Exercise

1. Invoke context events start(), stop() and close().

```
==========Refreshed Called By default==========
Mar 13, 2019 8:14:36 PM org.springframework.context.support.AbstractApplicationContext prep
INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@6d21714
Mar 13, 2019 8:14:36 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadB
INFO: Loading XML bean definitions from class path resource [database-config.xml]
ContextRefreshedEvent
==========Start Called Explicitly==========
ContextStartedEvent
==========Stop Called Explicitly==========
ContextStoppedEvent
==========Close Called Explicitly==========
ContextClosedEvent
```

2. Create listeners for spring events.

```
public class MyEventListener implements ApplicationListener<ApplicationContextEvent> {

    @Override
    public void onApplicationEvent(ApplicationContextEvent event) {
        System.out.println(event.getClass().getSimpleName());
    }
}
```

3. Create a CustomEvent which should get published when you invoke connect method of database bean.

```
public void connectService() throws IOException{
    applicationEventPublisher.publishEvent(new CustomEvent( source: this));
    throw new IOException();
}
```

```
public class CustomListener implements ApplicationListener<CustomEvent> {
    @Override
    public void onApplicationEvent(CustomEvent event) {
        System.out.println("Database Connected.");
    }
}
```

```
==========Getting instance==========
Instance : Database{name='mysql', port=3306}
==========Using Custom Event==========
Database Connected.
```

4. Create a logging aspect. Create a pointcut to log all methods Of services .

```
<!--Enables the use of components marked as Aspect and other related annotations-->
<aop:aspectj-autoproxy/>

<!--Registering Logging Aspect as bean-->
<bean class="aspect.LoggingAspect"/>
```

5. Add a logging statement before and after the method ends

```
@Aspect
public class LoggingAspect {
    @Around(value = "execution(* *Service*(..))")
    void aroundAdvice(ProceedingJoinPoint proceedingJoinPoint) throws Throwable {
        System.out.println("============================Aspect Before Service============================");
        proceedingJoinPoint.proceed();
        System.out.println("============================Aspect After Service============================");
    }
```

```
============================Aspect Before Service============================
Database Connected.
============================Aspect After Service============================
```

6. Log all the methods which throw IOException

```
    @AfterThrowing(pointcut = "execution(* *(..))", throwing = "e")
    void afterReturningAdvice(IOException e) {
        System.out.println("Running AfterThrowing " + e);
    }
```

```
============================Aspect Before Service============================
Database Connected.
Running AfterThrowing java.io.IOException
```

7. Apply execution, within, args, this and bean expressions on the before and after advice of service bean.

```
@Around(value = "execution(* *Service*(..))")
void aroundAdvice(ProceedingJoinPoint proceedingJoinPoint) throws Throwable {
    System.out.println("============================Aspect Before Service============================");
    proceedingJoinPoint.proceed();
    System.out.println("============================Aspect After Service============================");
}
```

```
@Before(value = "within(entity.*)")
void withinBeforeEntityPackageAdvice() {
    System.out.println("============================Before Anything within Entity Package============================")
}

@After(value = "within(entity.*)")
void withinAfterEntityPackageAdvice() {
    System.out.println("============================After Anything within Entity Package============================");
}
```

```
============================Before Anything within Entity Package============================
Database Connected.
============================After Anything within Entity Package============================
```

```
@Before(value = "args(Integer)")
void beforeIntegerArgumentAdvice() {
    System.out.println("============================Before Method with Integer Argument============
}

@After(value = "args(Integer)")
void afterIntegerArgumentAdvice() {
    System.out.println("============================After Method with Integer Argument============
}
```

```
============================Before Method with Integer Argument============================
============================After Method with Integer Argument============================
Port Updated : 27017
```

```
@Before(value = "this(entity.Database)")
void beforeDatabaseObjectAdvice() {
    System.out.println("============================Before Method with Database Object============================");
}

@After(value = "this(entity.Database)")
void afterDatabaseObjectAdvice() {
    System.out.println("============================After Method with Database Object============================");
}
```

```
============================Before Method with Database Object============================
============================After Method with Database Object============================
nstance : Database{name='mysql', port=3306}

rocess finished with exit code 0
```

```
@Before(value = "bean(database)")
void beforeUsingDatabaseBeanAdvice(JoinPoint joinPoint) {
    System.out.println("Before Using Database Bean : " + joinPoint.getThis().getClass().getSimpleName() +
            " having signature : "+joinPoint.getSignature()+" and Arguments : " +
            Arrays.toString(joinPoint.getArgs()));
}

@After(value = "bean(database)")
void afterUsingDatabaseBeanAdvice(JoinPoint joinPoint) {
    System.out.println("After Using Database Bean : " + joinPoint.getThis().getClass().getSimpleName());
}
```

```
Before Using Database Bean : Database$$EnhancerBySpringCGLIB$$d2d196ea
After Using Database Bean : Database$$EnhancerBySpringCGLIB$$d2d196ea
Instance : Database{name='mysql', port=3306}
```

8. Demonstrate the use of pointCut @Pointcut annotation and Reuse the expression created

```
@Pointcut("execution(* getPort())")
void afterBeforeGetPortMethod() {
}

@Pointcut("execution(* setPort(..))")
void afterBeforeSetPortMethod() {
}

@After("afterBeforeGetPortMethod() || afterBeforeSetPortMethod()")
void afterPortGetterSetter() {
    System.out.println("============================After after Port's Getter Setter============================");
}
```

```
============================After after Port's Getter Setter============================
============================After after Port's Getter Setter============================
Port Updated : 27017
```

9. Access the properties of the JoinPoint in before advice. Print Signature of method being called and its arguments

```
@Before(value = "bean(database)")
void beforeUsingDatabaseBeanAdvice(JoinPoint joinPoint) {
    System.out.println("Before Using Database Bean : " + joinPoint.getThis().getClass().getSimpleName() +
            " having signature : "+joinPoint.getSignature()+" and Arguments : " +
            Arrays.toString(joinPoint.getArgs()));
}
```

```
ContextRefreshedEvent
Before Using Database Bean : Database$$EnhancerBySpringCGLIB$$4d11a5c1 having signature : String entity.Database.toString() and Arguments : []
Instance : Database{name='mysql', port=3306}
```