Startup STEP 1

Spring Component Scanning

```
@Controller
public class RecordController {

    @Autowired
    RecordRepository r;
    @GetMapping("/index")
    ...
```

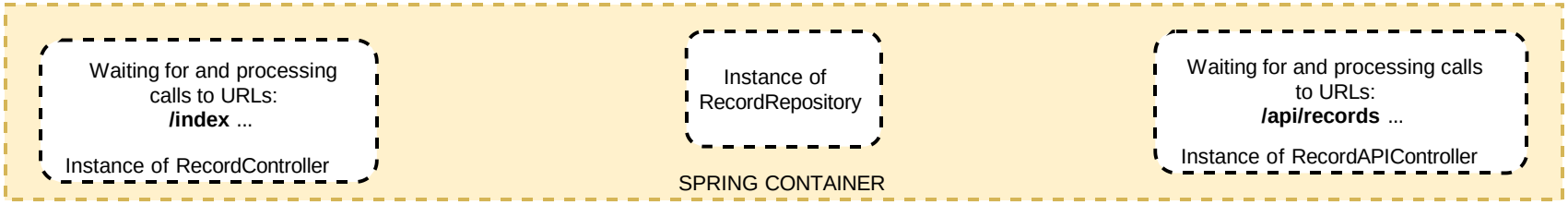
```
@RestController
@RequestMapping("/api")
public class RecordAPIController {

    @Autowired
    RecordRepositiry rp1;
    @GetMapping("/records")
    ...
```

```
@Repository
public interface RecordRepository extends
JpaRepository<Record,Long> {
```

When you start the application, Spring framework scans your code for @-annotations and registers them. So also the @Component annotations and the annotations derived from it such as @Controller or @Repository. In a @Controller or @RestController, the scan will also register which @...Mappings are there, and then register them as associated with that particular controller they are in.
In the end, on startup, the framework will have a full registry of all the @Controller, @Repository, ... classes, etc. in your project.

Startup STEP 2

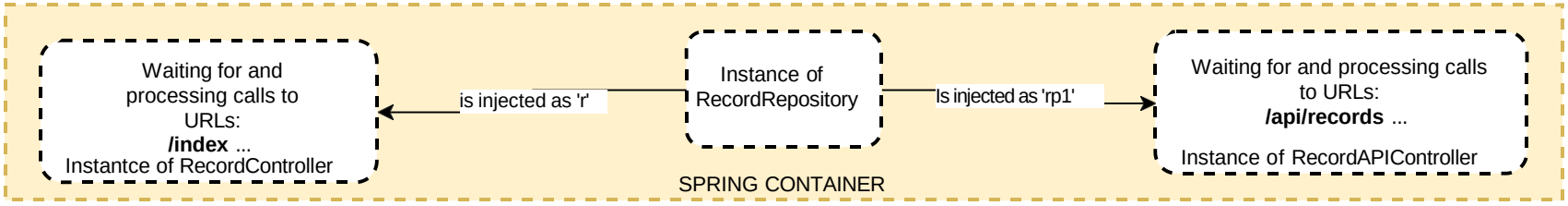
Inversion of Control



For all classes or interfaces with @Component or with annotations derived from it such as @Controller or @Repository, Spring will now perform **Inversion of Control**.
We used to be in control of creating and managing instances of classes by eg **new** RecordController(). Spring will now **take that control off your hands and turn it around** for the classes or interfaces with the annotations above. In other words Spring will itself create **one** instance of each Component, Controller, Repository, ... at startup and manage it itself in its Spring Container while the application is running. So calls to URLs will be handled by the one Controller responsible for those URLs. In turn, if instances of classes need to perform DB operations on Records, they will all use the one RecordRepository instance managed by Spring in the Spring Container.
Because only one instance of Components, Controllers and Repositories is created at a time, these are also **Singletons**.

Startup STEP 3

Depedency Injection



In the classes that have to use these DB operations, we never create separate Repositories per class (with eg **new** RecordRepository().)
Much more efficient is to use that single instance of RecordRepository in the Spring Container everywhere. This instance has already been created for us, so we just need to make the connection (hence the annotation @Autowired).
In other words we do not handle this **dependency** by creating an instance per class ourselves, but by always **injecting** a reference to an instance created for us.