Pearl Arora, 19csu208

Assignment-1

Q1. What is Spring Framework? How does it evolve in market? Explain the different features and architecture of spring framework.

Answer: Spring is the most popular application development framework for enterprise Java. Millions of developers around the world use Spring Framework to create high performing, easily testable, and reusable code.

The core features of the Spring Framework can be used in developing any Java application, but there are extensions for building web applications on top of the Java EE platform. Spring framework targets to make J2EE development easier to use and promotes good programming practices by enabling a POJO-based programming model.

FEATURES AND ARCHITECTURE OF SPRING FRAMEWORK

- Spring is a powerful framework, which address many common problems in Java EE. It includes support for managing business objects and exposing their services to presentation tier component.
- It facilitates good programming practice such as programming using interfaces instead of classes. Spring enables developers to develop enterprise applications using POJO and POJI model programming.
- It supports both XML- and annotation-based configuration.
- It is modular, allowing you to use only those parts that you need. It allows us to just choose any part of it in isolation.

Q2. What is Spring IOC Container? List some of the benefits of IoC

Answer: The Spring container is at the core of the Spring Framework. The container will create the objects, wire them together, configure them, and manage their complete life cycle from creation till destruction. The Spring container uses DI to manage the components that make up an application.

- Better clarity of the code – it is clear at first glance what dependencies the class (object) require for its functionality.

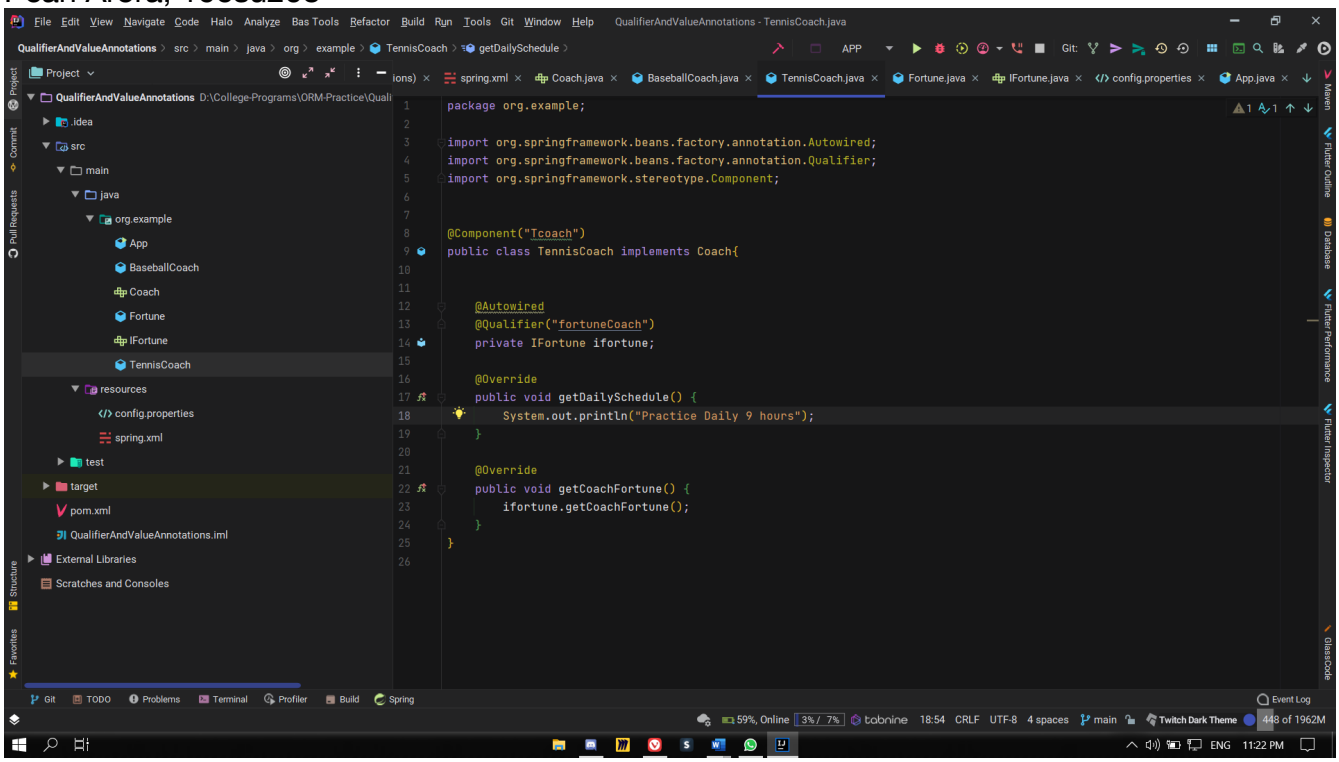
More simple structure of the application – application of the principles of IoC leads to the creation of reusable components.

Q3. What is dependency conflict? What are the strategies to remove the dependency conflicts when there is more than one bean for autowiring?

Answer: Conflicts will exist when two versions of a transitive dependency are required. The ClassNotFoundException you describe results from the app (or a dependency) attempting to use a class not available in the version of the conflicted dependency that actually gets used.

We can make dependency injection disembogues by using @Qualifier and autowiring by name as implemented below.

Pearl Arora, 19csu208



Q4. What if a class has two constructors with same number of input parameters with different types of arguments? In this case spring gets into ambiguous situation. How does spring resolves this ambiguity

Answer : To resolve this, we must specify constructor input of parameter type.

```
public class Example {


    private String name;
    private int number;
    private String name;

    public Article(String name, int number){
        this.name = name;
        this.number = number;
    }

    public Article(String name, String number){
        this.name = name;
        this.number = number;
    }
}
```

```
 <bean id="articleBean" class="com.bean.Article">

        <constructor-arg type="java.lang.String">
            <value>Pearl</value>
        </constructor-arg>

        <constructor-arg type="int">
```

```
        <value>40</value>
      </constructor-arg>
   </bean>
```

Q5. Are Singleton Beans Thread-Safe?
Answer: Singleton Beans is thread safe or not depends on how the class whose scope is singleton is written. Each calling thread will have its own execution and does not interfere with another thread's execution unless there is some code in the singleton scoped class which is shared by all calling threads.

Q6. What is circular dependency injection? Write an annotation based code to handle the circular dependency in the project.

Answer:

```
package org.example;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component("A")
public class A {

   public A()
   {
      System.out.println("In A");
   }
   @Autowired
   private B b;

}
```

```
package org.example;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Lazy;
import org.springframework.stereotype.Component;

@Component("B")
public class B {

   public B()
   {
      System.out.println("In B");
   }
   @Lazy
   @Autowired
   private A a;
}
```

```
package org.example;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

/**
 * Hello world!
 *
 */
```

```
public class App
{
    public static void main( String[] args )
    {
        ApplicationContext context = new ClassPathXmlApplicationContext("spring.xml");
        A a = context.getBean("A",A.class);
    }
}
```

Q7. Implement bean lifecycle using annotation based approach.

```
package org.example;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.annotation.Lazy;
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Component;

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


@Scope("singleton")
@Component("Bcoach")
public class BaseballCoach implements Coach{

    @PostConstruct
    public void startup()
    {
        System.out.println("Post Construct Method Called");
    }

    @PreDestroy
    public void end()
    {
        System.out.println("Pre Destroy Method");
    }


    @Autowired
    @Qualifier("fortuneCoach")
    private IFortune ifortune;

    @Override
    public void getDailySchedule() {
        System.out.println("Practice Daily 7 hours");
    }

    @Override
    public void getCoachFortune() {
        ifortune.getCoachFortune();
    }
}
```

Q8. For "prototype" scoped beans, Spring does not call the @PreDestroy method. How can you create code to call the destroy method on prototype scope beans.

Answer:

Pearl Arora, 19csu208
For this we use beanPostProcessor but DestructionAwareBeanPostProcessor are for singleton so we create out own implementation of BeanPostProcessor that handles bean distruction of prototype using beanFactory.isPrototype(beanName)
And synchronysed.

```
@Component
public class DestroyPrototypeBeansPostProcessor implements BeanPostProcessor,
BeanFactoryAware, DisposableBean {

    private BeanFactory beanFactory;

    private final List<Object> prototypeBeans = new LinkedList<>();

    @Override
    public Object postProcessBeforeInitialization(Object bean, String beanName) throws
BeansException {
        return bean;
    }

    @Override
    public Object postProcessAfterInitialization(Object bean, String beanName) throws BeansException
{
        if (beanFactory.isPrototype(beanName)) {
            synchronized (prototypeBeans) {
                prototypeBeans.add(bean);
            }
        }
        return bean;
    }

    @Override
    public void setBeanFactory(BeanFactory beanFactory) throws BeansException {
        this.beanFactory = beanFactory;
    }

    @Override
```

```
    public void destroy() throws Exception {
        synchronized (prototypeBeans) {
            for (Object bean : prototypeBeans) {
                if (bean instanceof DisposableBean) {
                    DisposableBean disposable = (DisposableBean)bean;
                    try {
                        disposable.destroy();
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
            }
            prototypeBeans.clear();
        }
    }
}
```