Exercises for lecture “Software Architecture“ – *06 – Additional Design Considerations*

1 Message Queueing

In moodle you find in the zip 06.1 two maven projects for a provider and a consumer. Import both maven projects. Start the producer and enter a text. Now start the consumer and observe its behavior. What would be a use case for this kind of communication?

2 A Rest-Service

In moodle you find in the zip 06.2. two maven projects for a service (based on Spring) and a client (using plain JDK). Import both maven projects. Start the server. Try out the service by calling the service in the browser http://localhost:8080/api/v1 . Next try out the documented interface in swagger: http://localhost:8080/api/v1/swagger-ui.html Finally, start the client which accesses the service. What would be a use case for this kind of communication?

3 Aspects with AspectJ

Implementing crosscutting concerns with AspectJ.

3.1 A Simple AspectJ Example: Setting up the Application

Create a simple maven project (“Skip Stereotypes”, group id: “de.stuttgart.hft”, project id “06.3.1.aspectj.sample”). Add AspectJ to the pom file (also in the cheat sheet):

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
     <modelVersion>4.0.0</modelVersion>

     <groupId>de.stuttgart.hft</groupId>
     <artifactId>aspectj.sample</artifactId>
     <version>0.0.1-SNAPSHOT</version>

     <properties>
          <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
          <aspectj.version>1.9.21.1</aspectj.version>
     </properties>

     <dependencies>
          <dependency>
               <groupId>org.aspectj</groupId>
               <artifactId>aspectjrt</artifactId>
               <version>${aspectj.version}</version>
          </dependency>
          <dependency> <!-- not really needed, but makes the tool explicit -->
```

```
                <groupId>org.aspectj</groupId>
                <artifactId>aspectjtools</artifactId>
                <version>${aspectj.version}</version>
            </dependency>

            <dependency>
                <groupId>org.codehaus.mojo</groupId>
                <artifactId>exec-maven-plugin</artifactId>
                <version>3.2.0</version>
            </dependency>

            <dependency>
                <groupId>org.junit.jupiter</groupId>
                <artifactId>junit-jupiter-engine</artifactId>
                <version>5.10.2</version>
                <scope>test</scope>
            </dependency>

        </dependencies>

        <build>
            <plugins>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-compiler-plugin</artifactId>
                    <version>3.12.1</version>
                    <configuration>
                        <release>21</release>
                    </configuration>
                </plugin>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-surefire-plugin</artifactId>
                    <version>3.2.5</version>
                    <dependencies>
                        <dependency>
                            <groupId>org.junit.jupiter</groupId>
                            <artifactId>junit-jupiter-engine</artifactId>
                            <version>5.10.2</version>
                        </dependency>
                    </dependencies>
                </plugin>
                <plugin>
                    <groupId>dev.aspectj</groupId>
                    <artifactId>aspectj-maven-plugin</artifactId>
                    <version>1.14</version>
                    <dependencies>
                        <dependency>
                            <groupId>org.aspectj</groupId>
                            <artifactId>aspectjtools</artifactId>
                            <version>${aspectj.version}</version>
                        </dependency>
                    </dependencies>
                    <configuration>
                        <complianceLevel>21</complianceLevel>
                        <showWeaveInfo>true</showWeaveInfo>
                        <verbose>true</verbose>
                        <Xlint>ignore</Xlint>
                        <encoding>UTF-8 </encoding>
                    </configuration>
                    <executions>
                        <!-- Warning maybe ignored in Eclipse preferences -->
                        <execution>
                            <goals>
                        <!-- use this goal to weave all your main classes -->
                                <goal>compile</goal>
                        <!-- use this goal to weave all your test classes -->
                                <goal>test-compile</goal>
                            </goals>
                        </execution>
                    </executions>
                </plugin>

                <plugin>
                    <groupId>org.codehaus.mojo</groupId>
                    <artifactId>exec-maven-plugin</artifactId>
                    <version>3.2.0</version>
                </plugin>
            </plugins>
        </build>

    </project>
```

Now create the lecture example with the following three classes in the package `sample.app.classes` (also in the cheat sheet):

```java
public class Hello {

    public static void sayHello() {
        System.out.println("Hello World ");
    }

    public static void sayGoodbye() {
        System.out.println("Goodbye, cruel World ");
    }
    public void printSomething(String s) {
        System.out.println(s);
    }

}
```

```java
public class HelloToo {

    public static void saySomething() {
        System.out.println("Hello World, too ");
    }

    public static void makeRemark() {
        System.out.println("Hi World ");
    }
    public void printSomething(String s) {
        System.out.println(s);
    }
}
```

```
public class Main {

     public static void main(String[] args) {
          Hello h = new Hello();
          HelloToo h2 = new  HelloToo();
          h.sayHello();
          h2.saySomething();
          h2.makeRemark();
          h.sayGoodbye();
          h2.printSomething("Hello, World");
          h.printSomething("secret");
     }
}
```

Try out the application.

3.2 A Simple AspectJ Example: Adding Aspects

After this create the following aspect in a package `sample.app.aspects`:

```
@Aspect
public class MonitorAspect {

     @Pointcut("execution(* *.say*(..))")
     public void executionOfSay() {}

     @Before("executionOfSay()")
     public void beforeExecution() {
          System.out.print("\nexecution started >>\n\t");
     }

     @After("executionOfSay()")
     public void afterExecution() {
          System.out.println("<< execution finished\n");
     }
}
```

To visualize (and debug) the compile process, execute “Run as … → Maven build … Enter goal compile”. Now execute the Main-class again.

Next, add the second aspect, compile and run again.

```
@Aspect
public class MonitorAspectOnTarget {

     @Pointcut("call(* *.*(String)) && args(s) && target(callee)")
     void anyCall(String s, HelloToo callee) {
     }

     @Before("anyCall(s, callee)")
     public void beforeAnyCall(String s, HelloToo callee) {
          System.out.println("Before method call with int argument: " +
               s + " outside of target: " + callee);
     }
```

```
	@Pointcut("execution(* *.*(String)) && args(s) && target(callee)")
	void anyExecution(String s, HelloToo callee) {
	}

	@Before("anyExecution(s, callee)")
	public void beforeAnyExecution(String s, HelloToo callee) {
		System.out.println("Before method execution with int argument: " + s
			+ " inside of target: " + callee);
	}
}
```

Finally, add the third aspect, compile and run again.

```
@Aspect
public class InterferingAspect {

	@Pointcut("within(sample.app.classes.HelloToo) && call(* *println(..))")
	public void interfere1() {
	}

	@Around("interfere1()")
	public void replace(JoinPoint joinPoint) throws Throwable {
		String str = joinPoint.getArgs()[0].toString();
		System.out.println("PRINTING: ***" + str + " ***");
	}

	@Pointcut("within(sample.app.classes.Hello) && call(* *println(..))")
	public void interfere2() {
	}

	@Around("interfere2()")
	public void around(ProceedingJoinPoint joinPoint) throws Throwable {
		String str = joinPoint.getArgs()[0].toString();
		if (str.equals("secret"))
			System.out.println("*******");
		else
			joinPoint.proceed();
	}
}
```

3.3 Logging with Aspects

Now return to the application of the previous (Pattern)-Task. Copy the project and add AspectJ to the pom-file.

Now create a logging monitor, which logs all database write actions with the value and time written (i.e. executions of `AppRepository.create` or `AppRepository.update` – hint: you can use || to combine two pointcuts). In what level should the aspect be placed?

In a first version just write the calls to the console. In a second version write the calls to a file; a method for writing could look like this:

```
private final String filename = "application.log";

private void appendMsg(Object ... args) {
   try {
      Path path = Path.of(filename);
      if(!Files.exists(path))
         Files.createFile(path);
      String msg = LocalDateTime.now() + " : " + Arrays.toString(args) + "\n";
      Files.writeString(path, msg, StandardOpenOption.APPEND);
   } catch (IOException e) {
      e.printStackTrace();
}
```

No create a second aspect, which filters certain reads from the database. If an id which is the current day of the month is entered, the read operation from the database should return null, otherwise it should proceed as normal. (This simulates the access restriction fro a user-profile). Hint: here you should use `@Around`. Again, in what level should the aspect be placed?