Sep 3rd Project spring-mvc-with-aop-demo

Create a Maven MVC project and deploy on an IDE Tomcat server. This document describes creating a MVC project in the SpringTool IDE. To begin the spring-mvc-orm-testing-demo project is copied. The lecture for this project on Sep 3rd failed to upload to the website. This document will start with coping the spring-mvc-orm-testing-demo then applying the changes seen in the new project for the spring-mvc-aop-demo.

The Spring Framework is an application framework and inversion of control container for the Java platform. The framework's core features can be used by any Java application, but there are extensions for building web applications on top of the Java EE.

This project is based on Object Relational Mapping (ORM) data access with integrated testing modules. ORM in this case is the Spring Framework integration with Hibernate using MariaDB and a memory DB for testing. There is first-class support with lots of IoC convenience features, addressing many typical Hibernate integration issues. All of these support packages for O/R (Object Relational) mappers comply with Spring's generic transaction and DAO exception hierarchies. There are usually two integration styles: either using Spring's DAO 'templates' or coding DAOs against plain Hibernate/JDO/TopLink/etc APIs. In both cases, DAOs can be configured through Dependency Injection and participate in Spring's resource and transaction management.

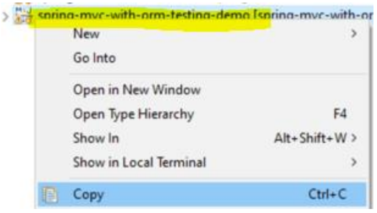
This project introduces Aspect Oriented Programming (AOP). AOP “entails breaking down program logic into distinct parts (so-called concerns, cohesive areas of functionality). Nearly all programming paradigms support some level of grouping and encapsulation of concerns into separate, independent entities by providing abstractions (e.g., functions, procedures, modules, classes, methods) that can be used for implementing, abstracting and composing these concerns. Some concerns "cut across" multiple abstractions in a program, and defy these forms of implementation. These concerns are called cross-cutting concerns or horizontal concerns” -- Wikipedia.

Table of Contents

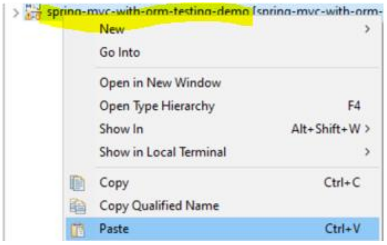
Create new Maven project to use AOP

Create the new project based on a previous ORM with testing project

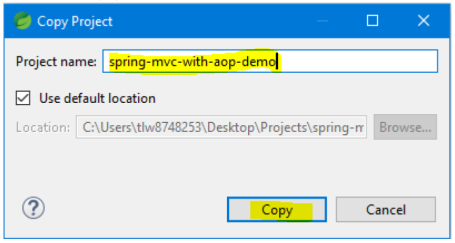
Copy the spring-mvc-orm-testing-demo project
Right click on the project
Select Copy



Right click again in navigation window
Select Paste
If there is an error try Ctrl+v



Name the project: "spring-mvc-with-aop-demo"
Click Copy

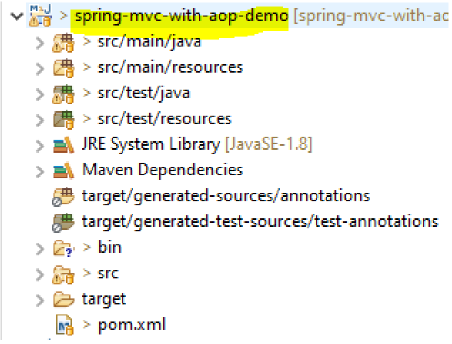


Project name: spring-mvc-with-aop-demo

☑ Use default location

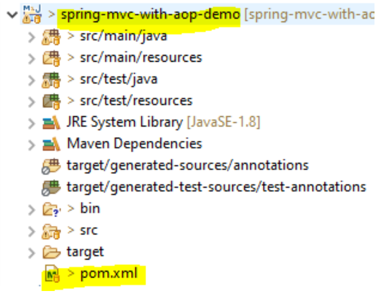
Location: C:\Users\tlw8748253\Desktop\Projects\spring-m

The folder structure to begin this project should look as follows:

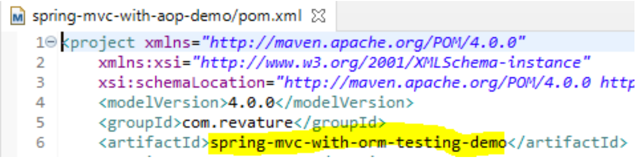

> spring-mvc-with-aop-demo [spring-mvc-with-ac
> src/main/java
> src/main/resources
> src/test/java
> src/test/resources
JRE System Library [JavaSE-1.8]
Maven Dependencies
target/generated-sources/annotations
target/generated-test-sources/test-annotations
> bin
> src
target
> pom.xml

Update the Project Name

Change the Project Name in POM.xml file



```
> spring-mvc-with-aop-demo [spring-mvc-with-ac
  > src/main/java
  > src/main/resources
  > src/test/java
  > src/test/resources
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  target/generated-sources/annotations
  target/generated-test-sources/test-annotations
  > bin
  > src
  target
  > pom.xml
```

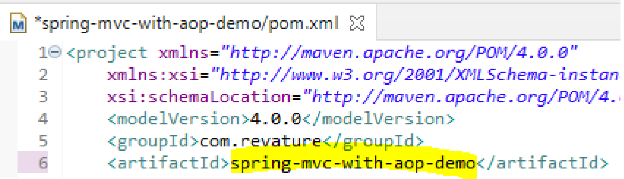


spring-mvc-with-aop-demo/pom.xml

```
<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 http
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.revature</groupId>
    <artifactId>spring-mvc-with-orm-testing-demo</artifactId>
```

To:

```
<artifactId>spring-mvc-with-aop-demo</artifactId>
```



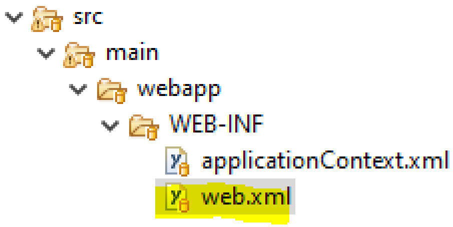
*spring-mvc-with-aop-demo/pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instan
    xsi:schemaLocation="http://maven.apache.org/POM/4.
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.revature</groupId>
    <artifactId>spring-mvc-with-aop-demo</artifactId>
```

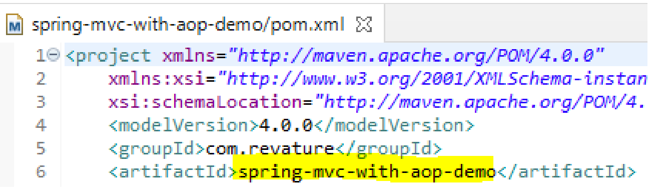
No further updates are required at this time. Updates will be made later for this project.

Change the Project name in WEB.xml file

Any time you open the web.xml file you might see an error. This is a bug with the IDE. Once the file is change in some way, add or delete a space, or make a needed change and the file is saved, the error should resolve.



```
src
  main
    webapp
      WEB-INF
        applicationContext.xml
        web.xml
```



spring-mvc-with-aop-demo/pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instan
    xsi:schemaLocation="http://maven.apache.org/POM/4.
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.revature</groupId>
    <artifactId>spring-mvc-with-aop-demo</artifactId>
```

To:

```
<display-name>spring-mvc-with-aop-demo</display-name>
```

Test the project name changes

When copying from a previous project we can expect issues with using the correct project URL to test the APIs. We want to test and resolve any issues up front before making modifications to the application.

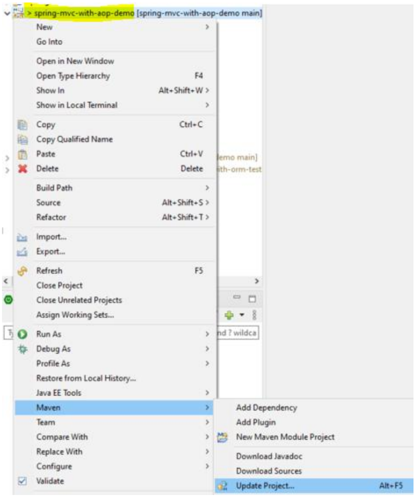
Since we did copy from the previous demo the database connectivity should be the same and any data in the database should also exists. We do not have to change configuration to “create” and leave it at “validate”.

Prelude to initial test

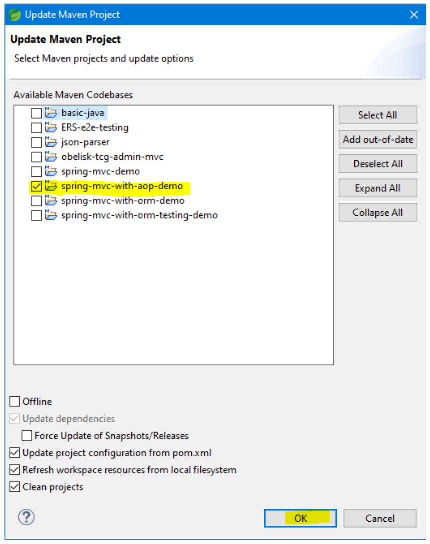
These few steps have worked to resolve the URL naming issue in a few projects. In case they do not work, additional steps are found in previous documents in their appendix.

Rebuild the project.

Right click on the project ➔ select “Maven” ➔ select “Update Project …”



The project should be selected by default.
Click “OK”


Update Maven Project

Update Maven Project

Select Maven projects and update options

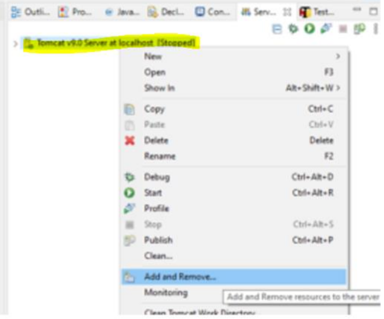
Available Maven Codebases

- ☐ basic-java
- ☐ ERS-e2e-testing
- ☐ json-parser
- ☐ obelisk-tcg-admin-mvc
- ☐ spring-mvc-demo
- ☑ spring-mvc-with-aop-demo
- ☐ spring-mvc-with-orm-demo
- ☐ spring-mvc-with-orm-testing-demo

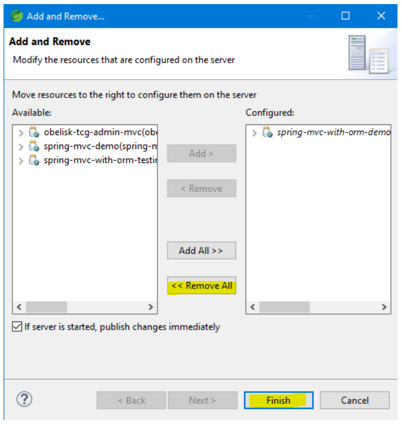
- ☐ Offline
- ☑ Update dependencies
  - ☐ Force Update of Snapshots/Releases
- ☑ Update project configuration from pom.xml
- ☑ Refresh workspace resources from local filesystem
- ☑ Clean projects

Update the Tomcat server

The location of the Server tab and window can vary depending on your IDE layout.
Select the Server tab
Right click the server ➔ select "Add and Remove…"



Remove all projects.
Click "Finish"


Add and Remove...

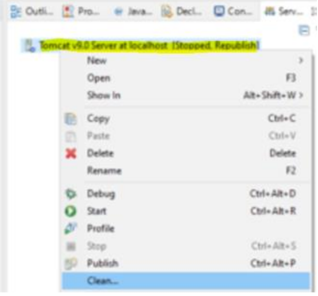
Add and Remove

Modify the resources that are configured on the server

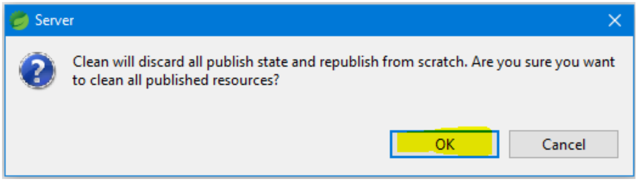
Move resources to the right to configure them on the server

☑ If server is started, publish changes immediately

Right click the server ➔ click "Clean…"

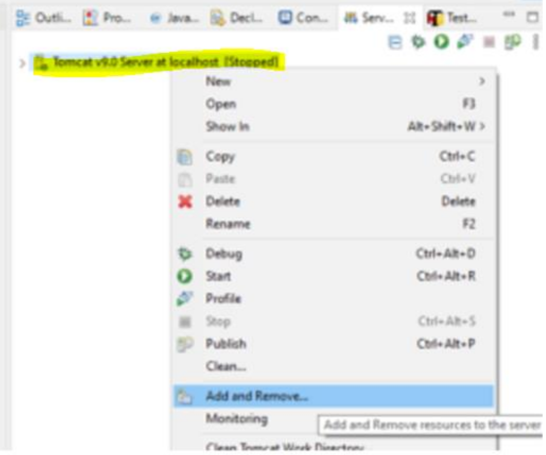


Click "OK"

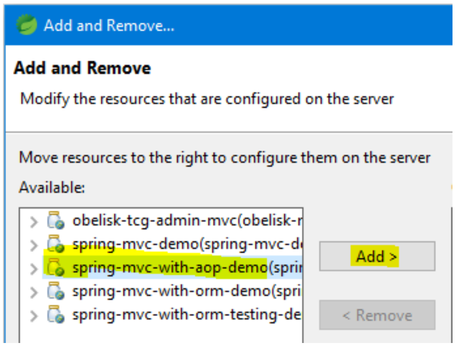

Server

Clean will discard all publish state and republish from scratch. Are you sure you want to clean all published resources?

Add this project to the server.
Right click the server ➔ select “Add and Remove…”



Select this project “spring-mvc-with-aop-demo”
Click “Add >”


Add and Remove...

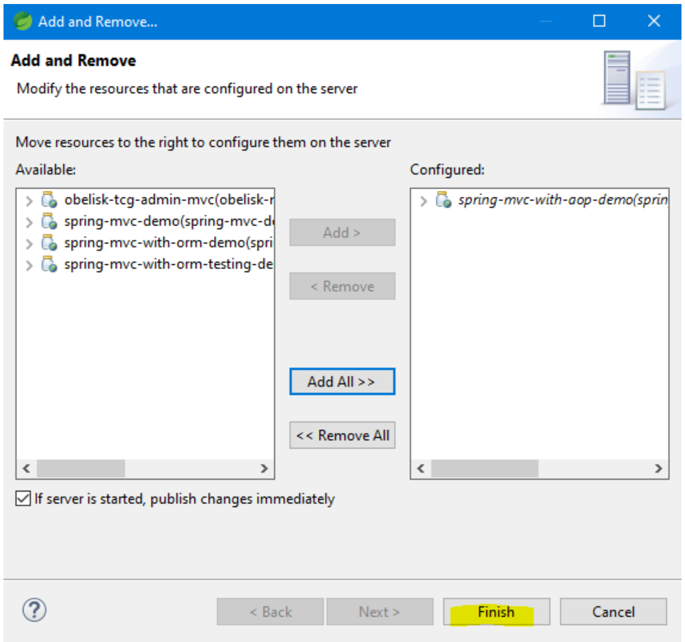
Add and Remove

Modify the resources that are configured on the server

Move resources to the right to configure them on the server

Available:
obelisk-tcg-admin-mvc(obelisk-r
spring-mvc-demo(spring-mvc-de
spring-mvc-with-aop-demo(sprir
spring-mvc-with-orm-demo(spri
spring-mvc-with-orm-testing-de
Add >
< Remove

Click “Finish”


Add and Remove...

Add and Remove

Modify the resources that are configured on the server

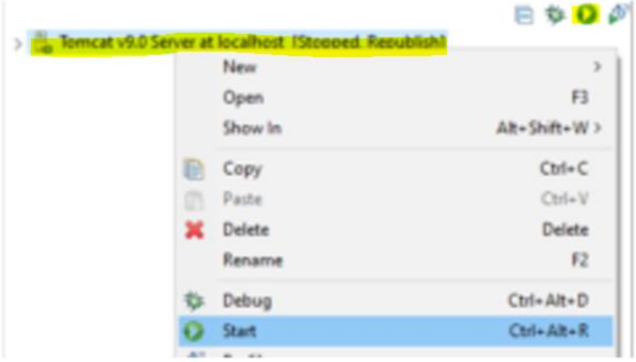
Move resources to the right to configure them on the server

Available:

Configured:
obelisk-tcg-admin-mvc(obelisk-r
spring-mvc-demo(spring-mvc-de
spring-mvc-with-orm-demo(spri
spring-mvc-with-orm-testing-de
Add >
< Remove
Add All >>
<< Remove All
spring-mvc-with-aop-demo(sprin

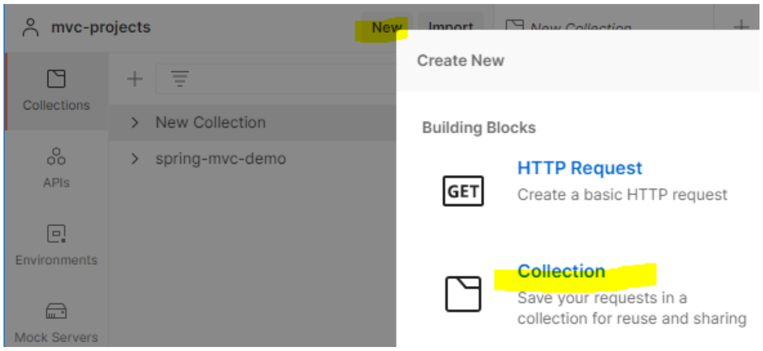
☑ If server is started, publish changes immediately
< Back
Next >
Finish
Cancel

Start the Tomcat server
Right click the server ➔ select "Start"

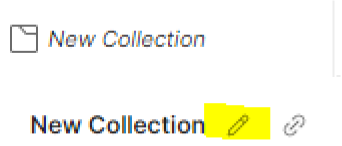


Test the Project Name Change

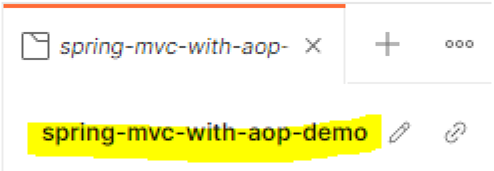
Open the Postman desktop application
Create a new collection



Edit the name



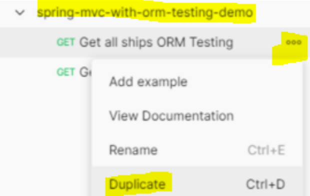
Enter the name: "spring-mvc-with-aop-demo"



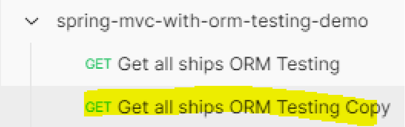
Test the GET API

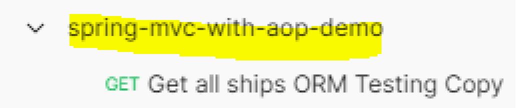
Using the Postman desktop application send in a request to get all ships. You can copy and modify the GET request from the last project.

Open Collection: spring-mvc-with-orm-testing-demo
Select the menu dots on GET all ships request
Select “Duplicate” GET all ships



Drag and drop the copy to new collection

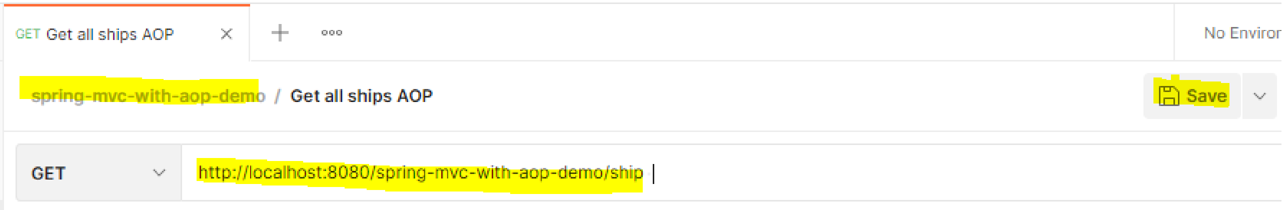

spring-mvc-with-orm-testing-demo
GET Get all ships ORM Testing
GET Get all ships ORM Testing Copy



Rename the test: “Get all ships AOP”
Use the project URL”

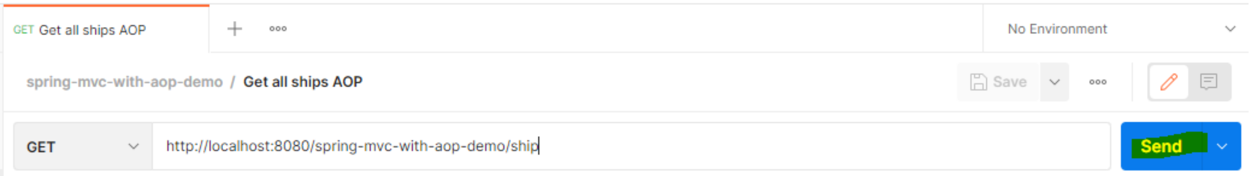
http://localhost:8080/spring-mvc-with-aop-demo/ship

Click “Save”



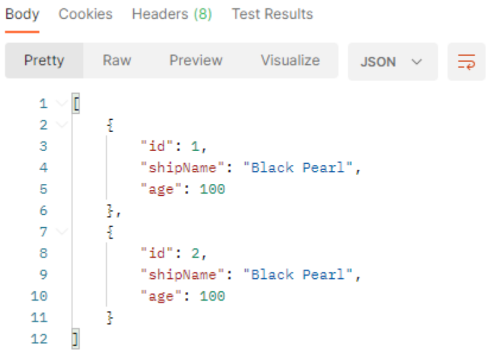
http://localhost:8080/spring-mvc-with-aop-demo/ship

Click “Send”



http://localhost:8080/spring-mvc-with-aop-demo/ship

Expect results (might vary depending on what is in your database).

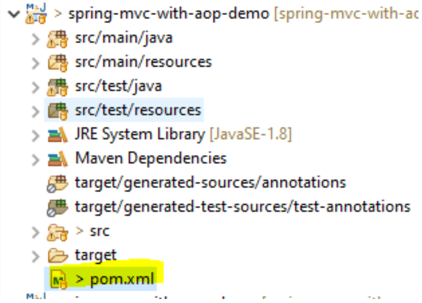


```
[
    {
        "id": 1,
        "shipName": "Black Pearl",
        "age": 100
    },
    {
        "id": 2,
        "shipName": "Black Pearl",
        "age": 100
    }
]
```

Add Project Related Dependencies

Add Project related dependencies to the POM.xml file

Open pom.xml file.



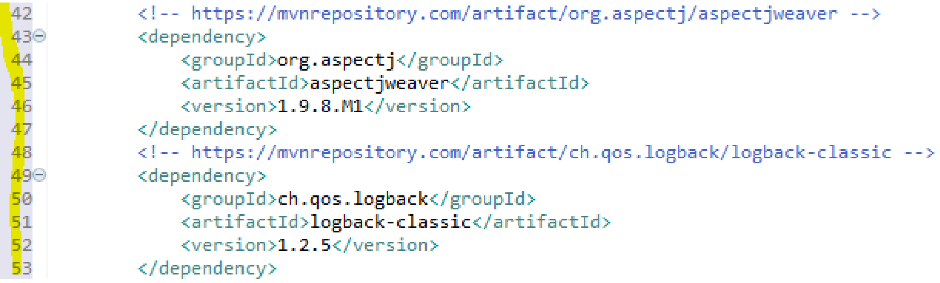
```
> spring-mvc-with-aop-demo [spring-mvc-with-ac
    src/main/java
    src/main/resources
    src/test/java
    src/test/resources
    JRE System Library [JavaSE-1.8]
    Maven Dependencies
    target/generated-sources/annotations
    target/generated-test-sources/test-annotations
    > src
    target
    > pom.xml
```

Add Aspect related dependency.
aspectj enables: - clean modularization of crosscutting concerns, such as error checking and handling, synchronization, context-sensitive behavior, performance optimizations, monitoring and logging, debugging support, and multi-object protocols

Add logging (logback) related dependency.
Next generation logging capability based on log4j.

Add the following dependencies at the location shown:

```
                <!-- https://mvnrepository.com/artifact/org.aspectj/aspectjweaver -->
                <dependency>
                        <groupId>org.aspectj</groupId>
                        <artifactId>aspectjweaver</artifactId>
                        <version>1.9.8.M1</version>
                </dependency>
                <!-- https://mvnrepository.com/artifact/ch.qos.logback/logback-classic -->
                <dependency>
                        <groupId>ch.qos.logback</groupId>
                        <artifactId>logback-classic</artifactId>
                        <version>1.2.5</version>
                </dependency>
```

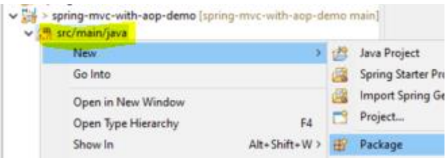


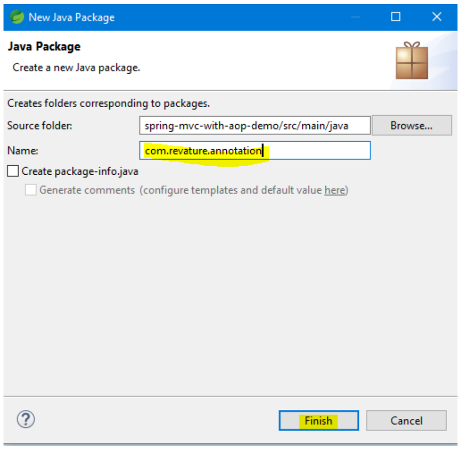
```
        <!-- https://mvnrepository.com/artifact/org.aspectj/aspectjweaver -->
        <dependency>
            <groupId>org.aspectj</groupId>
            <artifactId>aspectjweaver</artifactId>
            <version>1.9.8.M1</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/ch.qos.logback/logback-classic -->
        <dependency>
            <groupId>ch.qos.logback</groupId>
            <artifactId>logback-classic</artifactId>
            <version>1.2.5</version>
        </dependency>
```

Update AOP Project Components

Add new annotation package and interface

Add package: com.revature.annotation





Creates folders corresponding to packages.

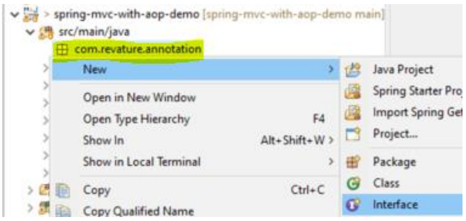
Source folder: spring-mvc-with-aop-demo/src/main/java

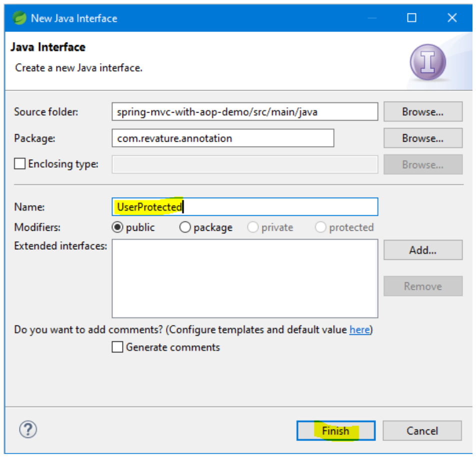
Name: com.revature.annotation

☐ Create package-info.java

☐ Generate comments (configure templates and default value here)

Add interface: UserProtected





Source folder: spring-mvc-with-aop-demo/src/main/java

Package: com.revature.annotation

☐ Enclosing type:

Name: UserProtected

Modifiers: ● public ○ package ○ private ○ protected

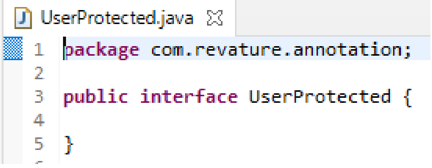
Extended interfaces:

Do you want to add comments? (Configure templates and default value here)

☐ Generate comments

Update annotation interface code shell: UserProtected

The interface wizard produces a code shell to start with:



UserProtected.java

```
package com.revature.annotation;

public interface UserProtected {

}
```

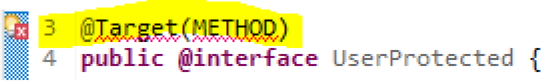
We update to an @interface annotation.

```
public @interface UserProtected {
```

The @ symbol denotes an annotation type definition. That means this is not really an interface, but rather a new annotation type -- to be used as a function modifier, such as @override.

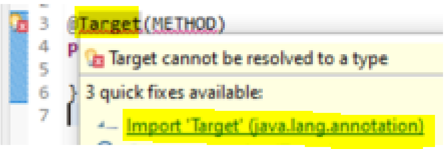
Add interface annotation:

```
@Target(METHOD)
```



```
@Target(METHOD)
public @interface UserProtected {
```

Correct the errors:



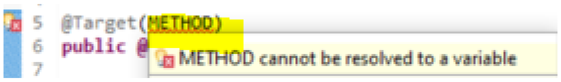
```
@Target(METHOD)
```

Target cannot be resolved to a type

3 quick fixes available:

Import 'Target' (java.lang.annotation)

The wizard is unable to determine the fix for this error:

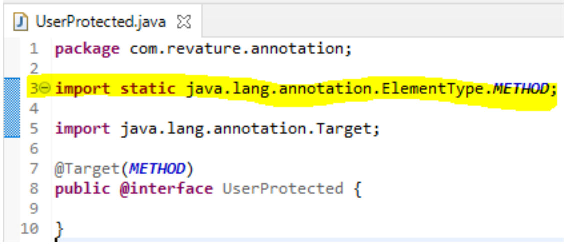


```
@Target(METHOD)
public @
```

METHOD cannot be resolved to a variable

Add the following import manually:

```
import static java.lang.annotation.ElementType.METHOD;
```

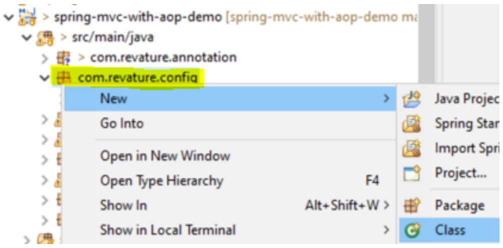


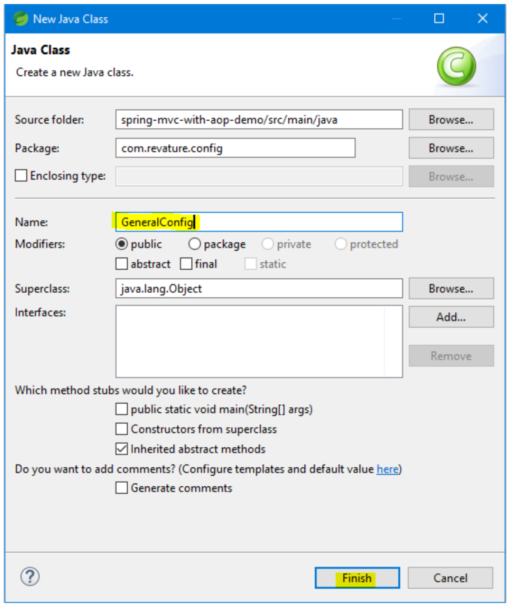
UserProtected.java

```
package com.revature.annotation;

import static java.lang.annotation.ElementType.METHOD;

import java.lang.annotation.Target;

@Target(METHOD)
public @interface UserProtected {

}
```

Add new configuration class

Add configuration class: GeneralConfig




New Java Class

Java Class

Create a new Java class.

Source folder: spring-mvc-with-aop-demo/src/main/java Browse...
Package: com.revature.config Browse...
□ Enclosing type: Browse...

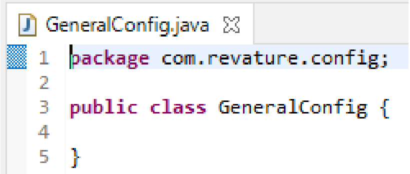
Name: GeneralConfig
Modifiers: ● public ○ package ○ private ○ protected
□ abstract □ final □ static
Superclass: java.lang.Object Browse...
Interfaces: Add... Remove

Which method stubs would you like to create?
□ public static void main(String[] args)
□ Constructors from superclass
☑ Inherited abstract methods
Do you want to add comments? (Configure templates and default value here)
□ Generate comments

Finish Cancel

Update interface code shell: GeneralConfig

The interface wizard produces a code shell to start with:

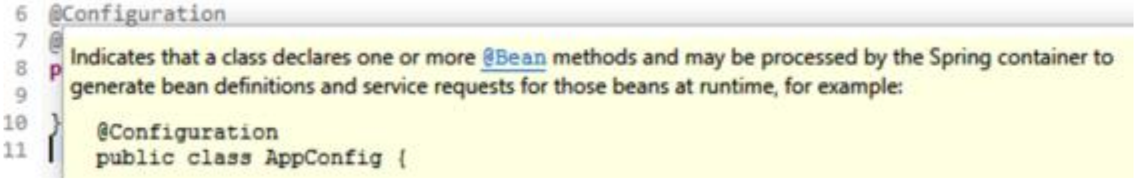

GeneralConfig.java

```java
package com.revature.config;

public class GeneralConfig {

}
```

Add the following annotation to the class:

```java
@Configuration
@EnableAspectJAutoProxy
```

@Configuration

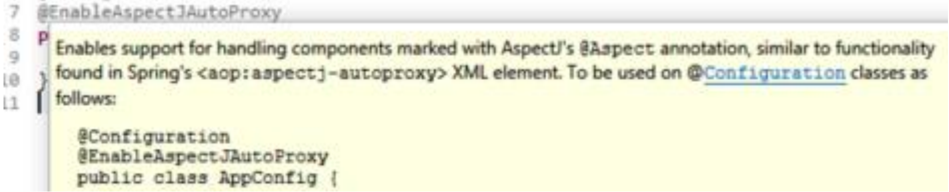


```
@Configuration
```

Indicates that a class declares one or more @Bean methods and may be processed by the Spring container to generate bean definitions and service requests for those beans at runtime, for example:

```
@Configuration
public class AppConfig {
```

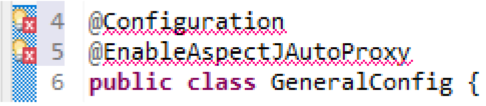
@EnableAspectJAutoProxy



```
@EnableAspectJAutoProxy
```

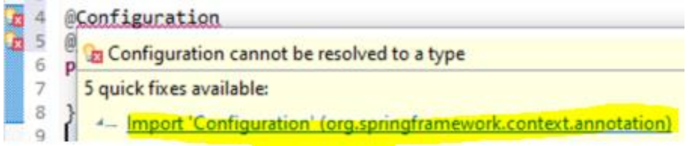
Enables support for handling components marked with AspectJ's @Aspect annotation, similar to functionality found in Spring's <aop:aspectj-autoproxy> XML element. To be used on @Configuration classes as follows:

```
@Configuration
@EnableAspectJAutoProxy
public class AppConfig {
```



```
@Configuration
@EnableAspectJAutoProxy
public class GeneralConfig {
```

Resolve the errors:

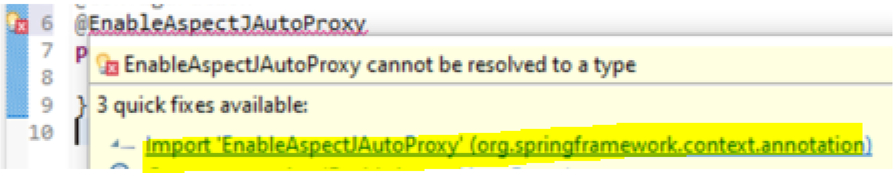


```
@Configuration
```

Configuration cannot be resolved to a type

5 quick fixes available:

Import 'Configuration' (org.springframework.context.annotation)

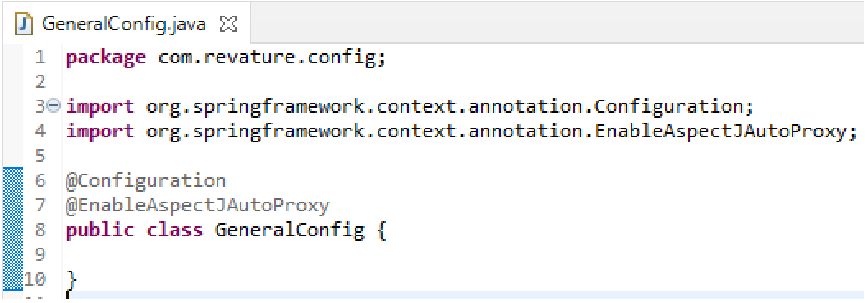


```
@EnableAspectJAutoProxy
```

EnableAspectJAutoProxy cannot be resolved to a type

3 quick fixes available:

Import 'EnableAspectJAutoProxy' (org.springframework.context.annotation)



GeneralConfig.java

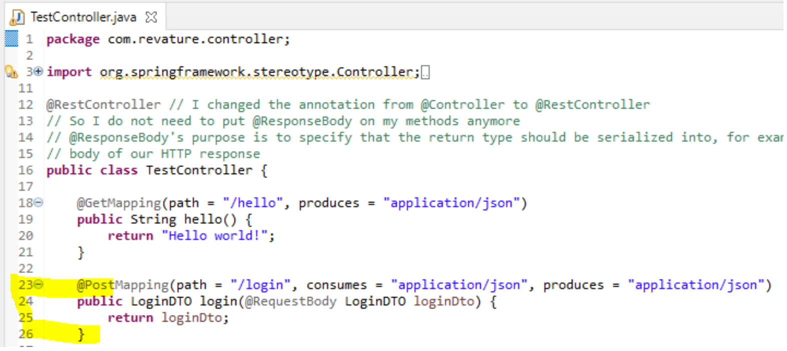
```java
package com.revature.config;

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

@Configuration
@EnableAspectJAutoProxy
public class GeneralConfig {

}
```

There are no further updates to this class. The class body remains empty.

Update controller package classes

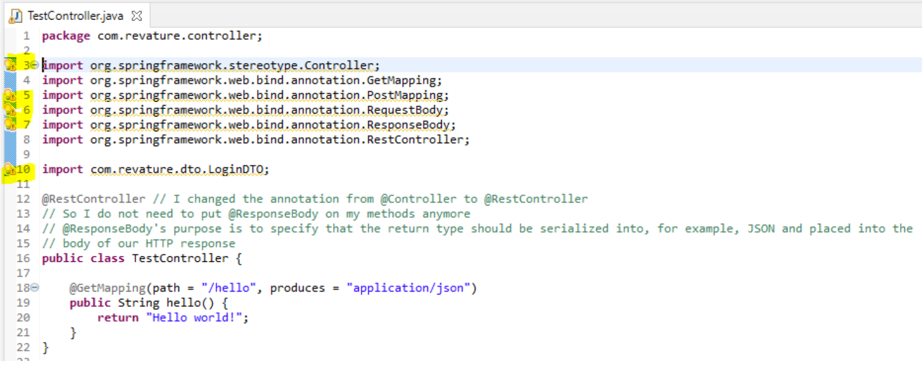
Update controller class: TestController

Remove the login post mapping method:



```
package com.revature.controller;

import org.springframework.stereotype.Controller;

@RestController // I changed the annotation from @Controller to @RestController
// So I do not need to put @ResponseBody on my methods anymore
// @ResponseBody's purpose is to specify that the return type should be serialized into, for exar
// body of our HTTP response
public class TestController {

    @GetMapping(path = "/hello", produces = "application/json")
    public String hello() {
        return "Hello world!";
    }

    @PostMapping(path = "/login", consumes = "application/json", produces = "application/json")
    public LoginDTO login(@RequestBody LoginDTO loginDto) {
        return loginDto;
    }
```

You can also remove unused imports:

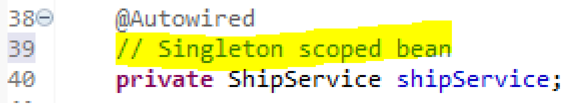


```
package com.revature.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.RestController;

import com.revature.dto.LoginDTO;

@RestController // I changed the annotation from @Controller to @RestController
// So I do not need to put @ResponseBody on my methods anymore
// @ResponseBody's purpose is to specify that the return type should be serialized into, for example, JSON and placed into the
// body of our HTTP response
public class TestController {

    @GetMapping(path = "/hello", produces = "application/json")
    public String hello() {
        return "Hello world!";
    }
}
```

Update controller class: ShipController

Add comment to class variable:

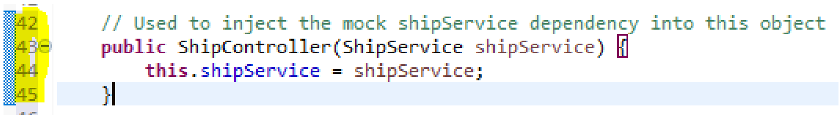
```
// Singleton scoped bean
```



```
@Autowired
// Singleton scoped bean
private ShipService shipService;
```

Add an injection constructor:

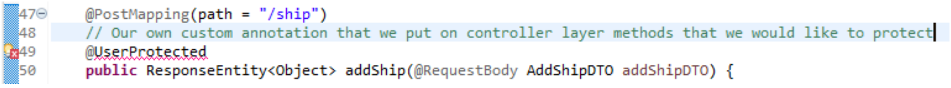
```
        // Used to inject the mock shipService dependency into this object
        public ShipController(ShipService shipService) {
                this.shipService = shipService;
        }
```



```
    // Used to inject the mock shipService dependency into this object
    public ShipController(ShipService shipService) {
        this.shipService = shipService;
    }
```

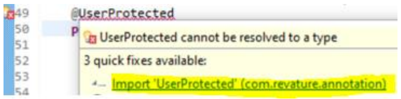
Add annotation to add ship post mapping:

```
        // Our own custom annotation that we put on controller layer methods that we would like to protect
        @UserProtected
```



```
    @PostMapping(path = "/ship")
    // Our own custom annotation that we put on controller layer methods that we would like to protect
    @UserProtected
    public ResponseEntity<Object> addShip(@RequestBody AddShipDTO addShipDTO) {
```

Correct the error:



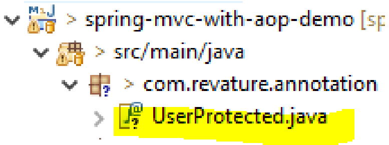
```
    @UserProtected
    p
```

UserProtected cannot be resolved to a type

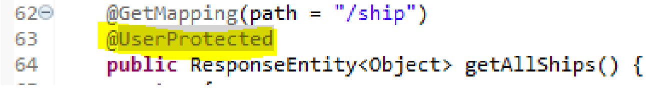
3 quick fixes available:

Import 'UserProtected' (com.revature.annotation)

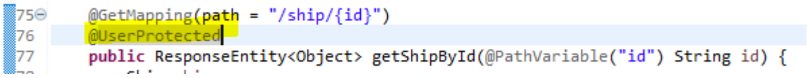
NOTE: we are importing our @interface class from our annotation package.



Add the @UserProtected to the other two methods in the class:



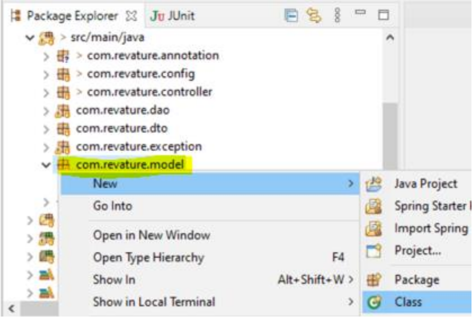
```
    @GetMapping(path = "/ship")
    @UserProtected
    public ResponseEntity<Object> getAllShips() {
```

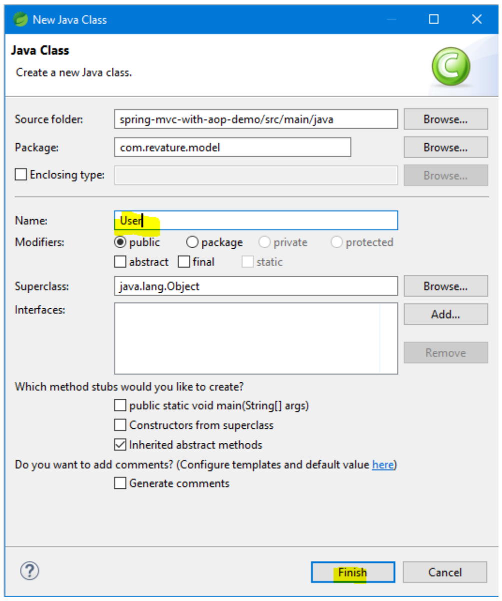


```
    @GetMapping(path = "/ship/{id}")
    @UserProtected
    public ResponseEntity<Object> getShipById(@PathVariable("id") String id) {
```

Add new Model class

Add model class: User





Source folder: spring-mvc-with-aop-demo/src/main/java

Package: com.revature.model

□ Enclosing type:

Name: User

Modifiers: ● public ○ package ○ private ○ protected

□ abstract □ final □ static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

□ public static void main(String[] args)

□ Constructors from superclass

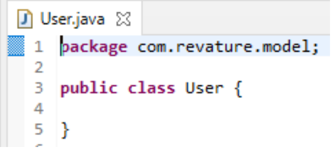
☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

□ Generate comments

Update model code shell: User

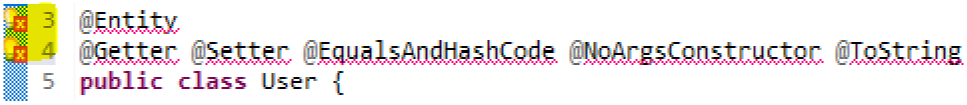
The interface wizard produces a code shell to start with:



```java
package com.revature.model;

public class User {

}
```

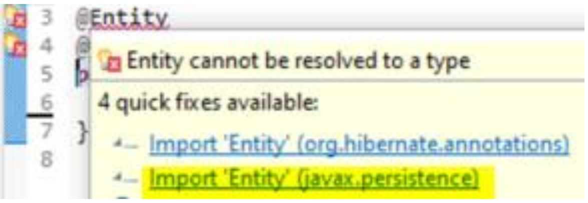
The model class User will create a new table in our database for storing user login information. It uses Hibernate annotation on the class and it variables to map to the database table and fields:

Add class level annotation for Hibernate and Lombok:



```java
@Entity
@Getter @Setter @EqualsAndHashCode @NoArgsConstructor @ToString
public class User {
```

Resolve the errors:

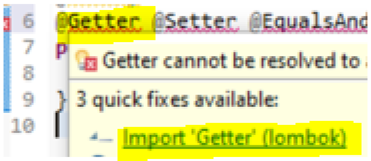


```
@Entity
```

Entity cannot be resolved to a type

4 quick fixes available:

- Import 'Entity' (org.hibernate.annotations)
- Import 'Entity' (javax.persistence)



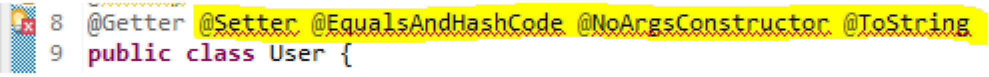
```
@Getter @Setter @EqualsAnd
```

Getter cannot be resolved to

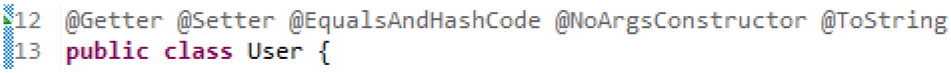
3 quick fixes available:

- Import 'Getter' (lombok)

Repeat for each the Lombok annotations.



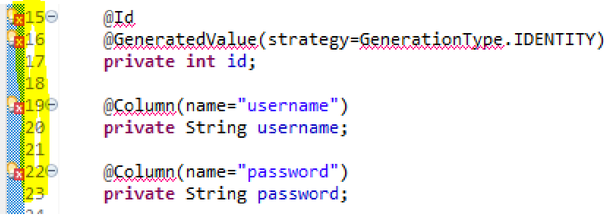
```
@Getter @Setter @EqualsAndHashCode @NoArgsConstructor @ToString
public class User {
```



```
@Getter @Setter @EqualsAndHashCode @NoArgsConstructor @ToString
public class User {
```

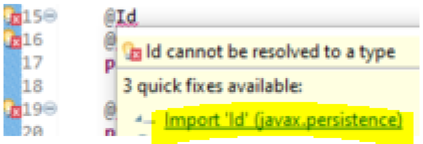
Add class variables and Hibernate annotations for database column mappings:

```
            @Id
            @GeneratedValue(strategy=GenerationType.IDENTITY)
            private int id;

            @Column(name="username")
            private String username;

            @Column(name="password")
            private String password;
```



```
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int id;

    @Column(name="username")
    private String username;

    @Column(name="password")
    private String password;
```

Resolve the errors:

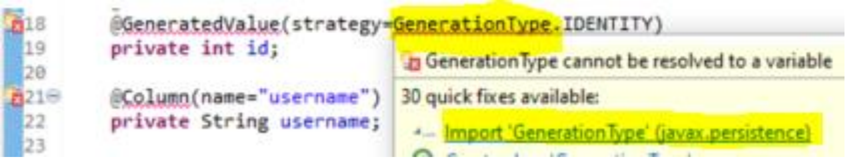


```
@Id
```

Id cannot be resolved to a type

3 quick fixes available:

- Import 'Id' (javax.persistence)

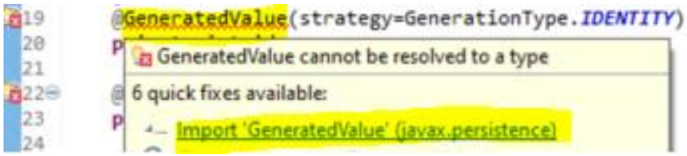


```
@GeneratedValue(strategy=GenerationType.IDENTITY)
private int id;

@Column(name="username")
private String username;
```

GenerationType cannot be resolved to a variable

30 quick fixes available:

- Import 'GenerationType' (javax.persistence)

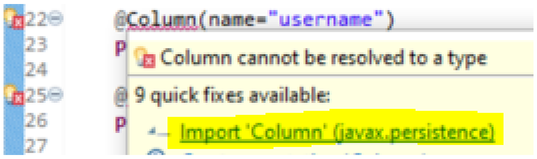


```
@GeneratedValue(strategy=GenerationType.IDENTITY)
```

GeneratedValue cannot be resolved to a type

6 quick fixes available:

Import 'GeneratedValue' (javax.persistence)



```
@Column(name="username")
```

Column cannot be resolved to a type

9 quick fixes available:

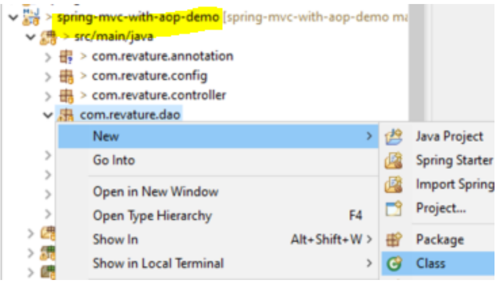
Import 'Column' (javax.persistence)

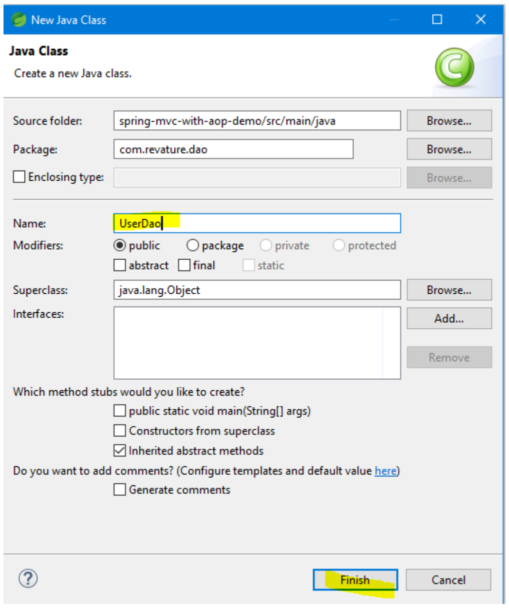
Add an constructor with arguments:

```
        public User(String username, String password) {
                this.username = username;
                this.password = password;
        }
```

Add new DAO class

Add DAO class: UserDao





Source folder: spring-mvc-with-aop-demo/src/main/java

Package: com.revature.dao

□ Enclosing type:

Name: UserDao

Modifiers: ● public ○ package ○ private ○ protected

□ abstract □ final □ static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

□ public static void main(String[] args)

□ Constructors from superclass

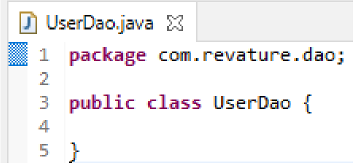
☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

□ Generate comments

Update DAO class code shell: UserDao

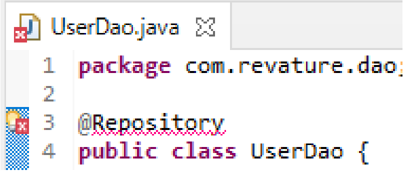
The class wizard produces a code shell to start with:



```
package com.revature.dao;

public class UserDao {

}
```

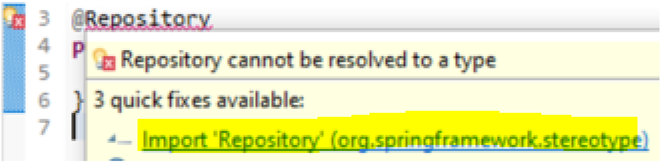
Since this is a DAO class interacting with the database we add the `@Repository` annotation at the class level:

```
@Repository
```



```
package com.revature.dao;

@Repository
public class UserDao {
```

Correct the error:



```
@Repository
p
}
```

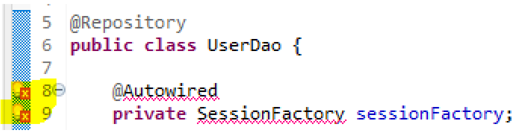
Repository cannot be resolved to a type

3 quick fixes available:

Import 'Repository' (org.springframework.stereotype)

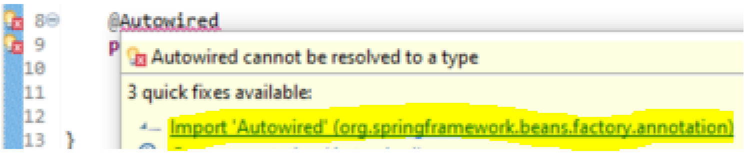
Add the class autowired variable:

```
        @Autowired
        private SessionFactory sessionFactory;
```



```
@Repository
public class UserDao {

    @Autowired
    private SessionFactory sessionFactory;
```

Resolves the error:

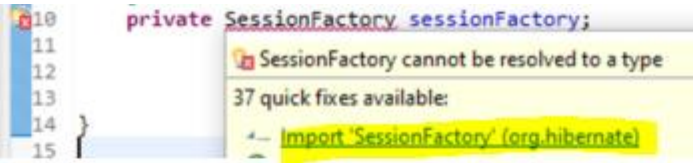


```
    @Autowired
    p
}
```

Autowired cannot be resolved to a type

3 quick fixes available:

Import 'Autowired' (org.springframework.beans.factory.annotation)



```
    private SessionFactory sessionFactory;

}
```

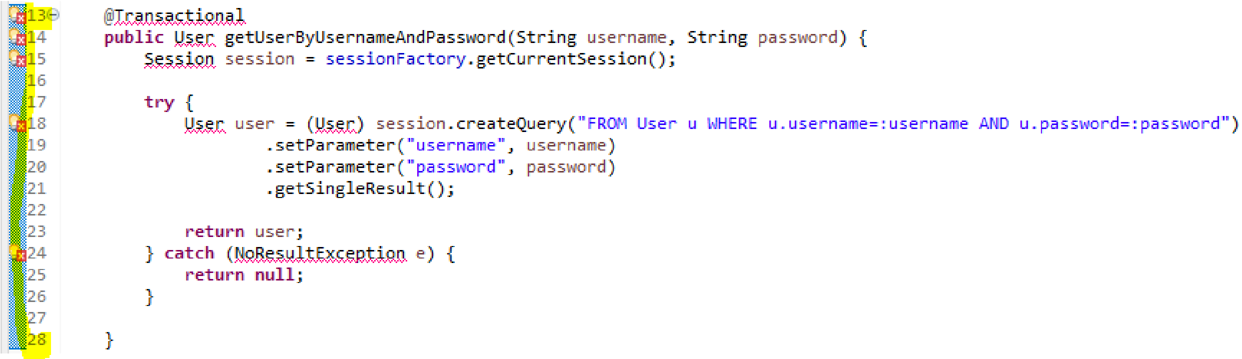
SessionFactory cannot be resolved to a type

37 quick fixes available:

Import 'SessionFactory' (org.hibernate)

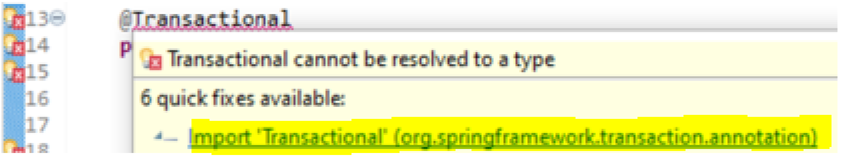
Add an @Transactional method: getUserByUsernameAndPassword()

```
        @Transactional
        public User getUserByUsernameAndPassword(String username, String password) {
                Session session = sessionFactory.getCurrentSession();

                try {
                        User user = (User) session.createQuery("FROM User u WHERE
u.username=:username AND u.password=:password")
                                        .setParameter("username", username)
                                        .setParameter("password", password)
                                        .getSingleResult();

                        return user;
                } catch (NoResultException e) {
                        return null;
                }

        }
```



```
@Transactional
public User getUserByUsernameAndPassword(String username, String password) {
    Session session = sessionFactory.getCurrentSession();

    try {
        User user = (User) session.createQuery("FROM User u WHERE u.username=:username AND u.password=:password")
                .setParameter("username", username)
                .setParameter("password", password)
                .getSingleResult();

        return user;
    } catch (NoResultException e) {
        return null;
    }

}
```

Resolve the errors:

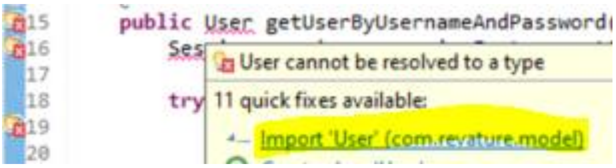


```
@Transactional
```

Transactional cannot be resolved to a type

6 quick fixes available:

Import 'Transactional' (org.springframework.transaction.annotation)

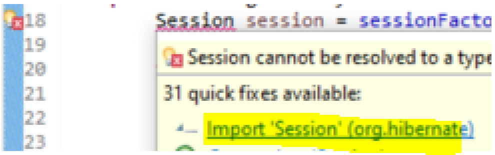


```
public User getUserByUsernameAndPassword(
```

User cannot be resolved to a type

11 quick fixes available:

Import 'User' (com.revature.model)

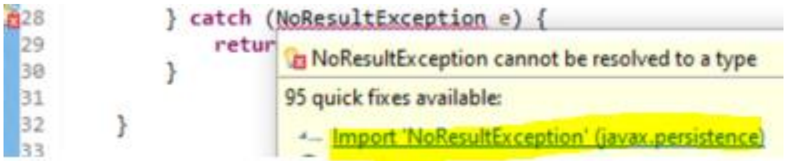


```
Session session = sessionFacto
```

Session cannot be resolved to a type

31 quick fixes available:

Import 'Session' (org.hibernate)



```
} catch (NoResultException e) {
```

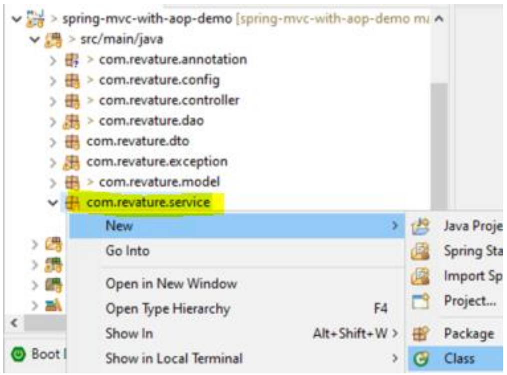
NoResultException cannot be resolved to a type

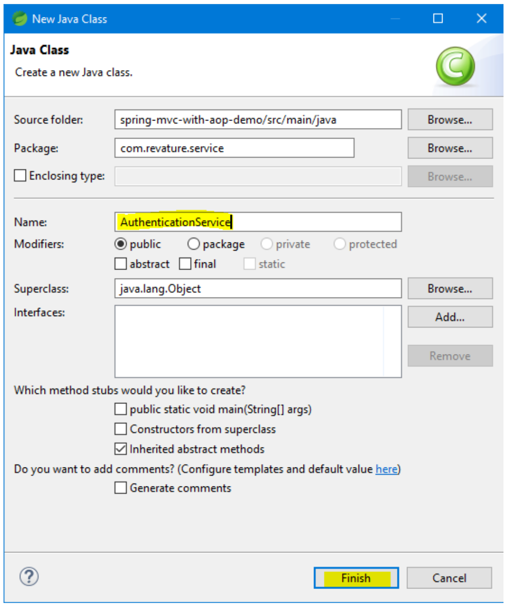
95 quick fixes available:

Import 'NoResultException' (javax.persistence)

Add new Service class

Add service class: AuthenticationService





Java Class

Create a new Java class.

Source folder: spring-mvc-with-aop-demo/src/main/java

Package: com.revature.service

☐ Enclosing type:

Name: AuthenticationService

Modifiers: ● public ○ package ○ private ○ protected

☐ abstract ☐ final ☐ static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Constructors from superclass

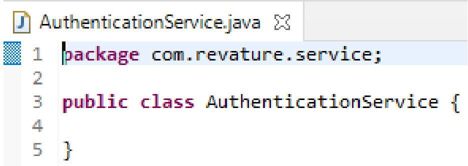
☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

☐ Generate comments

Update service class code shell: AuthenticationService

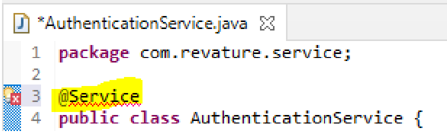
The class wizard produces a code shell to start with:



```
package com.revature.service;

public class AuthenticationService {

}
```

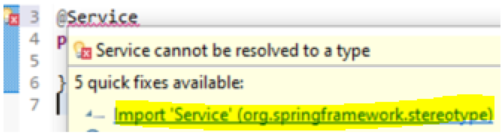
Add the @Service annotation that identifies this class as a service:

```
@Service
```



```
package com.revature.service;

@Service
public class AuthenticationService {
```

Correct the error:



```
@Service
p
}
```

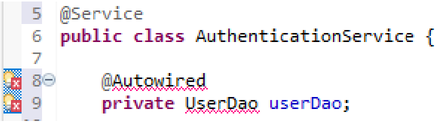
Service cannot be resolved to a type

5 quick fixes available:

Import 'Service' (org.springframework.stereotype)

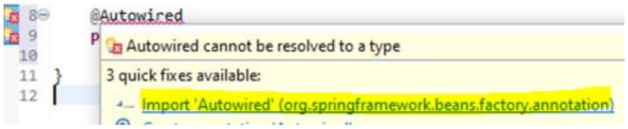
Add the class autowired variable:

```
        @Autowired
        private UserDao userDao;
```



```
@Service
public class AuthenticationService {

    @Autowired
    private UserDao userDao;
```

Resolves the error:

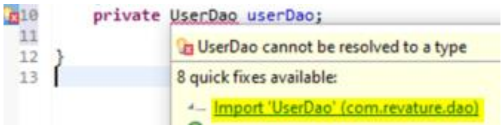


```
@Autowired
p
}
```

Autowired cannot be resolved to a type

3 quick fixes available:

Import 'Autowired' (org.springframework.beans.factory.annotation)



```
    private UserDao userDao;
}
```

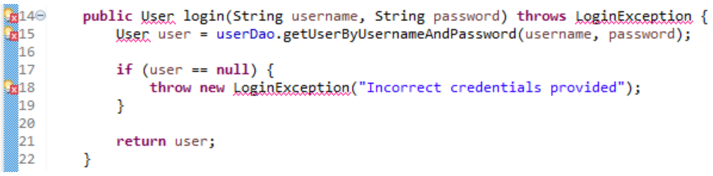
UserDao cannot be resolved to a type

8 quick fixes available:

Import 'UserDao' (com.revature.dao)

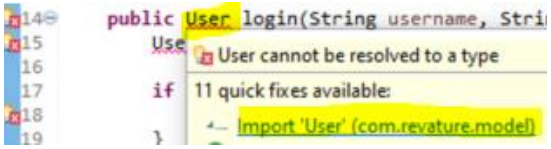
Add the following service method: login()

```
        public User login(String username, String password) throws LoginException {
                User user = userDao.getUserByUsernameAndPassword(username, password);

                if (user == null) {
                        throw new LoginException("Incorrect credentials provided");
                }

                return user;
        }
```



```
public User login(String username, String password) throws LoginException {
    User user = userDao.getUserByUsernameAndPassword(username, password);

    if (user == null) {
        throw new LoginException("Incorrect credentials provided");
    }

    return user;
}
```

Resolve the errors:

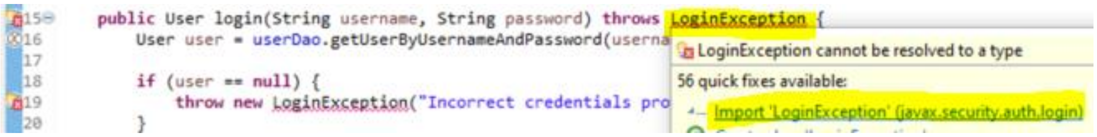


```
public User login(String username, Stri
    Use
    if
    }
```

User cannot be resolved to a type

11 quick fixes available:

Import 'User' (com.revature.model)



```
public User login(String username, String password) throws LoginException {
    User user = userDao.getUserByUsernameAndPassword(userna

    if (user == null) {
        throw new LoginException("Incorrect credentials pro
    }
```

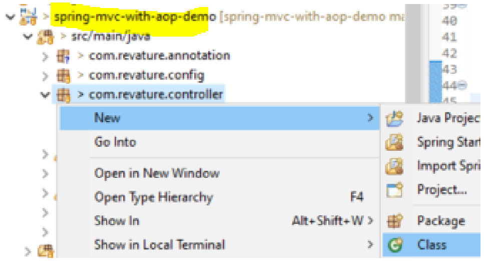
LoginException cannot be resolved to a type

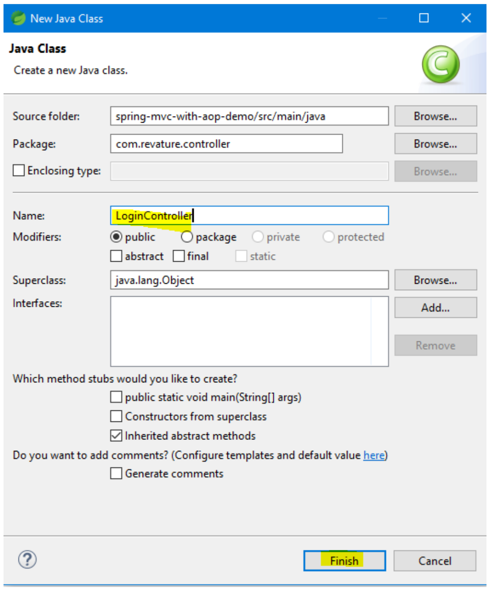
56 quick fixes available:

Import 'LoginException' (javax.security.auth.login)

Add new Controller class

Add new controller class: LoginController





Java Class

Create a new Java class.

Source folder: spring-mvc-with-aop-demo/src/main/java

Package: com.revature.controller

□ Enclosing type:

Name: LoginController

Modifiers: ● public ○ package ○ private ○ protected

□ abstract □ final □ static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

□ public static void main(String[] args)

□ Constructors from superclass

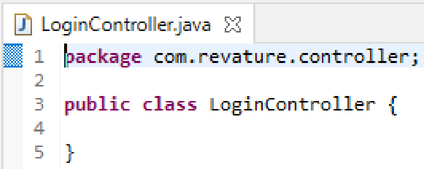
☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

□ Generate comments

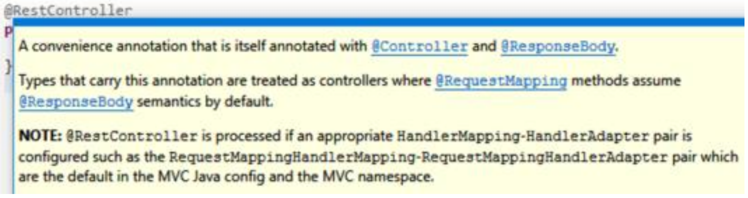
Update controller class code shell: LoginController

The class wizard produces a code shell to start with:



```java
package com.revature.controller;

public class LoginController {

}
```

This is a controller class so we add the @RestController annotation:

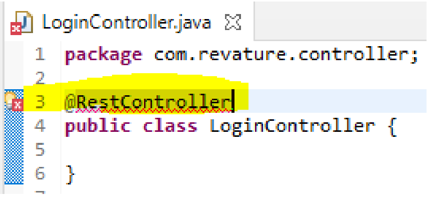


```java
@RestController
```

A convenience annotation that is itself annotated with @Controller and @ResponseBody.

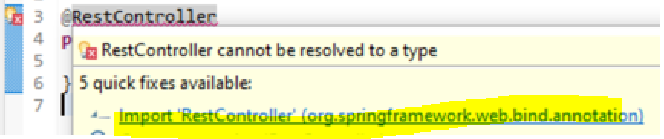
Types that carry this annotation are treated as controllers where @RequestMapping methods assume @ResponseBody semantics by default.

NOTE: @RestController is processed if an appropriate HandlerMapping-HandlerAdapter pair is configured such as the RequestMappingHandlerMapping-RequestMappingHandlerAdapter pair which are the default in the MVC Java config and the MVC namespace.



```
package com.revature.controller;

@RestController
public class LoginController {

}
```

Correct the error:



```
@RestController
```

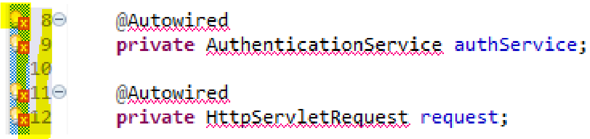
RestController cannot be resolved to a type

5 quick fixes available:

Import 'RestController' (org.springframework.web.bind.annotation)

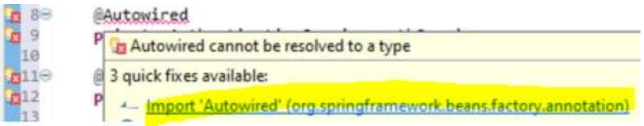
Add class autowired variables:

```
	@Autowired
	private AuthenticationService authService;

	@Autowired
	private HttpServletRequest request;
```



```
	@Autowired
	private AuthenticationService authService;

	@Autowired
	private HttpServletRequest request;
```

Resolve the errors:

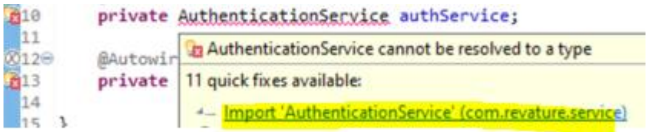


```
	@Autowired
```

Autowired cannot be resolved to a type

3 quick fixes available:

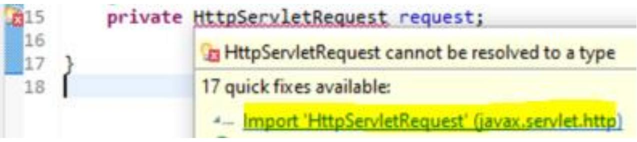
Import 'Autowired' (org.springframework.beans.factory.annotation)



```
	private AuthenticationService authService;

	@Autowir
	private
```

AuthenticationService cannot be resolved to a type

11 quick fixes available:



```
	private HttpServletRequest request;

}
```

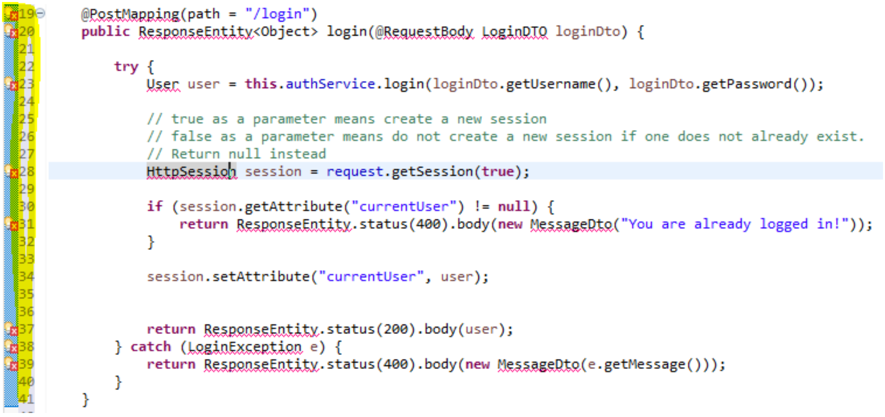
HttpServletRequest cannot be resolved to a type

17 quick fixes available:

Add a post mapping method to login: login()

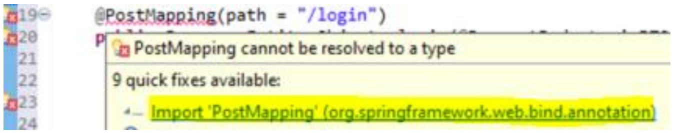
```
	@PostMapping(path = "/login")
	public ResponseEntity<Object> login(@RequestBody LoginDTO loginDto) {

		try {
			User user = this.authService.login(loginDto.getUsername(), loginDto.getPassword());

			// true as a parameter means create a new session
			// false as a parameter means do not create a new session if one does not already exist.
			// Return null instead
			HttpSession session = request.getSession(true);

			if (session.getAttribute("currentUser") != null) {
				return ResponseEntity.status(400).body(new MessageDto("You are already logged in!"));
			}

			session.setAttribute("currentUser", user);
```

```
                        return ResponseEntity.status(200).body(user);
                } catch (LoginException e) {
                        return ResponseEntity.status(400).body(new MessageDto(e.getMessage()));
                }
        }
```



```
    @PostMapping(path = "/login")
    public ResponseEntity<Object> login(@RequestBody LoginDTO loginDto) {

        try {
            User user = this.authService.login(loginDto.getUsername(), loginDto.getPassword());

            // true as a parameter means create a new session
            // false as a parameter means do not create a new session if one does not already exist.
            // Return null instead
            HttpSession session = request.getSession(true);

            if (session.getAttribute("currentUser") != null) {
                return ResponseEntity.status(400).body(new MessageDto("You are already logged in!"));
            }

            session.setAttribute("currentUser", user);


            return ResponseEntity.status(200).body(user);
        } catch (LoginException e) {
            return ResponseEntity.status(400).body(new MessageDto(e.getMessage()));
        }
    }
```

Resolve the errors:

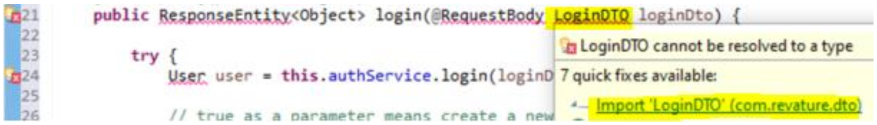


```
@PostMapping(path = "/login")
```

PostMapping cannot be resolved to a type

9 quick fixes available:

Import 'PostMapping' (org.springframework.web.bind.annotation)

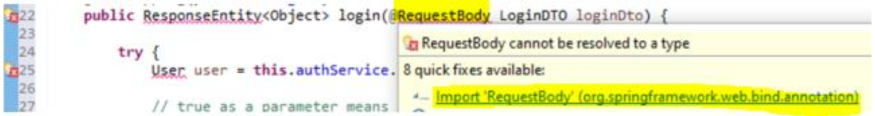


```
public ResponseEntity<Object> login(@RequestBody LoginDTO loginDto) {

    try {
        User user = this.authService.login(loginD

        // true as a parameter means create a new
```

LoginDTO cannot be resolved to a type

7 quick fixes available:

Import 'LoginDTO' (com.revature.dto)

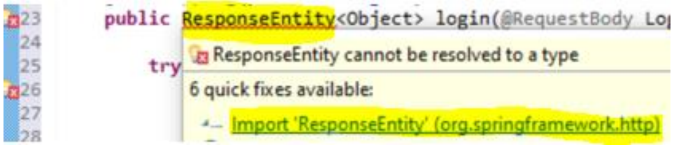


```
public ResponseEntity<Object> login(@RequestBody LoginDTO loginDto) {

    try {
        User user = this.authService.

        // true as a parameter means
```

RequestBody cannot be resolved to a type

8 quick fixes available:

Import 'RequestBody' (org.springframework.web.bind.annotation)

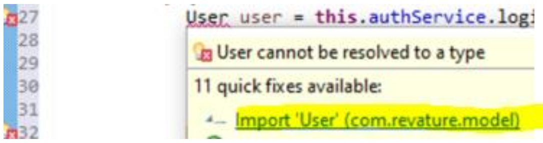


```
public ResponseEntity<Object> login(@RequestBody Lo

    try
```

ResponseEntity cannot be resolved to a type

6 quick fixes available:

Import 'ResponseEntity' (org.springframework.http)

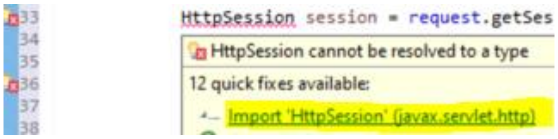


```
User user = this.authService.logi
```

User cannot be resolved to a type

11 quick fixes available:

Import 'User' (com.revature.model)

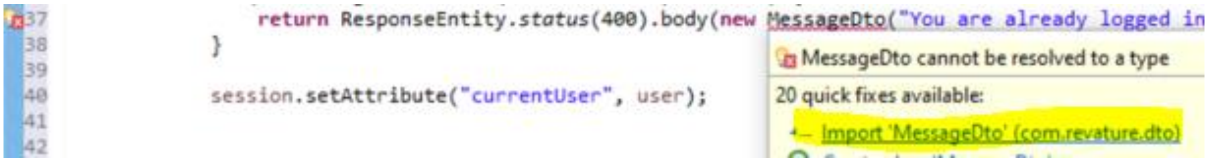


```
HttpSession session = request.getSes
```

HttpSession cannot be resolved to a type

12 quick fixes available:

Import 'HttpSession' (javax.servlet.http)

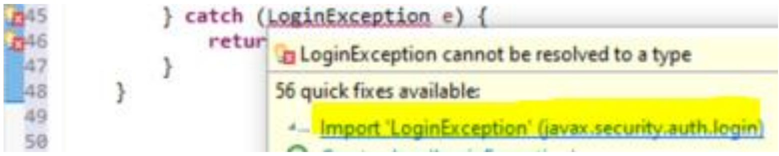


```
            return ResponseEntity.status(400).body(new MessageDto("You are already logged in
        }

        session.setAttribute("currentUser", user);
```

MessageDto cannot be resolved to a type

20 quick fixes available:

Import 'MessageDto' (com.revature.dto)



```
        } catch (LoginException e) {
            retur
        }
    }
```

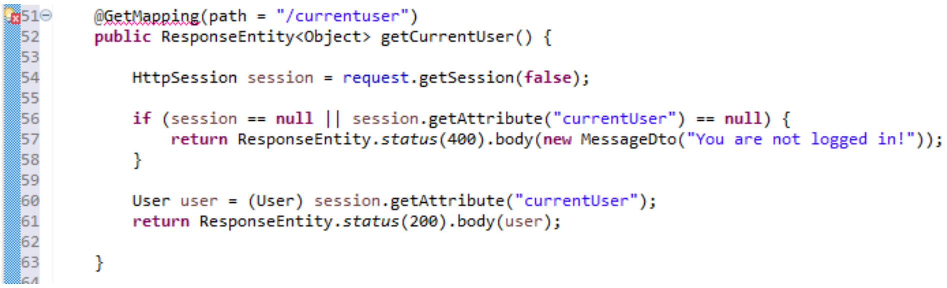
LoginException cannot be resolved to a type

56 quick fixes available:

Import 'LoginException' (javax.security.auth.login)

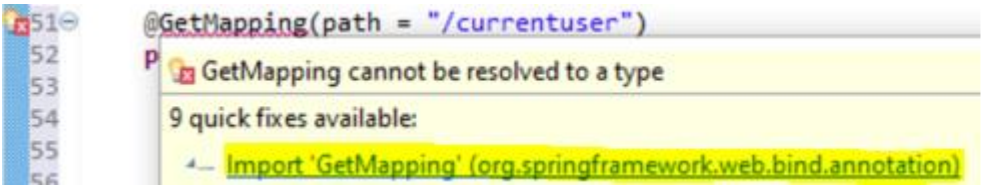
Add a get mapping method to retrieve current user: getCurrentUser()

```
	@GetMapping(path = "/currentuser")
	public ResponseEntity<Object> getCurrentUser() {

		HttpSession session = request.getSession(false);

		if (session == null || session.getAttribute("currentUser") == null) {
			return ResponseEntity.status(400).body(new MessageDto("You are not logged in!"));
		}

		User user = (User) session.getAttribute("currentUser");
		return ResponseEntity.status(200).body(user);

	}
```



```
    @GetMapping(path = "/currentuser")
    public ResponseEntity<Object> getCurrentUser() {

        HttpSession session = request.getSession(false);

        if (session == null || session.getAttribute("currentUser") == null) {
            return ResponseEntity.status(400).body(new MessageDto("You are not logged in!"));
        }

        User user = (User) session.getAttribute("currentUser");
        return ResponseEntity.status(200).body(user);

    }
```

Correct the error:



```
    @GetMapping(path = "/currentuser")
    p
```

GetMapping cannot be resolved to a type

9 quick fixes available:

Import 'GetMapping' (org.springframework.web.bind.annotation)

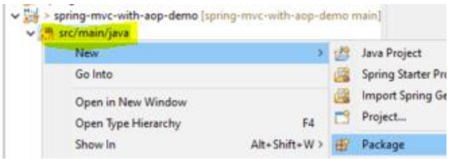
Add Aspect Related Project Components

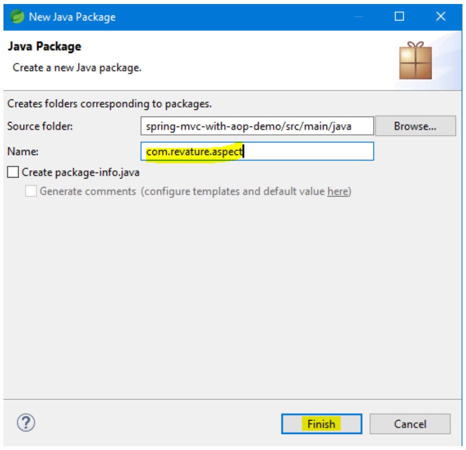
With respect with AOP we will break down program logic into cross-cutting-concerns. For this demo we break down for two concerns, logging and security.

Breaking down concerns into units are known as “Aspects”. It is a class that contains different “Advice” structured as methods. The class itself carries the annotation @Aspect to address a particular concern.

Add new aspect package and classes

Add package: com.revature.aspect





Java Package

Create a new Java package.

Creates folders corresponding to packages.

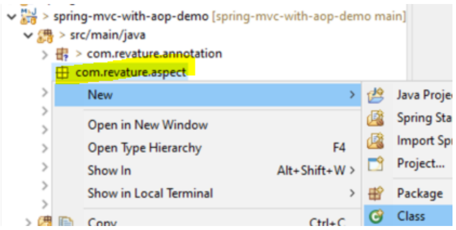
Source folder: spring-mvc-with-aop-demo/src/main/java

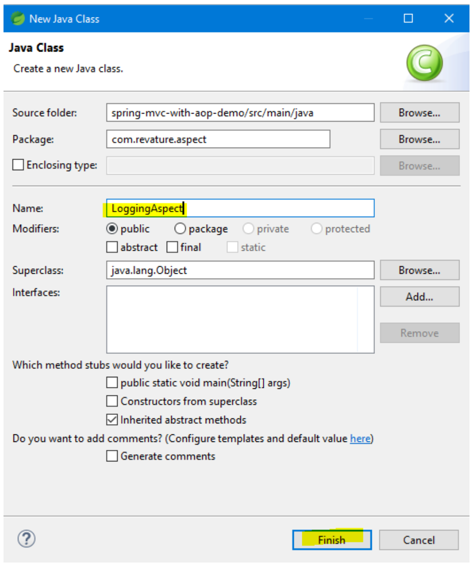
Name: com.revature.aspect

☐ Create package-info.java

☐ Generate comments (configure templates and default value here)

Add aspect class: LoggingAspect





New Java Class

Java Class

Create a new Java class.

Source folder: spring-mvc-with-aop-demo/src/main/java

Package: com.revature.aspect

□ Enclosing type:

Name: LoggingAspect

Modifiers: ● public ○ package ○ private ○ protected

□ abstract □ final □ static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

□ public static void main(String[] args)

□ Constructors from superclass

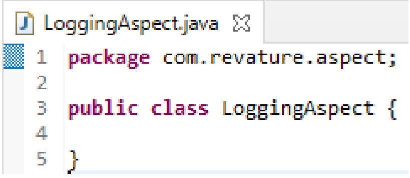
☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

□ Generate comments

Update aspect class code shell: LoggingAspect

The code wizard produces a code shell to start with:

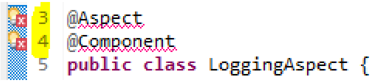


LoggingAspect.java

```
package com.revature.aspect;

public class LoggingAspect {

}
```

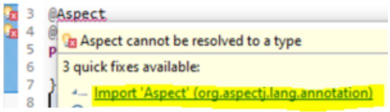
Add class annotations:

```
@Aspect
@Component
```



```
@Aspect
@Component
public class LoggingAspect {
```

Resolve the errors:

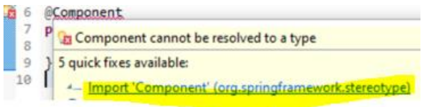


@Aspect

Aspect cannot be resolved to a type

3 quick fixes available:

Import 'Aspect' (org.aspectj.lang.annotation)



@Component

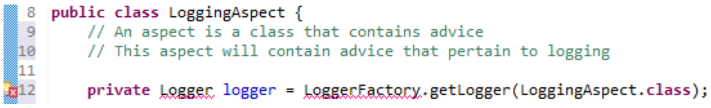
Component cannot be resolved to a type

5 quick fixes available:

Import 'Component' (org.springframework.stereotype)

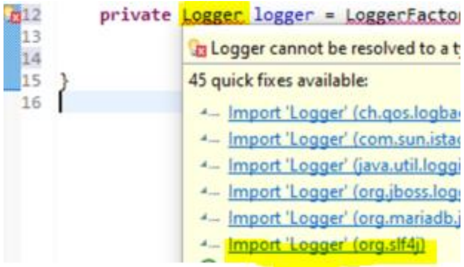
Add class level comments and variable:

```
        // An aspect is a class that contains advice
        // This aspect will contain advice that pertain to logging

        private Logger logger = LoggerFactory.getLogger(LoggingAspect.class);
```



```
public class LoggingAspect {
    // An aspect is a class that contains advice
    // This aspect will contain advice that pertain to logging

    private Logger logger = LoggerFactory.getLogger(LoggingAspect.class);
```

Resolve the errors:

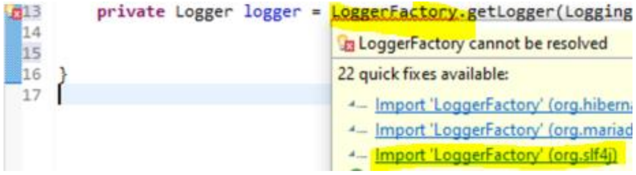


```
    private Logger logger = LoggerFacto

}
```

Logger cannot be resolved to a t

45 quick fixes available:

- Import 'Logger' (ch.qos.logba
- Import 'Logger' (com.sun.ista
- Import 'Logger' (java.util.loggi
- Import 'Logger' (org.jboss.log
- Import 'Logger' (org.mariadb.j
- Import 'Logger' (org.slf4j)



```
    private Logger logger = LoggerFactory.getLogger(Logging

}
```

LoggerFactory cannot be resolved

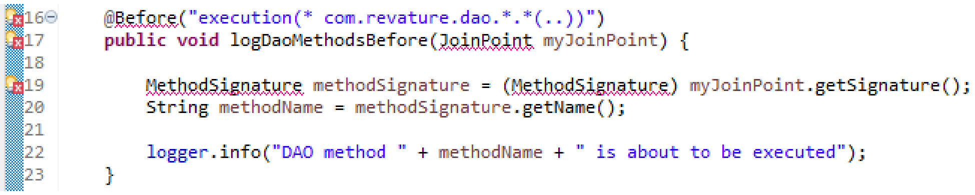
22 quick fixes available:

- Import 'LoggerFactory' (org.hibern
- Import 'LoggerFactory' (org.mariad
- Import 'LoggerFactory' (org.slf4j)

Add @Before annotation method: logDaoMethodsBefore()

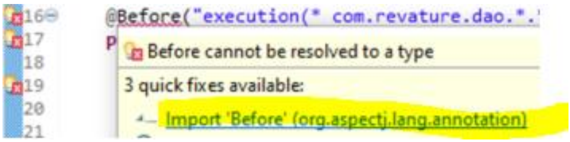
@Before - Advice that will execute before the JoinPoint that is to be intercepted

```
        @Before("execution(* com.revature.dao.*.*(..))")
        public void logDaoMethodsBefore(JoinPoint myJoinPoint) {

                MethodSignature methodSignature = (MethodSignature) myJoinPoint.getSignature();
                String methodName = methodSignature.getName();

                logger.info("DAO method " + methodName + " is about to be executed");
        }
```



```
    @Before("execution(* com.revature.dao.*.*(..))")
    public void logDaoMethodsBefore(JoinPoint myJoinPoint) {

        MethodSignature methodSignature = (MethodSignature) myJoinPoint.getSignature();
        String methodName = methodSignature.getName();

        logger.info("DAO method " + methodName + " is about to be executed");
    }
```

Resolve the errors:

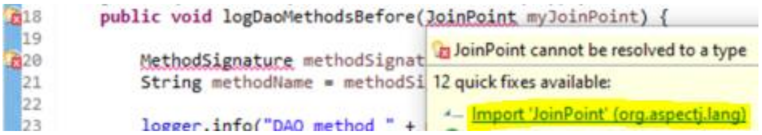


```
    @Before("execution(* com.revature.dao.*.'
    p
```

Before cannot be resolved to a type

3 quick fixes available:

- Import 'Before' (org.aspectj.lang.annotation)

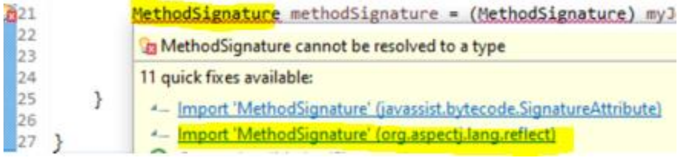


```
    public void logDaoMethodsBefore(JoinPoint myJoinPoint) {

        MethodSignature methodSignat
        String methodName = methodSi

        logger.info("DAO method " +
```

JoinPoint cannot be resolved to a type

12 quick fixes available:

- Import 'JoinPoint' (org.aspectj.lang)



```
        MethodSignature methodSignature = (MethodSignature) myJ




    }
}
```

MethodSignature cannot be resolved to a type

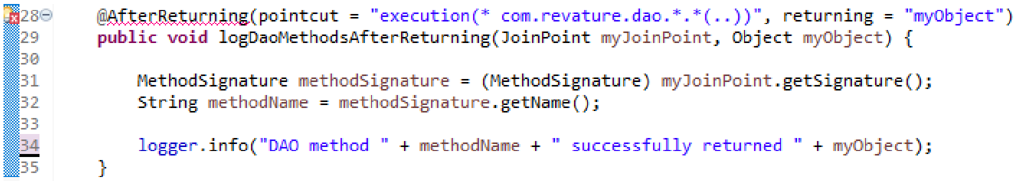
11 quick fixes available:

- Import 'MethodSignature' (javassist.bytecode.SignatureAttribute)
- Import 'MethodSignature' (org.aspectj.lang.reflect)

Add @AfterReturning annotation method: logDaoMethodsAfterReturning()

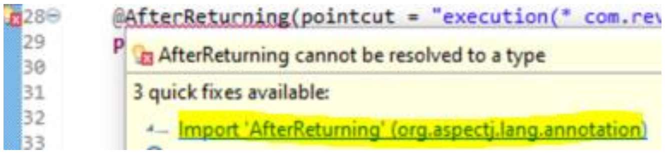
@AfterReturning - Advice that will execute after a method returns successfully

```
	@AfterReturning(pointcut = "execution(* com.revature.dao.*.*(..))", returning = "myObject")
	public void logDaoMethodsAfterReturning(JoinPoint myJoinPoint, Object myObject) {

		MethodSignature methodSignature = (MethodSignature) myJoinPoint.getSignature();
		String methodName = methodSignature.getName();

		logger.info("DAO method " + methodName + " successfully returned " + myObject);

	}
```



```
    @AfterReturning(pointcut = "execution(* com.revature.dao.*.*(..))", returning = "myObject")
    public void logDaoMethodsAfterReturning(JoinPoint myJoinPoint, Object myObject) {

        MethodSignature methodSignature = (MethodSignature) myJoinPoint.getSignature();
        String methodName = methodSignature.getName();

        logger.info("DAO method " + methodName + " successfully returned " + myObject);
    }
```

Correct the error:



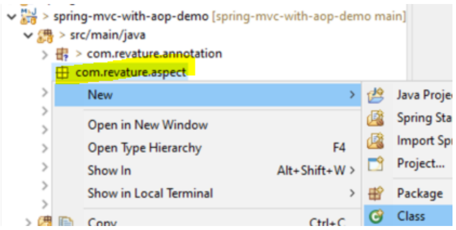
```
    @AfterReturning(pointcut = "execution(* com.rev
```

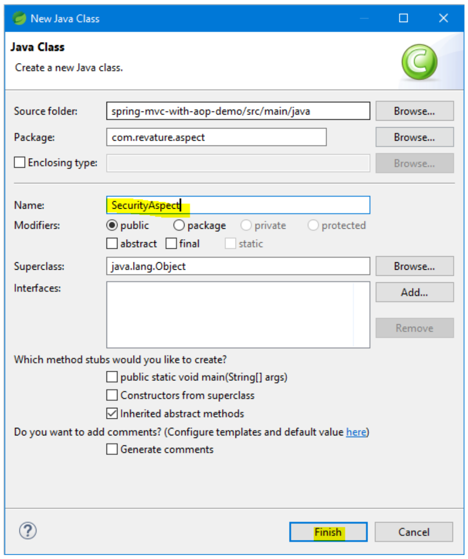
AfterReturning cannot be resolved to a type

3 quick fixes available:

Import 'AfterReturning' (org.aspectj.lang.annotation)

Add aspect class: SecurityAspect





New Java Class

Java Class

Create a new Java class.

Source folder: spring-mvc-with-aop-demo/src/main/java

Package: com.revature.aspect

□ Enclosing type:

Name: SecurityAspect

Modifiers: ● public ○ package ○ private ○ protected

□ abstract □ final □ static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

□ public static void main(String[] args)

□ Constructors from superclass

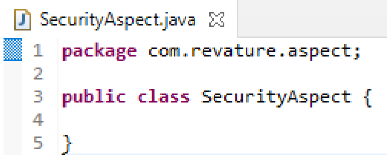
☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

□ Generate comments

Update aspect class code shell: SecurityAspect

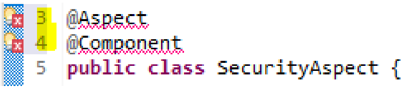
The code wizard produces a code shell to start with:



```
package com.revature.aspect;

public class SecurityAspect {

}
```

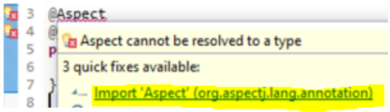
Add class annotations:

```
@Aspect
@Component
```



```
@Aspect
@Component
public class SecurityAspect {
```

Resolve the errors:

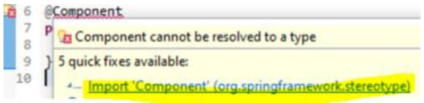


```
@Aspect
```

Aspect cannot be resolved to a type

3 quick fixes available:

Import 'Aspect' (org.aspectj.lang.annotation)



```
@Component
```

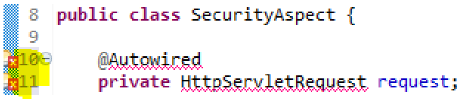
Component cannot be resolved to a type

5 quick fixes available:

Import 'Component' (org.springframework.stereotype)

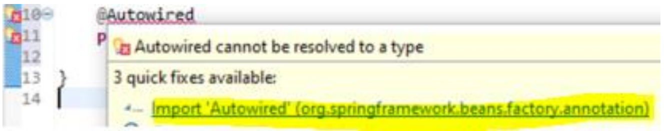
Add class level autowired variable:

```
	@Autowired
	private HttpServletRequest request;
```



```
public class SecurityAspect {

	@Autowired
	private HttpServletRequest request;
```

Resolve the errors:

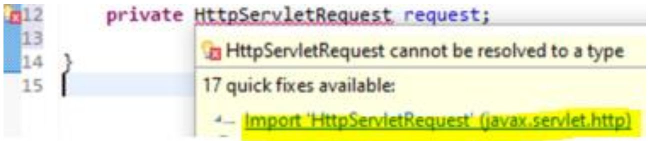


```
	@Autowired
```

Autowired cannot be resolved to a type

3 quick fixes available:

Import 'Autowired' (org.springframework.beans.factory.annotation)



```
	private HttpServletRequest request;
```

HttpServletRequest cannot be resolved to a type

17 quick fixes available:

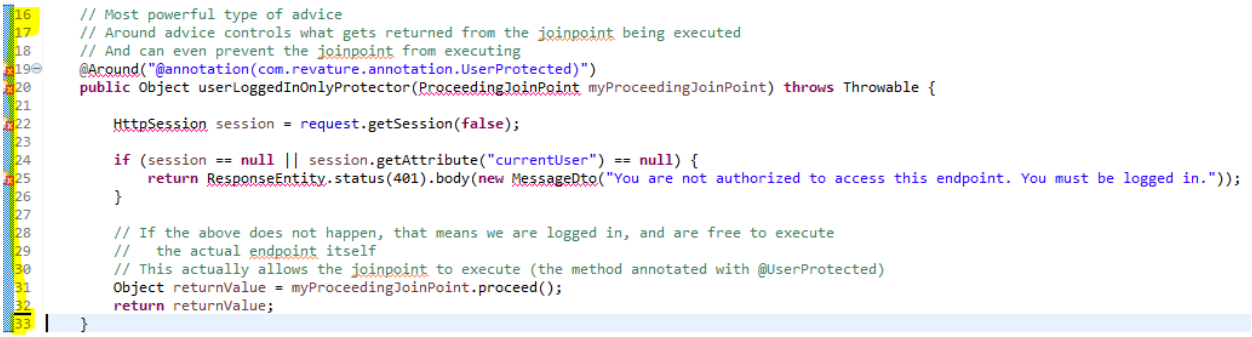
Import 'HttpServletRequest' (javax.servlet.http)

Add @Around annotation method: userLoggedInOnlyProtector()

@Around- Allows for this advice to intercept a method both before and after. Most powerful type of advice, and can do things like stopping the execution of the joinpoint method, stopping an exception from propagating, etc.

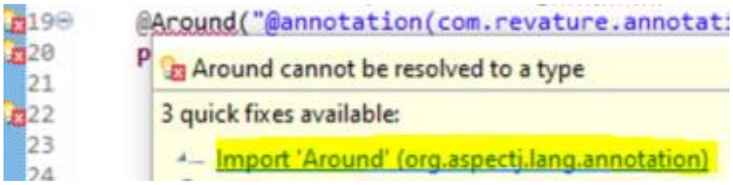
```
	// Most powerful type of advice
	// Around advice controls what gets returned from the joinpoint being executed
	// And can even prevent the joinpoint from executing
	@Around("@annotation(com.revature.annotation.UserProtected)")
	public Object userLoggedInOnlyProtector(ProceedingJoinPoint myProceedingJoinPoint) throws Throwable {
```

```
                    HttpSession session = request.getSession(false);

                    if (session == null || session.getAttribute("currentUser") == null) {
                              return ResponseEntity.status(401).body(new MessageDto("You are not authorized to access this endpoint.
You must be logged in."));
                    }

                    // If the above does not happen, that means we are logged in, and are free to execute
                    //   the actual endpoint itself
                    // This actually allows the joinpoint to execute (the method annotated with @UserProtected)
                    Object returnValue = myProceedingJoinPoint.proceed();
                    return returnValue;
          }
```



```
    // Most powerful type of advice
    // Around advice controls what gets returned from the joinpoint being executed
    // And can even prevent the joinpoint from executing
    @Around("@annotation(com.revature.annotation.UserProtected)")
    public Object userLoggedInOnlyProtector(ProceedingJoinPoint myProceedingJoinPoint) throws Throwable {

        HttpSession session = request.getSession(false);

        if (session == null || session.getAttribute("currentUser") == null) {
            return ResponseEntity.status(401).body(new MessageDto("You are not authorized to access this endpoint. You must be logged in."));
        }

        // If the above does not happen, that means we are logged in, and are free to execute
        //   the actual endpoint itself
        // This actually allows the joinpoint to execute (the method annotated with @UserProtected)
        Object returnValue = myProceedingJoinPoint.proceed();
        return returnValue;
    }
```

Resolve the errors:

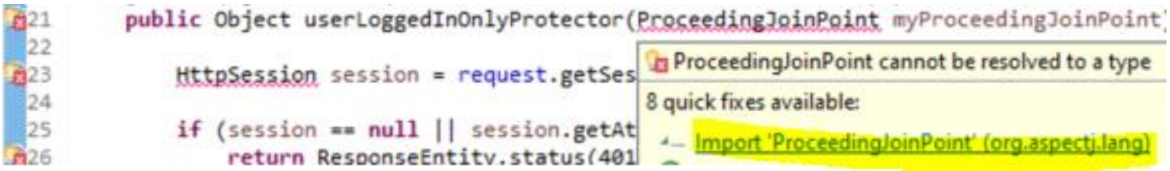


```
    @Around("@annotation(com.revature.annotat
    p
```

Around cannot be resolved to a type

3 quick fixes available:

Import 'Around' (org.aspectj.lang.annotation)

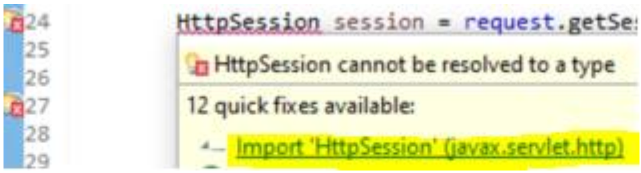


```
    public Object userLoggedInOnlyProtector(ProceedingJoinPoint myProceedingJoinPoint

        HttpSession session = request.getSes

        if (session == null || session.getAt
            return ResponseEntity.status(401
```

ProceedingJoinPoint cannot be resolved to a type

8 quick fixes available:

Import 'ProceedingJoinPoint' (org.aspectj.lang)

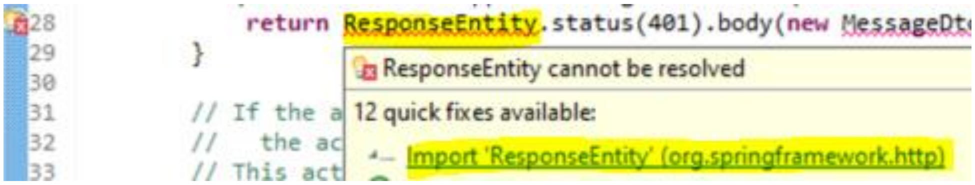


```
        HttpSession session = request.getSe
```

HttpSession cannot be resolved to a type

12 quick fixes available:

Import 'HttpSession' (javax.servlet.http)

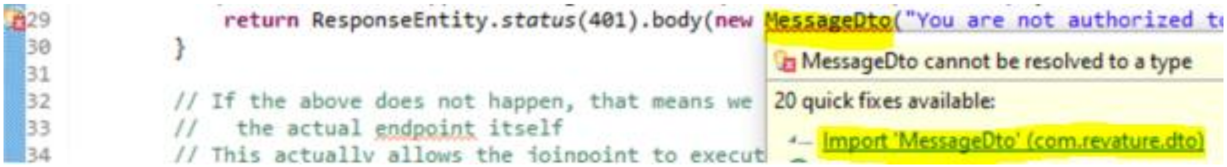


```
            return ResponseEntity.status(401).body(new MessageDt
        }

        // If the a
        //   the ac
        // This act
```

ResponseEntity cannot be resolved

12 quick fixes available:

Import 'ResponseEntity' (org.springframework.http)



```
            return ResponseEntity.status(401).body(new MessageDto("You are not authorized t
        }

        // If the above does not happen, that means we
        //   the actual endpoint itself
        // This actually allows the joinpoint to execut
```

MessageDto cannot be resolved to a type

20 quick fixes available:

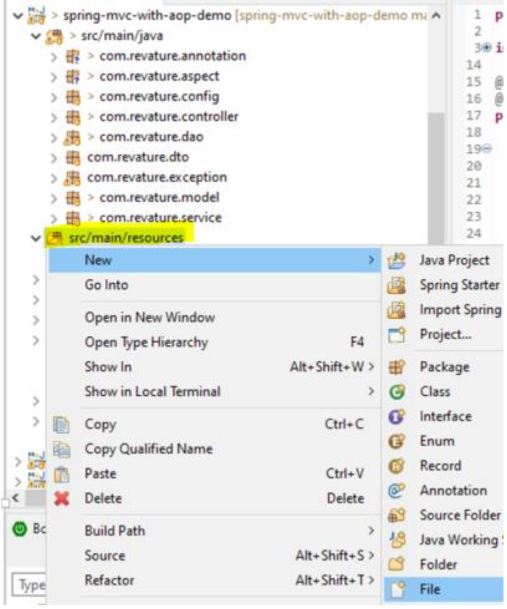
Import 'MessageDto' (com.revature.dto)

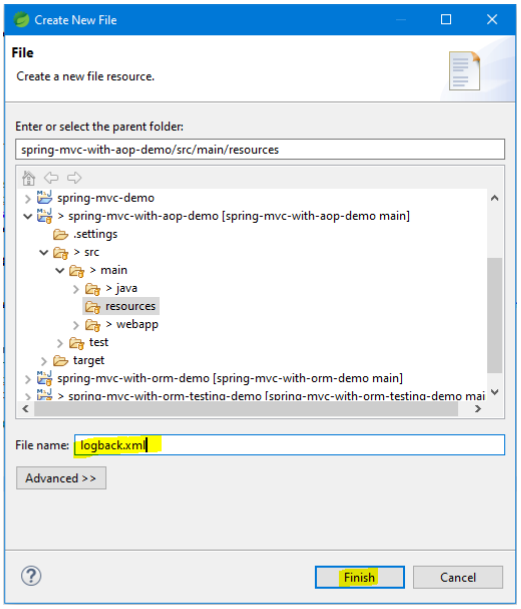
Add logging properties file

In a previous section we added a logging aspect class (LoggingAspect). This class uses logback the next generation of log4j. In this section we add a property file for logback.

Create a logback properties file: logback.xml

Create a resource file: logback.xml




Create New File
File
Create a new file resource.
spring-mvc-demo
spring-mvc-with-aop-demo [spring-mvc-with-aop-demo main]
.settings
src
main
java
resources
webapp
test
target
spring-mvc-with-orm-demo [spring-mvc-with-orm-demo main]
spring-mvc-with-orm-testing-demo [spring-mvc-with-orm-testing-demo mai
Advanced >>
Finish
Cancel

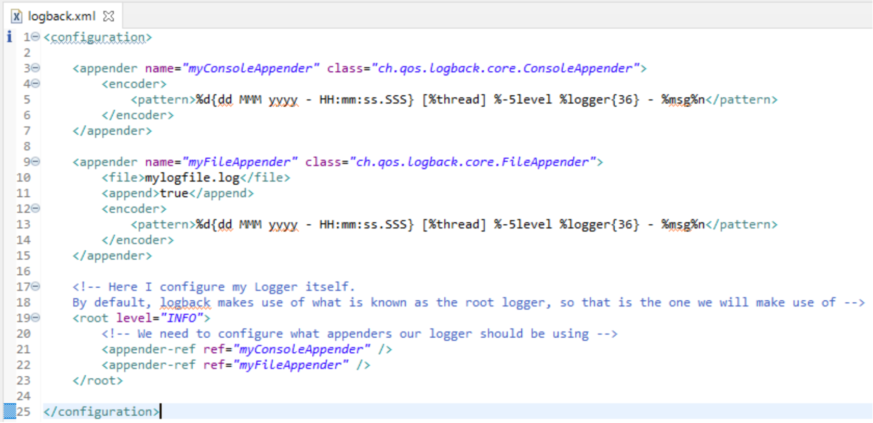
Enter or select the parent folder: spring-mvc-with-aop-demo/src/main/resources

File name: logback.xml

Update the resource blank file shell: logback.xml

Copy and paste the following into the file shell:

```
<configuration>

        <appender name="myConsoleAppender" class="ch.qos.logback.core.ConsoleAppender">
                <encoder>
                        <pattern>%d{dd MMM yyyy - HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>
                </encoder>
        </appender>

        <appender name="myFileAppender" class="ch.qos.logback.core.FileAppender">
                <file>mylogfile.log</file>
                <append>true</append>
                <encoder>
                        <pattern>%d{dd MMM yyyy - HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>
                </encoder>
        </appender>

        <!-- Here I configure my Logger itself.
        By default, logback makes use of what is known as the root logger, so that is the one we will make use of -->
        <root level="INFO">
                <!-- We need to configure what appenders our logger should be using -->
                <appender-ref ref="myConsoleAppender" />
                <appender-ref ref="myFileAppender" />
        </root>

</configuration>
```



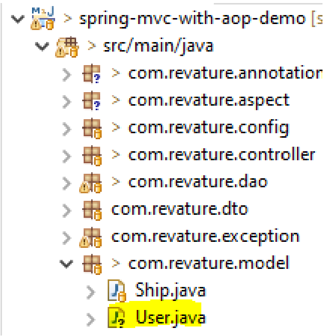
```
<configuration>

    <appender name="myConsoleAppender" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>%d{dd MMM yyyy - HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>
        </encoder>
    </appender>

    <appender name="myFileAppender" class="ch.qos.logback.core.FileAppender">
        <file>mylogfile.log</file>
        <append>true</append>
        <encoder>
            <pattern>%d{dd MMM yyyy - HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>
        </encoder>
    </appender>

    <!-- Here I configure my Logger itself.
    By default, logback makes use of what is known as the root logger, so that is the one we will make use of -->
    <root level="INFO">
        <!-- We need to configure what appenders our logger should be using -->
        <appender-ref ref="myConsoleAppender" />
        <appender-ref ref="myFileAppender" />
    </root>

</configuration>
```

Running the AOP Program

Initial Run with AOP Components

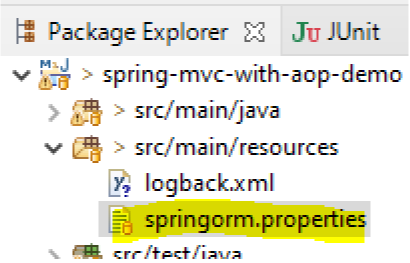
This program should already be in the IDE Tomcat server. It was added during the initial test after copy the program from a previous demo.

We added a new model file User which requires a database table. We need to update a property file to create database components for this program. Any existing data will be lost.



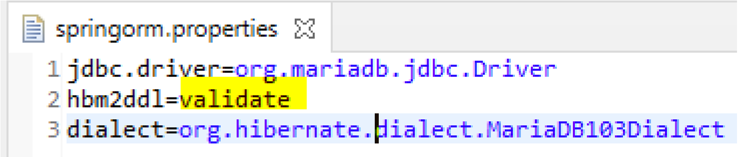
```
> spring-mvc-with-aop-demo [s
  > src/main/java
    > com.revature.annotatior
    > com.revature.aspect
    > com.revature.config
    > com.revature.controller
    > com.revature.dao
    com.revature.dto
    com.revature.exception
    > com.revature.model
      Ship.java
      User.java
```

Edit the property file: springorm.properties



```
> spring-mvc-with-aop-demo
  > src/main/java
  > src/main/resources
      logback.xml
      springorm.properties
  src/test/java
```

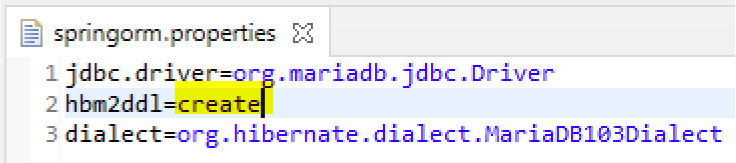
Change hbm2ddl property from “validate”



springorm.properties

```
jdbc.driver=org.mariadb.jdbc.Driver
hbm2ddl=validate
dialect=org.hibernate.dialect.MariaDB103Dialect
```

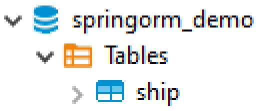
To “create”



springorm.properties

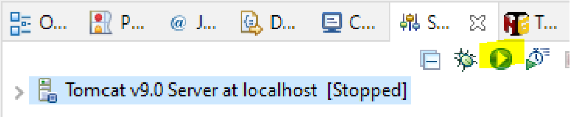
```
jdbc.driver=org.mariadb.jdbc.Driver
hbm2ddl=create
dialect=org.hibernate.dialect.MariaDB103Dialect
```

Open DBeaver and look at the existing schema for this program. We did not change the database name so it still points to the springorm_demo. The schema should only contain a ship table.



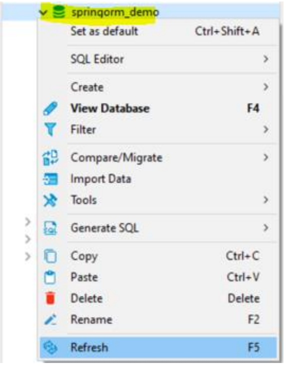
```
springorm_demo
  Tables
    ship
```

Start the Tomcat server

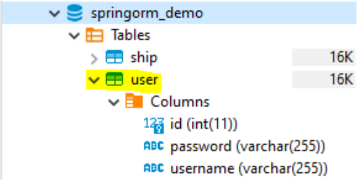


Refresh the schema in DBeaver

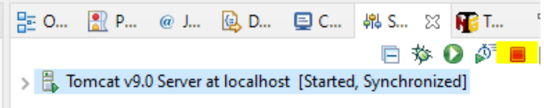
Right click database ➔ click “Refresh”



Verify the User table was created:

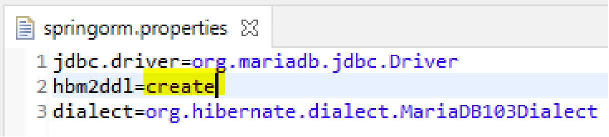


Stop the Tomcat server



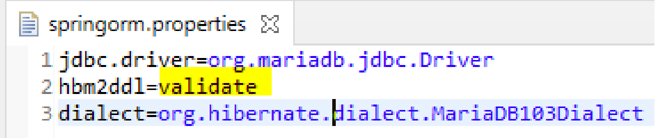
Change back the property file: springorm.properties

Change hbm2ddl property from “create”



```
jdbc.driver=org.mariadb.jdbc.Driver
hbm2ddl=create
dialect=org.hibernate.dialect.MariaDB103Dialect
```

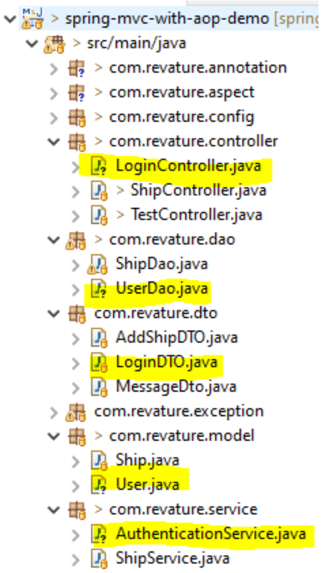
To “validate”



```
jdbc.driver=org.mariadb.jdbc.Driver
hbm2ddl=validate
dialect=org.hibernate.dialect.MariaDB103Dialect
```

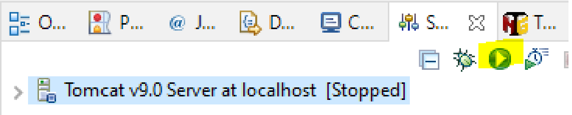
Run with Aspect AOP Components

The program added log in and authentication features. However the program did not any feature to add a user to the database. So we need to manually add a user to the user table.



```
spring-mvc-with-aop-demo [spring
  src/main/java
    com.revature.annotation
    com.revature.aspect
    com.revature.config
    com.revature.controller
      LoginController.java
      ShipController.java
      TestController.java
    com.revature.dao
      ShipDao.java
      UserDao.java
    com.revature.dto
      AddShipDTO.java
      LoginDTO.java
      MessageDto.java
    com.revature.exception
    com.revature.model
      Ship.java
      User.java
    com.revature.service
      AuthenticationService.java
      ShipService.java
```

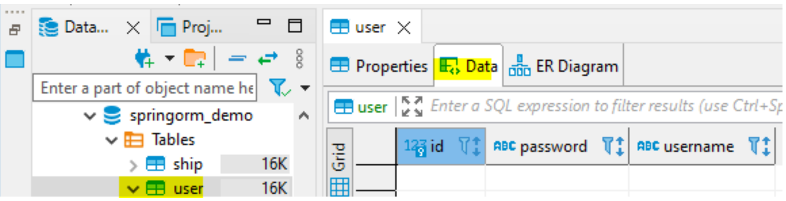
Start the Tomcat server



Add User to Database

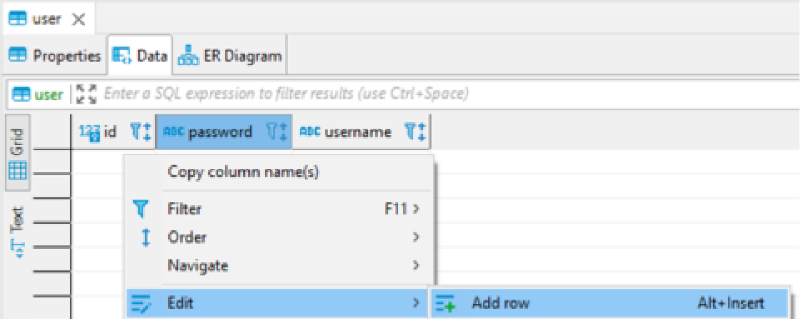
Using DBeaver, open the user table.

Double click the “user” table ➔ select “Data” tab

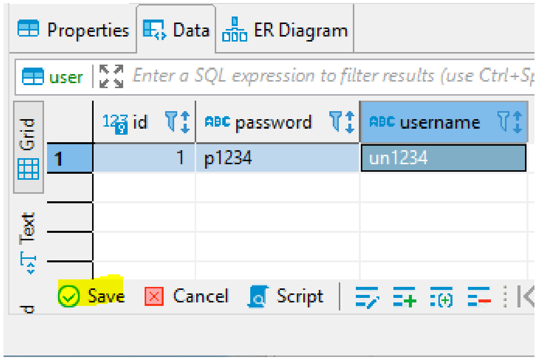


Add a row in the table

Right click in the Grid ➔ highlight “Edit” ➔ select “Add Row”



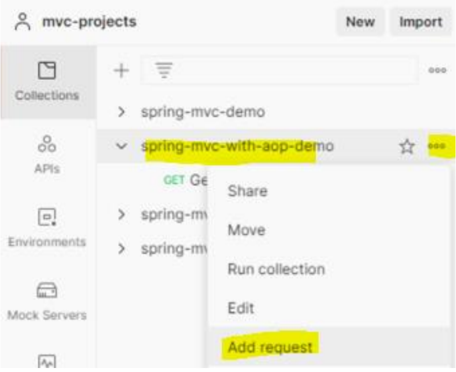
Enter password: “p1234”
Enter username: “un1234”
Click “Save”



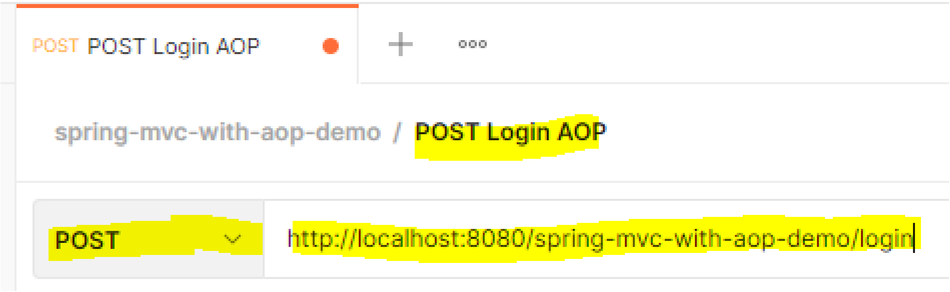
	id	password	username
1	1	p1234	un1234

Create Login POST Request

Open Postman desktop agent.
Create the Login POST Request



Edit name: “POST Login AOP”
Select request type: “POST”
Enter login URL: “http://localhost:8080/spring-mvc-with-aop-demo/login”

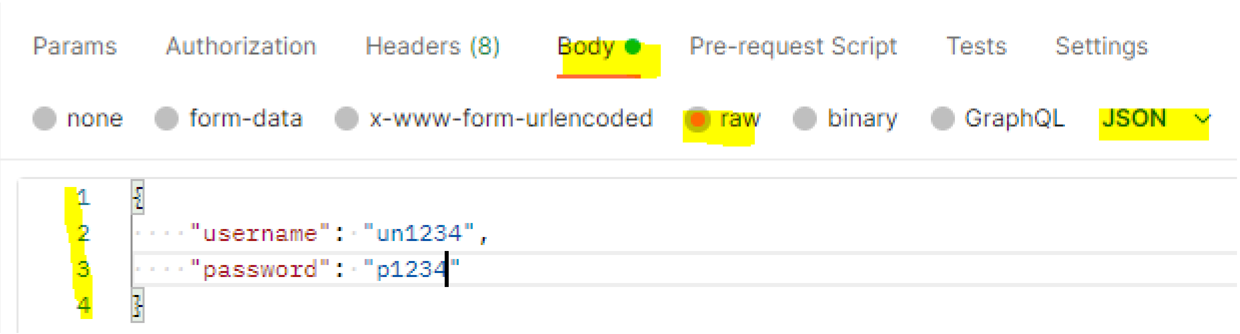


spring-mvc-with-aop-demo / POST Login AOP

POST http://localhost:8080/spring-mvc-with-aop-demo/login

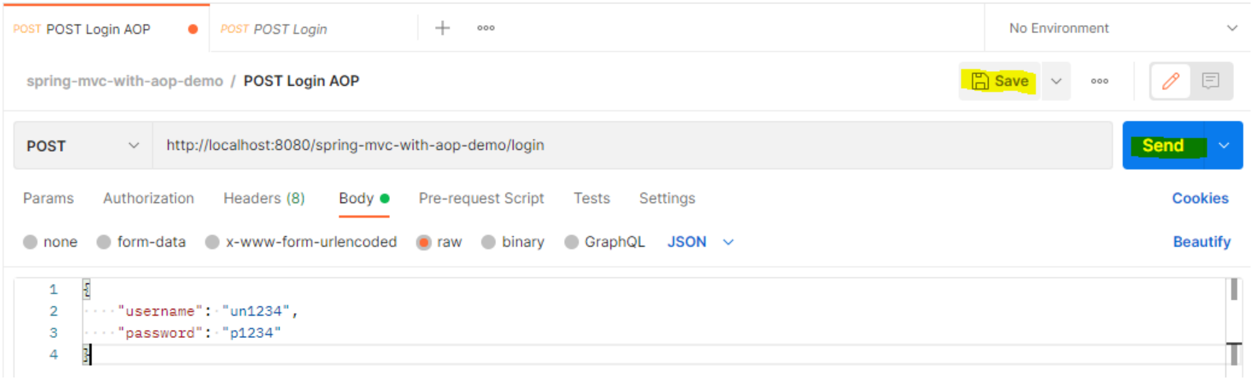
Add request body.
Select tab: “Body”
Select type: “raw”
Select from dropdown: “JSON”
Enter body as define in login DTO:

```
{
    "username": "un1234",
    "password": "p1234"
}
```



```
{
    "username": "un1234",
    "password": "p1234"
}
```

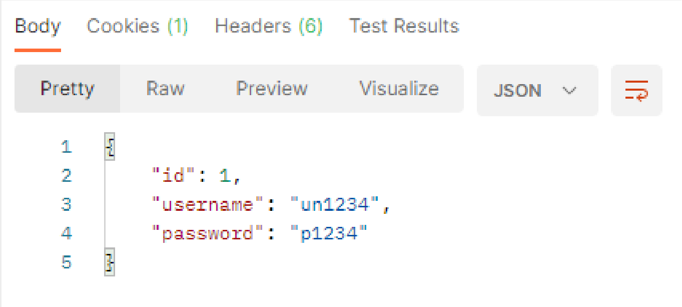
Save the request.
Send the request.



POST http://localhost:8080/spring-mvc-with-aop-demo/login

```
{
    "username": "un1234",
    "password": "p1234"
}
```

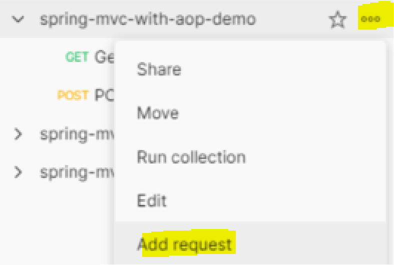
The login POST response echoes back the database record:



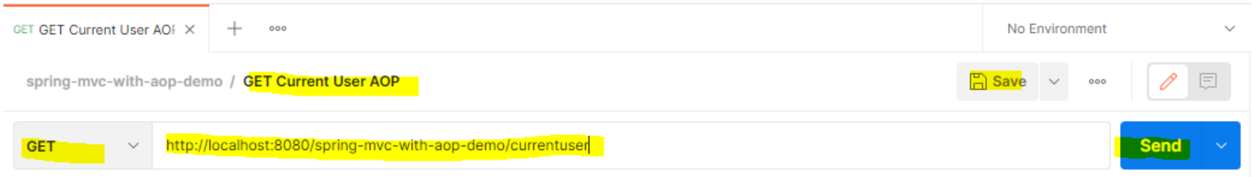
```
{
    "id": 1,
    "username": "un1234",
    "password": "p1234"
}
```

Create Current User GET Request

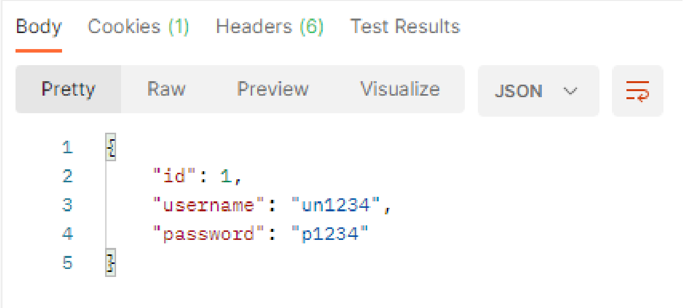
Add a request



Edit name: “GET Current User AOP”
Leave request type as: “GET”
Enter Current User URL: “http://localhost:8080/spring-mvc-with-aop-demo/currentuser”
Save the request.
Send the request.

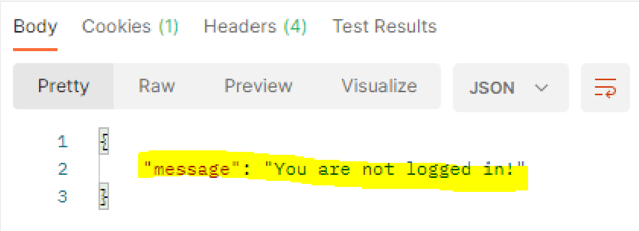

GET GET Current User AOI
No Environment
spring-mvc-with-aop-demo / GET Current User AOP
Save
GET
http://localhost:8080/spring-mvc-with-aop-demo/currentuser
Send

The current user GET response echoes back the database record when a user has a session:



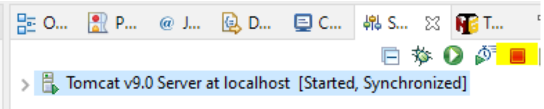
```
{
    "id": 1,
    "username": "un1234",
    "password": "p1234"
}
```

If no user is currently logged in, the following message is sent in the response:



```
{
    "message": "You are not logged in!"
}
```

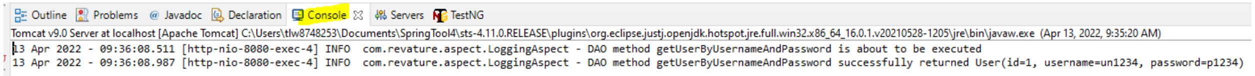
Stop the Tomcat server



Tomcat v9.0 Server at localhost [Started, Synchronized]

Aspect AOP Logging

The LoggingAspect class should performed certain logging messages without having logging code in other classes. There should be messages in the server console and in a log file “mylogfile.log”.



```
Tomcat v9.0 Server at localhost [Apache Tomcat] C:\Users\tlw8748253\Documents\SpringTool4\sts-4.11.0.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_16.0.1.v20210528-1205\jre\bin\javaw.exe  (Apr 13, 2022, 9:35:20 AM)
13 Apr 2022 - 09:36:08.511 [http-nio-8080-exec-4] INFO  com.revature.aspect.LoggingAspect - DAO method getUserByUsernameAndPassword is about to be executed
13 Apr 2022 - 09:36:08.987 [http-nio-8080-exec-4] INFO  com.revature.aspect.LoggingAspect - DAO method getUserByUsernameAndPassword successfully returned User(id=1, username=un1234, password=p1234)
```