



# Cucumber With Spring Boot

# Spring

The Spring logo, consisting of a horizontal bar with a teal segment on the left and an orange segment on the right.

01

Most Popular Java Framework

02

Mature & Lightweight

03

Inversion Of Control

04

Dependency Injection

# Spring Boot



01

Provides Rapid Access  
Development to Spring Framework

02

Opinionated

03

Increased Developer  
Productivity

# Demo



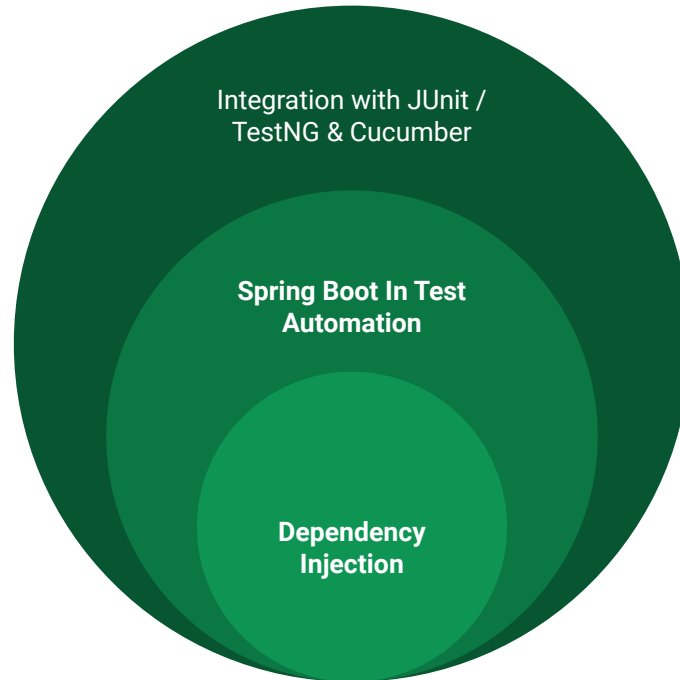
- A Simple MicroService development!!

# Goal



- Let spring handle the routine task!
- Manage the webdriver instance
- Manage Page objects / fragments
- Executing tests in multiple test environments like **dev / qa / stg / prod**
- Localization Testing for multiple languages
- Data driven testing by loading a CSV file in to a runtime DB
- Accessing files
- Automatic **window**/frame switching
- Taking screenshots
- Parallel test execution
- Executing cucumber feature files with automatic dependency injection

# Coverage



# Prerequisites & Assumptions



Spring Boot	0 knowledge!!!
<u>Java</u>	<u>Should be very comfortable with Java 8</u>
Selenium	Should know basics.
Cucumber	Not really required. Not an exhaustive cucumber course.
JUnit / TestNG	All the demos will be in TestNG / notes will be there for Junit users.

# Coverage



Spring Boot Basics	No test automation examples. Just basics of Spring Boot
Spring Boot with Selenium	Managing WebDriver, Automatic Dependency Injection of WebDriver, Page Objects etc  Executing tests in different env by using command line - dev/qa/stg/prd
Spring boot bean scope	Bean Scopes & Parallel test execution
Aspect Oriented Programming	Window Switch / Take Screenshot just with custom annotation
Accessing Files	How to easily access files using Spring Boot
Spring Data JPA	Data driven testing, Test data management, Querying test data
Logging	Logging, writing into a file
Cucumber	Complete integration of Feature files, Spring Boot, Parallel testing etc



# Softwares Required



Java 8 or above
Maven 3.3 or above
IntelliJ / Eclipse / IDE

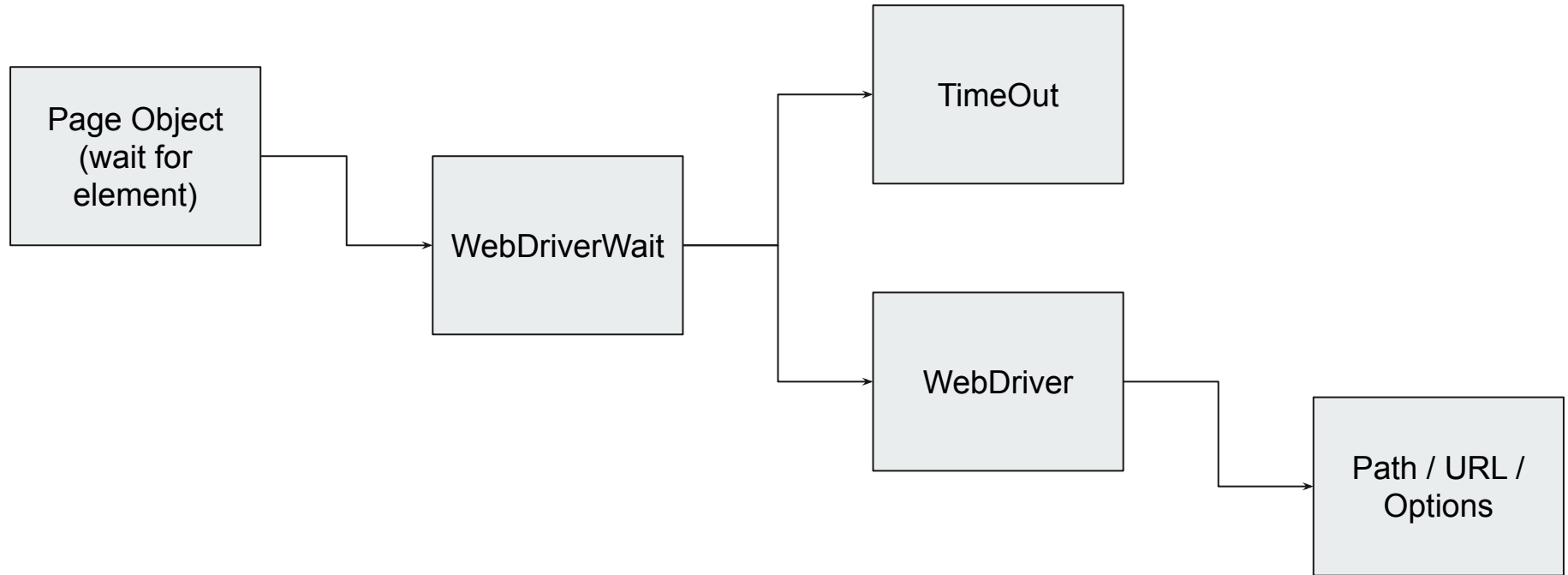


# Spring Boot - Basics

# Object Dependency



# Object Dependency



# Tight Coupling



```
class LoginPage{

    @FindBy(id = "username")
    private WebElement username;

    @FindBy(id = "password")
    private WebElement password;

    @FindBy(id = "signin")
    private WebElement signin;

    private WebDriverWait wait;

    LoginPage(){
        System.setProperty("webdriver.chrome.driver", "/home/qa/drivers/chromedriver");
        WebDriver driver = new ChromeDriver();
        wait = new WebDriverWait(driver, 30);
        PageFactory.initElements(driver, this);
    }

    ...
    ...
    ...
}
```

# Loose Coupling



```
class LoginPage{

    @FindBy(id = "username")
    private WebElement username;

    @FindBy(id = "password")
    private WebElement password;

    @FindBy(id = "signin")
    private WebElement signin;

    private WebDriverWait wait;


    LoginPage(WebDriver driver, int timeout){
        wait = new WebDriverWait(driver, timeout);
        PageFactory.initElements(driver, this);
    }

    ...
    ...
    ...
}
```

# Passing the Dependencies



- Constructor
- Setter methods
- Field injection



```
private WebDriver driver;  
private WebDriverWait wait;  
  
public void setWebDriver(WebDriver driver){  
    this.driver = driver;  
}
```

# Field Injection



```
class LoginPage{

    @FindBy(id = "username")
    private WebElement username;

    @FindBy(id = "password")
    private WebElement password;

    @FindBy(id = "signin")
    private WebElement signin;

    private WebDriverWait wait;

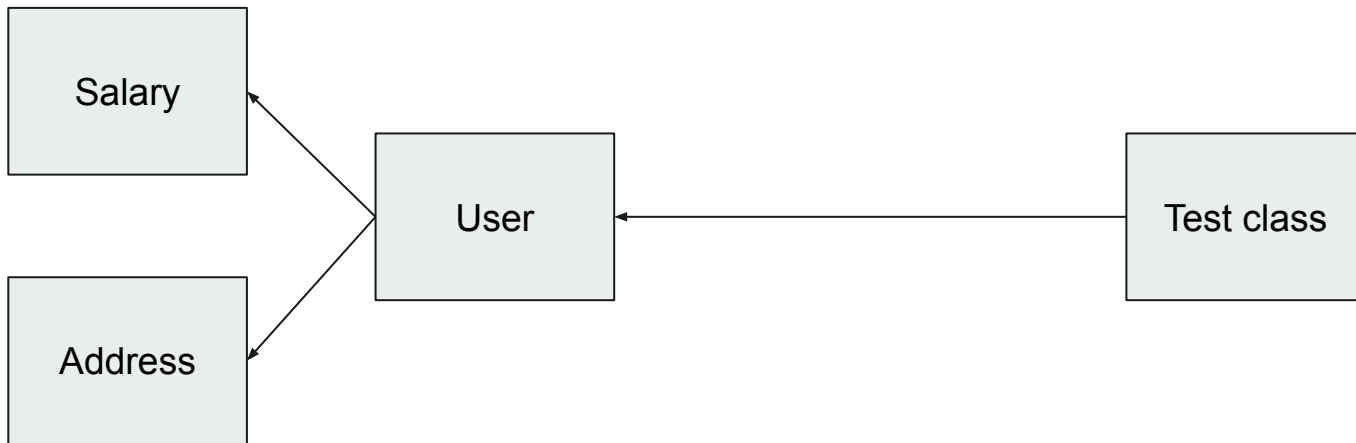
    LoginPage(WebDriver driver, int timeout){
        wait = new WebDriverWait(driver, timeout);
        PageFactory.initElements(driver, this);
    }

    ...
    ...
    ...
}
```

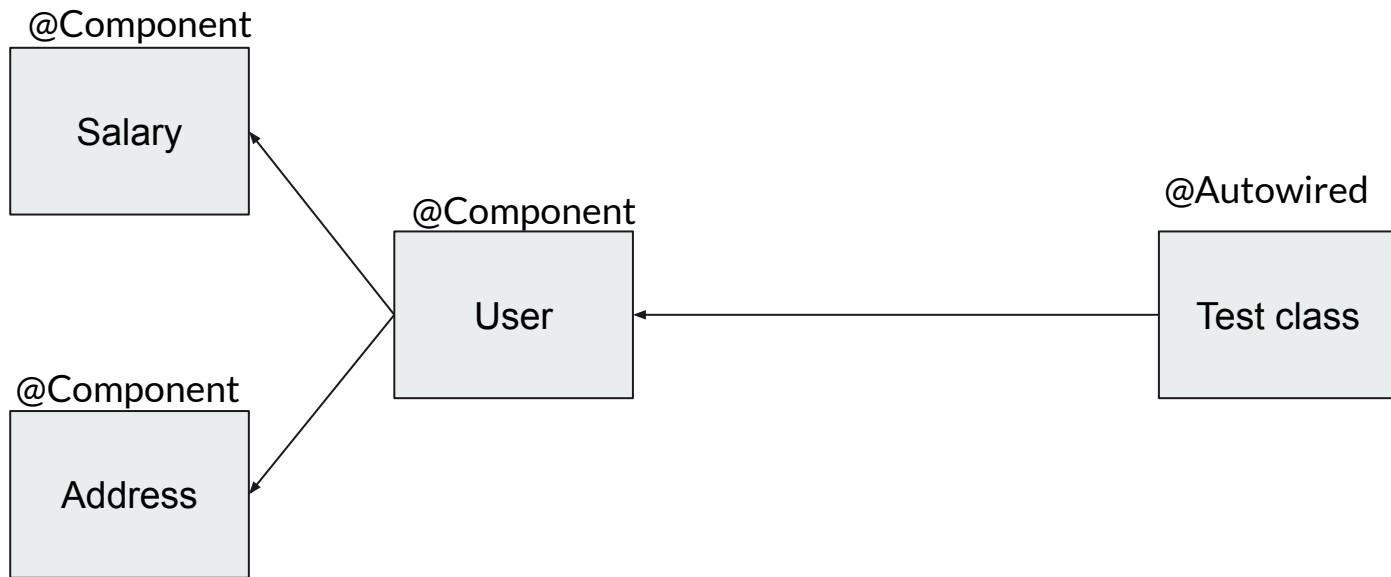


# Dependency Injection - @Autowired

- Injecting your application classes - @Component



# Dependency Injection - @Autowired



# Dependency Injection - @Value



- Primitive types
- Properties
- Environment Variables
- URL
- Arrays / Lists
- File / Path
- Default values

# Constructor / Setter / Field Injection



Constructor	Setter	Field
Boilerplate code / verbose	Boilerplate code / verbose	Short & Neat
Less readable	Less readable	More readable
Easy to unit test with custom parameters	Easy to unit test with custom parameters	Works only with DI frameworks
Immutable	Mutable	<del>(Im)</del> mutable

# Page Object - Constructor Injection(!!)



```
class LoginPage{  
    private WebElement username;  
    private WebElement password;  
    private WebElement signin;  
    private WebDriver driver;  
    private WebDriverWait wait;  
  
    LoginPage(WebDriver driver, int timeout, WebElement username, WebElement password, WebElement signin){  
        this.driver = driver;  
        this.wait = new WebDriverWait(driver, timeout);  
        this.username = username;  
        this.password = password;  
        this.signin = signin;  
    }  
}
```

# External Libraries - Dependency Injection

---

- Classes from external libraries
  - @Bean
  - @Configuration



# @Bean vs @Component



- @Component - Your classes (automatic configuration)
- @Bean - You explicitly create a spring bean by doing your own configuration
  - Why? - because we can not modify 3rd party classes source code. Instead we create an instance and request spring to manage it for us!
- **Both are Spring managed beans!**

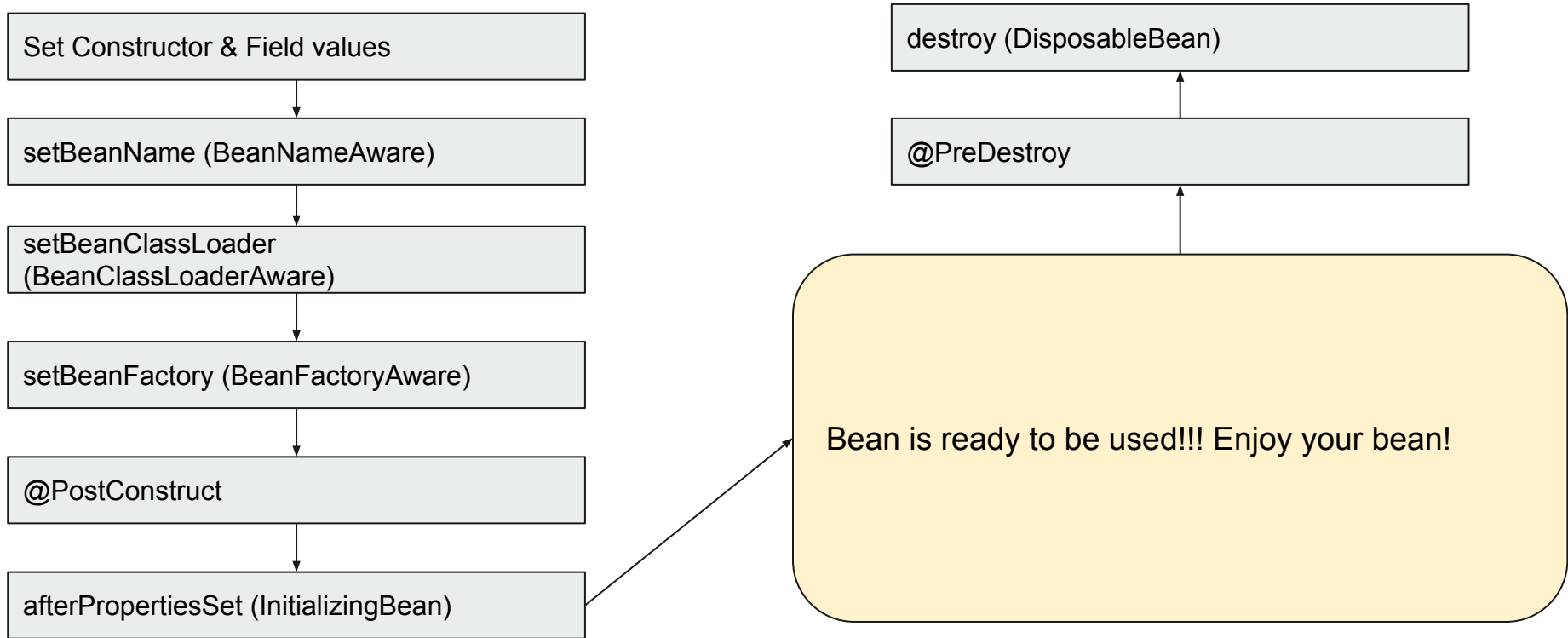
# @Component/@Bean for all!?



```
for(User user: users){  
    Email email = new Email(user.getEmailAddress());  
    email.setSubject(user.getOrder().getNumber());  
    email.setBody("Thanks for the purchase");  
    EmailUtil.send(email);  
}
```

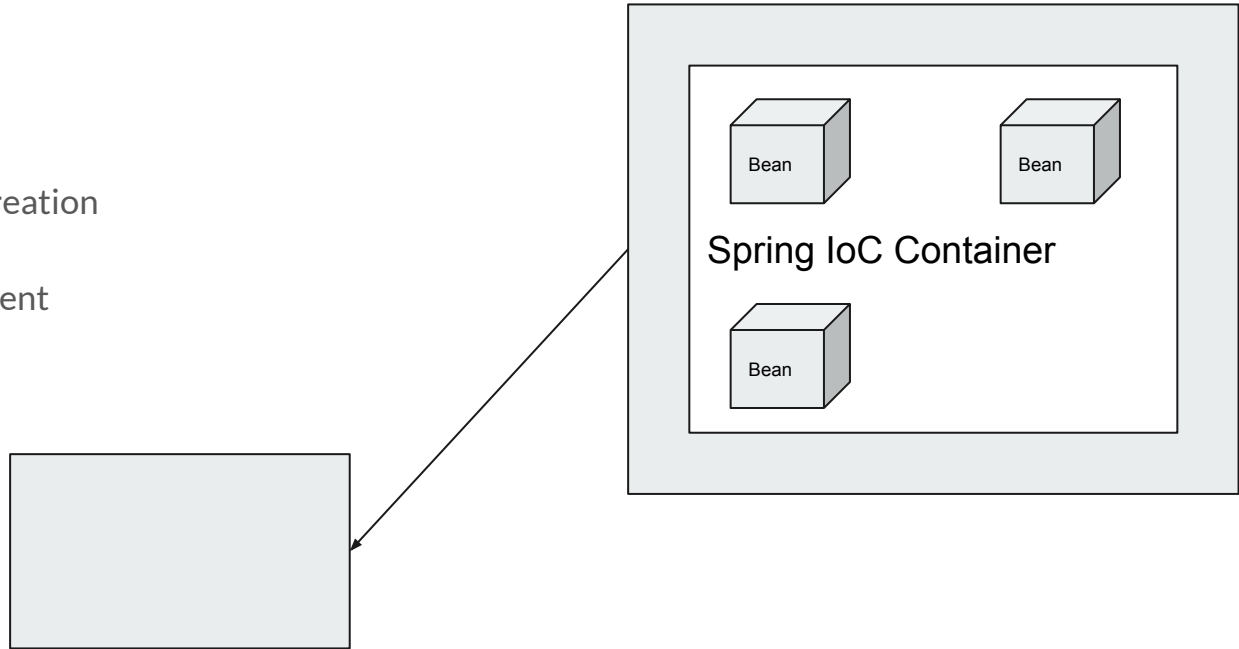


# Bean Lifecycle



# Inversion Of Control


- Automatic object creation
- Method invocation
- Life cycle management



# Summary



- Spring allows loose coupling by injecting the dependencies
- Spring boot is an extension for rapid development
- Constructor / Setter / Field injections
- **Bean** is a Java Object managed by Spring
  - **@Component** - for automatic configuration
  - **@Configuration & @Bean** for manual configuration
- **@Autowired** for spring beans
- **@Value** - from application.properties, environment variables etc
- Bean has a lifecycle
  - **@PostConstruct**
  - **@PreDestroy**
- Spring is IoC container



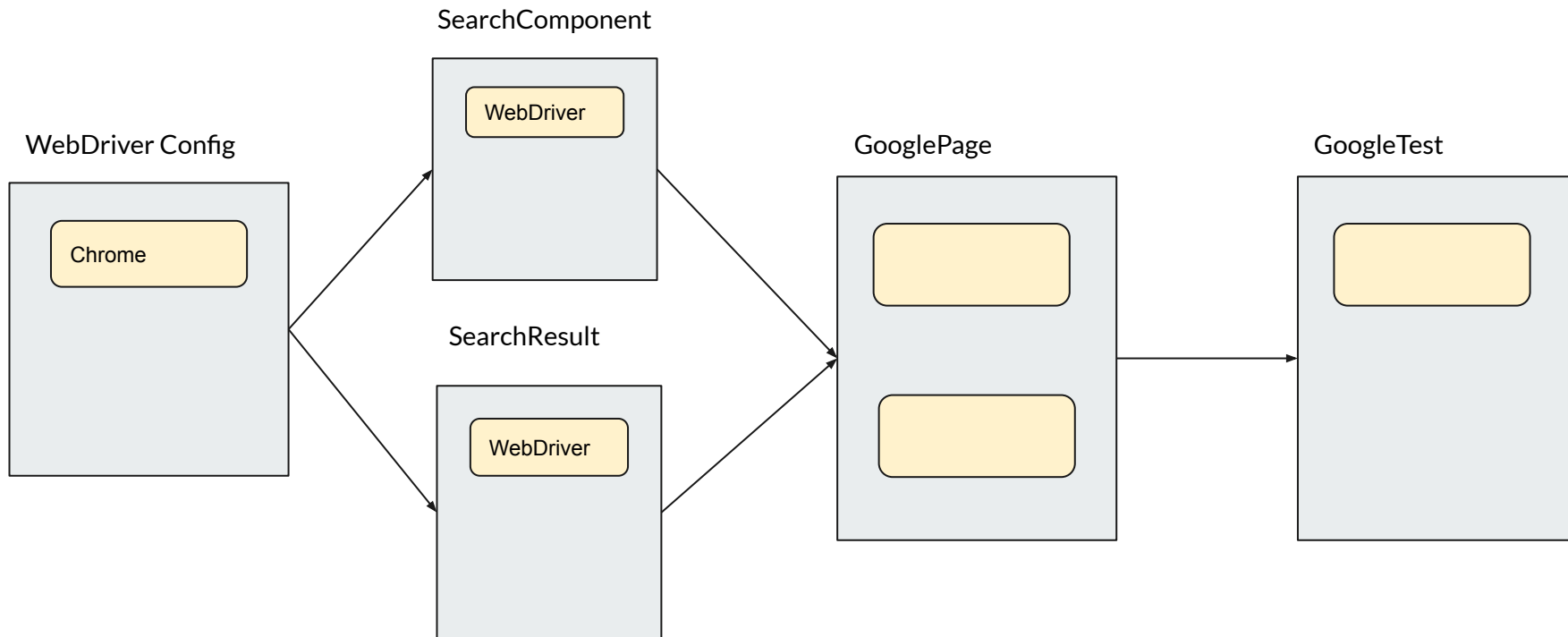
# Spring Boot - Integration With Selenium

# WebDriver - Bean



- Bean
  - FirefoxDriver
  - ChromeDriver

# Page Objects / Fragments



# Lazy Bean



- Beans can be lazy!
- @Lazy should be used along with @Component/@Bean & @Autowired

# Unique Bean!



- Spring should be able to inject a bean uniquely!!!

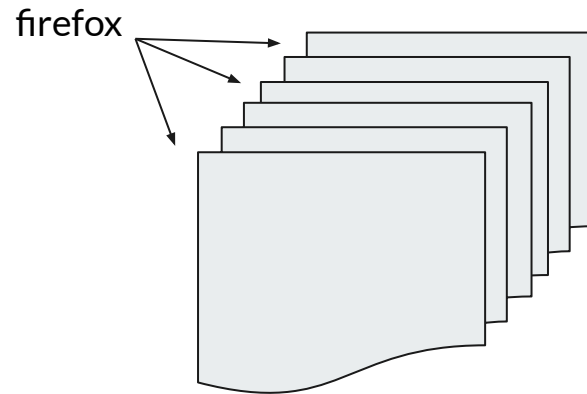
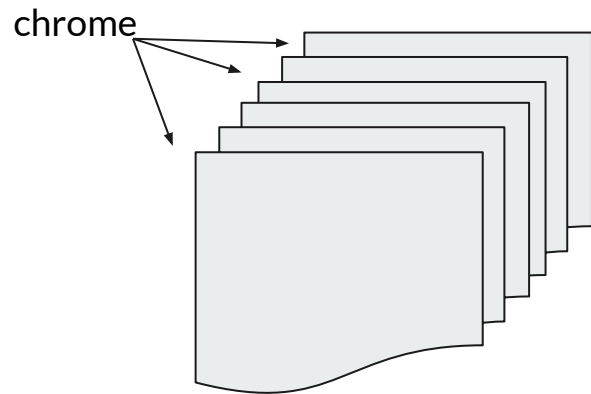


# Unique Bean!



- @Primary
- Conditions
  - @ConditionalOnProperty
  - @ConditionalOnExpression
  - @ConditionalOnMissingBean
- @Qualifier

# WebDriver - Managed By Spring



# WebDriver - Managed By You!



```
@Component
public class WebDriverFactory {

    public WebDriver getDriver(String browser){

        if("chrome".equals(browser)){
            return new ChromeDriver();
        }else{
            return new FirefoxDriver();
        }

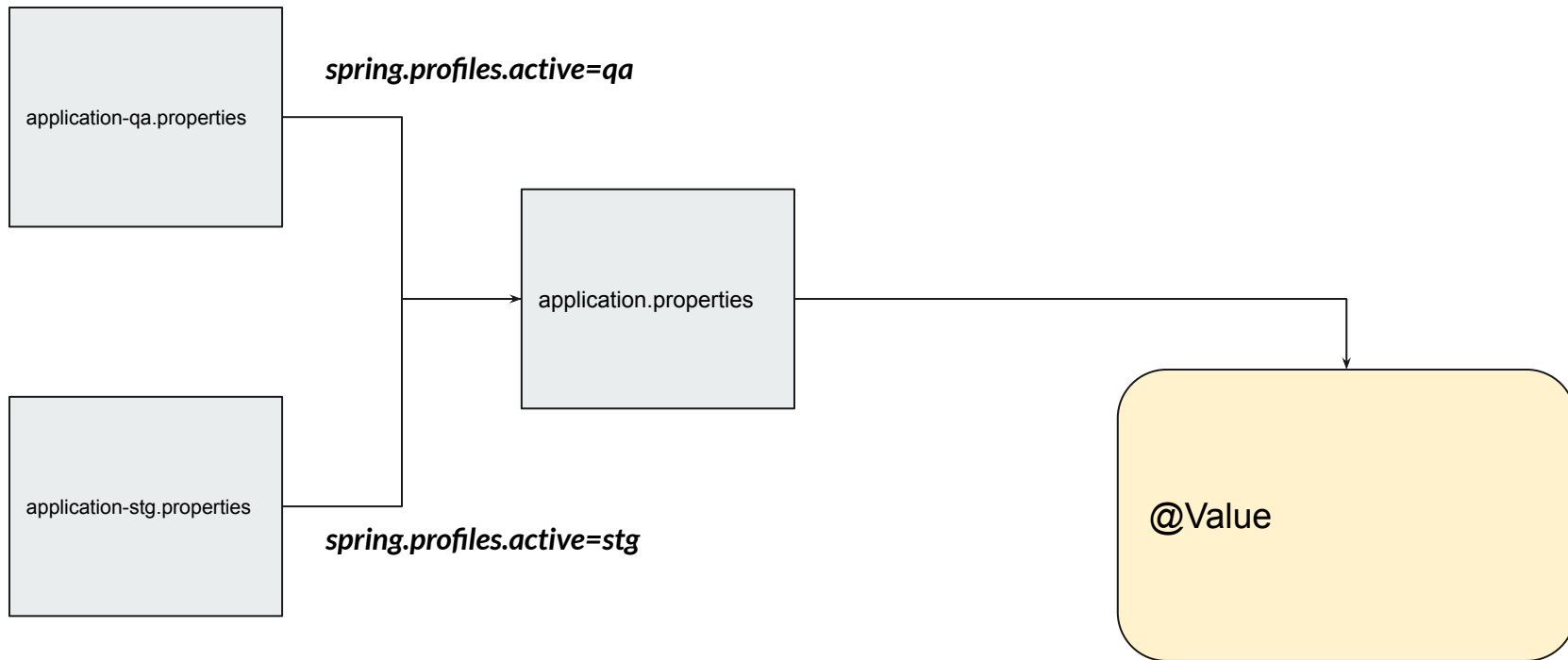
    }

}
```

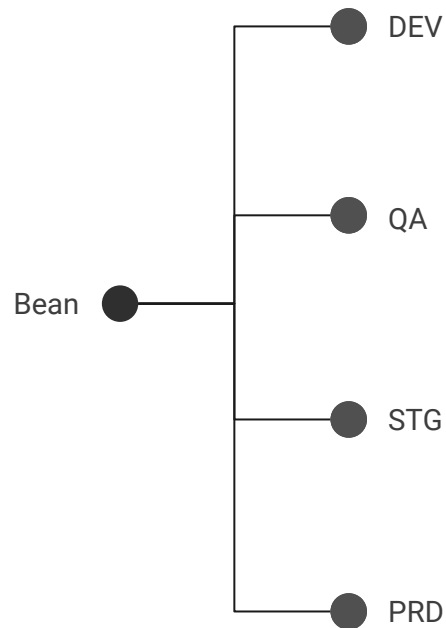
```
@Autowired
private WebDriverFactory driverFactory;

...
...
this.driverFactory.getDriver("chrome");
this.driverFactory.getDriver("firefox");
```

# Profile - Properties



# Profile - @Component / @Bean



# Summary



- **WebDriver** Bean - Managed by Spring
- **WebDriverFactory** - *Managed by Spring & You create the Driver*
- Page Objects / Page Components can be autowired
  - Calendar widget - can be a spring component
- Remember - Spring Boot will crate all the beans before your test starts!
- Bean should be Unique
  - @Primary
  - Conditions
    - @ConditionalOnProperty
    - @ConditionalOnExpression
    - @ConditionalOnMissingBean
  - @Qualifier
  - @Profile

# Summary



- Profiles can be used for environment specific details
  - URL
  - Username/Passwords
- Profiles can be combined
  - Remote + qa
  - Local + stg
- `spring.profiles.active=remote,qa`
- Bean can be `@Lazy`
  - Do not forget to add `@Lazy` in all the places `@Autowired` & `@Bean/@Component`

# Assignments!



- Create a new page object by autowiring WebDriver for this site - <http://newtours.demoaut.com/mercurywelcome.php>
- Practice to run the same test by passing browser / profiles via command line





# Spring Boot - Bean Scope & Parallel Testing

# Bean Scope



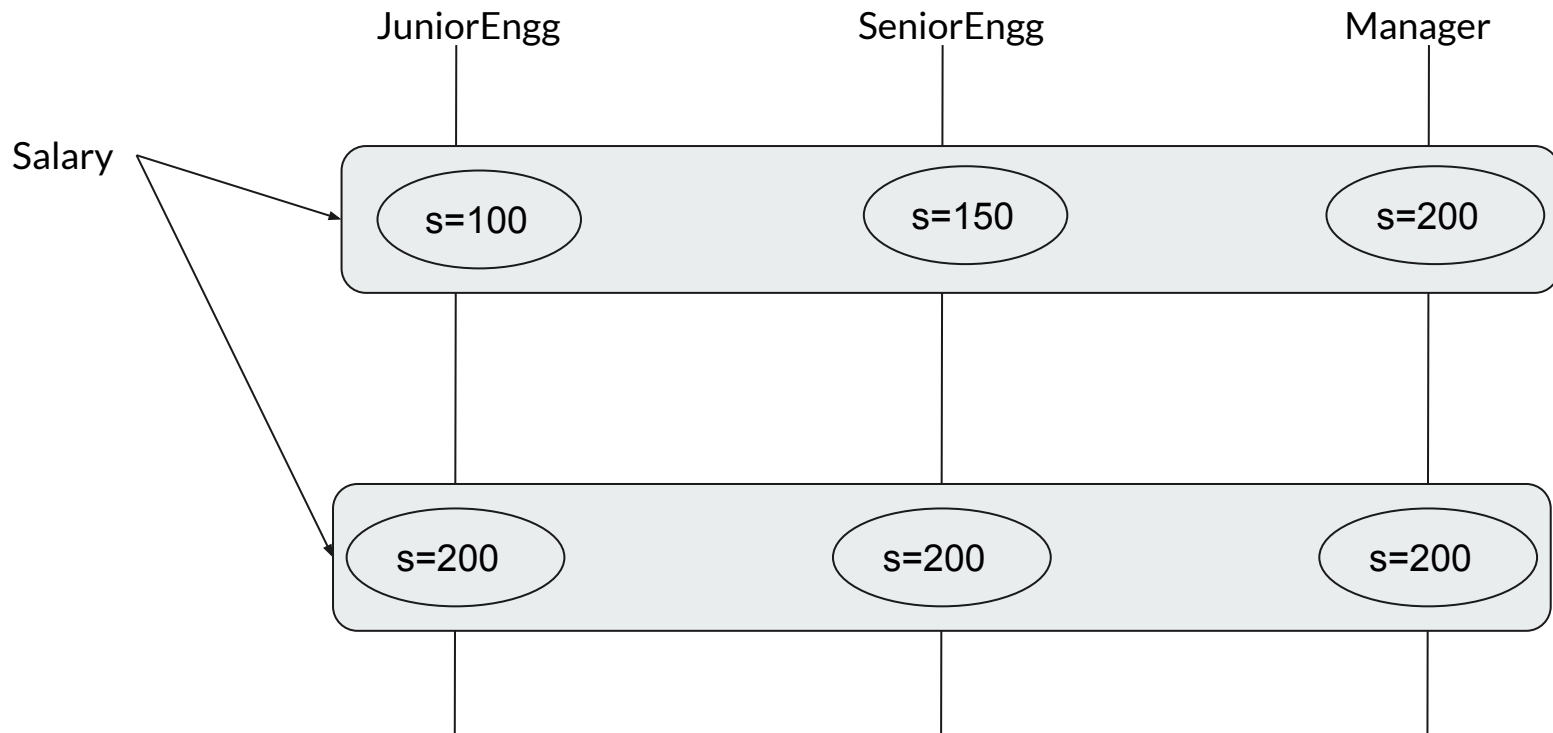
- Singleton (default)
- Prototype

# Singleton



- Only one instance of the Bean
- Shared with all the objects
- Shared with all the threads
- Good for Reporting / Logging
- Avoid mutating instance variables
- Your responsibility to keep it thread safe!

# Singleton

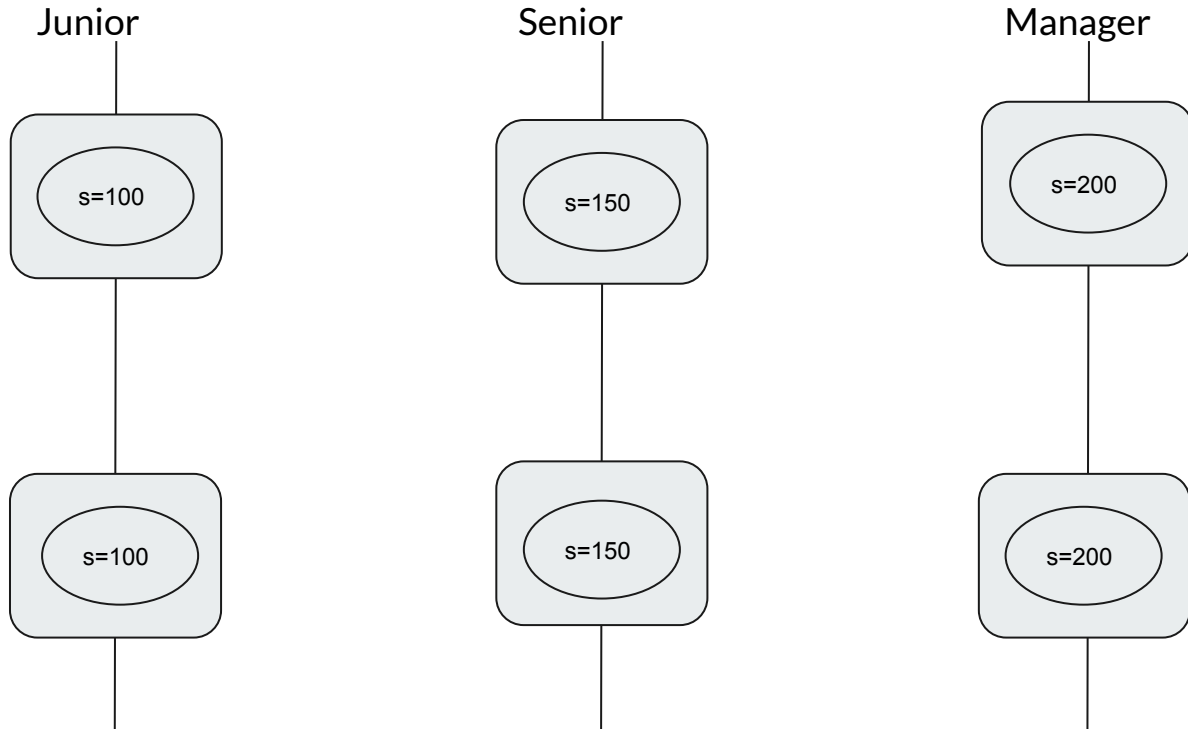


# Prototype



- Fresh instance for all!
- Note: *@PreDestroy* is not invoked!!

# Prototype



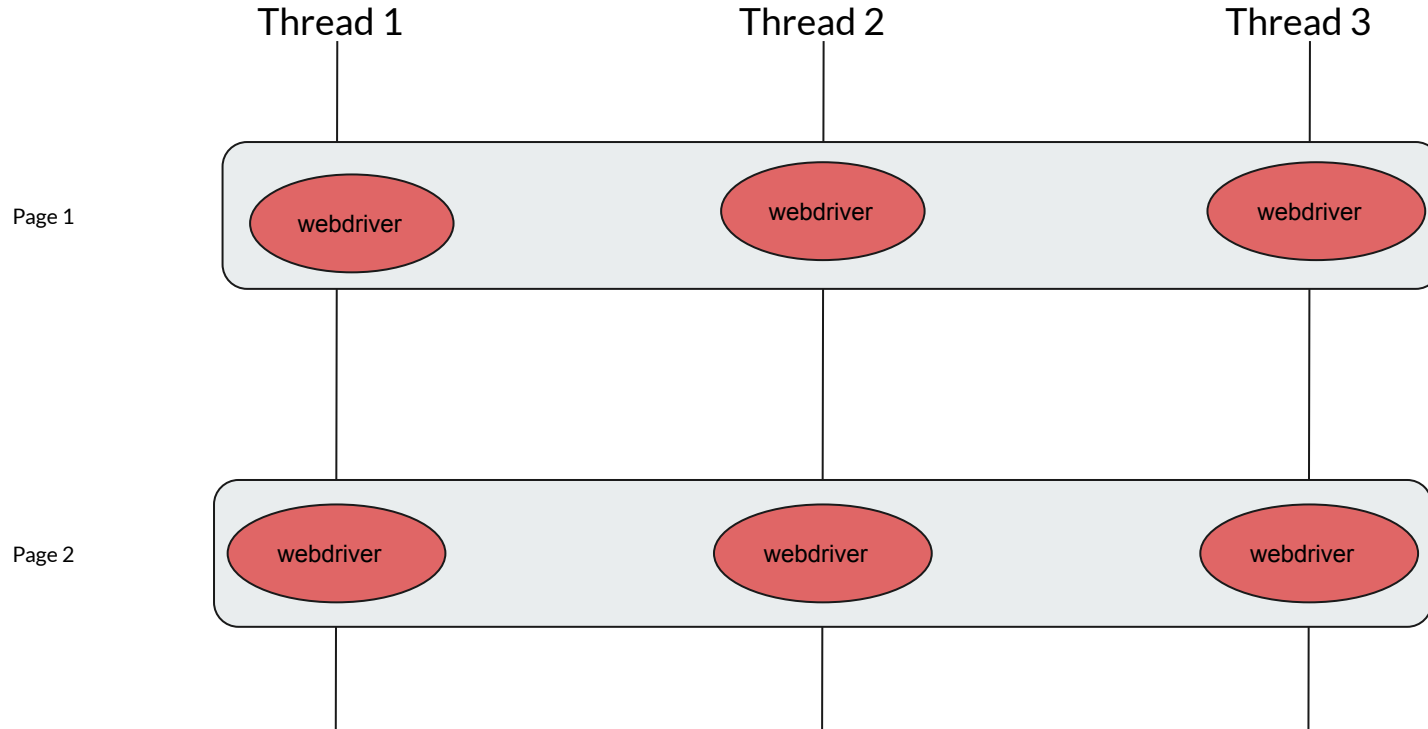
# Custom Annotation



- Group annotations and create custom annotations

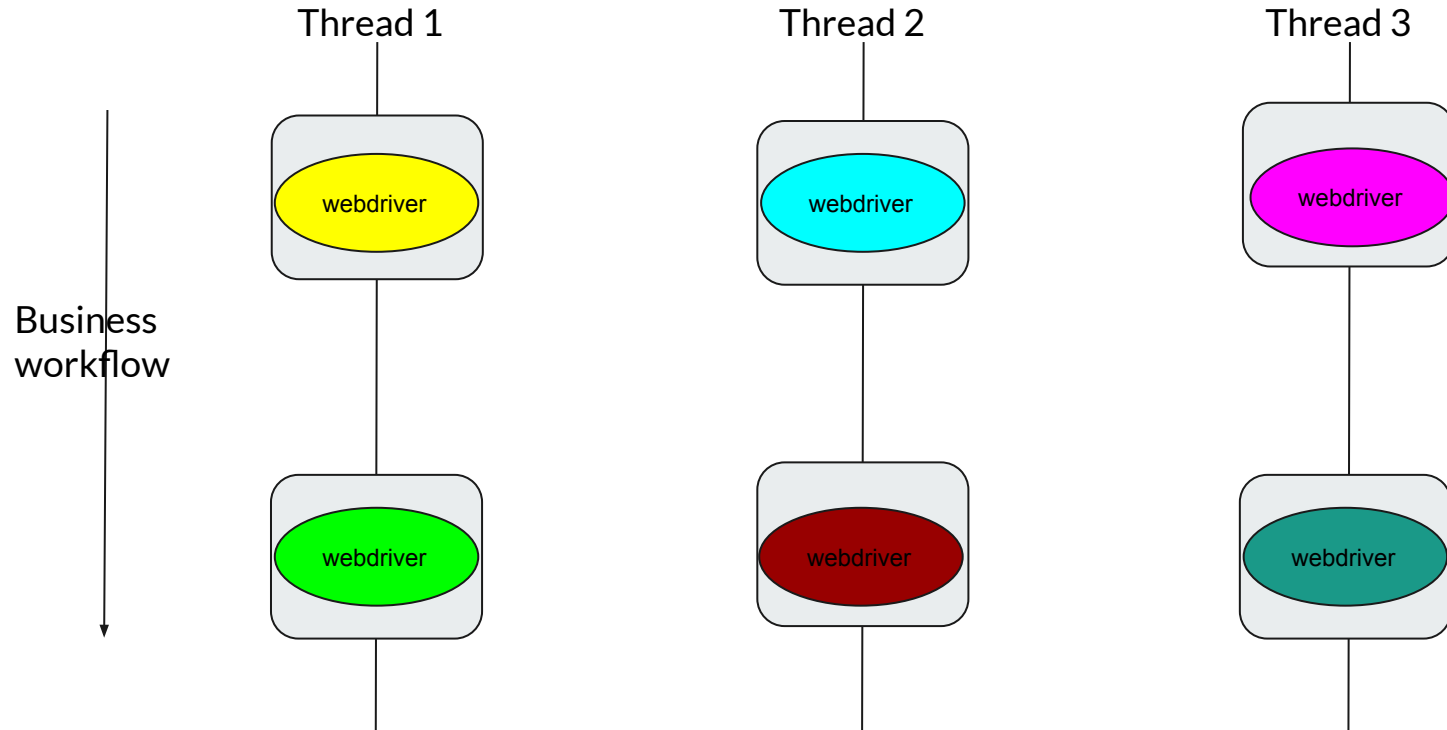
```
@Target({ElementType.TYPE})  
@Retention(RetentionPolicy.RUNTIME)  
@Documented  
@Lazy  
@Component  
@Scope(scopeName = "prototype")  
public @interface Page {  
}
```

# WebDriver - Singleton

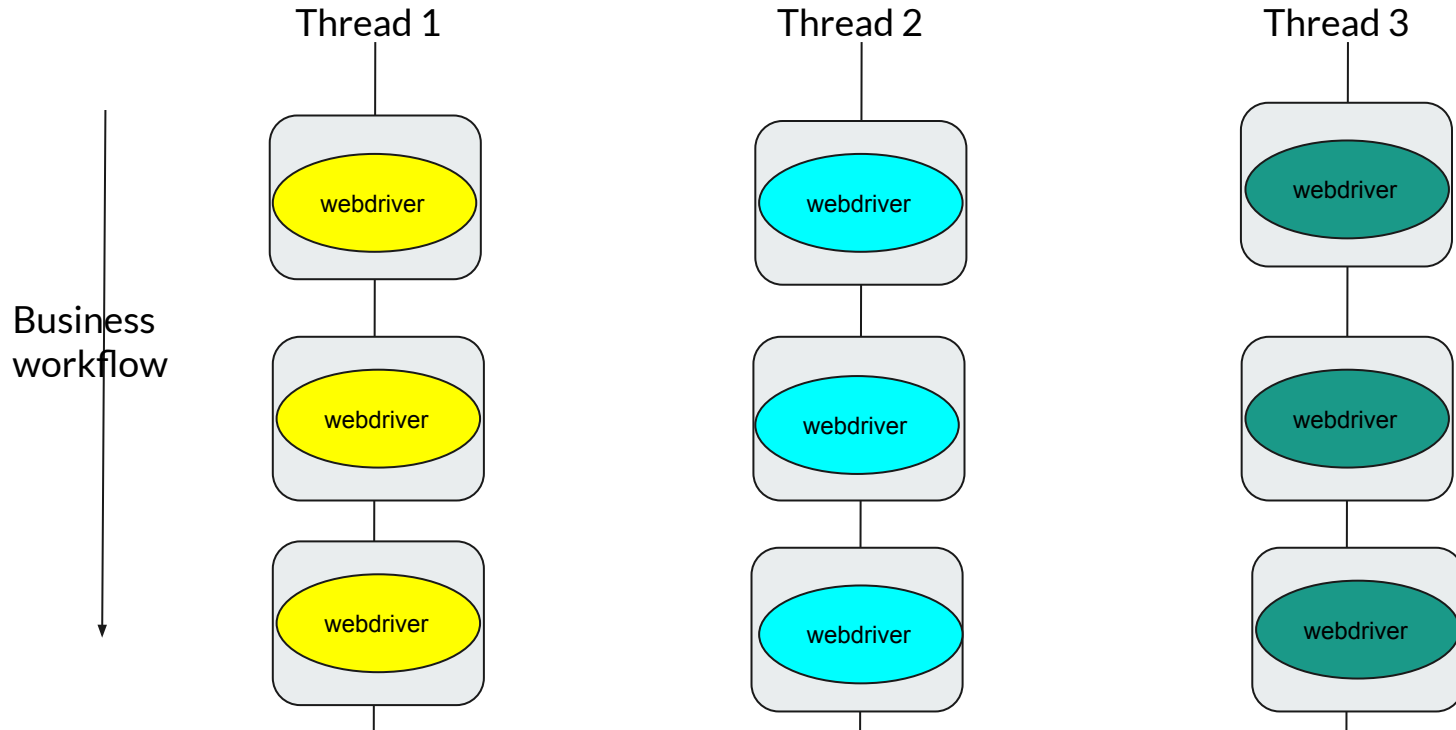




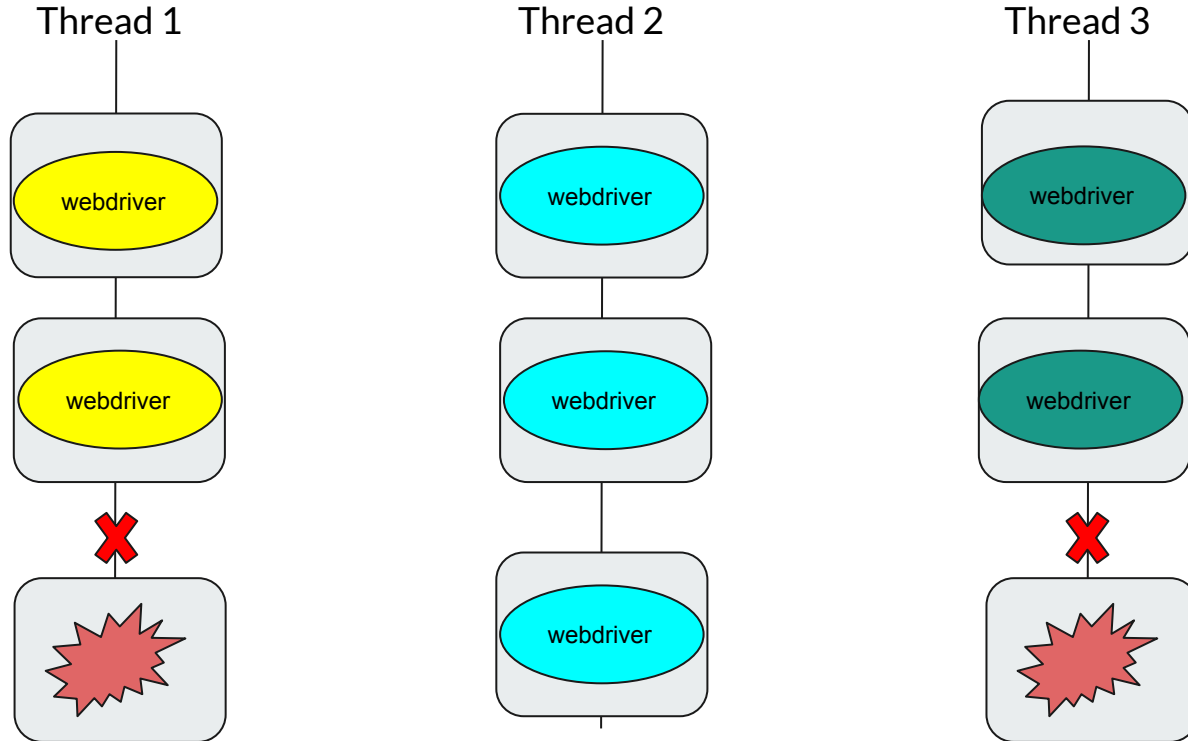
# WebDriver - Prototype



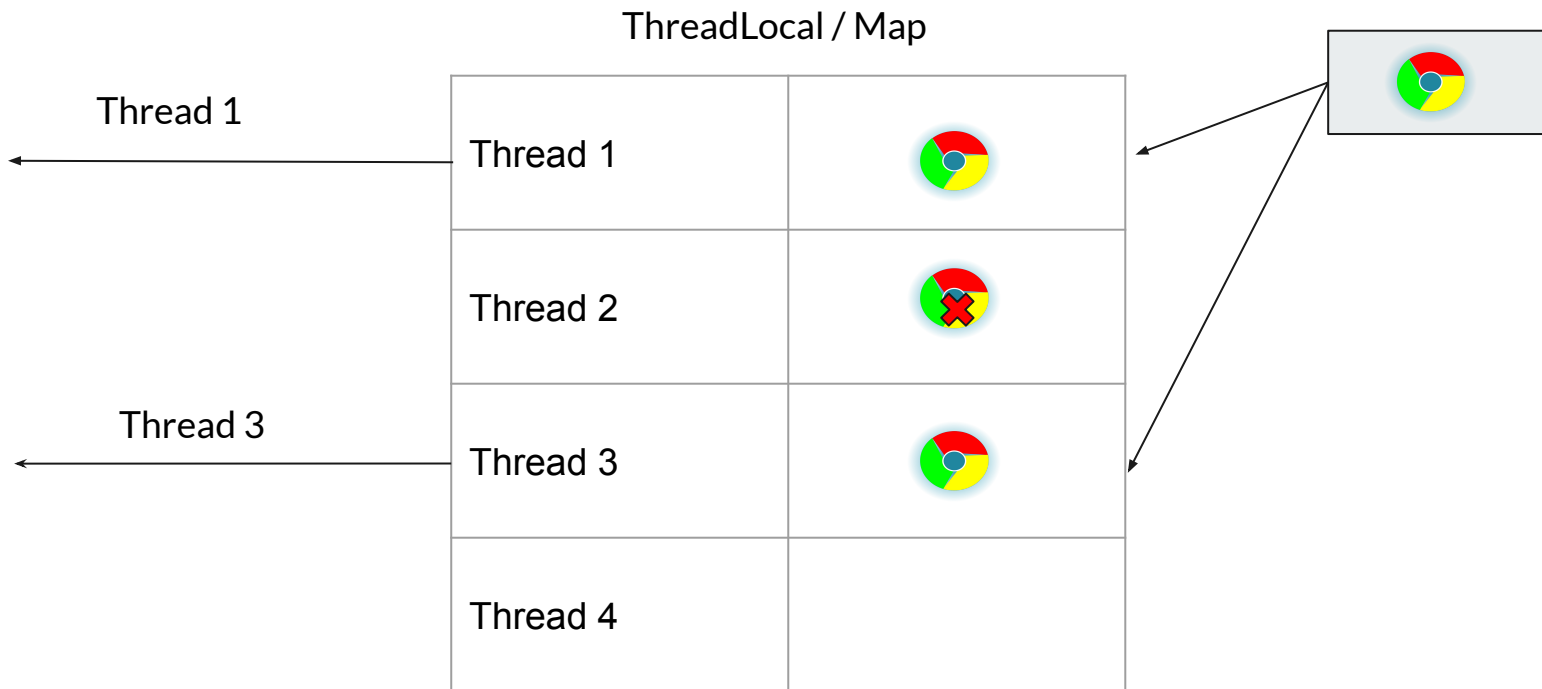
# WebDriver - Thread Scope!



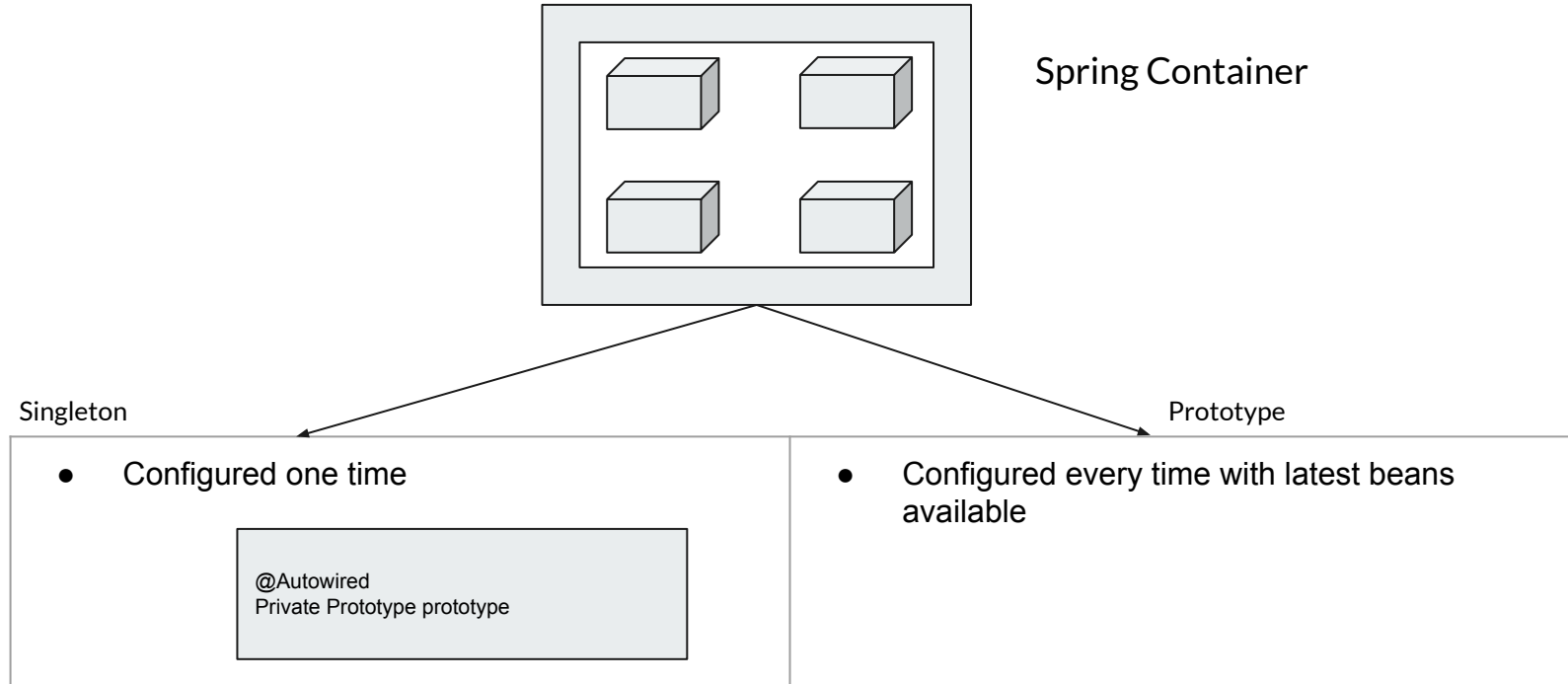
# WebDriver - Thread Scope???



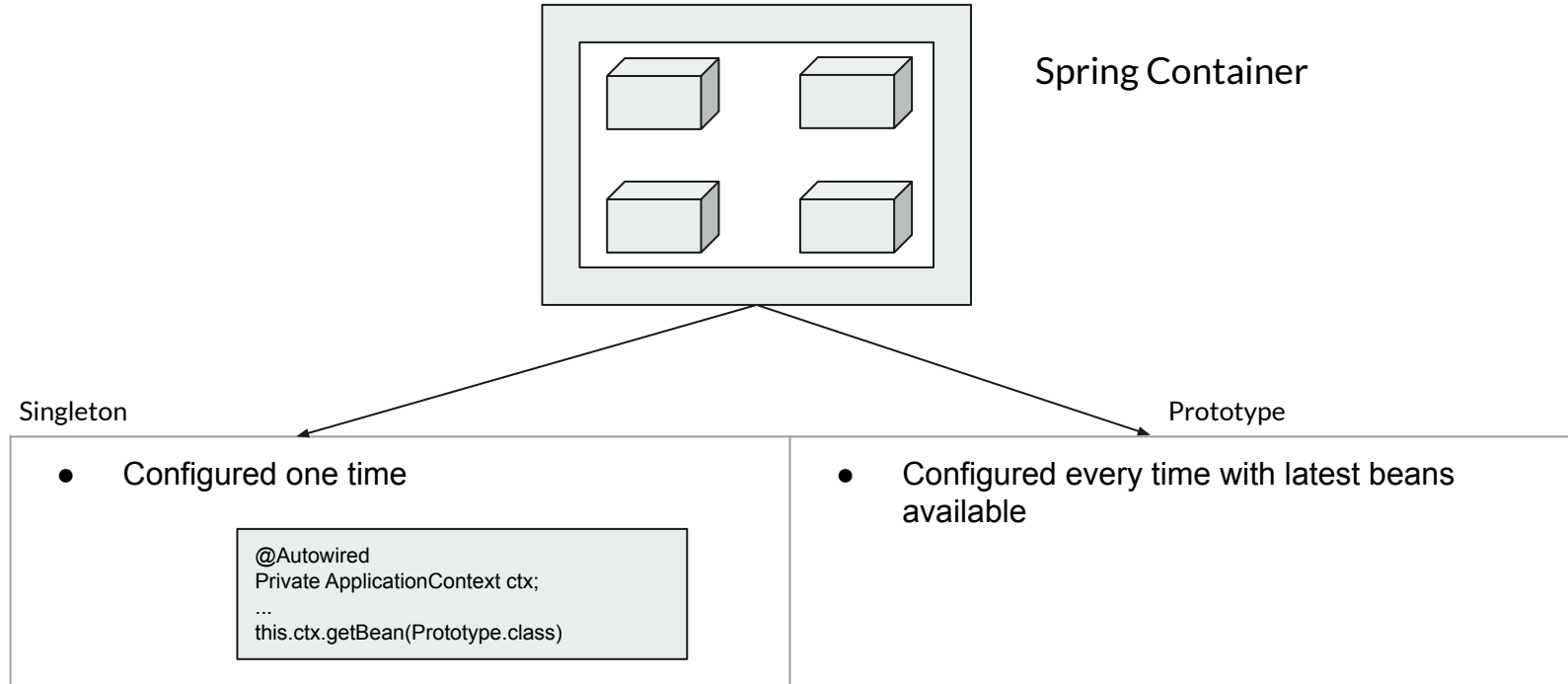
# Custom ThreadScope



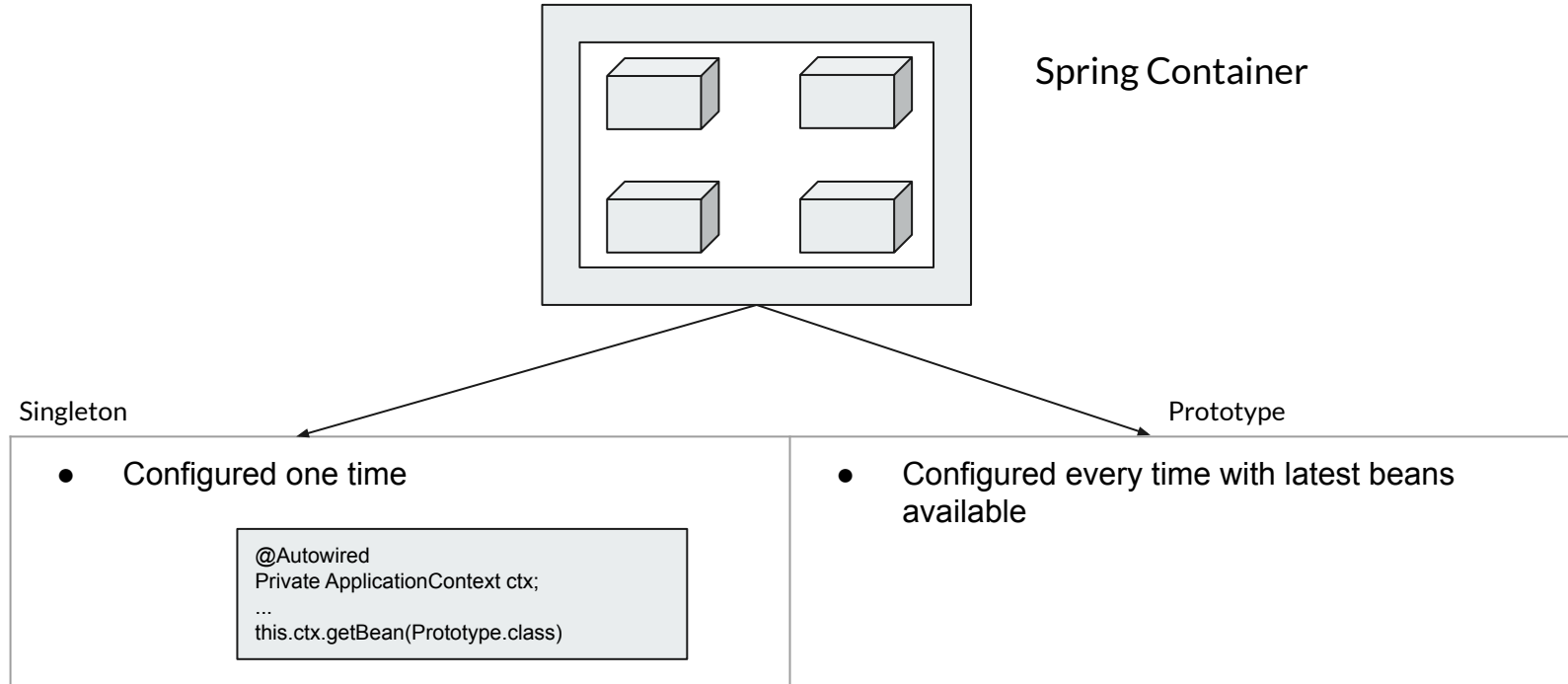
# Application Context



# Application Context



# WebDriverFactory



# Summary



- ***Bean Scope***
  - Singleton
  - Prototype
  - ThreadScope - custom scope. We have to register ourselves!
- Parallel Browser Testing
- ApplicationContext
  - To get bean ourselves
  - `getBean("method", WebDriver.class)`
- ***WebDriverFactory - Managed by Spring & You***
- Custom Annotations
  - *We will do something cool with this!!*
- Kelvin.....!!!!





# **Spring Boot & Selenium - Files & Properties**

# Accessing Resources



Prefix	Example
classpath	classpath:data/udemy.txt
file:	file:c:/some/path/udemy.txt
http:	http://somesite.com/udemy.txt

```
@Value("classpath:features/data.csv")  
private Resource classpathResource;  
  
@Value("https://www.google.com")  
private Resource urlResource;  
  
@Value("file:/home/udemy/data.csv")  
private Resource fileResource;
```

# AWS S3 Resources



- We could easily access S3 resources as well
  - No additional code required
  - We need to include AWS core dependency
  - (You should have the permission)

```
@Value("s3://my-bucket/s3-demo")  
private Resource s3resource;
```

## AWS Core

AWS native services from Spring  
Cloud for AWS



# ResourceLoader



- To get resources at runtime (similar to application context to get beans at runtime)
- To copy files

```
FileCopyUtils.copy(  
    resourceLoader.getResource(url).getInputStream(),  
    Files.newOutputStream(path.resolve(saveAs))  
);
```

# PropertiesLoaderUtils



- To load N number of property files at runtime!

# @PropertySource



- To read any additional property files which are not part of application.properties.
- Instead of injecting values in multiple places, just inject once!
- Properties into a Java object!

```
@Component
@PropertySource("language/${app.locale}.properties")
public class FlightAppDetails {





    @Value("${flight.app.url}")
    private String url;


    @Value("${flight.app.labels}")
    private List<String> labels;

    public String getUrl() {
        return url;
    }

    public List<String> getLabels() {
        return labels;
    }
}
```

# Localization



 Search flights	 Manage booking/Check in	 What's on your flight	 Flight status
--	---	---	---

 Flight

Vacations package





[Log in to earn and spend Skywards Miles](#) [Advanced search: multi-city, promo codes, partner airlines](#) >

Departure Airport

 Houston (IAH) 



Arrival Airport

Continue

 Rechercher des vols	 Gérer une réservation/s'enregistrer	 Services disponibles sur votre vol	 Statut du vol
---	---	--	---

[Connectez-vous et dépensez des Miles Skywards](#) [Recherche avancée : multi-destinations, codes promo, compagnies aériennes partenaires](#) >

Aéroport de départ

 Houston (IAH) 


Aéroport d'arrivée

Continuer

# Locale Properties




en-us.properties



```
label.username=Username  
label.password=Password
```

es.properties



```
label.username=Nombre de Usuario  
label.password=Contraseña
```




# @TestPropertySource



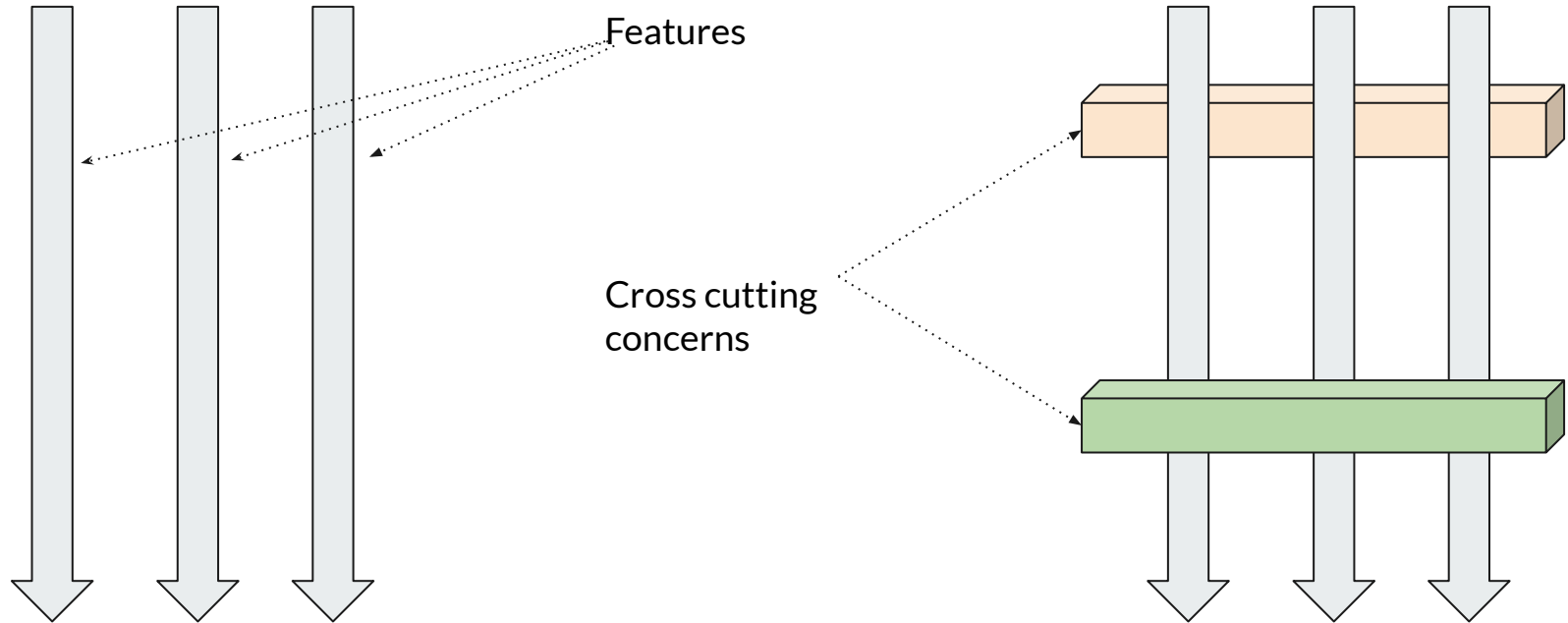
- To set test specific properties

```
@TestPropertySource(properties = {"app.locale=id", "browser=firefox"})  
public class IdTest extends FlightTest{  
}
```

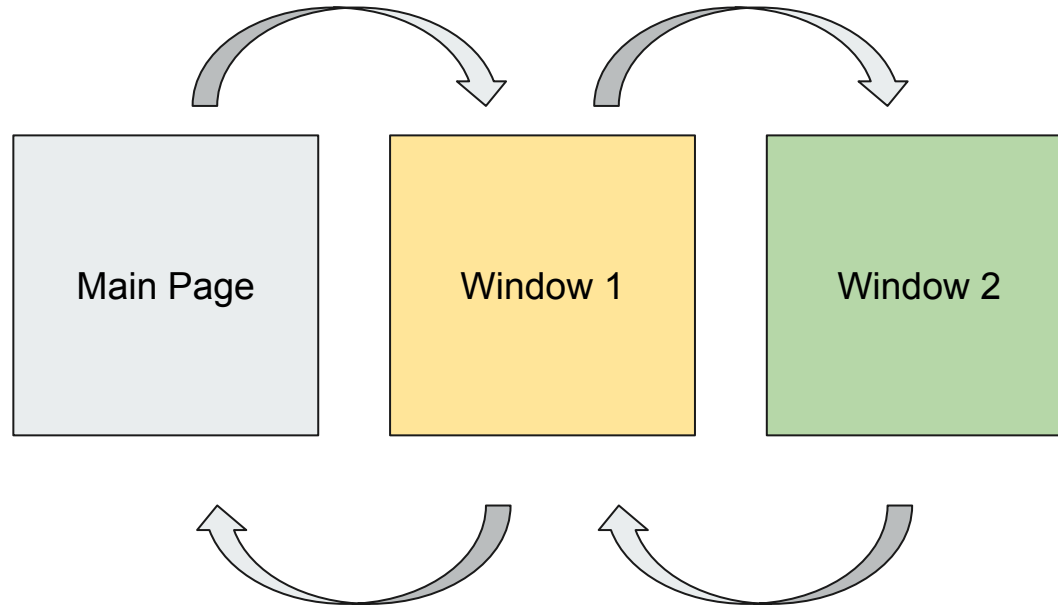


# **Spring Boot & Selenium - Aspect Oriented Programming**

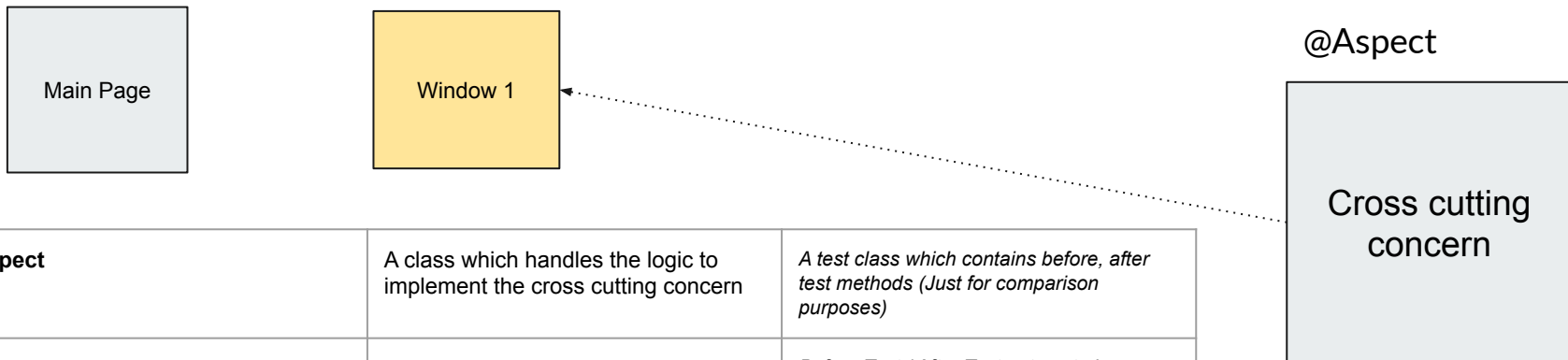
# Cross Cutting Concern



# Aspect Oriented Programming (AOP)



# Aspect Oriented Programming (AOP)



<b>Aspect</b>	A class which handles the logic to implement the cross cutting concern	<i>A test class which contains before, after test methods (Just for comparison purposes)</i>
<b>Advice</b>	The action taken by the Aspect / a method in the Aspect	<i>Before Test / After Test - steps to be performed</i>
<b>JoinPoint</b>	A point of execution / an event in the program - like method invocation / exception being thrown / field being modified.	<i>@Test method execution</i>
<b>PointCut</b>	At which JoinPoint we need the advice!	

```
@Before("@target(window) && within(com.udemy.spring.spring selenium..*)" )  
public void before(Window window){
```

# AOP - Sample PointCut Expressions



When To Invoke	@Before the method execution @After the method execution @Around the method execution
A target class with annotation	@target(com.udemy.spring.springselenium.kelvin.annotation.Window)
A target class with annotation	@target(window) (here we assume that the method has a parameter called window. The type of the window would be used as target class)
A method with the annotation	@annotation(com.udemy.spring.Something)
A method with the annotation	@annotation(something)

Complete Doc: <https://docs.spring.io/spring/docs/2.0.x/reference/aop.html>

# Summary



- *AOP*
- Cross cutting concern
  - Separate core and secondary concerns
- Window Switch
- Frame Switch
- Taking Screenshot
- Logging
- Re-execute the method if the result is not as expected
- Do a data setup / make a REST call before method/after method invocation



# Spring Boot - Data JPA

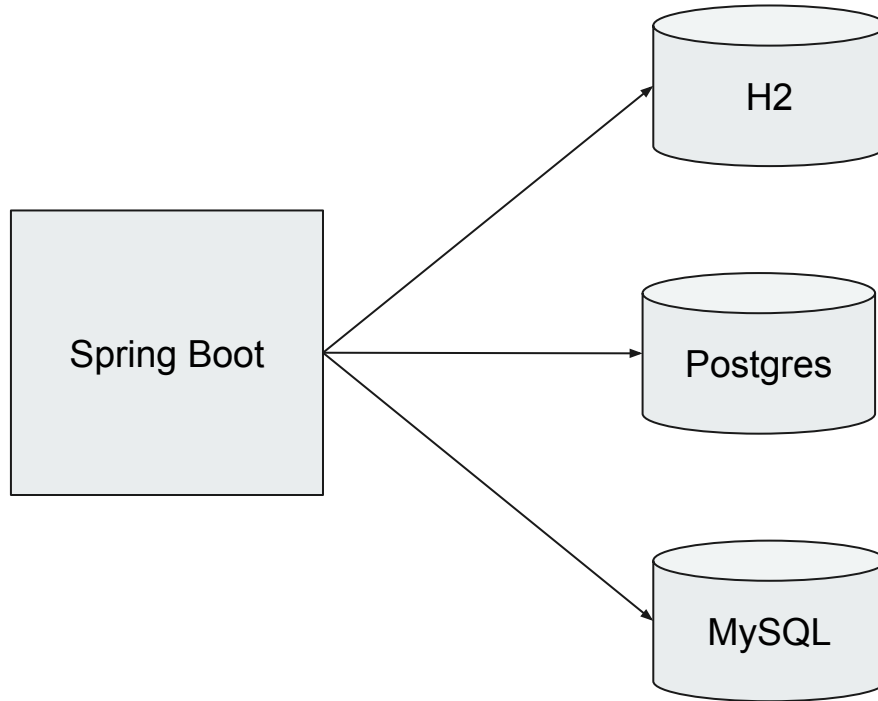


# Spring Data JPA



- JPA - Java Persistence API
- Provides a nice abstraction to connect to various SQL/NoSQL Databases
- Do not write a single line of SQL (Hibernate lib)
- Select / Insert / Update / Delete
- Where / Aggregation

# Spring Data JPA

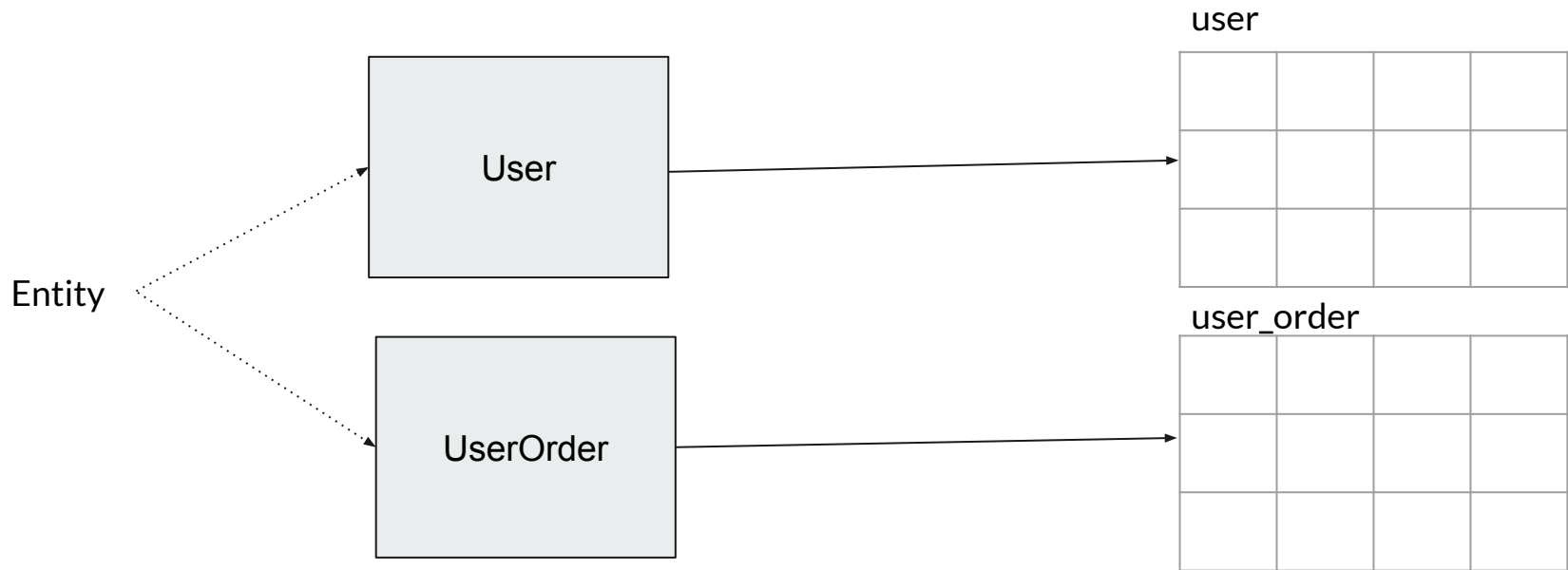


# Spring Data In Testing

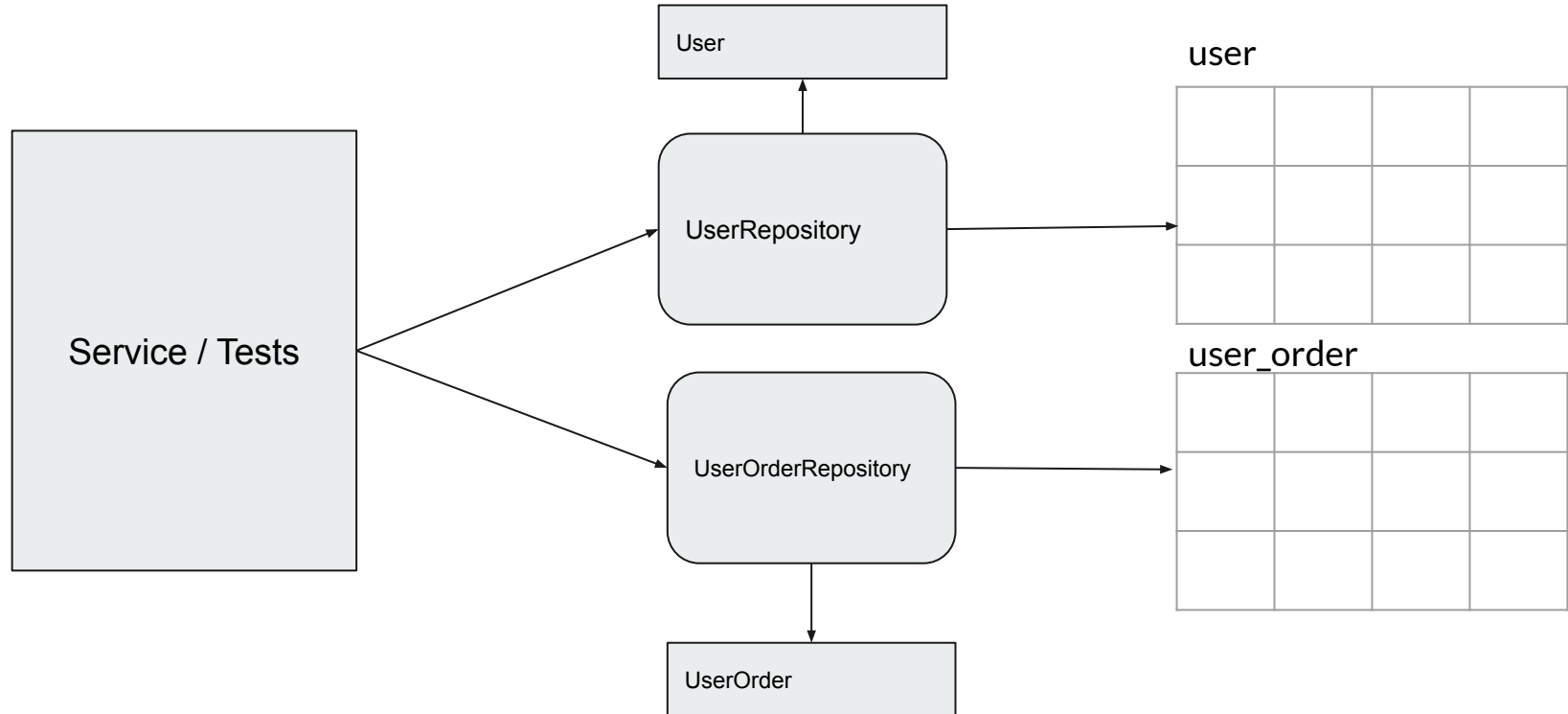


- Data Driven Tests
- Query test data
- Let spring handle routine activities!
- CSV → Table

# Spring Data JPA - Basics



# Spring Data JPA - Basics



# Spring Data JPA - Naming Conventions



DB Table / Column Name	Java Class / Field Name
user_visa (table)	UserVisa (class)
first_name	firstName
last_name	lastName
dob	dob

# Spring Data JPA - Summary



- JPA - Java Persistence API
- Provides a nice abstraction to connect to various SQL/NoSQL Databases
- CRUD w/o any SQL
- Data driven tests
- To treat CSV like a table.

# Spring Data JPA - Query

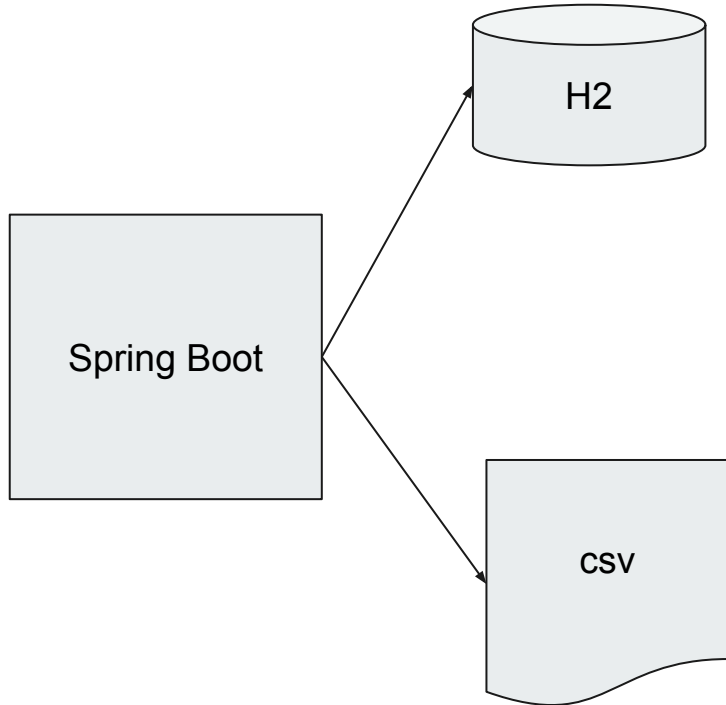
Table 3. Supported keywords inside method names

Keyword	Sample	JPQL snippet
And	findByLastnameAndFirstname	... where x.lastname = ?1 and x.firstname = ?2
Or	findByLastnameOrFirstname	... where x.lastname = ?1 or x.firstname = ?2
Is , Equals	findByFirstname , findByFirstnameIs , findByFirstnameEquals	... where x.firstname = ?1
Between	findByStartDateBetween	... where x.startDate between ?1 and ?2
LessThan	findByAgeLessThan	... where x.age < ?1
LessThanEqual	findByAgeLessThanEqual	... where x.age <= ?1

1



# Spring Data JPA - H2



```
spring.datasource.url=jdbc:postgresql://10.11.12.13:5432/appdb  
spring.datasource.username=vinsguru  
spring.datasource.password=admin
```

# Spring Data JPA - DML

---

- To update
- To Delete
- To insert

```
u.setComments("new comment");
```

```
this.repository.deleteById(85);
```

```
User newUser = new User();  
newUser.setFirstName("vinoth");  
this.repository.save(newUser);
```

# Spring Data JPA - Curious to know more!?

## Spring Data JPA Using Hibernate

Learn and use the most popular ORM Frameworks in easy steps

BESTSELLER



4.4 (803 ratings)

4,243 students enrolled

Created by Bharath Thippireddy

Last updated 5/2020

English

English, Polish [Auto-generated]



Preview this course

### What you'll learn

- ✓ Master the concepts of ORM, Spring Data JPA and Hibernate
- ✓ Configure auto generated IDs for the Primary Key fields
- ✓ Load data from database without implementing any code or SQL
- ✓ Execute native SQL queries from your Spring Data Application
- ✓ Learn the different types of Hibernate
- ✓ Perform CRUD operations against a database with two simple steps
- ✓ Realize the power of Spring Data Finder methods
- ✓ Learn and use JPQL - Java Persistence Query Language
- ✓ Use Paging and Sorting
- ✓ Implement Component Mapping

[See more](#)

**\$11.99** ~~\$29.99~~ 60% off

🕒 1 day left at this price!


Add to cart

Buy now

30-Day Money-Back Guarantee

This course includes

- 📺 8 hours on-demand video
- 📄 1 article
- 📁 16 downloadable resources
- 🔒 Full lifetime access
- 📱 Access on mobile and TV



# Spring Boot & Selenium - Logging

# Logging



- We can create logger for any class.
- Default - console.output

```
private static final Logger logger = LoggerFactory.getLogger(GooglePage.class);|
```

# Logging Levels



- ERROR
- WARN
- INFO
  - *DEBUG*
  - *TRACE*

# Logger Properties



- `logging.level.root=INFO`
- `logging.level.com.udemy=WARN`
- `logging.file.name=test.log`
- `logging.file.max-size=10MB`

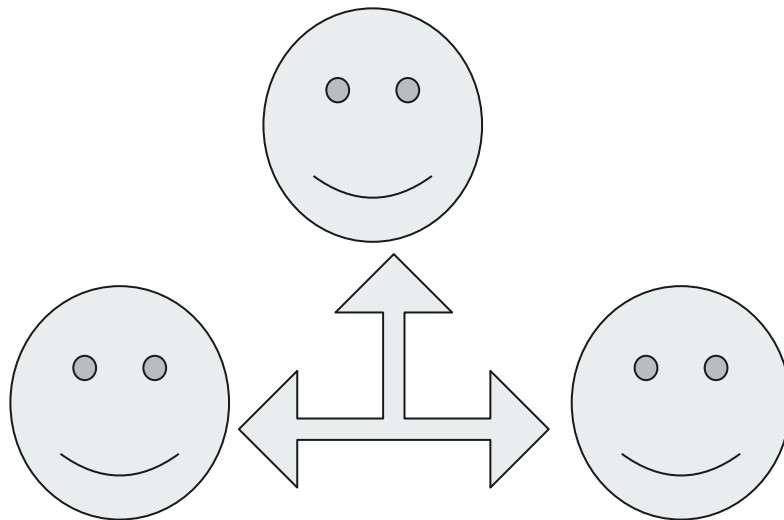
<https://docs.spring.io/spring-boot/docs/2.1.9.RELEASE/reference/html/boot-features-logging.html>



# **Spring Boot, Selenium & Cucumber Integration**



# Behavior Driven Development




# Behavior Driven Development



- Feature - a business functionality (Login feature / Search feature)
- Scenario - a specific case within the feature. Feature will have at least 1 scenario
  - Check for the successful login with valid user credentials
- Given / When / Then / And - Steps to form a meaningful sentence
  - **Given** I am on the google site
  - **When** I enter 'spring boot' as a keyword
  - **And** I click on the search button
  - **Then** I should see at least 2 results
- Scenario Outline - data driven scenarios with placeholders
  - **Given** I am on the google site
  - **When** I enter <keyword> as a keyword
  - **And** I click on the search button
  - **Then** I should see at least <resultcount> results
- Gherkin - a plain english language to write feature files

keyword	resultcount
Spring boot	5
selenium	3

# Cucumber - Tags

`-Dcucumber.options`  `-Dcucumber.filter.tags="@google"`

To run only @google	@google
either @google or @visa	@google or @visa
@google scenarios which are also marked as @visa	@google and @visa
All scenarios which are not marked as @visa	not @visa
Some advanced	(@front or @backend) and (not @slow)

# Cucumber - Data Table → UserRepository

Examples:

fromCountry	toCountry	dateOfBirth	firstName	lastName	email	phone	comments
Isle of Man	Mali	2011-05-31	Kraig	Wiza	Kraig@nobody.com	1-000-884-1373	comment1
Lithuania	Mexico	2001-01-01	Houston	Kertzmann	Houston@nobody.com	284.864.6580	
Somalia	Greece	2004-07-02	Ruthie	Stamm	Ruthie@nobody.com	1-209-813-9712	comment2
Christmas Island	French Southern Territories	2019-04-05	Shonna	Nolan	Shonna@nobody.com	(162) 387-0305	

```
public class VisaSteps {  
  
    @LazyAutowired  
    private VisaRegistrationPage registrationPage;  
  
    @LazyAutowired  
    private UserRepository repository;  
}
```

# Summary



- Integrated with Cucumber
- **Dependency Injection** in the Step Definitions
- Reporting
- Failed tests screenshots
- Parallel scenarios run
- Filtering tags
- Cucumber Runner & Command line execution
- Closing browsers

```
@LazyAutowired  
private ScreenshotService screenshotService;
```

```
@LazyAutowired  
private GooglePage googlePage;
```

# JVM Shutdown Hook



```
@ThreadScopeBean
@ConditionalOnMissingBean
public WebDriver chromeDriver(){
    WebDriverManager.chromedriver().version("77.0.3865.40").setup();
    WebDriver driver = new ChromeDriver();
    Runtime.getRuntime().addShutdownHook(new Thread(driver::quit));
    return driver;
}
```



# **Spring Boot & Selenium - Miscellaneous**

# Docker Integration



- You need to update the <build> section in the pom file with appropriate plugins
- *CucumberRunner* will NOT be used for running the tests.
- We would be using below command
- Parameterize glue, tags, threads, browser, profiles etc

```
java -cp spring-boot-selenium.jar:spring-boot-selenium-tests.jar:libs/* \  
  -Dbrowser=chrome \  
  io.cucumber.core.cli.Main \  
  classpath:features \  
  --glue com.udemy.spring.springselenium.bdd \  
  --tags "@google" \  
  --threads 2
```

```
.  
..  
classes  
generated-sources  
generated-test-sources  
libs  
maven-archiver  
maven-status  
spring-boot-selenium.jar  
spring-boot-selenium-tests.jar  
test-classes
```





# Thank You!