Spring Framework Testing Context

Make 2 word classes implementing WordProducer which just prints word

```
package org.springframework.samples.petclinic.sfg;

public interface WordProducer {
    String getWord();
}
```

```
package org.springframework.samples.petclinic.sfg;

import org.springframework.context.annotation.Primary;
import org.springframework.stereotype.Component;

@Component
@Primary
public class LaurelWordProducer implements WordProducer {
    @Override
    public String getWord() {
        return "Laurel";
    }
}
```

```
package org.springframework.samples.petclinic.sfg;

import org.springframework.stereotype.Component;

@Component
public class YannyWordProducer implements WordProducer {
    @Override
    public String getWord() {
        return "Yanny";
    }
}
```

Here we have a core class dependent on wordProducer

```
package org.springframework.samples.petclinic.sfg;

import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.stereotype.Service;

@Service
public class HearingInterpreter {

    private final WordProducer wordProducer;


    @Autowired
    public HearingInterpreter(WordProducer wordProducer) {
        this.wordProducer = wordProducer;
    }

    public String whatIHeard() {
        String word = wordProducer.getWord();
        System.out.println(word);
        return word;
    }

}
```

Set up configuration classes which we can use for test classes.
We are setting up this in test context folder so we don't want this going out with the larger app.
So we will start laying down initial Spring configuration here.

```
package org.springframework.samples.petclinic.sfg;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class BaseConfig {

    @Bean
    HearingInterpreter hearingInterpreter(WordProducer wordProducer) {
        return new HearingInterpreter(wordProducer);
    }

}
```

```
package org.springframework.samples.petclinic.sfg;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
```

```
@Configuration
public class LaurelConfig {

    @Bean
    LaurelWordProducer laurelWordProducer() {
        return new LaurelWordProducer();
    }
}
```

Now we have to tell unit test about it. Convert unit test into integration test. We will allow the Spring framework, the IOC container Spring framework to manage our test dependency.
Manage the tests to pick up the configuration classes.

Junit does not know anything about this, so we have to tell to @RunWith

@RunWith(SpringRunner.class)
Now tell about the configuration classes, because right we got our spring context but we have not pointed to any configuration.

@ContextConfiguration(classes = {BaseConfig.class, LaurelConfig.class})

So we are using spring framework dependency management and ioc to manage our test components.

```
package org.springframework.samples.petclinic.sfg;

import org.junit.jupiter.api.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringRunner;

import static org.junit.jupiter.api.Assertions.*;


@RunWith(SpringRunner.class)
@ContextConfiguration(classes = {LaurelConfig.class, BaseConfig.class})
class HearingInterpreterTest {

    @Autowired
    private HearingInterpreter hearingInterpreter;

    @Test
```

```
    void whatIHeard() {
        String word = hearingInterpreter.whatIHeard();
        assertEquals("Laurel", word);
    }
}
```

This is all Junit4 testing....

For Junit 5 there is a change.

```
package org.springframework.samples.petclinic.sfg.junit5;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.samples.petclinic.sfg.BaseConfig;
import org.springframework.samples.petclinic.sfg.HearingInterpreter;
import org.springframework.samples.petclinic.sfg.LaurelConfig;
import org.springframework.test.context.junit.jupiter.SpringJUnitConfig;

@SpringJUnitConfig(classes = {BaseConfig.class, LaurelConfig.class})
class HearingInterpreterLaurelTest {

    @Autowired
    private HearingInterpreter hearingInterpreter;
    @Test
    void whatIheard() {
        String word = hearingInterpreter.whatIheard();
        Assertions.assertEquals("Laurel", word);
    }
}
```

@SpringJUnitConfig is a combination of @ExtendWith({SpringExtension.*class*})
@ContextConfiguration

It came out in spring 2.2

There should be a valid question as why the real stereotypes are not getting scanned ?

In Spring applications, component scanning, which detects classes annotated with stereotypes like @Component, @Service, @Repository, etc., is typically enabled to automatically register these classes as beans in the Spring context. However, in tests, the component scanning behavior may not be automatically applied, and you may need to explicitly configure the test context to pick up these stereotypes.

Here's why you might need to declare config classes for tests and why the Spring context may not pick up stereotypes for real classes:

- **Scope of Component Scanning**: Component scanning is typically limited to specific packages or base packages configured in the application context. If your test classes are located in different packages or are not included in the base packages for component scanning, Spring may not automatically detect them as beans.
- **Testing Specific Configurations**: In tests, you may want to customize the application context for testing purposes. This could include mocking certain dependencies, providing test-specific configurations, or loading only relevant beans for testing a specific component or layer of the application. Declaring config classes for tests allows you to explicitly define these configurations.
- **Isolation of Tests**: Tests should ideally be isolated from the actual application context to prevent unintended side effects or dependencies on external resources. By explicitly declaring config classes for tests, you can control which beans are loaded into the test context and ensure that tests run in isolation without affecting the actual application context.
- **Performance Considerations**: Component scanning can be resource-intensive, especially in large applications with many classes. By explicitly configuring the test context with only the necessary beans, you can improve the performance of test execution.

To address these concerns and ensure that stereotypes are picked up for real classes and configurations are applied correctly in tests, you can follow these best practices:

- Ensure that test classes are located in packages covered by component scanning or explicitly include them in the base packages for scanning.
- Declare config classes specifically for tests (@Configuration classes annotated with @SpringJUnitConfig or similar) and customize the test context as needed.
- Use mocking frameworks like Mockito or configure test-specific beans to isolate tests and control dependencies.
- Consider the performance implications of component scanning in tests and optimize the test context configuration accordingly.

By following these practices, you can effectively manage the Spring context in tests and ensure that stereotypes are picked up for real classes while maintaining control and isolation in the test environment.

Using Inner classes for configuration:

It is used when we have to do small configuration tweaks or overriding some

configuration. It gives lot of flexibility that is how our test gonna be brought up and run.

```
package org.springframework.samples.petclinic.sfg.junit5;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.samples.petclinic.sfg.HearingInterpreter;
import org.springframework.samples.petclinic.sfg.LaurelWordProducer;
import org.springframework.test.context.junit.jupiter.SpringJUnitConfig;

import static org.junit.jupiter.api.Assertions.*;


@SpringJUnitConfig(classes =
HearingInterpreterLaurelInnerTest.TestConfig.class)
class HearingInterpreterLaurelInnerTest {

    @Configuration
    static class TestConfig{
        @Bean
        HearingInterpreter hearingInterpreter() {
            return new HearingInterpreter(new LaurelWordProducer());
        }
    }

    @Autowired
    private HearingInterpreter hearingInterpreter;
    @Test
    void whatIheard() {
        String word = hearingInterpreter.whatIheard();
        assertEquals("Laurel", word);
    }
}
```

Using Component Scans:

Till now we have been using strict configuration path.
We have not seen anything from the configuration of the base classes.
This will also add test environment package to scan.

```
package org.springframework.samples.petclinic.sfg.junit5;
```

```
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.samples.petclinic.sfg.HearingInterpreter;
import org.springframework.test.context.junit.jupiter.SpringJUnitConfig;

import static org.junit.jupiter.api.Assertions.*;

@SpringJUnitConfig
class HearingInterpreterTest {

    @Configuration
    @ComponentScan("org.springframework.samples.petclinic.sfg")
    static class TestConfig {

    }

    @Autowired
    private HearingInterpreter hearingInterpreter;

    @Test
    void whatIheard() {
        String word = hearingInterpreter.whatIheard();
        assertEquals("Laurel", word);
    }
}
```

Setting active profiles for Tests

If you have set one class as `@Primary` and the other as `@Profile`, and during testing you activate the profile corresponding to the `@Profile` annotation, the `@Primary` annotation will still take precedence.

The `@Primary` annotation indicates that a bean should be given preference when multiple beans of the same type are present in the Spring context. It always takes precedence over other annotations such as `@Profile`.

So, even if you activate a profile during testing, if there is a bean marked as `@Primary`, it will be selected regardless of the active profile.

If you want to ensure that the bean corresponding to the active profile is selected during testing, you should not mark the other bean with `@Primary`. Instead, you can use `@Qualifier` or other means to specify which bean you want to use. Alternatively, you can create separate Spring contexts for testing

purposes where the active profile is set differently.

```
package org.springframework.samples.petclinic.sfg;

import org.springframework.context.annotation.Primary;
import org.springframework.context.annotation.Profile;
import org.springframework.stereotype.Component;

/**
 * Created by jt on 2019-02-16.
 */
@Component
@Profile("yanny")
public class YannyWordProducer implements WordProducer {
    @Override
    public String getWord() {
        return "Yanny";
    }
}
```

```
package org.springframework.samples.petclinic.sfg;

import org.springframework.context.annotation.Primary;
import org.springframework.context.annotation.Profile;
import org.springframework.stereotype.Component;

/**
 * Created by jt on 2019-02-16.
 */
@Component
@Primary
@Profile("laurel")
public class LaurelWordProducer implements WordProducer {
    @Override
    public String getWord() {
        return "Laurel";
    }
}
```

```
package org.springframework.samples.petclinic.sfg.junit5;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.samples.petclinic.sfg.HearingInterpreter;
```

```
import org.springframework.samples.petclinic.sfg.LaurelWordProducer;
import org.springframework.test.context.ActiveProfiles;
import org.springframework.test.context.junit.jupiter.SpringJUnitConfig;

import java.lang.reflect.Field;
import java.lang.reflect.Method;
import java.lang.reflect.Parameter;

import static org.junit.jupiter.api.Assertions.*;

@SpringJUnitConfig(HearingInterpreterComponentScanTest.TestConfig.class)
@ActiveProfiles("yanny")
class HearingInterpreterComponentScanTest {

    @Configuration
    @ComponentScan("org.springframework.samples.petclinic.sfg")
    static class TestConfig {

    }

    @Autowired
    private HearingInterpreter hearingInterpreter;

    @Test
    void whatIheard(){
        String word = hearingInterpreter.whatIheard();
        assertEquals("Yanny", word);
    }
}
```

Problem is created by this test:

```
@SpringJUnitConfig(classes =
HearingInterpreterLaurelInnerTest.TestConfig.class)
class HearingInterpreterLaurelInnerTest {

    @Configuration
    static class TestConfig{
        @Bean
        HearingInterpreter hearingInterpreter() {
            return new HearingInterpreter(new LaurelWordProducer());
        }
    }

    @Autowired
    private HearingInterpreter hearingInterpreter;
    @Test
```

```
    void whatIheard() {
        String word = hearingInterpreter.whatIheard();
        assertEquals("Laurel", word);
    }
}
```

Configuration classes in test environment is overriding the contents of config classes in base packages (on which component scan is done).
To solve this conflicting configurations, we can use profiles to solve this.

All test will pass by:

```
package org.springframework.samples.petclinic.sfg.junit5;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Profile;
import org.springframework.samples.petclinic.sfg.HearingInterpreter;
import org.springframework.test.context.ActiveProfiles;
import org.springframework.test.context.junit.jupiter.SpringJUnitConfig;

import static org.junit.jupiter.api.Assertions.*;

@SpringJUnitConfig(HearingInterpreterComponentScanTest.TestConfig.class)
@ActiveProfiles("component-scan")
class HearingInterpreterComponentScanTest {

    @Profile("component-scan")
    @Configuration
    @ComponentScan("org.springframework.samples.petclinic.sfg")
    static class TestConfig {

    }

    @Autowired
    private HearingInterpreter hearingInterpreter;

    @Test
    void whatIheard(){
        String word = hearingInterpreter.whatIheard();
        assertEquals("Yanny", word);
    }
}

package org.springframework.samples.petclinic.sfg.junit5;
```

```
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Profile;
import org.springframework.samples.petclinic.sfg.HearingInterpreter;
import org.springframework.samples.petclinic.sfg.LaurelWordProducer;
import org.springframework.test.context.ActiveProfiles;
import org.springframework.test.context.junit.jupiter.SpringJUnitConfig;

import static org.junit.jupiter.api.Assertions.*;


@SpringJUnitConfig(classes =
HearingInterpreterLaurelInnerTest.TestConfig.class)
@ActiveProfiles("inner-class")
class HearingInterpreterLaurelInnerTest {

    @Profile("inner-class")
    @Configuration
    static class TestConfig{
        @Bean
        HearingInterpreter hearingInterpreter() {
            return new HearingInterpreter(new LaurelWordProducer());
        }
    }

    @Autowired
    private HearingInterpreter hearingInterpreter;
    @Test
    void whatIheard() {
        String word = hearingInterpreter.whatIheard();
        assertEquals("Laurel", word);
    }
}
```

But it is good to give all the configuration a different profile.

e.g.

```
package org.springframework.samples.petclinic.sfg.junit5;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.samples.petclinic.sfg.BaseConfig;
import org.springframework.samples.petclinic.sfg.HearingInterpreter;
import org.springframework.samples.petclinic.sfg.YannyConfig;
import org.springframework.test.context.ActiveProfiles;
import org.springframework.test.context.junit.jupiter.SpringJUnitConfig;

@ActiveProfiles("base-yanny")
@SpringJUnitConfig(classes = {BaseConfig.class, YannyConfig.class})
class HearingInterpreterLaurelTest {

    @Autowired
    private HearingInterpreter hearingInterpreter;
    @Test
    void whatIheard() {
        String word = hearingInterpreter.whatIheard();
        Assertions.assertEquals("Yanny", word);
    }
}

package org.springframework.samples.petclinic.sfg;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Profile;

@Configuration
@Profile("base-yanny")
public class YannyConfig {
    @Bean
    YannyWordProducer yannyWordProducer() {
        return new YannyWordProducer();
    }
}


package org.springframework.samples.petclinic.sfg;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Profile;

/**
 * Created by jt on 2019-02-16.
 */
@Configuration
@Profile("base-yanny")
```

```
public class BaseConfig {

    @Bean
    HearingInterpreter hearingInterpreter(WordProducer wordProducer){
        return new HearingInterpreter(wordProducer);
    }
}
```

After giving everyone a profile we can have an error...
As after giving active profile no default bean is active, so we have to include "yanny" profile

```
package org.springframework.samples.petclinic.sfg.junit5;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Profile;
import org.springframework.samples.petclinic.sfg.HearingInterpreter;
import org.springframework.test.context.ActiveProfiles;
import org.springframework.test.context.junit.jupiter.SpringJUnitConfig;

import static org.junit.jupiter.api.Assertions.*;

@SpringJUnitConfig(HearingInterpreterComponentScanTest.TestConfig.class)
@ActiveProfiles({"component-scan","yanny"})
class HearingInterpreterComponentScanTest {

    @Profile("component-scan")
    @Configuration
    @ComponentScan("org.springframework.samples.petclinic.sfg")
    static class TestConfig {

    }

    @Autowired
    private HearingInterpreter hearingInterpreter;

    @Test
    void whatIheard(){
        String word = hearingInterpreter.whatIheard();
        assertEquals("Yanny", word);
    }
}
```

So if a class does not have a profile it will be picked up in any test...

Spring Test Properties:

@TestPropertySource is used to set up things like url, data sources, etc. We can connect to different type of data source and so.

```
package org.springframework.samples.petclinic.sfg;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Profile;
import org.springframework.stereotype.Component;

@Profile("externalized")
@Component
public class PropertiesWordProducer implements WordProducer {

    private String word;

    @Value("${actual.word}")
    public void setWord(String word) {
        this.word = word;
    }

    @Override
    public String getWord() {
        return word;
    }
}
```

```
package org.springframework.samples.petclinic.sfg;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Profile;
import org.springframework.test.context.ActiveProfiles;
import org.springframework.test.context.TestPropertySource;
import org.springframework.test.context.junit.jupiter.SpringJUnitConfig;

import static org.junit.jupiter.api.Assertions.*;

@SpringJUnitConfig(PropertiesWordProducerTest.TestConfig.class)
@ActiveProfiles({"property-test","externalized"})
@TestPropertySource("classpath:yannyProperties.properties")
```

```
class PropertiesWordProducerTest {

    @Autowired
    private HearingInterpreter hearingInterpreter;

    @Test
    void getWord() {
        String word = hearingInterpreter.whatIheard();
        assertEquals("YaNNy", word);
    }

    @Configuration
    @Profile("property-test")
    @ComponentScan("org.springframework.samples.petclinic.sfg")
    static class TestConfig {
    }
}
```

Make yannyProperties.properties under resources section
And put actual.word=YaNNy