Java 4 WEB

Lesson 15 – Spring Framework

Lesson goals

- Inversion of Control
- Dependency Injection
- Spring Core
- XML config vs Annotation config vs Java Config

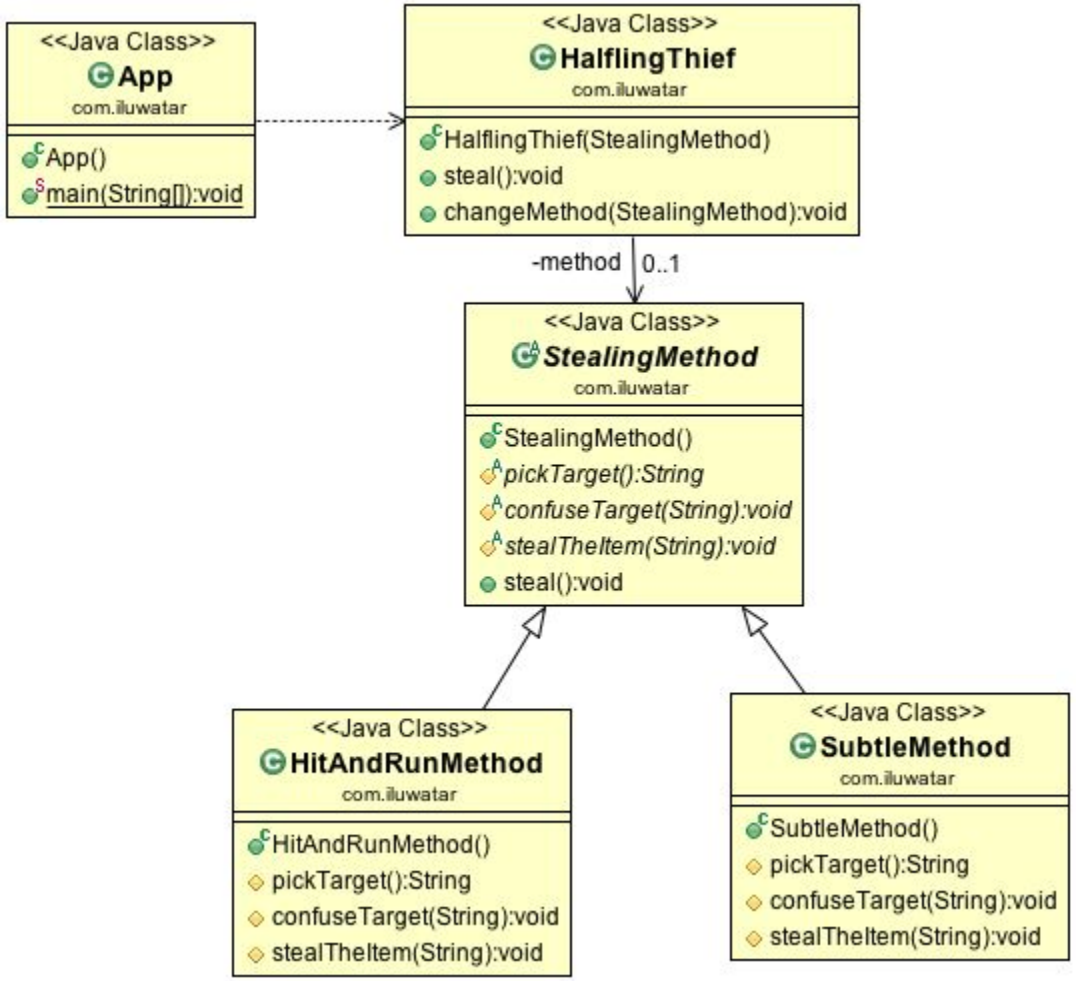
Inversion of Control

In software engineering, inversion of control (IoC) is a design principle in which custom-written portions of a computer program receive the flow of control from a generic framework. A software architecture with this design inverts control as compared to traditional procedural programming: in traditional programming, the custom code that expresses the purpose of the program calls into reusable libraries to take care of generic tasks, but with inversion of control, it is the framework that calls into the custom, or task-specific, code.

Inversion of Control

- Principle helping to write loose coupled code
- In object-oriented programming, there are several basic techniques to implement inversion of control. These are:
  - Using a **service locator pattern**
  - Using **dependency injection**, for example
    - Constructor injection
    - Parameter injection
    - Setter injection
    - Interface injection
  - Using **template method design pattern**
  - Using **strategy design pattern**

Inversion of Control. Template Method

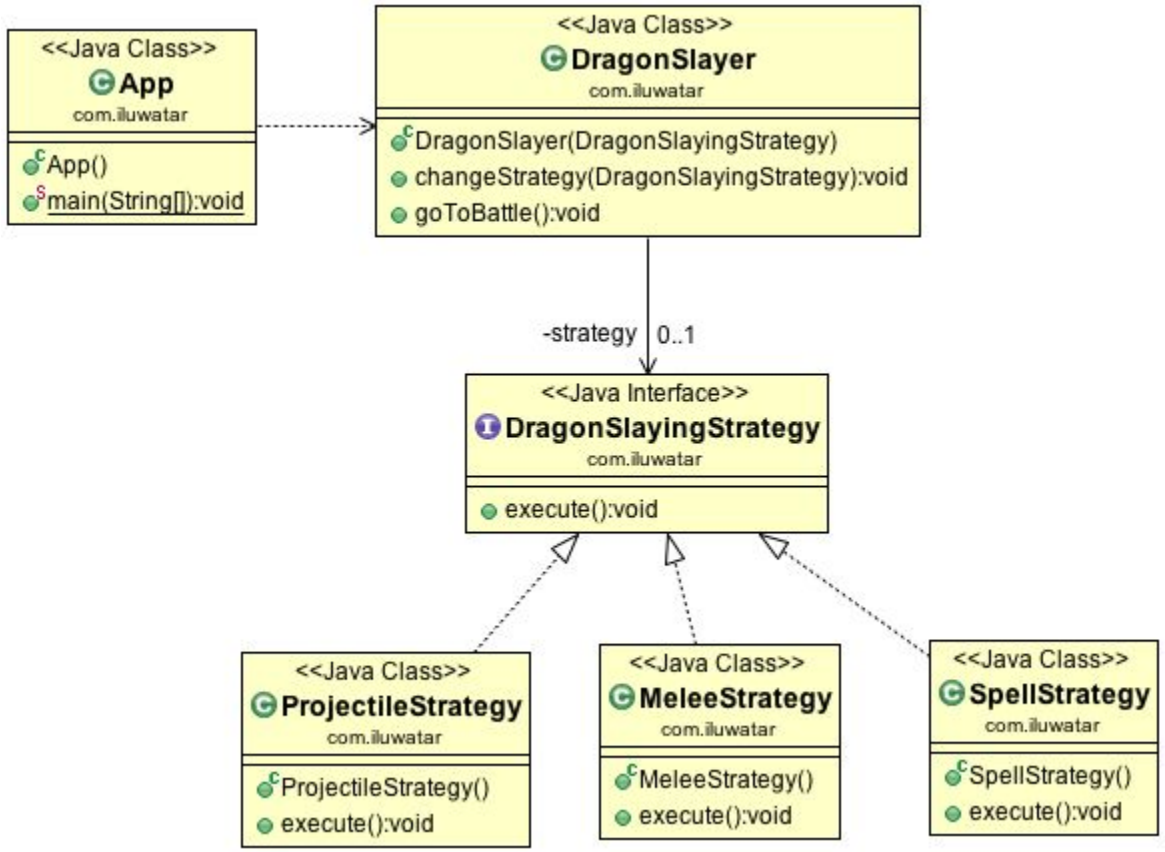


Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.

Inversion of Control. Template Method

Code example

Inversion of Control. Strategy

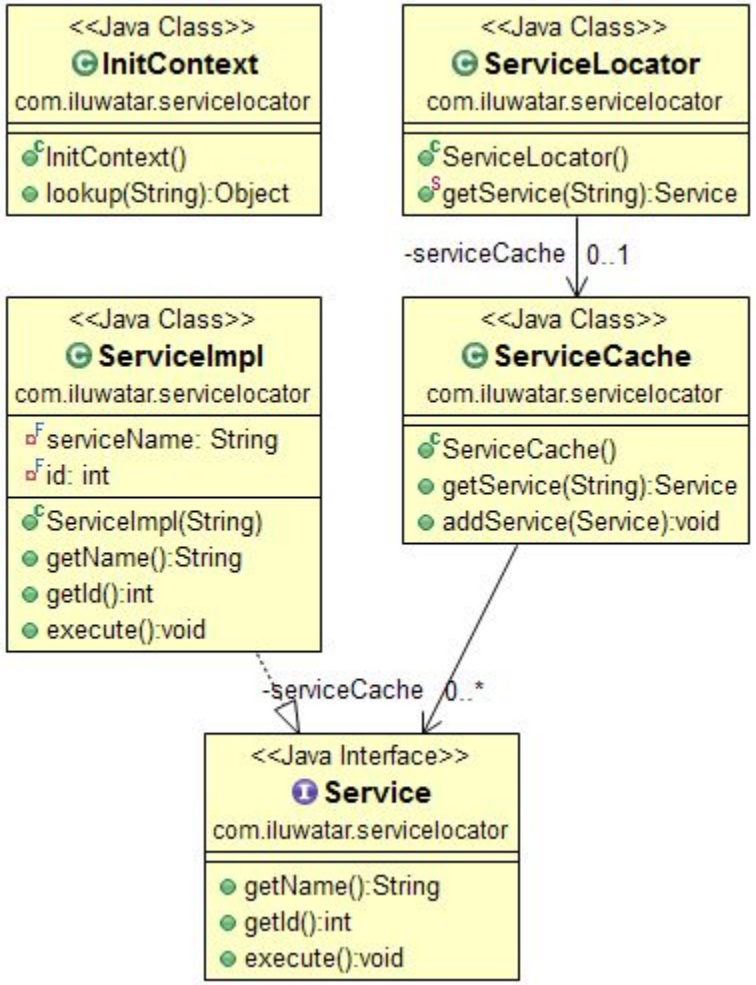


Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.

Inversion of Control. Strategy

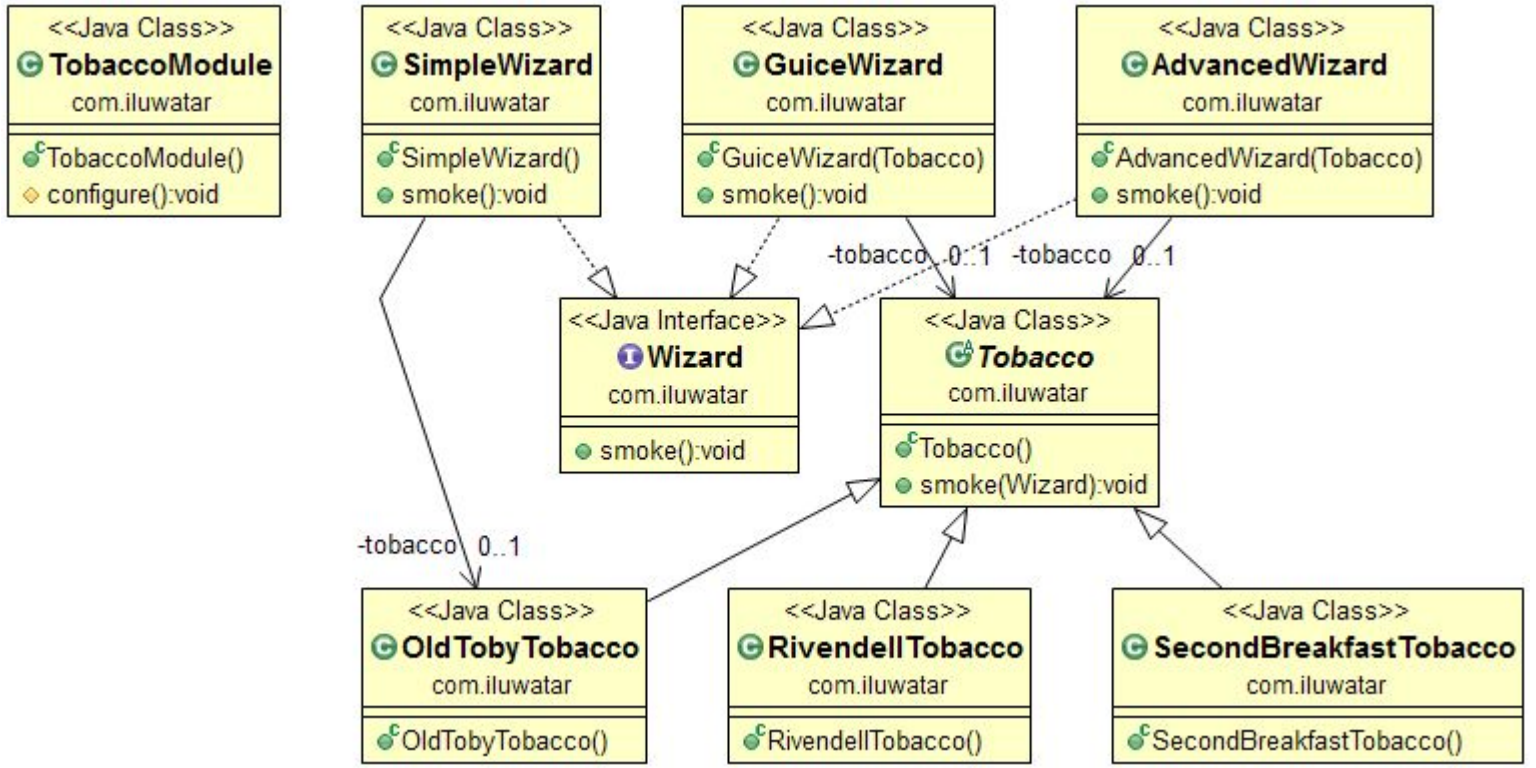
Code example

Inversion of Control. Service Locator



The service locator pattern is a design pattern used in software development to encapsulate the processes involved in obtaining a service with a strong abstraction layer. This pattern uses a central registry known as the "service locator", which on request returns the information necessary to perform a certain task. Note that many consider service locator to actually be an anti-pattern.

Inversion of Control. Dependency Injection



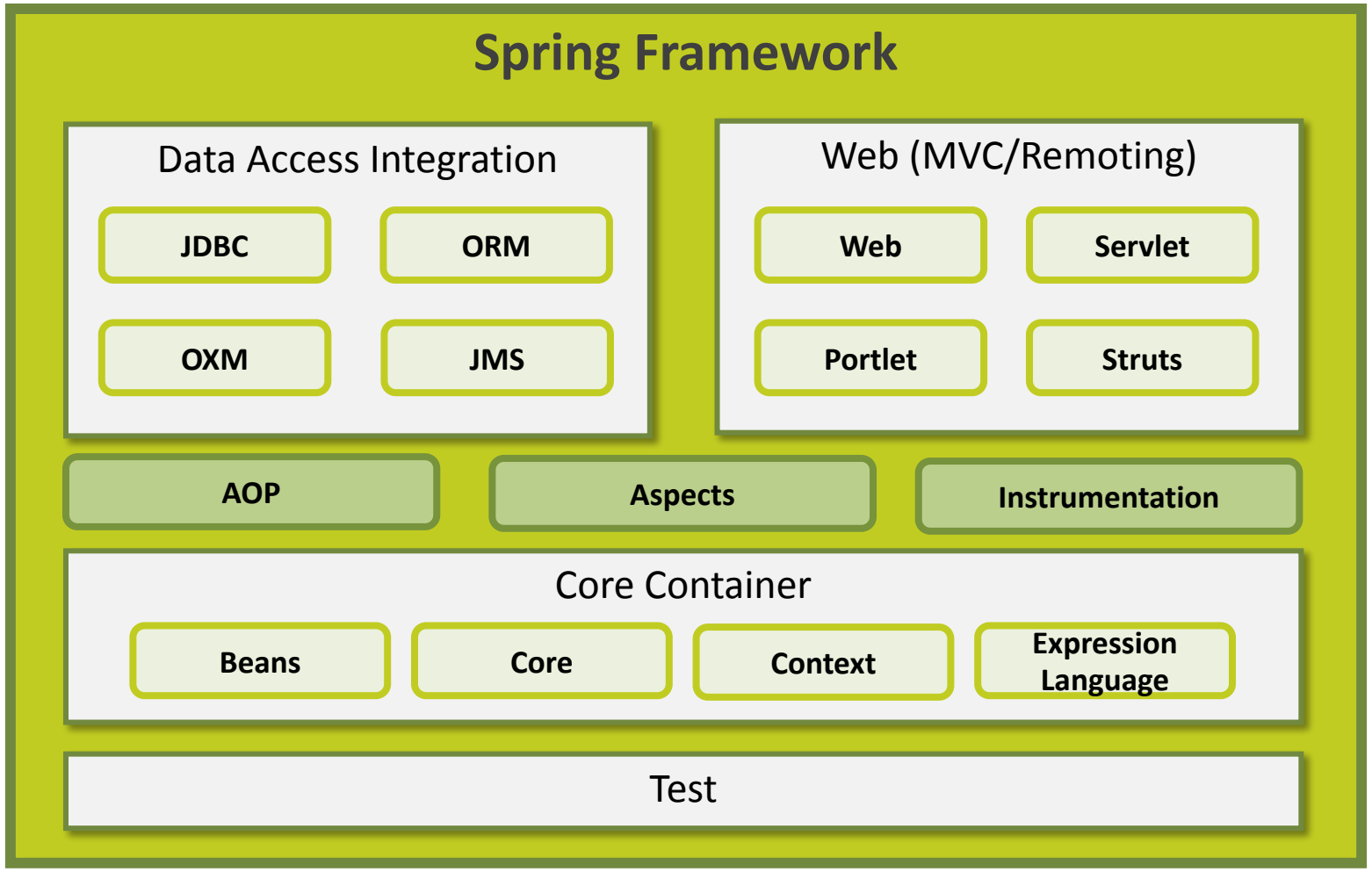
- Dependency Injection is a software design pattern in which one or more dependencies (or services) are injected, or passed by reference, into a dependent object (or client) and are made part of the client's state. The pattern separates the creation of a client's dependencies from its own behavior, which allows program designs to be loosely coupled and to follow the inversion of control and single responsibility principles.

Code example

Spring Core

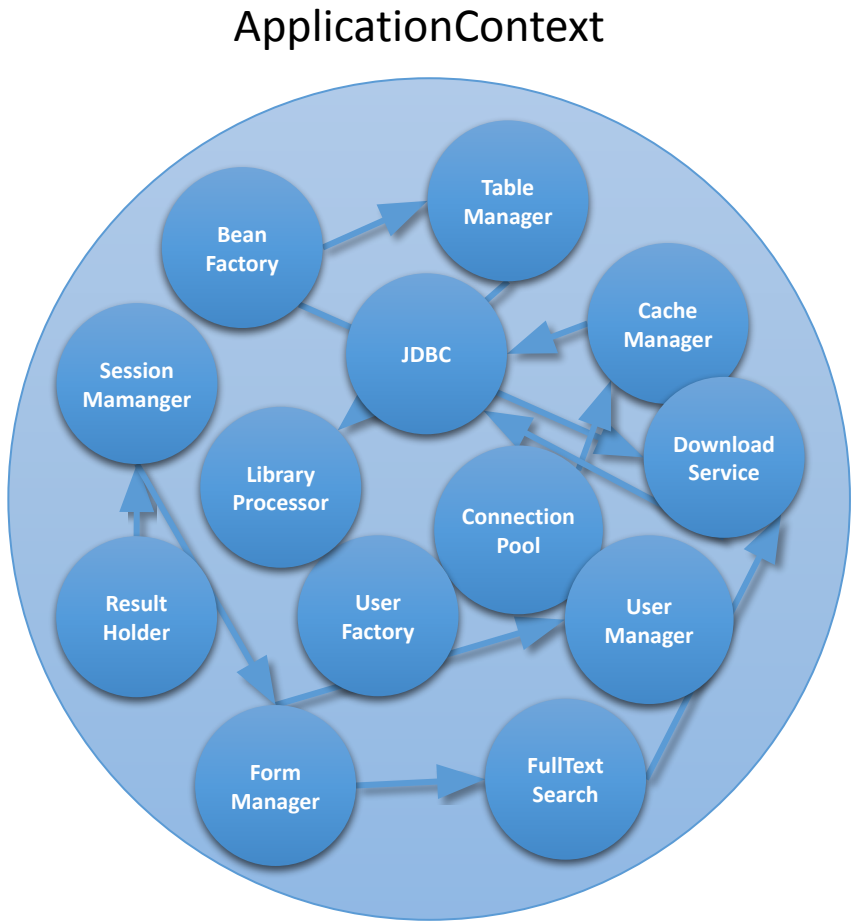
The Spring Framework is an application framework and inversion of control container for the Java platform. The framework's core features can be used by any Java application, but there are extensions for building web applications on top of the Java EE (Enterprise Edition) platform. Although the framework does not impose any specific programming model, it has become popular in the Java community as an addition to, or even replacement for the Enterprise JavaBeans (EJB) model. The Spring Framework is open source.

Spring Core. Architecture



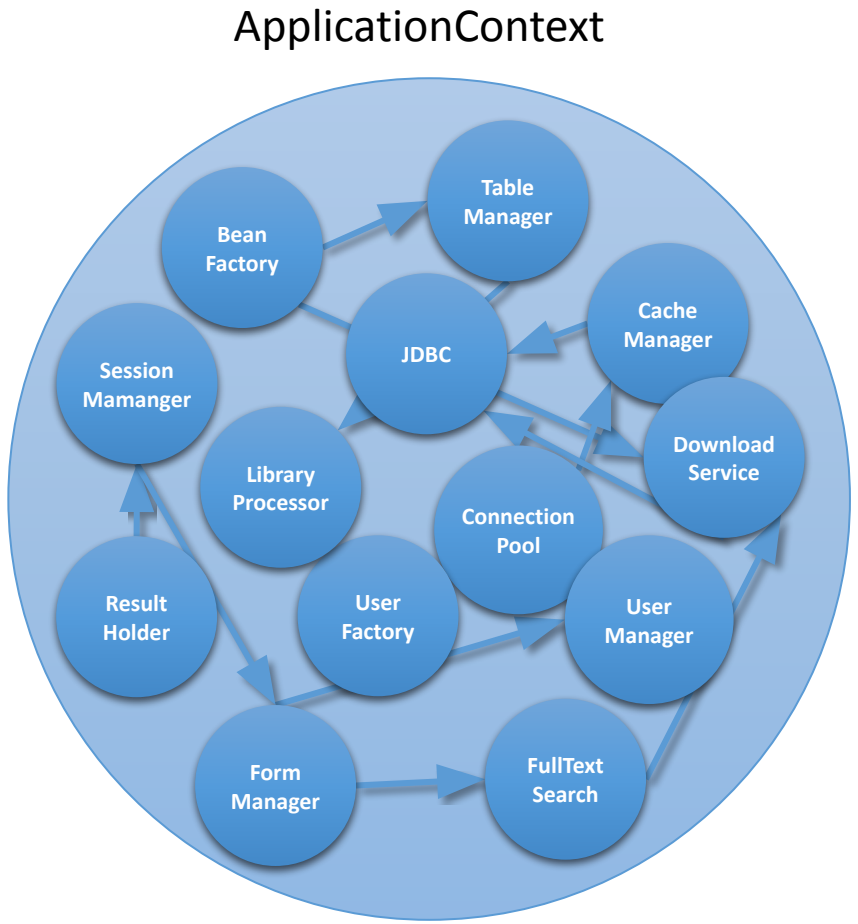
Spring Core. ApplicationContext

- Container of all beans and their dependencies.
- The ApplicationContext is the central interface within a Spring application for providing configuration information to the application. It is read-only at run time, but can be reloaded if necessary and supported by the application. A number of classes implement the ApplicationContext interface, allowing for a variety of configuration options and types of applications.



Spring Core. ApplicationContext

- The ApplicationContext provides:
  - Bean factory methods for accessing application components.
  - The ability to load file resources in a generic fashion.
  - The ability to publish events to registered listeners.
  - The ability to resolve messages to support internationalization.
  - Inheritance from a parent context.



Spring Core. Bean Scopes

Scope	Description
singleton	Scopes a single bean definition to a single object instance per Spring IoC container.
prototype	Scopes a single bean definition to any number of object instances. New object will be created every time on getting from ApplicationContext.
request	Scopes a single bean definition to the lifecycle of a single HTTP request; that is each and every HTTP request will have its own instance of a bean created off the back of a single bean definition. Only valid in the context of a web-aware Spring ApplicationContext.
session	Scopes a single bean definition to the lifecycle of a HTTP Session. Only valid in the context of a web-aware Spring ApplicationContext.
global session	Scopes a single bean definition to the lifecycle of a global HTTP Session. Typically only valid when used in a portlet context. Only valid in the context of a web-aware Spring ApplicationContext.

Spring Core. Bean Definition

property	Description
class	This attribute is mandatory and specifies the bean class to be used to create the bean.
name	This attribute specifies the bean identifier uniquely. In XML-based configuration metadata, you use the id and/or name attributes to specify the bean identifier(s).
scope	This attribute specifies the scope of the objects created from a particular bean definition and it will be discussed in bean scopes chapter.
initialization method	A callback to be called just after all necessary properties on the bean have been set by the container. It will be discussed in bean life cycle chapter.
destruction method	A callback to be used when the container containing the bean is destroyed. It will be discussed in bean life cycle chapter.

Spring Xml Config

context.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">
    <bean id="worker" class="com.geekhub.examples.xmlconfig.Worker">
        <constructor-arg ref="primeNumberPrinter"/>
    </bean>

    <bean id="loggerService" class="com.geekhub.examples.xmlconfig.logger.LoggerServiceImpl" init-method="init" destroy-method="destroy"/>

    <bean id="primeNumberPrinter" class="com.geekhub.examples.xmlconfig.printer.PrimeNumberPrinterImpl">
        <property name="logger" ref="loggerService"/>
    </bean>
</beans>
```

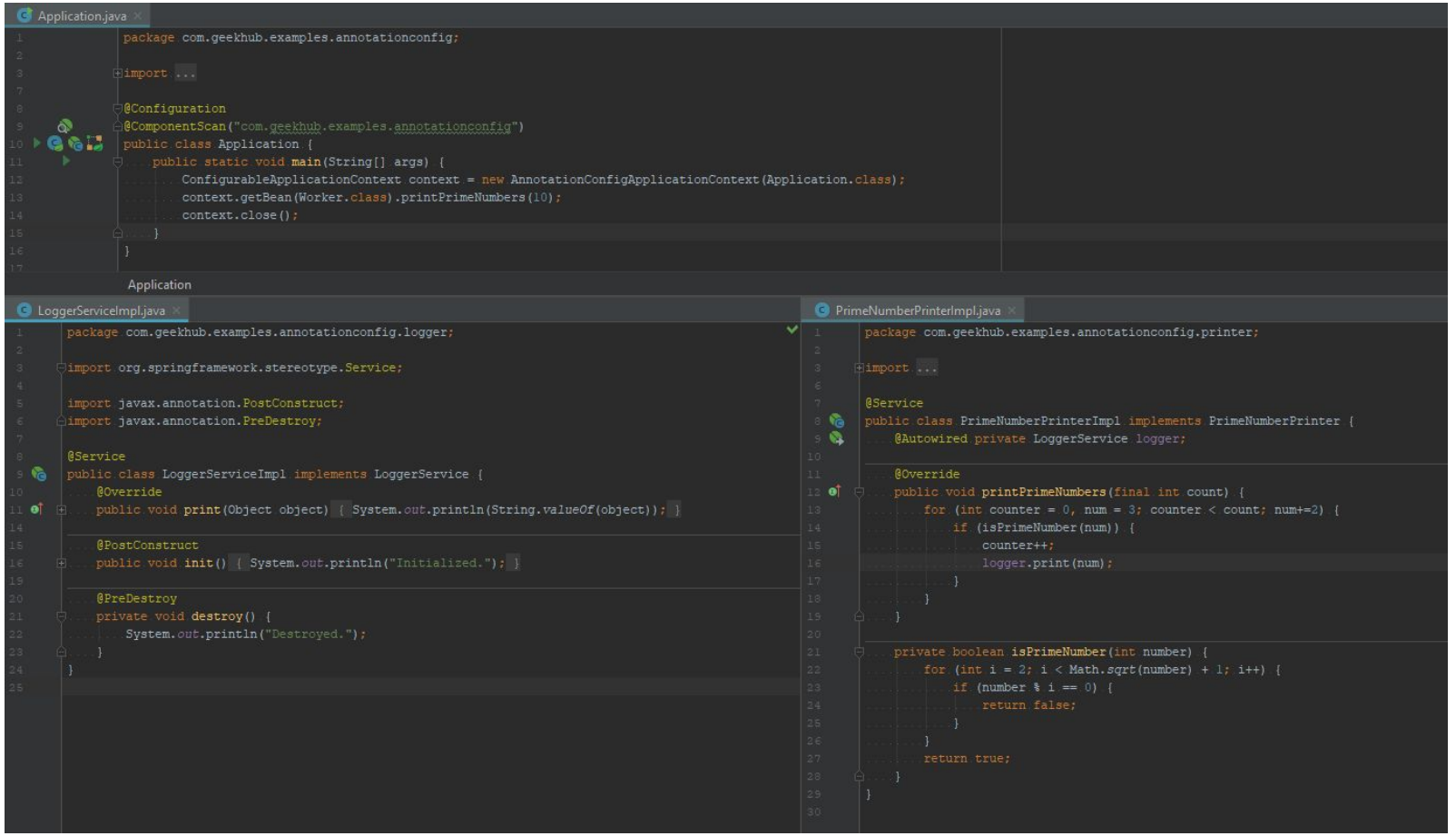
Application.java

```java
package com.geekhub.examples.xmlconfig;

import ...

public class Application {
    public static void main(String[] args) {
        ConfigurableApplicationContext context = new ClassPathXmlApplicationContext( configLocation: "com/geekhub/examples/xmlconfig/context.xml");
        context.getBean(Worker.class).printPrimeNumbers(10);
        context.close();
    }
}
```

Spring Annotation Config



Application.java

```java
package com.geekhub.examples.annotationconfig;

import ...

@Configuration
@ComponentScan("com.geekhub.examples.annotationconfig")
public class Application {
    public static void main(String[] args) {
        ConfigurableApplicationContext context = new AnnotationConfigApplicationContext(Application.class);
        context.getBean(Worker.class).printPrimeNumbers(10);
        context.close();
    }
}
```

Application

LoggerServiceImpl.java

```java
package com.geekhub.examples.annotationconfig.logger;

import org.springframework.stereotype.Service;

import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;

@Service
public class LoggerServiceImpl implements LoggerService {
    @Override
    public void print(Object object) { System.out.println(String.valueOf(object)); }

    @PostConstruct
    public void init() { System.out.println("Initialized."); }

    @PreDestroy
    private void destroy() {
        System.out.println("Destroyed.");
    }
}
```

PrimeNumberPrinterImpl.java

```java
package com.geekhub.examples.annotationconfig.printer;

import ...

@Service
public class PrimeNumberPrinterImpl implements PrimeNumberPrinter {
    @Autowired private LoggerService logger;

    @Override
    public void printPrimeNumbers(final int count) {
        for (int counter = 0, num = 3; counter < count; num+=2) {
            if (isPrimeNumber(num)) {
                counter++;
                logger.print(num);
            }
        }
    }

    private boolean isPrimeNumber(int number) {
        for (int i = 2; i < Math.sqrt(number) + 1; i++) {
            if (number % i == 0) {
                return false;
            }
        }
        return true;
    }
}
```

Spring Java Config

Application.java

```java
package com.geekhub.examples.javaconfig;

import ...

@Configuration
public class Application {
    @Bean
    public Worker worker(PrimeNumberPrinter primeNumberPrinter) { return new Worker(primeNumberPrinter); }

    @Bean(initMethod = "init", destroyMethod = "destroy")
    public LoggerService loggerService() { return new LoggerServiceImpl(); }

    @Bean
    public PrimeNumberPrinter primeNumberPrinter(LoggerService loggerService) {
        return new PrimeNumberPrinterImpl(loggerService);
    }

    public static void main(String[] args) {
        ConfigurableApplicationContext context = new AnnotationConfigApplicationContext(Application.class);
        context.getBean(Worker.class).printPrimeNumbers(10);
        context.close();
    }
}
```

Spring IoC Annotations

```
@Component
@Scope("session")
public class JDBC {
   //Spring bean component, does not require to be declared in app context
}

public class UserManager {
    //be sure jdbc will be initialized before you start using it
    @Autowired private JDBC jdbc;
}

<beans>
    <context:annotation-config/>
    <context:component-scan base-package="com.beans"/>
</beans>
```

Spring Life Cycle Annotations

```
public class UserManager {
    @PostConstruct
    public void init() {
        //do some initialization work
    }

    @PreDestroy
    public void destroy() {
        //release all resources
    }
}
```

Literature

- Java Design Patterns - Service Locator
- Java Design Patterns - Dependency Injection
- Java Design Patterns - Template Method
- Java Design Patterns - Strategy
- Spring Tutorial
- Spring Framework Docs
- Inversion of Control and Dependency Injection in Spring

Homework 1

Implement user management API protected with authentication by login and password.

Requirements