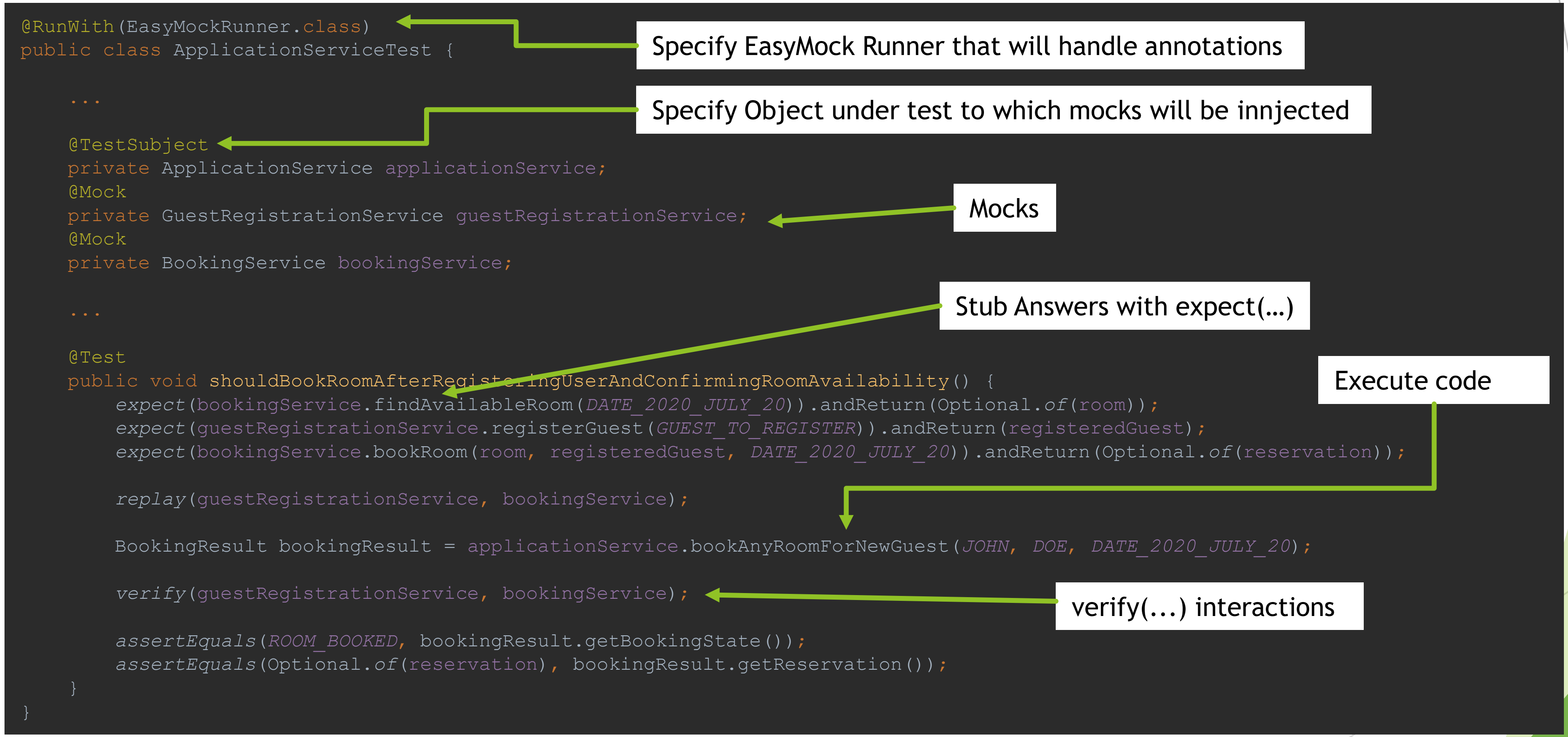
Inject all dependencies from context

Inject Mock from Configuration with Mock

Create Mock manually as `@Bean`

Question 05 - How are mock frameworks such as Mockito or EasyMock used?

EasyMock usage in Unit Test – for full documentation go to https://easymock.org/

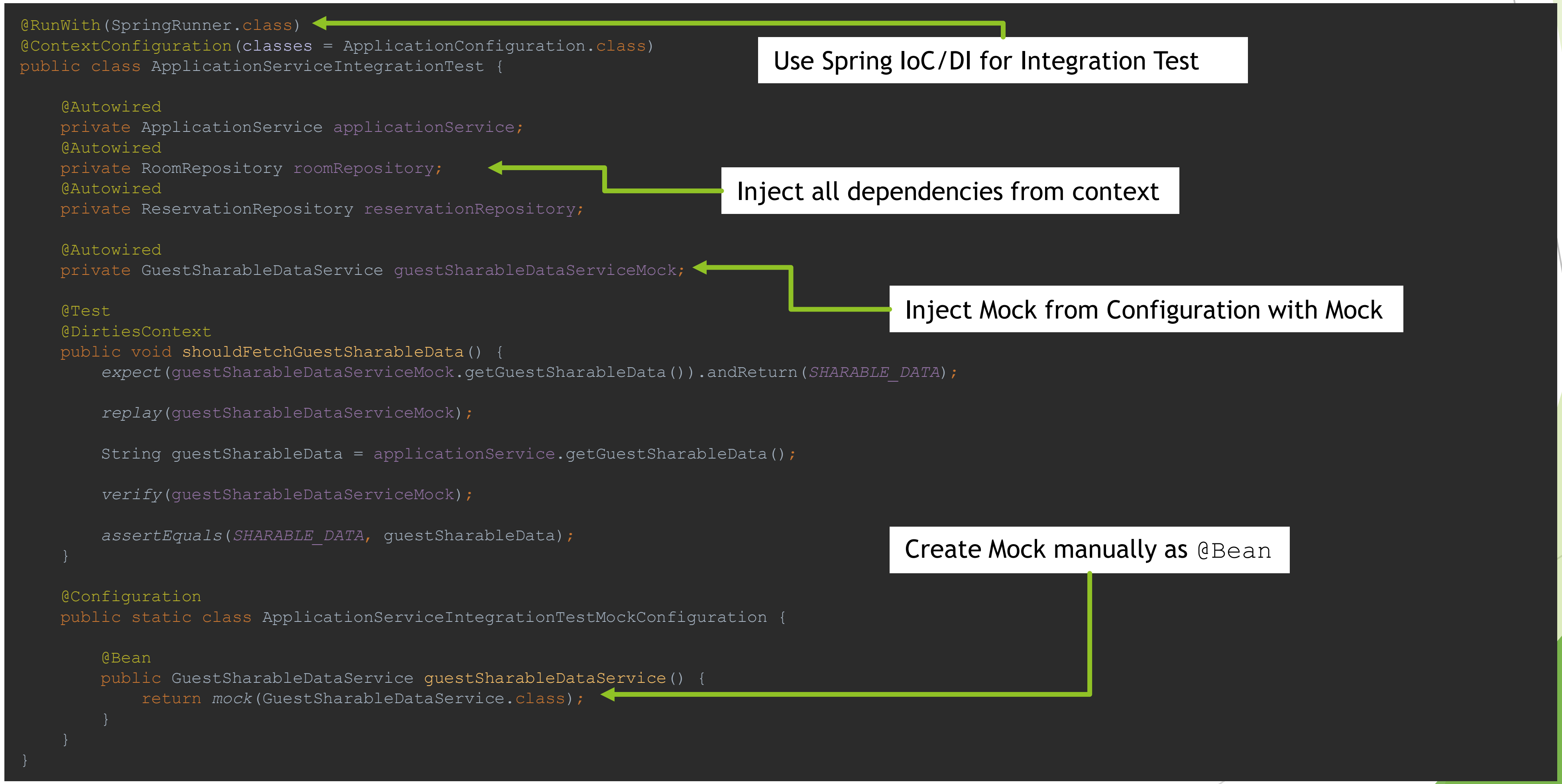


```
@RunWith(EasyMockRunner.class)
public class ApplicationServiceTest {

    ...

    @TestSubject
    private ApplicationService applicationService;
    @Mock
    private GuestRegistrationService guestRegistrationService;
    @Mock
    private BookingService bookingService;

    ...

    @Test
    public void shouldBookRoomAfterRegisteringUserAndConfirmingRoomAvailability() {
        expect(bookingService.findAvailableRoom(DATE_2020_JULY_20)).andReturn(Optional.of(room));
        expect(guestRegistrationService.registerGuest(GUEST_TO_REGISTER)).andReturn(registeredGuest);
        expect(bookingService.bookRoom(room, registeredGuest, DATE_2020_JULY_20)).andReturn(Optional.of(reservation));

        replay(guestRegistrationService, bookingService);

        BookingResult bookingResult = applicationService.bookAnyRoomForNewGuest(JOHN, DOE, DATE_2020_JULY_20);

        verify(guestRegistrationService, bookingService);

        assertEquals(ROOM_BOOKED, bookingResult.getBookingState());
        assertEquals(Optional.of(reservation), bookingResult.getReservation());
    }
}
```

Specify EasyMock Runner that will handle annotations
Specify Object under test to which mocks will be injnected
Mocks
Stub Answers with expect(…)
Execute code
verify(…) interactions

Question 05 - How are mock frameworks such as Mockito or EasyMock used?

EasyMock usage in Integration Test



```java
@RunWith(SpringRunner.class)
@ContextConfiguration(classes = ApplicationConfiguration.class)
public class ApplicationServiceIntegrationTest {

    @Autowired
    private ApplicationService applicationService;
    @Autowired
    private RoomRepository roomRepository;
    @Autowired
    private ReservationRepository reservationRepository;

    @Autowired
    private GuestSharableDataService guestSharableDataServiceMock;

    @Test
    @DirtiesContext
    public void shouldFetchGuestSharableData() {
        expect(guestSharableDataServiceMock.getGuestSharableData()).andReturn(SHARABLE_DATA);

        replay(guestSharableDataServiceMock);

        String guestSharableData = applicationService.getGuestSharableData();

        verify(guestSharableDataServiceMock);

        assertEquals(SHARABLE_DATA, guestSharableData);
    }

    @Configuration
    public static class ApplicationServiceIntegrationTestMockConfiguration {

        @Bean
        public GuestSharableDataService guestSharableDataService() {
            return mock(GuestSharableDataService.class);
        }
    }
}
```

Use Spring IoC/DI for Integration Test

Inject all dependencies from context

Inject Mock from Configuration with Mock

Create Mock manually as `@Bean`