Spring Boot

AOP, Request Methods, Stream

Aspect Oriented Programming

Aspects

AOP

Aspect-Oriented Programming (AOP) is a programming paradigm that complements object-oriented programming (OOP) by allowing the separation of cross-cutting concerns.

Cross-cutting concerns are aspects of a program that affect multiple parts of the application but do not belong to the main business logic.

Examples include logging, transaction management, security, and error handling.

Spring AOP (Aspect-Oriented Programming) is a module in the Spring Framework that provides support for AOP.

Key Concepts of AOP

- Aspect
- Join Point:
- Advice:
- Pointcut:
- Weaving:

Key Concepts of AOP

Aspect:

An aspect is a module that encapsulates a cross-cutting concern.

It can be seen as a class in which advice (action to be performed) is applied at certain join points.

Join Point:

A join point is a point in the execution of a program, such as method execution, object construction, or field assignment.

In Spring AOP, join points typically represent method executions.

Key Concepts of AOP

Advice:

Advice is the action taken at a particular join point. It defines what should happen when the program reaches the join point. There are several types of advice:

- Before Advice: Runs before the method execution.
- After Advice: Runs after the method execution, regardless of its outcome.
- After Returning Advice: Runs after the method returns successfully.
- After Throwing Advice: Runs when a method throws an exception.
- Around Advice: Runs before and after the method execution, allowing the advice to control whether the method proceeds.

Key Concepts of AOP

Pointcut:
A pointcut is an expression that matches join points.
It defines where an advice should be applied in the application.
Pointcuts are usually expressed using method names, annotations, or regular expressions.

Weaving:
Weaving is the process of linking aspects with the application code.
This can happen at various stages of the program lifecycle:

- Compile-time weaving
- Load-time weaving
- Runtime weaving (commonly used in Spring AOP)

Advice

```
@Before("execution(* com.demo.service.*.*(..))")
public void logBeforeMethodExecution() {
   System.out.println("Before advice executed");
}

@After("execution(* com.demo.service.*.*(..))")
public void logAfterMethodExecution() {
   System.out.println("After advice executed");
}

@AfterReturning("execution(* com.demo.service.*.*(..))")
public void logAfterReturningMethodExecution() {
   System.out.println("After returning advice executed");
}
```

```
@AfterThrowing(
value = "execution(* com.demo.service.*.*(..))",
throwing = "ex")
public void logAfterThrowingMethodExecution(Exception ex) {
   System.out.println("Exception thrown: " + ex.getMessage());
}

@Around("execution(* com.demo.service.*.*(..))")
public Object logAroundMethodExecution(ProceedingJoinPoint
joinPoint) throws Throwable {
   System.out.println("Before method execution");
   Object result = joinPoint.proceed();  // Proceed with execution
   System.out.println("After method execution");
   return result;
}
```

Pointcut Expression

- Pointcut expressions in Spring AOP are used to define where advice should be applied. They specify which methods or types should be targeted by the aspect.

The general syntax for a pointcut expression in Spring AOP is:

```
execution(
        modifiers-pattern?
        return-type-pattern
        declaring-type-pattern?
        method-name-pattern(param-pattern)
        throws-pattern?
)
```

Pointcut Expression

- **modifiers-pattern?**: Optionally matches method modifiers like public, private, protected, etc.
- **return-type-pattern**: Matches the return type of the method.
- **declaring-type-pattern?**: Optionally matches the type declaring the method.
- **method-name-pattern**: Matches the method name.
- **param-pattern**: Optionally matches method parameters by type.
- **throws-pattern?**: Optionally matches thrown exceptions.

Matching Execution

- Matching all methods in a class
- @Before("execution(* com.service.*.*(..))")

- Matching methods with specific return type
- @Before("execution(String com.service.*.*(..))")

- Matching methods with specific parameters
- @Before("execution(* com.service.*.processPayment(String, double))")

- Matching methods that throw a specific exception
- @Before("execution(* com.service.*.*(..)) throws com.exception.PaymentException")

Matching Execution

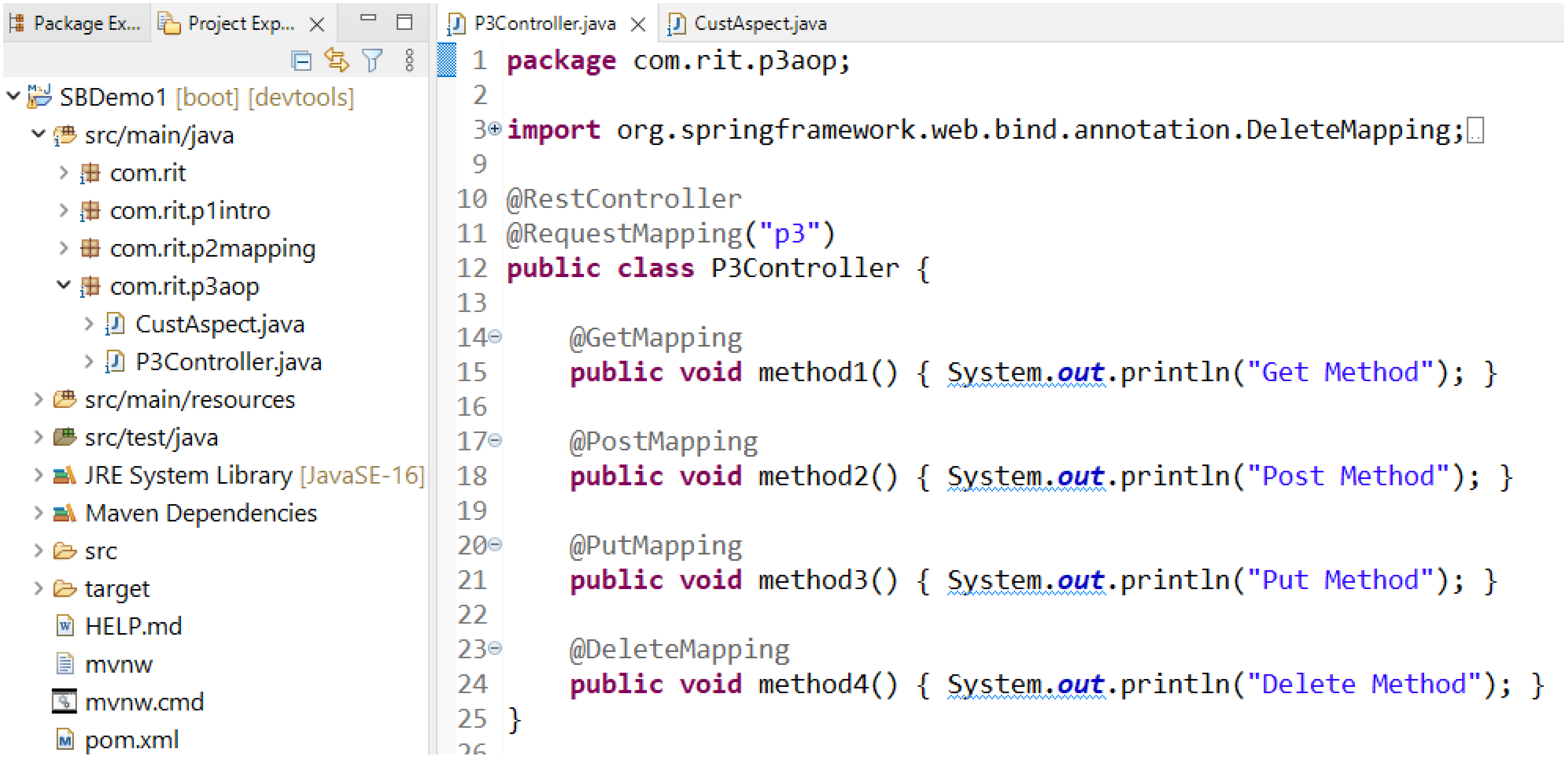
- Using && for Combining Pointcuts

```
@Before("
          execution( * com.service.*.*(..)) **&&**
         @annotation(com.annotation.Loggable)
")
```

This matches methods in the com.service package that are annotated with @Loggable.

- Combining Required and Optional Parameters

```
@Before("execution(* com.service.*.*(String, ..))")
```

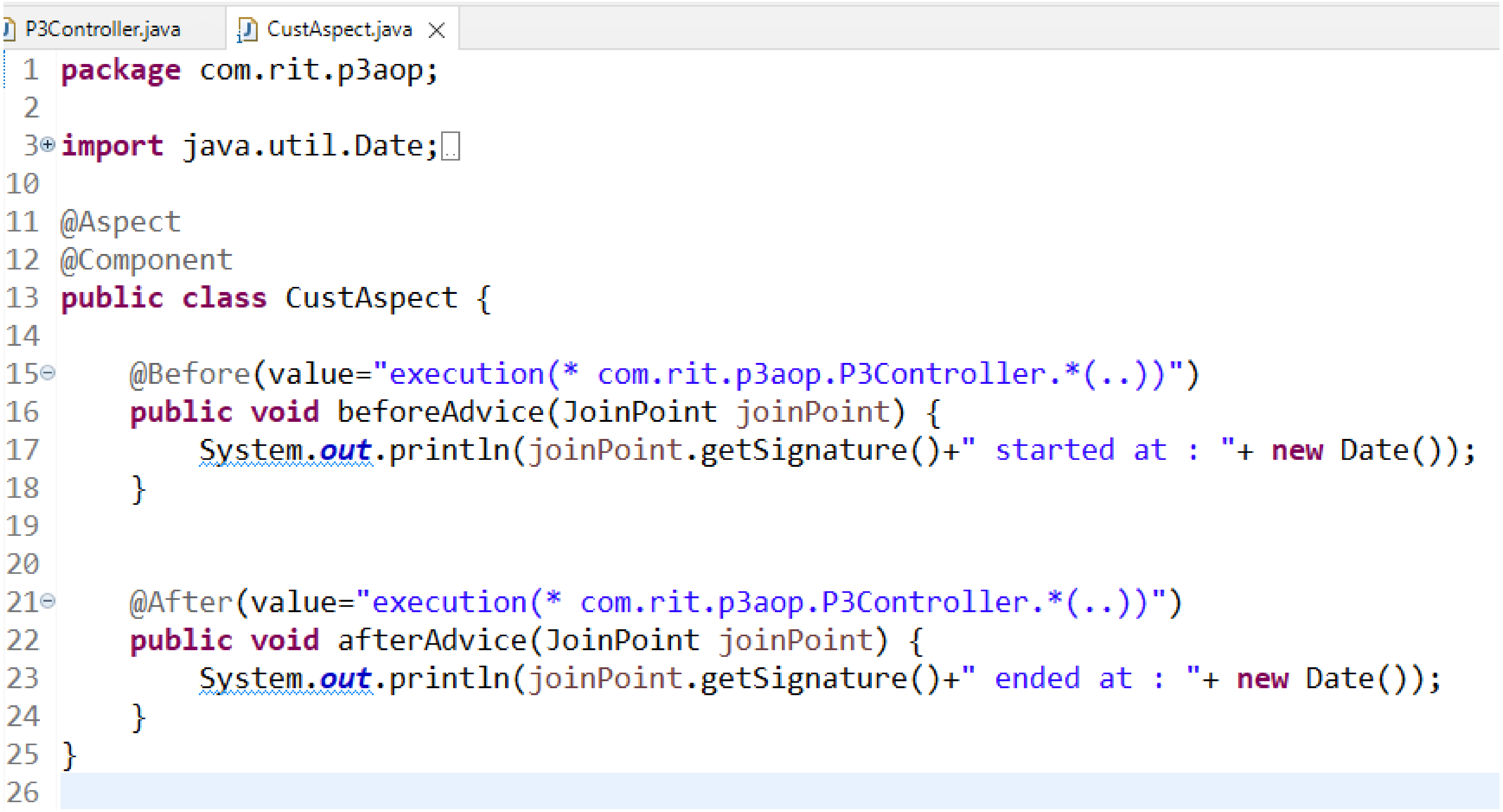


Package Ex... Project Exp...

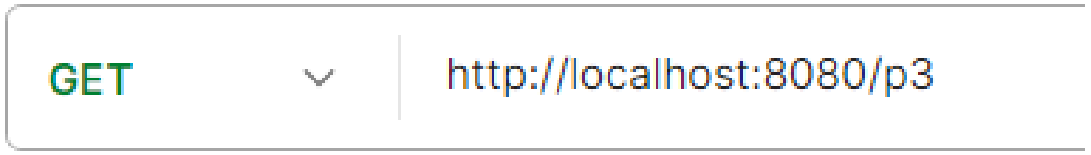
SBDemo1 [boot] [devtools]
- src/main/java
  - com.rit
  - com.rit.p1intro
  - com.rit.p2mapping
  - com.rit.p3aop
    - CustAspect.java
    - P3Controller.java
- src/main/resources
- src/test/java
- JRE System Library [JavaSE-16]
- Maven Dependencies
- src
- target
- HELP.md
- mvnw
- mvnw.cmd
- pom.xml

P3Controller.java CustAspect.java

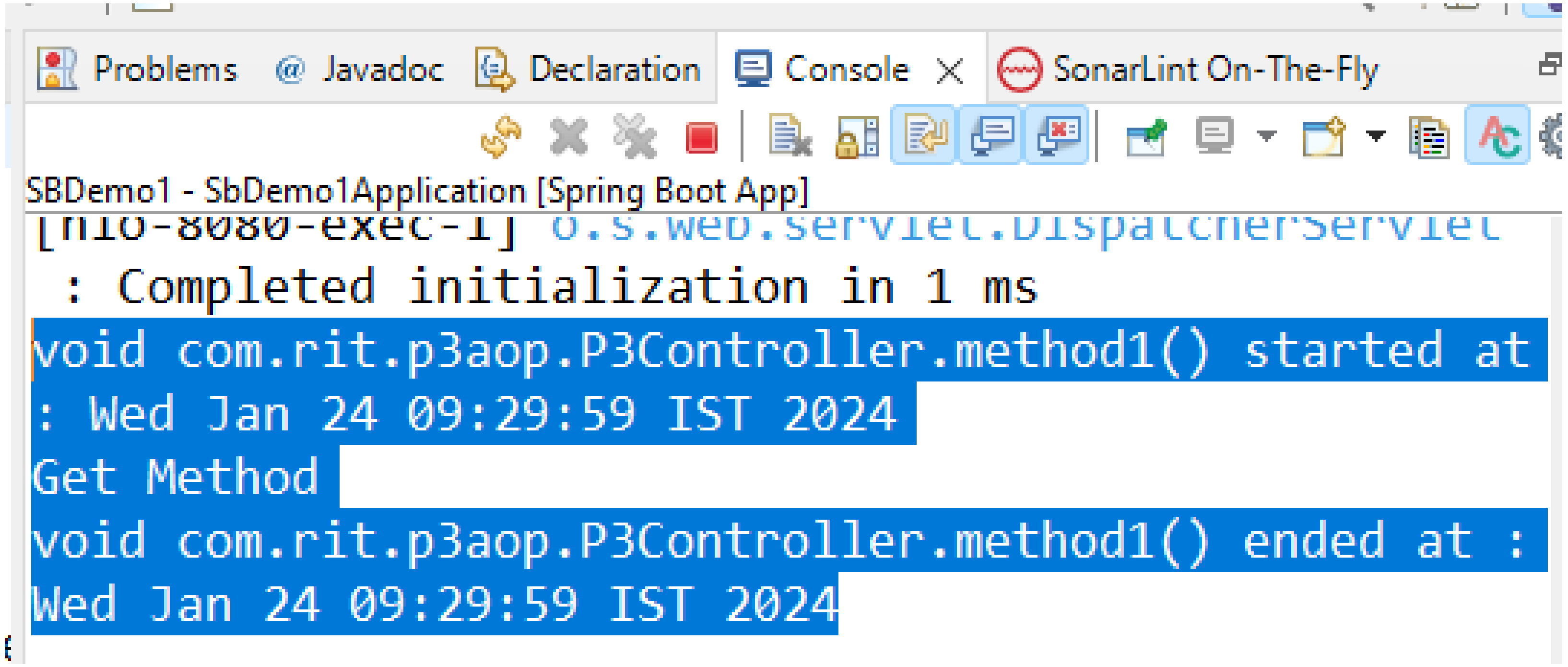
```java
package com.rit.p3aop;

import org.springframework.web.bind.annotation.DeleteMapping;

@RestController
@RequestMapping("p3")
public class P3Controller {

    @GetMapping
    public void method1() { System.out.println("Get Method"); }

    @PostMapping
    public void method2() { System.out.println("Post Method"); }

    @PutMapping
    public void method3() { System.out.println("Put Method"); }

    @DeleteMapping
    public void method4() { System.out.println("Delete Method"); }
}
```



```java
package com.rit.p3aop;

import java.util.Date;

@Aspect
@Component
public class CustAspect {

    @Before(value="execution(* com.rit.p3aop.P3Controller.*(..))")
    public void beforeAdvice(JoinPoint joinPoint) {
        System.out.println(joinPoint.getSignature()+" started at : "+ new Date());
    }


    @After(value="execution(* com.rit.p3aop.P3Controller.*(..))")
    public void afterAdvice(JoinPoint joinPoint) {
        System.out.println(joinPoint.getSignature()+" ended at : "+ new Date());
    }
}
```



```
http://localhost:8080/p3
```



SBDemo1 - SbDemo1Application [Spring Boot App]

```
[nio-8080-exec-1] o.s.web.servlet.DispatcherServlet
 : Completed initialization in 1 ms
void com.rit.p3aop.P3Controller.method1() started at
: Wed Jan 24 09:29:59 IST 2024
Get Method
void com.rit.p3aop.P3Controller.method1() ended at :
Wed Jan 24 09:29:59 IST 2024
```

Thank you