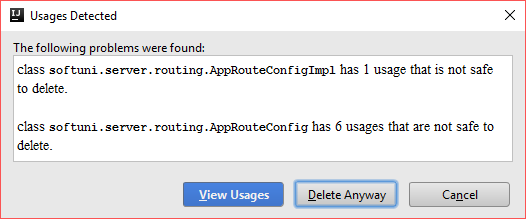
# Lab: Web Server Annotation Routing

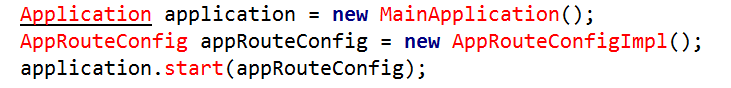
This **tutorial** provides step-by-step guidelines to change the routing of the web server we've created earlier with annotations.

## Delete Routing

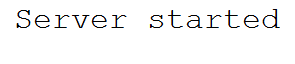
First, we must delete the old routing. Delete the AppRouteConfig interface and the AppRouteConfigImpl class. You will get the following warning:



Now delete the MainApplication class and the Application interface. Open the Main class and you should see this:



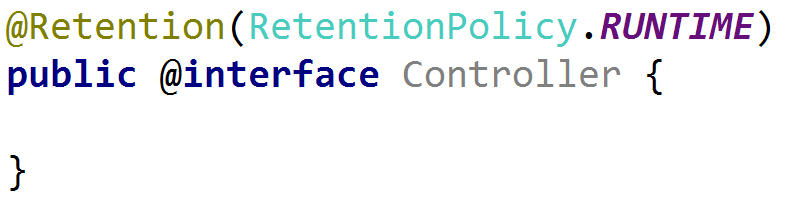
Delete those lines. Now remove the AppRouteConfig from the ServerImpl and ServerRouteConfigImpl constructors. From the ServerRouteConfigImpl delete the initializeServerConfig() and parseRoute() methods. Now the server should be able to start:



Now we need to create the new routing.

## Controller Annotation

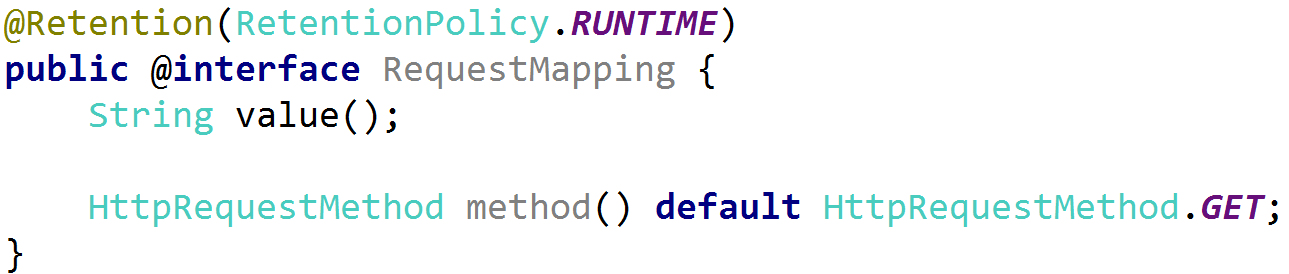
In the server.routing package, create the following annotation:



We will use this to identify our controllers later on. Only classes with this annotation will be able to declare routings.

## Request Mapping Annotation

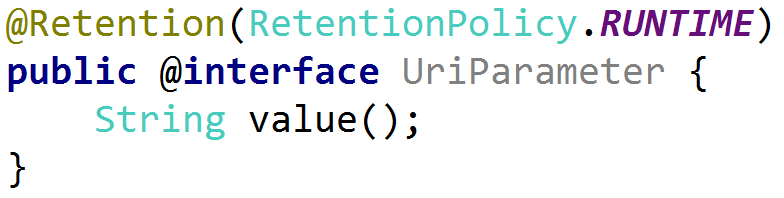
In the server.routing package, create the following annotation:



We will need this annotation in order to **define the routings**. It will be placed above each action (method).

## Uri Parameter Annotation

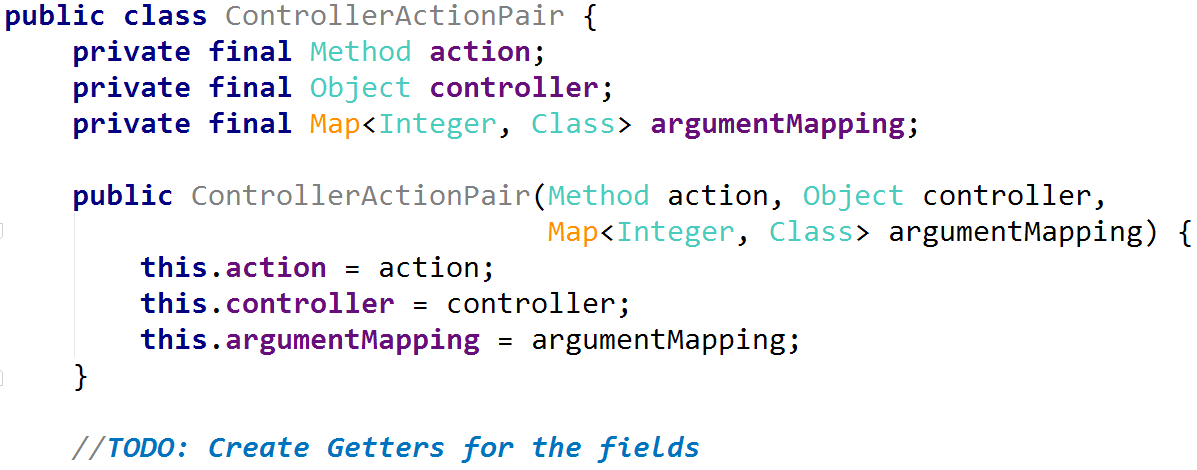
In the server.routing package, create the following annotation:



This annotation will be used to define the action parameters. It will tell our server that we need to pass a certain parameter from the url to the method.

## Controller Action Pair

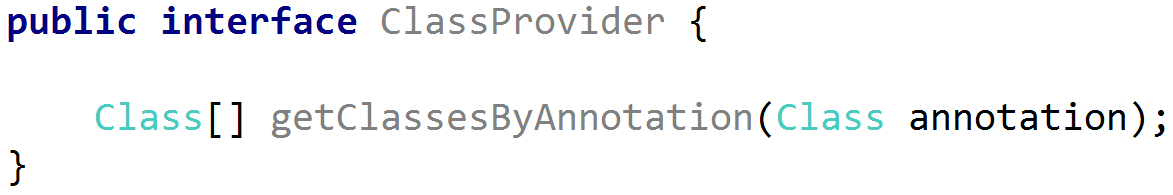
Again, in the routing package, create a new class called ControllerActionPair:



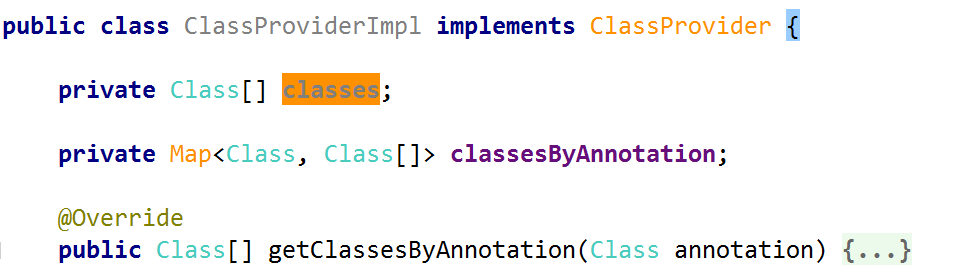
This class will keep our routing info. The action field will be the method that we are going to invoke for the given route. The controller field keeps the metadata about our controller. The argumentMapping map keeps the index of the parameter and its type.

## Class Provider

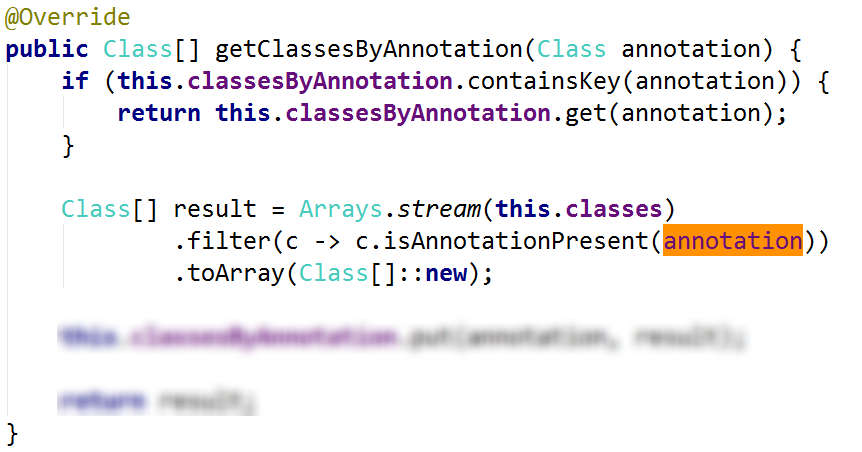
In the server package, create a new package called provider. Inside, create a new interface called ClassProvider:



Now we need to create the implementation:

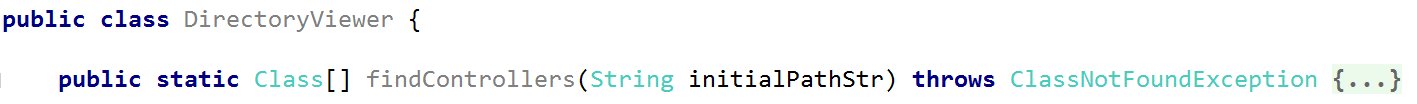


We need two private fields. One for the classes we are going to traverse in a moment and one for the classes that contain given annotation. We will use the map in order to make the method faster. The getClassesByAnnotation() method should have the following code:



## Traversing Classes

Before we continue with our class provider, we need to implement a way to traverse our classes. Implement a new class in the util package called DirectoryViewer. It should contain static method called findControllers():



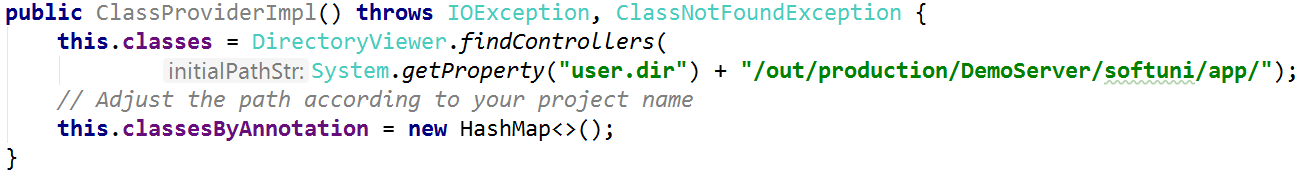
The implementation of the findControllers() method should be like this:



Now we can go back to the routing.

## Completing the Class Provider

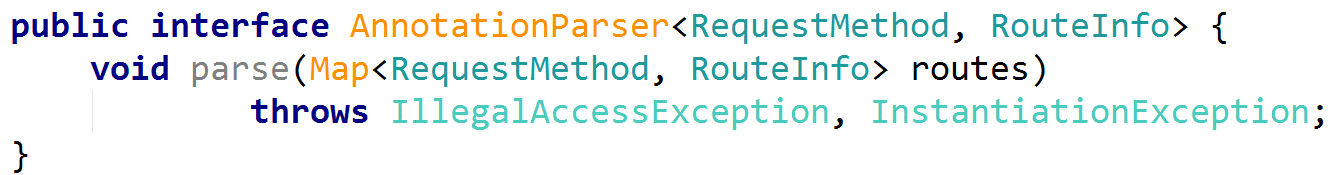
In the ClassProvider class, create a constructor that initializes the private fields:



With this, the class provider is ready.

## Creating Annotation Parser Interface

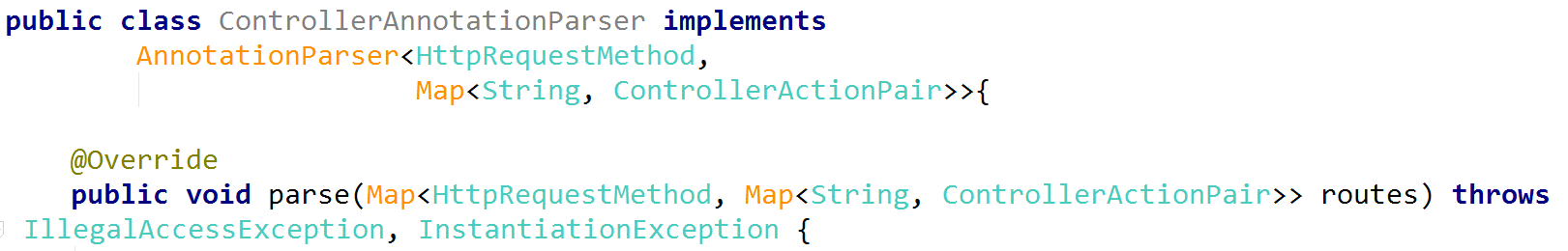
In the server package, create a new package called parser. Create a new generic interface with two parameters called AnnotationParser:



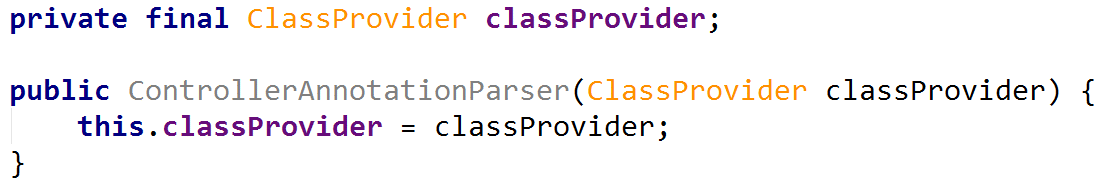
It should contain only one method called parse(), which will parse the annotation. It should accept a map with the generics from the interface.

## Creating Controller Annotation Parser

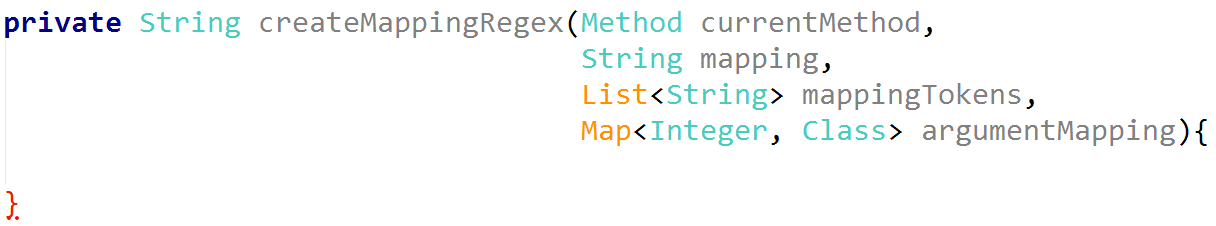
Create a new class called ControllerAnnotationParser. It should implement the AnnotationParser. The parameters for the AnnotationParser should be <HttpRequestMethod, Map<String, ControllerActionPair>>:



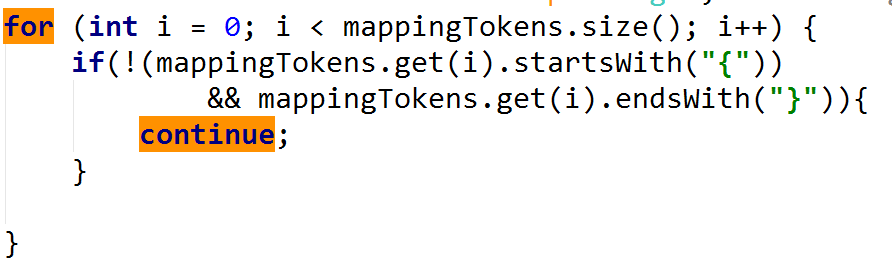
Before we implement the parse() method, we want to create a constructor that takes a ClassProvider as a parameter:



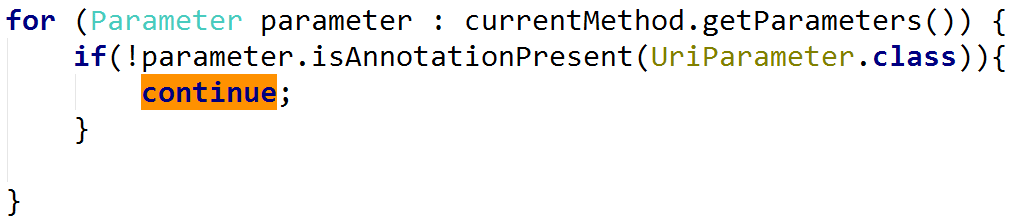
Now, we need to implement another method called createMappingRegex():



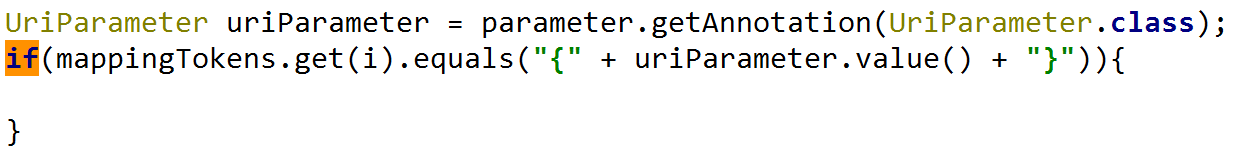
This method will **get our routing** and **create regex** that we are going to use for our router. We need the current method, the URL, the different parts of the URL in the mappingTokens collection and the map which will keep the type of our parameters. First, we need to make sure that we are only checking the parts of our routing that are parameters. A parameter is identified by "{" and "}" in the begging and in the end:



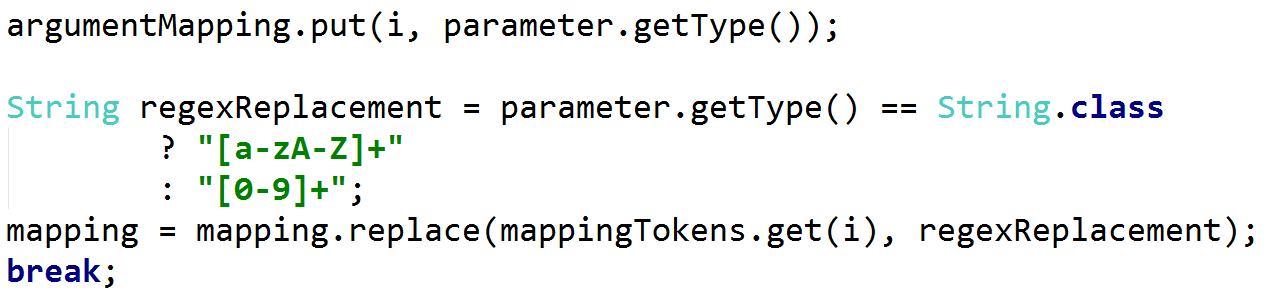
After the if statement, we need to iterate over the parameters for our current method. We only need the parameter if it contains the UriParameter annotation:



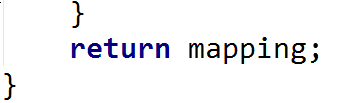
Now we need to get the value of our UriParameter and check if the current URL part is the same as our UriParameter:



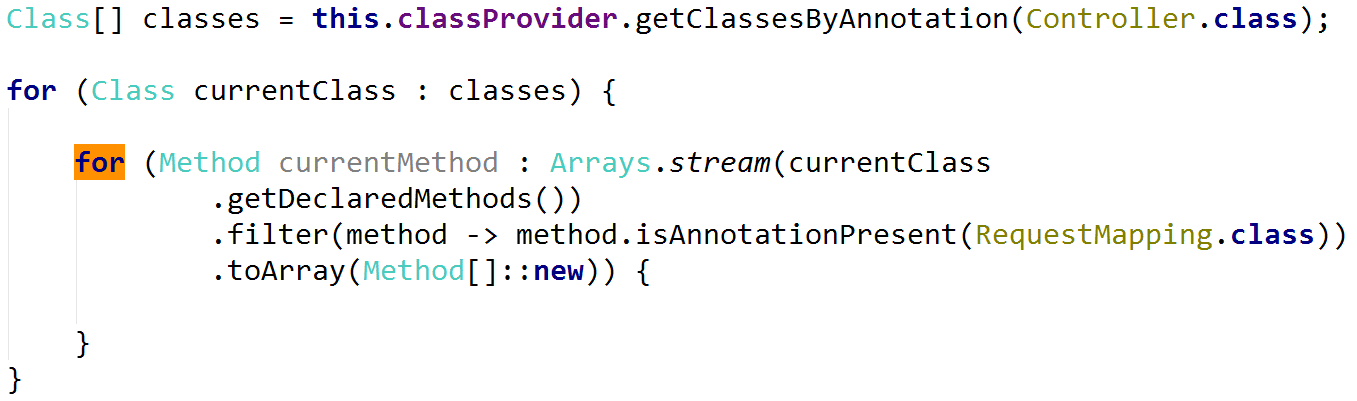
When we find the right UriParameter, we want to save it to our map. Then we need to replace the parameter from the URL with the regex we want to match. We will only have strings and numbers, so we are going to use [a-zA-Z]+ and [0-9]+. After this is done, we can exit the loop:



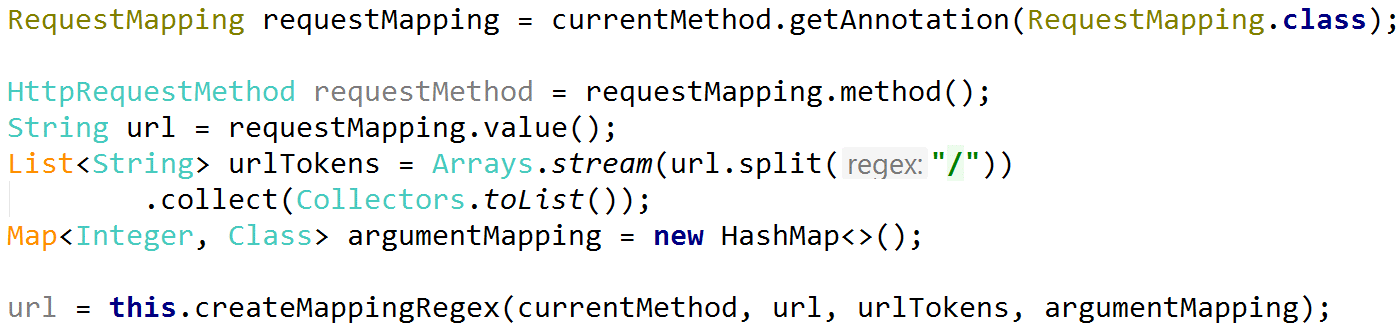
Finally, the method will return the mapping string, which corresponds to the URL:



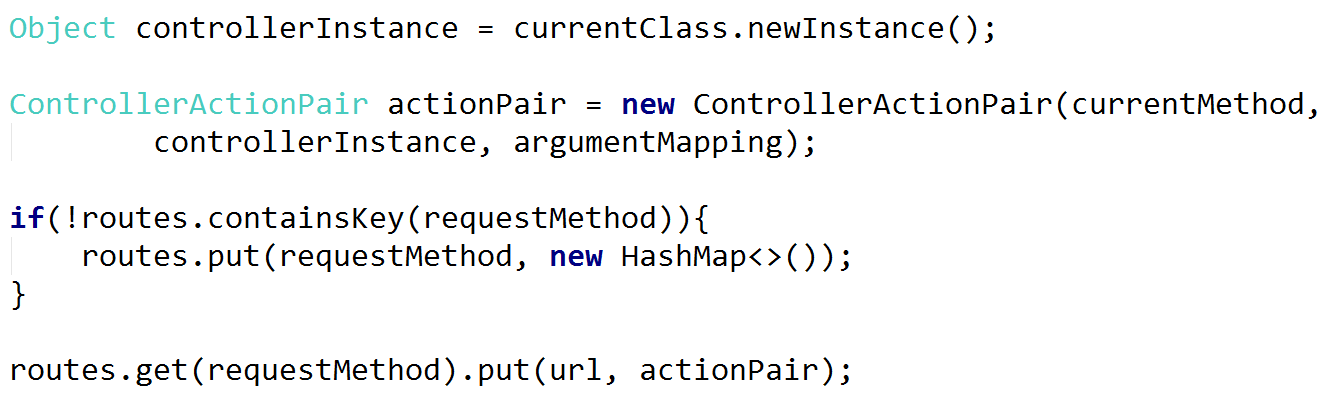
Now we can finally go back to our parse() method. It should get all the classes with the Controller annotation. We will get them, using the ClassProvider. Then we are going to iterate over all of them and for each one we are going to iterate over its methods that contain the RequestMapping annotation:



For every method, we need to create a mapping regex, using the method we've created earlier. In order to do that, we need to initialize the parameters that we are going to pass to that method:



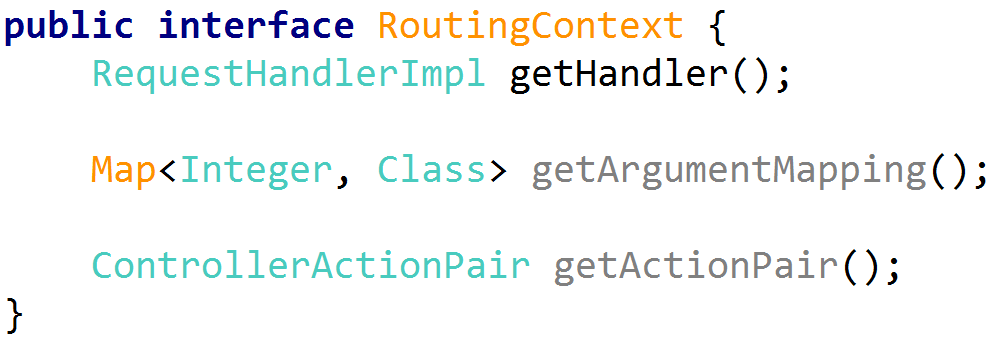
Now, we only need to create new instance of the class, new ControllerActionPair and add it to our routes map:



This should be our controller annotation parser done. Now we need to put it to good use.

## Changing the Routing Context Interface

Change your current RoutingContext interface to this:



You need to reflect those changes in the RoutingContextImpl.

## Changing the Request Handler Implementation

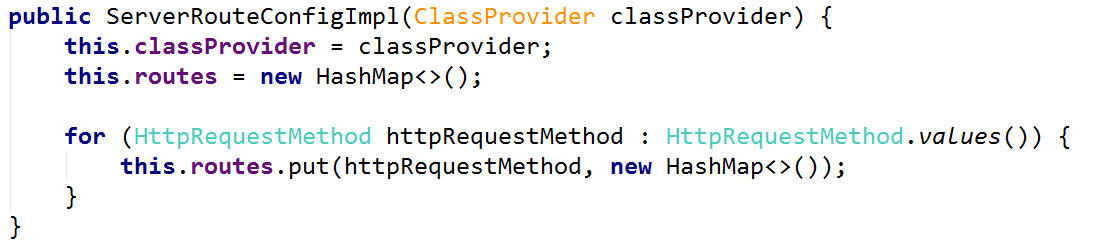
For our RequestHandlerImpl we need to create an empty constructor. Now we need to create a setter for our function:



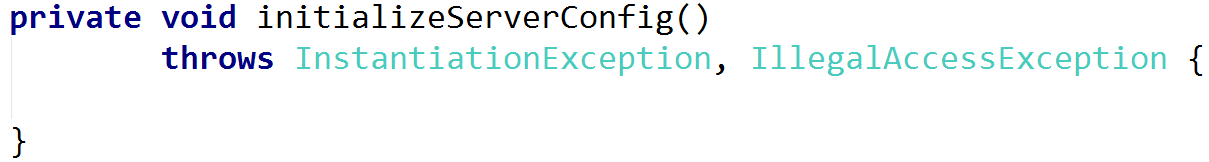
Create empty constructors in the GetHandler and PostHandler classes.

## Changing the Server Route Config

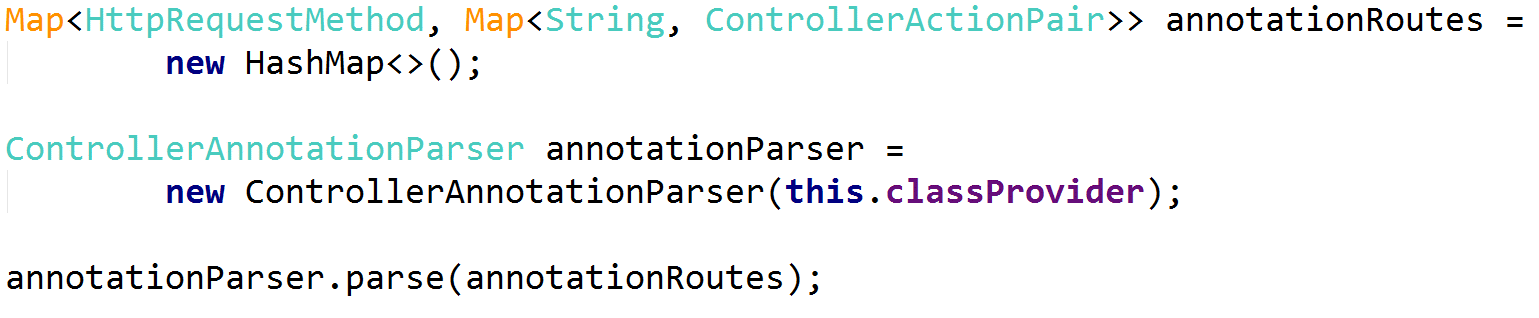
First of all, now our constructor should accept a ClassProvider, that we need to keep in a field:



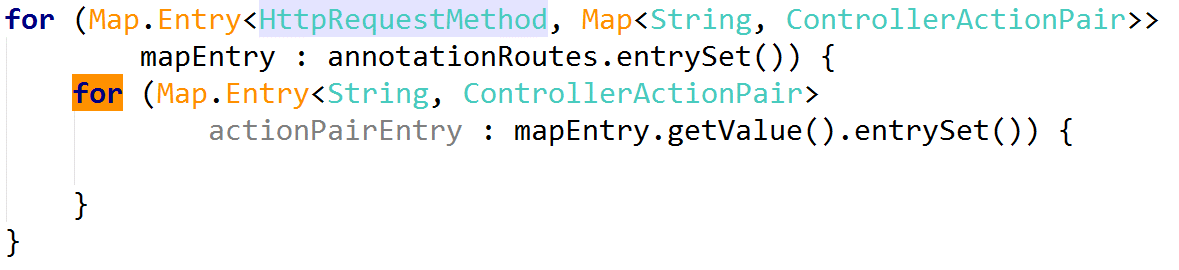
Create a new method called initializeServerConfig():



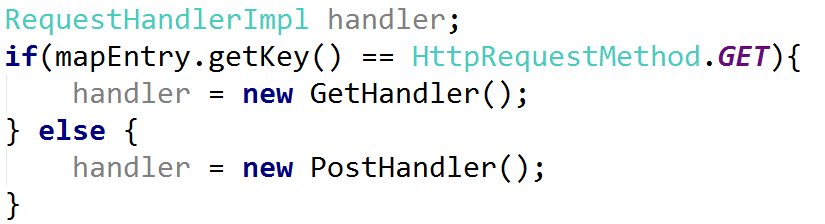
In this method, we will initialize our ControllerAnnotationParser and we are going to save the routes it creates in a map.



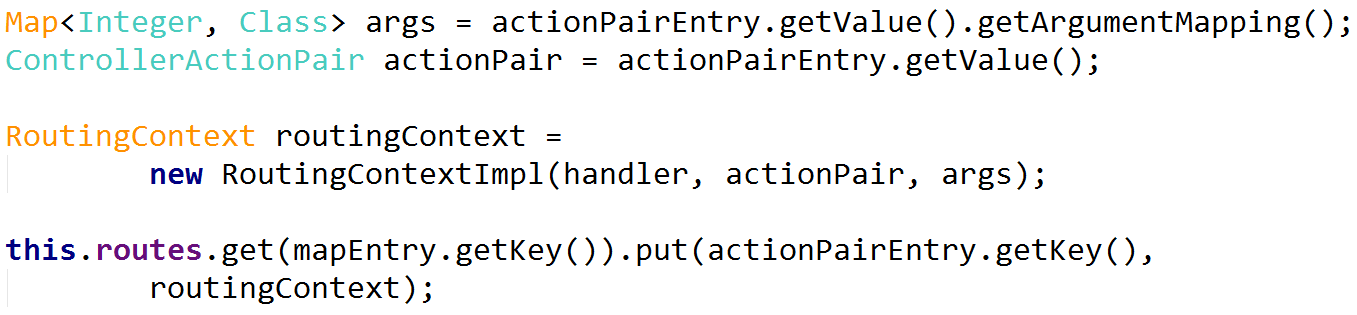
Now, we need to traverse that map and create our server routes, based on the annotation routes. We need to iterate our inner map:



First, we need to create a new RequestHandlerImpl based on the request method:



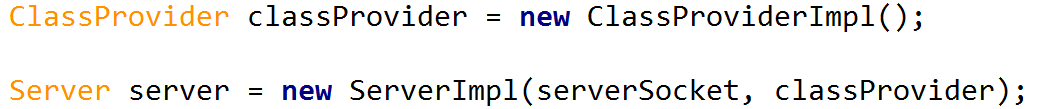
Now we need to create a new RoutingContext and save the route in our private field:



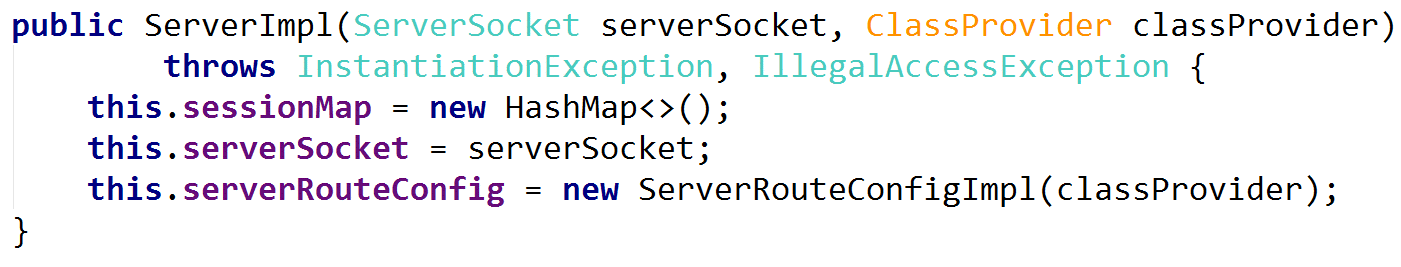
Finally, we need to **call this method** from our constructor.

## Creating the Server Route Config

In our Main class, we need to instantiate new ClassProvider and pass it to the ServerImpl:



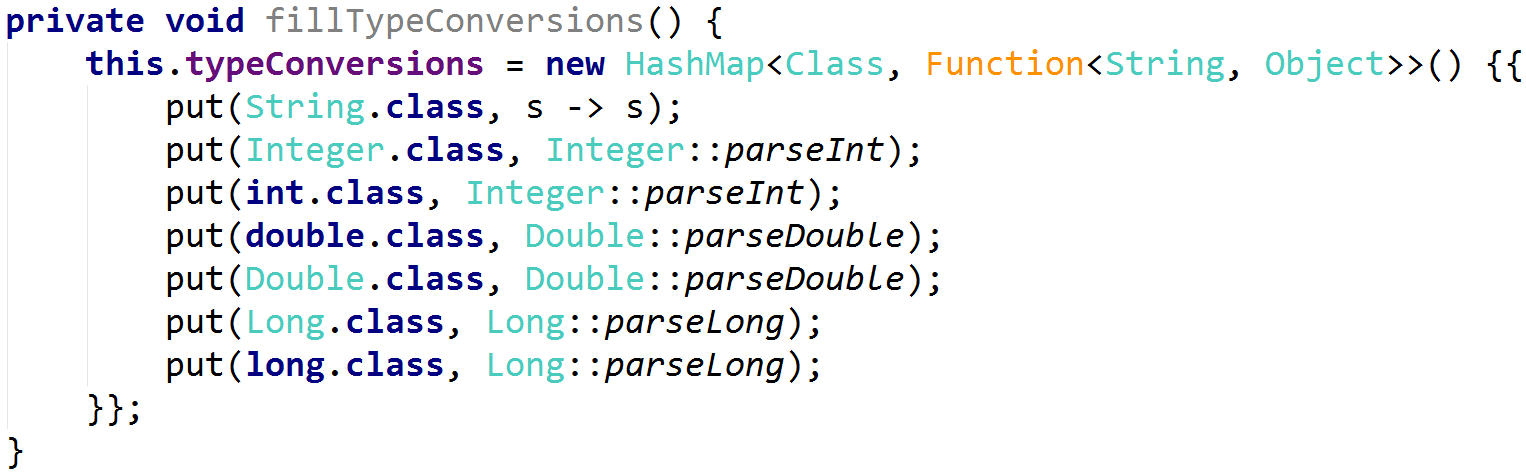
In our ServerImpl constructor, we only need to pass the ClassProvider to our ServerRouteConfigImpl:



This is it. The next thing we need to edit is the HttpHandler.

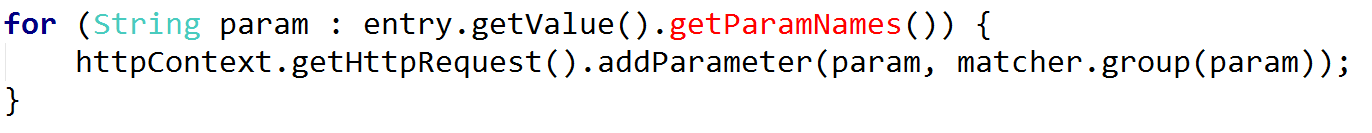
## Changing the Http Handler

Our handle() method currently isn't working, but we don't care about that for now. First, we want to create a new method called fillTypeConversions(), which will parse our parameters. Create new private field holding Map<Class, Function<String, Object>>:

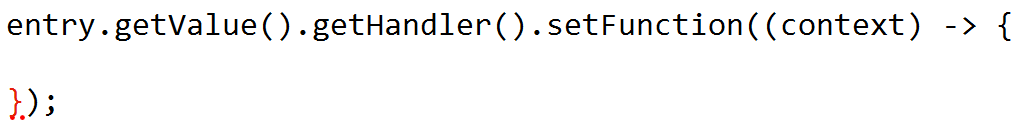


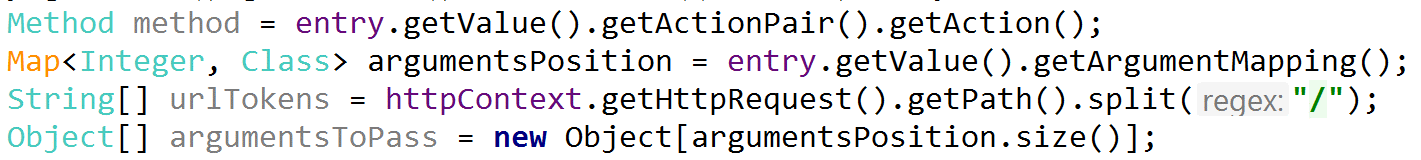
Call this method in your constructor. Now we can take a look at handle().

First, **delete** the for loop that iterate over non-existing collection:

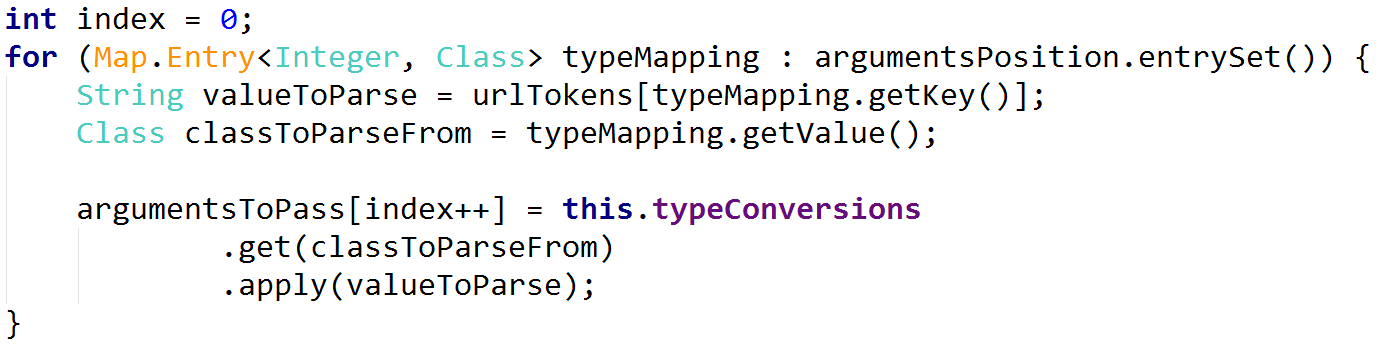


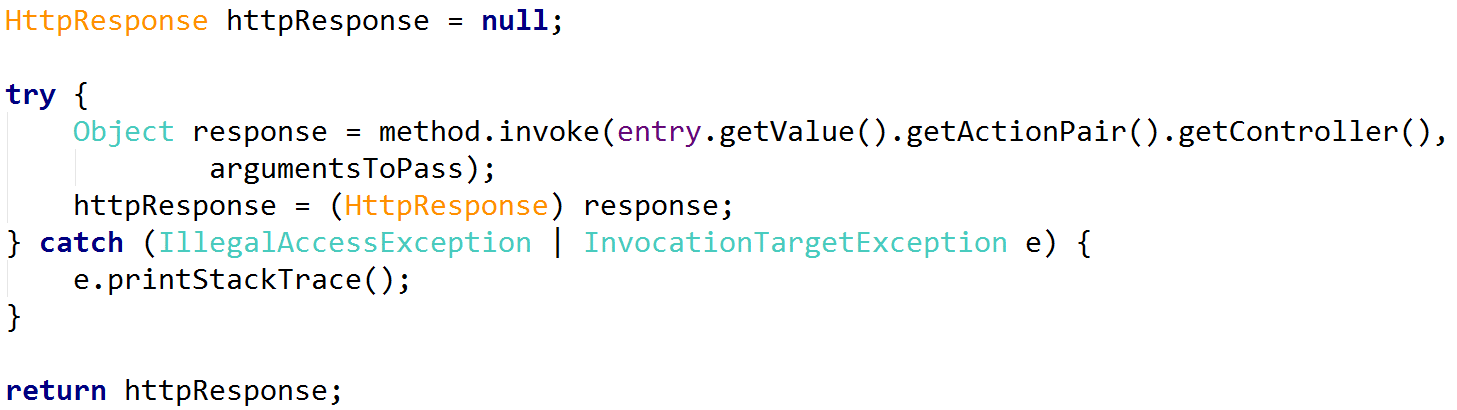
In the same place call the setFunction() of the RequestHandlemImpl:



This function will be relatively long, so get ready. We want to get our **method**, the **current URL** and the **parameters positions** in the URL:

Now we want to parse our parameters, that we are receiving from the URL:

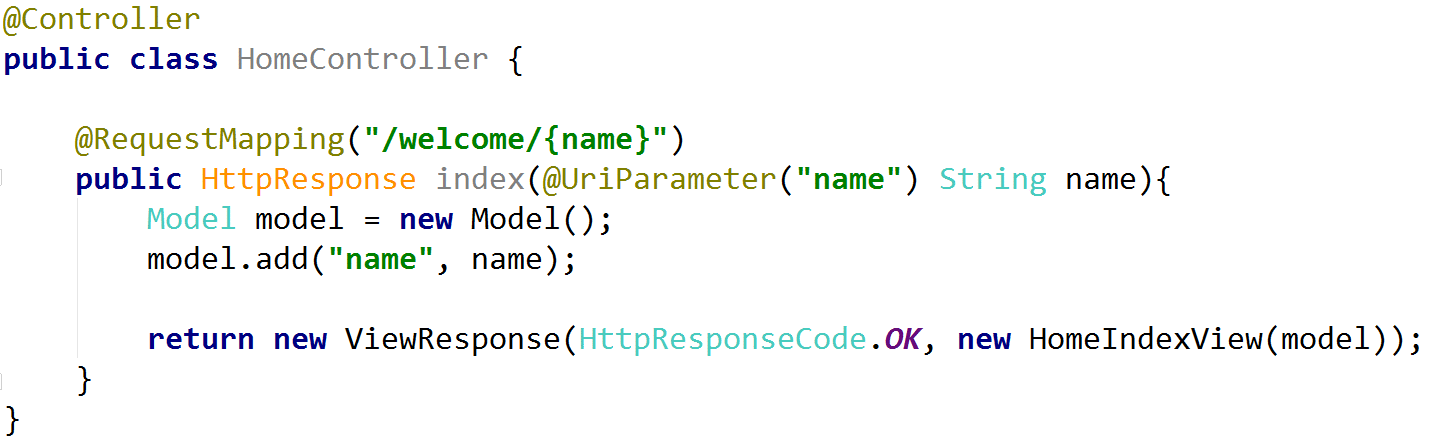


We are using our typeConversions function to parse the current parameter. Finally, we want to invoke our method and get the HttpResponse it returns. Then we **return that response** to our function:

Now we should be able to start our server again. Before we do that, let's add a route.

## Adding Our First Route

Open your HomeController. Add the Controller annotation to our class. You should have at least one method. Add the RequestMapping annotation to it. Give it a proper route. Finally, if your method takes any parameters, add the UriParameter annotation to them. The UriParameter value should correspond to the RequestMapping parameter name:



You can start the server and see if it works:



That’s it! Now you can create your own app.

## Exercise: Create a TODO List Application

Create a To Do list application, in which a user should be able to **login** and **logout**. If he is **logged in**, he should be able to **add items in a list**. If he **logouts**, he **loses all of the current items**.