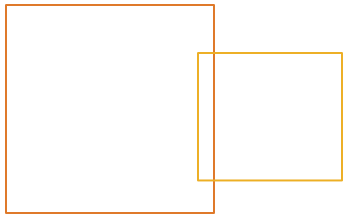
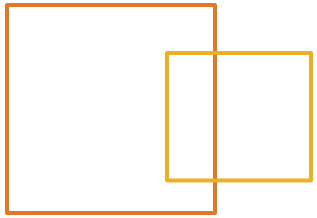
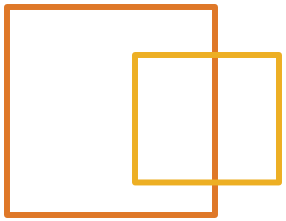
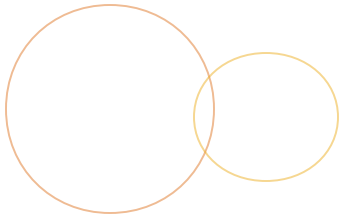
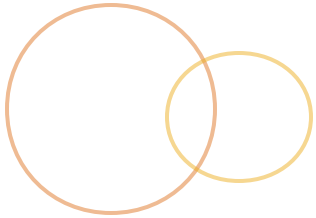
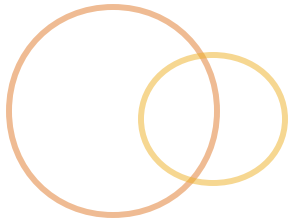





Fast Track to Java

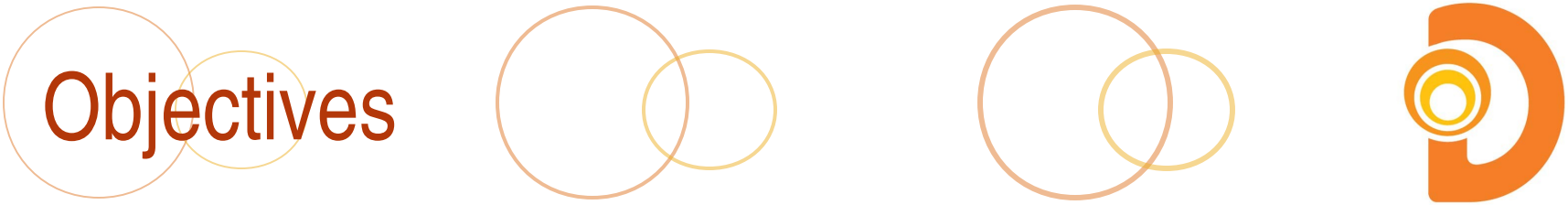
Customized for Starbucks

Delivered by DevelopIntelligence


Introduction To Spring

Objectives



At the end of this module you should:

- Have some basic knowledge about what Spring is
- Be able to use Spring to create objects
- Understand the idea behind Dependency Injection
- Be able to use Spring to wire up networks of objects
- Become familiar with Spring support for unit testing with JUnit.

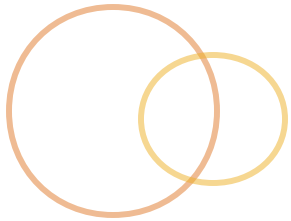
A Brief Spring Refresher

- At it's core, Spring is a factory for creating and wiring up objects.
- We most often deal with the factory using the **ApplicationContext**
- Spring beans can be configured using xml files and/or annotations
- Must know terms – **Dependency Injection, Inversion of Control**

A Brief Spring Refresher

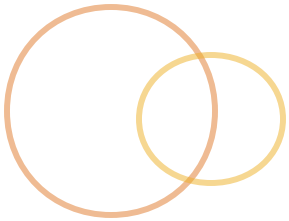
- What is a Container?
  - Repository and factory for creating and retrieving objects (beans)
  - Various implementations provided by the framework.
- What are Beans?
  - Objects that you ask the container to create for you
  - You can specify about what to create and how, using xml and/or annotation based metadata.
  - Beans can have names and aliases which you can use when retrieving them from the container.

A Brief Spring Refresher



- Instantiation and Initialization options
  - Default Constructors
  - Property based initialization
  - Constructors
  - Factory Methods
  - Inner Beans
  - Collections
  - Aliases
  - Parent/Child definitions
  - Lifetime – lazy init
  - Scope – singleton, prototype

Spring Talking Points



- Lifecycle callbacks
  - {init, destroy}-method
  - @PostConstruct, @PreDestroy
  - InitializingBean, DisposableBean
- Annotation based configuration
  - @Component, @Controller
  - @Resource
  - @Autowired
- All Java configuration
  - @Configuration

A Brief Spring Refresher

```
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:context="http://www.springframework.org/schema/context"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
            http://www.springframework.org/schema/context
            http://www.springframework.org/schema/context/spring-context-3.0.xsd">

      <context:component-scan base-package="ttl.larku" />

      <bean id="courseService" class="ttl.larku.service.CourseService">
            <property name="courseDAO" ref="courseDAO" />
      </bean>


      <bean id="courseDAO" class="ttl.larku.dao.inmemory.InMemoryCourseDAO" />
      <bean id="studentDAO" class="ttl.larku.dao.inmemory.InMemoryStudentDAO" />
</beans>
```

Needed for annotations

Dependency Injection

```
@Component
public class StudentService {

      @Resource(name="studentDAO")
      private BaseDAO<Student> studentDAO;
```

Bean Declaration – bean name is “studentService”

Dependency Injection

Tricks with @Autowired

```
@Component
public class ClassService {

    @Autowired
    private CourseService courseService;

    @Autowired(required="false")
    private CourseService courseService;

    @Autowired
    private CourseService[] courseServices;

    @Autowired
    private ApplicationContext context;

    @Autowired
    @Qualifier(“studentDAO”)
    private BaseDAO<Student> studentDao;
```

Better be exactly 1 bean of **type** CourseService.

No Exception if 0 beans of **type** CourseService.

All beans of type CourseService.

Useful for injecting “well known” objects

Qualification

Using an ApplicationContext

```
public static void main(String [] args) {
    ClassPathXmlApplicationContext appContext =
        new ClassPathXmlApplicationContext("larkUContext.xml");

    StudentService studentService =
        appContext.getBean("studentService", StudentService.class);

    Student student1 = studentService.createStudent("Sammy");

    Student student2 = studentService.getStudent(student1.getId());

    System.out.println("student 2 is " + student2);
}
```

JUnit and Spring

- One issue with using JUnit and Spring is how does the ApplicationContext get created.
- You can do it yourself, or you can use some Spring magic to have it created for you:
  - @RunWith(SpringJUnit4ClassRunner.class)
  - @ContextConfiguration("classpath:larkUContext.xml")
- You can then also inject references to required beans into your test:
  - @Resource(name="courseService")

    private CourseService courseService;

JUnit and Spring

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:larkUContext.xml")
public class ClassServiceTest {

    @Resource(name="classService")
    private ClassService classService;

    @Resource(name="courseService")
    private CourseService courseService;

    @Resource(name="studentService")
    private StudentService studentService;

    @Autowired
    private ApplicationContext context;
```

If you need it. To create instances of prototypes, for example

@DirtiesContext

- By default, SpringJUnit4RunnerClass will create the application context **once** at the beginning of the test run.
- The same context will be used for all the tests in the class.
- Done for performance reasons.
- This is only an issue if tests change the state of beans, but subsequent tests depend on the beans being in some initial state.
- In that case, **@DirtiesContext** is your friend.

JUnit and Spring

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:larkUContext.xml")
public class CourseDAOTest {
     @Resource
     Private InMemoryCourseDAO courseDAO;

     @Test
     public void testCreate() {
          int newId = dao.create(course1);

          Course resultCourse = dao.get(newId);

          assertEquals(newId, resultCourse.getId());
     }

     @Test
     public void testDelete() {
          int id1 = dao.create(course1);
          int id2 = dao.create(course2);

          assertEquals(2, dao.getAll().size());

          dao.delete(id2);

          assertEquals(1, dao.getAll().size());
          assertEquals(title1, dao.get(id1).getTitle());
     }
}
```

This will cause an assertion error, since the DAO already has the course added in the previous test

JUnit and Spring

```
@DirtiesContext
@Test
public void testCreate() {
     int newId = dao.create(course1);

     Course resultCourse = dao.get(newId);

     assertEquals(newId, resultCourse.getId());
}

@Test
public void testDelete() {
     int id1 = dao.create(course1);
     int id2 = dao.create(course2);

     assertEquals(2, dao.getAll().size());

     dao.delete(id2);

     assertEquals(1, dao.getAll().size());
     assertEquals(title1, dao.get(id1).getTitle());
}
```

This will cause the ApplicationContext to be destroyed and re-created after this test.

JUnit and Spring

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:larkUContext.xml")
@DirtiesContext(classMode=ClassMode.AFTER_EACH_TEST_METHOD)
public class CourseDAOTest {

    …
}
```

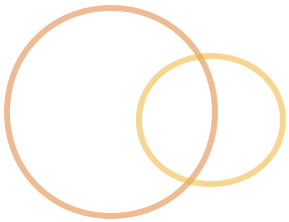
This will cause the ApplicationContext to be destroyed and re-created after **every** test in this class.

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:larkUContext.xml")
@DirtiesContext()
public class CourseDAOTest {

    …
}
```

This will cause the ApplicationContext to be destroyed and re-created after **all** tests in this class have run.

Lab 13 – Basic Spring



- In this Lab, you will finish the wiring up of a Spring application. You will use both XML and annotation based configuration. The end goal is to make a suite of Junit tests run successfully.
- Instructions start on the next page

Lab 13 – Basic Spring

1. Create a new Eclipse workspace at some location of your choosing, e.g. C:\SpringMVCEssenstials
2. Import the Eclipse project **Basic-Spring** into your workspace.
3. You may need to set up or configure some Libraries. If you are unsure about how to do this, ask your Instructor.
4. Examine the code. Source code is in **src/main/java**, configuration resources are in **src/main/resources**, and Junit tests are in **src/test/java**.
5. Run any of the **service** tests in **src/test/java** (right click and choose **Run As → Junit Test**)
6. You will find see a whole bunch of errors in the Junit console.
7. Your job is to fix the errors for all the service tests.

Lab 13 – Basic Spring

8. You will **NOT** need to make any changes to the code itself. All your changes will be to Spring configuration elements.
9. You will need to make changes in the Spring config file **src/main/resources/larkUContext.xml.**
10. You will also need to make annotation based changes to some of the Junit test cases.
11. There are some **TODO** comments in various source files that provide hints about what needs to be done.
12. You will probably need to iterate through a sequence of changes, fixing errors one at a time. In some cases, one fix will cause a bunch of errors to go away.
13. Your goal is to see that lovely green bar indicating a successful Junit test run.
14. A good strategy would be to proceed a test at a time.