ITWS Final exam

Spring projects

Student: Sara Paunović   
ID: 2019/230075  
Email: sara.paunovic.19@singimail.rs

Student: Stefan Cliff   
ID: 2019/230449  
Email: stefan.cliff.19@singimail.rs

Contents

[Introduction 3](#_Toc99548834)

[Technology 4](#_Toc99548835)

[Spring beans and dependacny injection; Constructo and setter injection 4](#_Toc99548836)

[Collection Injection 5](#_Toc99548837)

[Auto-wiring 5](#_Toc99548838)

[Implementation 6](#_Toc99548839)

[Project #1 6](#_Toc99548840)

[beans.xml 6](#_Toc99548841)

[PC.java 7](#_Toc99548842)

[User.java 7](#_Toc99548843)

[Server\_App1.java 8](#_Toc99548844)

[app1Console.java 8](#_Toc99548845)

[Project #2 9](#_Toc99548846)

[menu.xml 9](#_Toc99548847)

[Customer.java 10](#_Toc99548848)

[Reservation.java 11](#_Toc99548849)

[Menu.java 12](#_Toc99548850)

[App2.java 13](#_Toc99548851)

[App2\_Console 14](#_Toc99548852)

[Project #3 15](#_Toc99548853)

[beans.xml 15](#_Toc99548854)

[Car.java 16](#_Toc99548855)

[App3.java 17](#_Toc99548856)

[App3\_Console 17](#_Toc99548857)

[Conclusion & Future upgrades 18](#_Toc99548858)

# Introduction

Our final project is one that is actually a collection of three different projects that aren’t actually connected really by logic. We did it this way as we wanted each point/request/checkbox of the requirements for the final exam to be crossed off.

The three projects are:

1. The first project is a kind of server log type of system;   
   (*Spring beans and dependency injection, both constructor and setter injection*)
2. The second serves as a restaurant tally-up for the customers and their orders;

(*Injecting collections*)

1. The third one works similarly to a garage stock style of application, where it writes out what vehicles exist and their colour.

(*Auto-wiring*)

# Technology

## Spring beans and dependency injection; Constructor and setter injection

Dependency injection is an approach to implement loose coupling among the classes in an application.

* Dependency: An object usually requires objects of other classes to perform its operations. We call these objects dependencies.
* Injection: The process of providing the required dependencies to an object.

Thus dependency injection helps in implementing inversion of control (IoC). This means that the responsibility of object creation and injecting the dependencies is given to the framework (i.e. Spring) instead of the class creating the dependency objects by itself.

We can implement dependency injection with:

* constructor-based injection
* setter-based injection

Difference between Setter and Constructor Injection in Spring framework:

* In setter injection strategy, we trust the Inversion of control (IoC) container that it will first create the bean first but will do the injection right before using the bean using the setter methods. And the injection is done according to your configuration. If you somehow misses to specify any beans to inject in the configuration, the injection will not be done for those beans and your dependent bean will not function accordingly when it will be in use!
* But in constructor injection strategy, container imposes (or must impose) to provide the dependencies properly while constructing the bean. This was addressed as "container-agnostic manner", as we are required to provide dependencies while creating the bean, thus making the visibility of dependency, independent of any IoC container.

## Collection Injection

If we want to pass plural values like Java Collection types such as List, Set, Map, and Properties, Spring offers four types of collection configuration elements:

1. <list>

This helps in wiring i.e. injecting a list of values, allowing duplicates.

1. <set>

This helps in wiring a set of values but without any duplicates.

1. <map>

This can be used to inject a collection of name-value pairs where name and value can be of any type.

1. <props>

This can be used to inject a collection of name-value pairs where the name and value are both Strings.

## Auto-wiring

Spring provides a way to automatically detect the relationships between various beans.

This can be done by declaring all the bean dependencies in Spring configuration file.

So, Spring is able to utilize the Bean-Factory to know the dependencies across all the used beans. The XML-configuration-based auto-wiring functionality has several modes, and we'll use two of them:

* byName: The byName mode injects the object dependency according to name of the bean. In such a case, the property and bean name should be the same.   
  It internally calls the setter method.
* byType: The byType mode injects the object dependency according to type. So it can have a different property and bean name. It internally calls the setter method.
* constructor: The constructor mode injects the dependency by calling the constructor of the class. It calls the constructor having a large number of parameters.

# Implementation

Originally when we began working on our project we struggled to come up with an idea that would cover all the points we need to have, so instead we created separate projects for each requirement. That way we could make them the star of their own project.

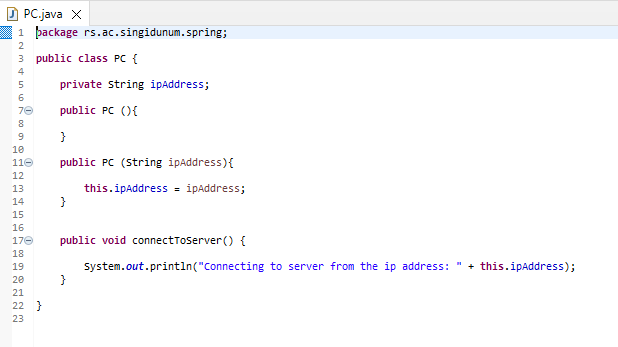
## Project #1

This project was created with the idea of showing some form of communication between a PC and a server reading and monitoring the connected devices and their users.

### beans.xml



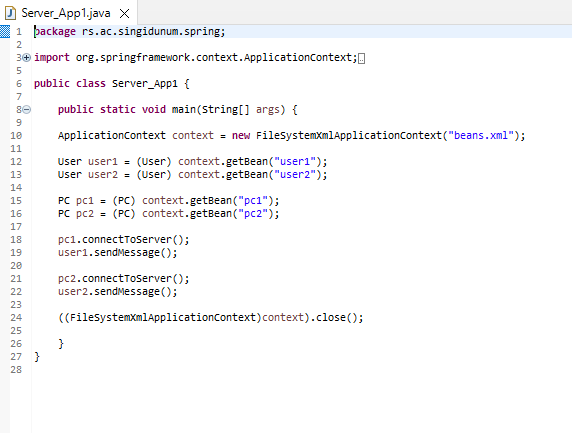
### PC.java



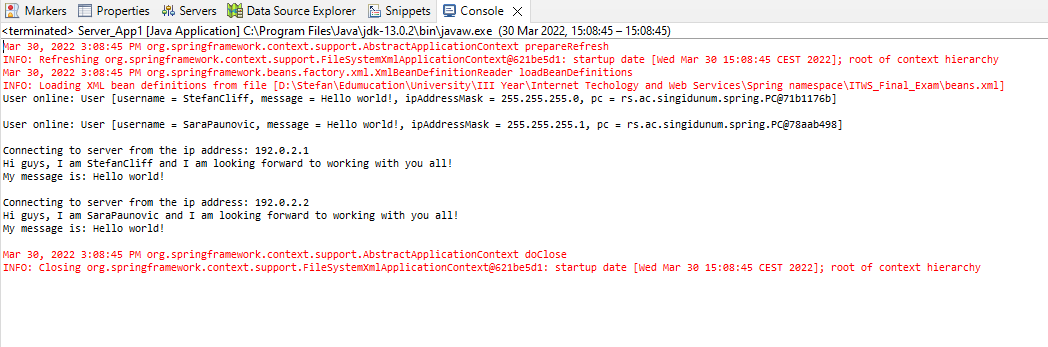
### User.java



### Server\_App1.java



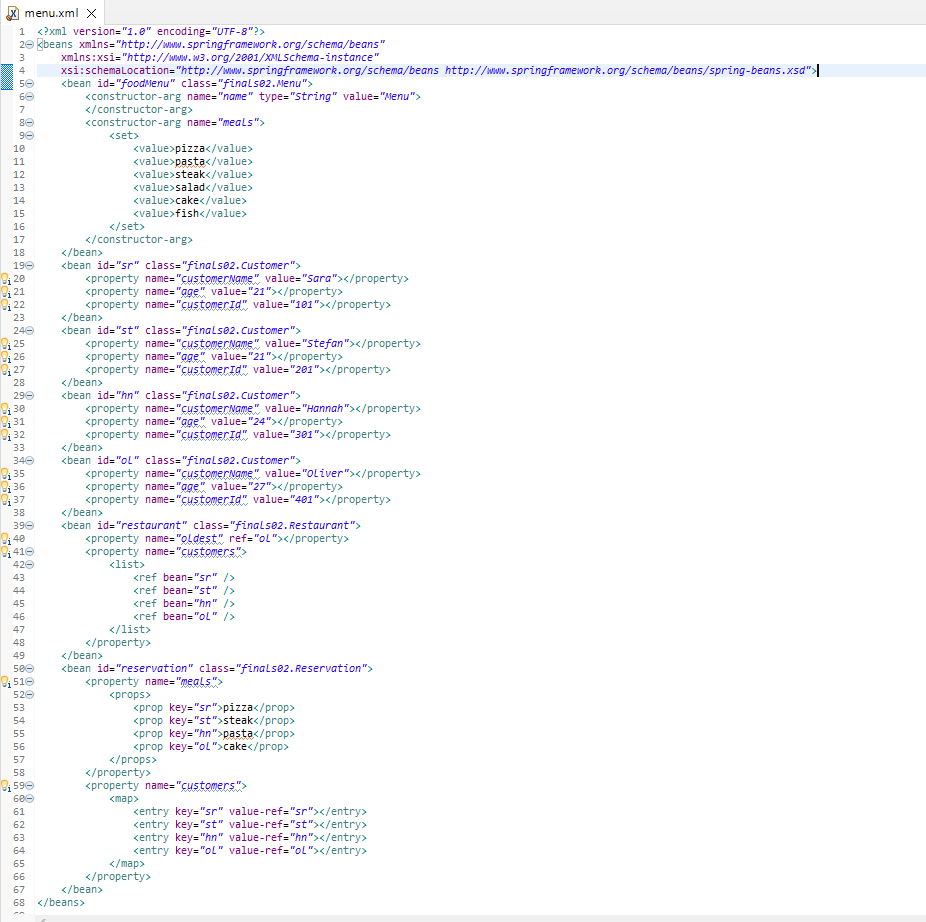
### app1Console.java



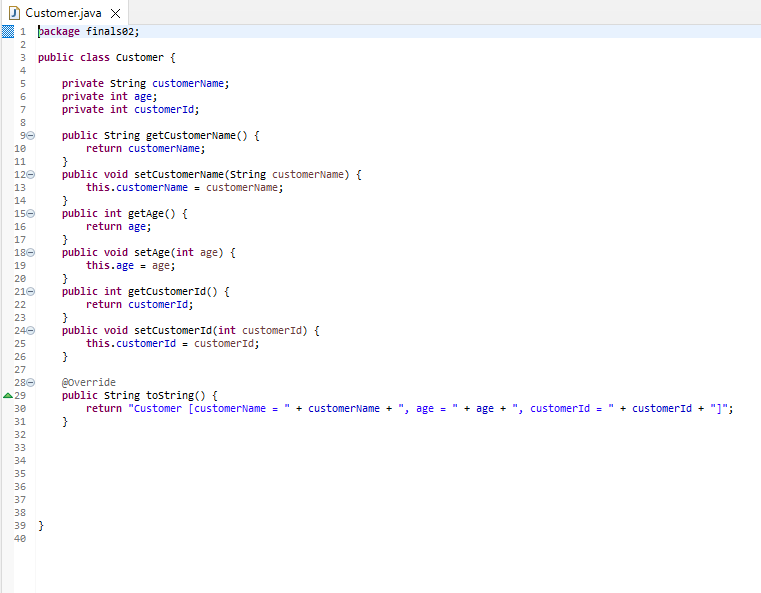
## Project #2

This projects theme was a restaurant reservation/bill kind of idea, where all the customers and their orders are printed out.

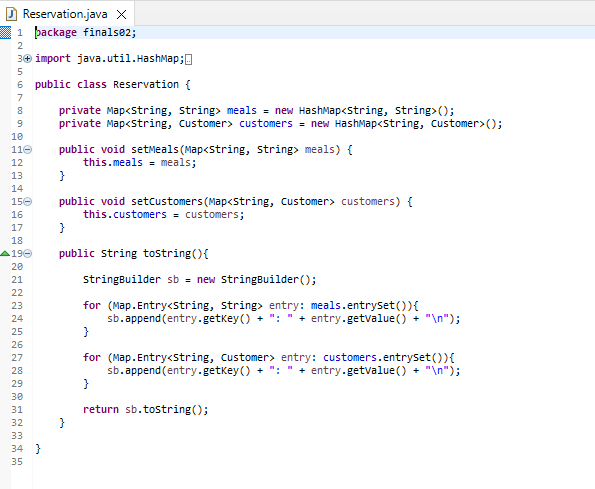
### menu.xml



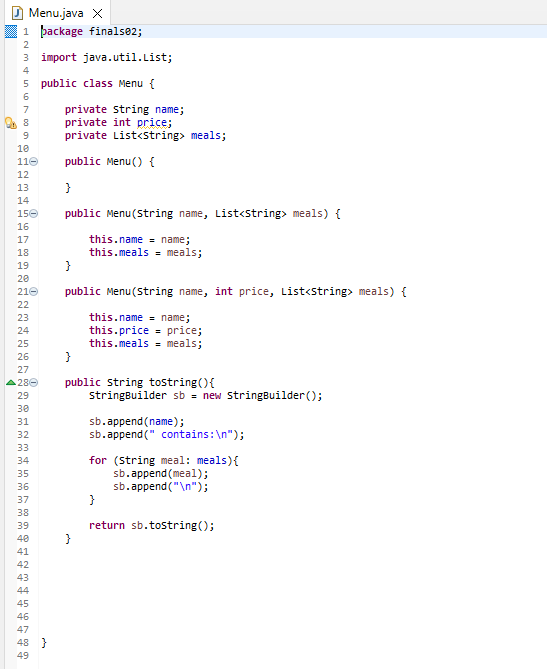
### Customer.java



### Reservation.java



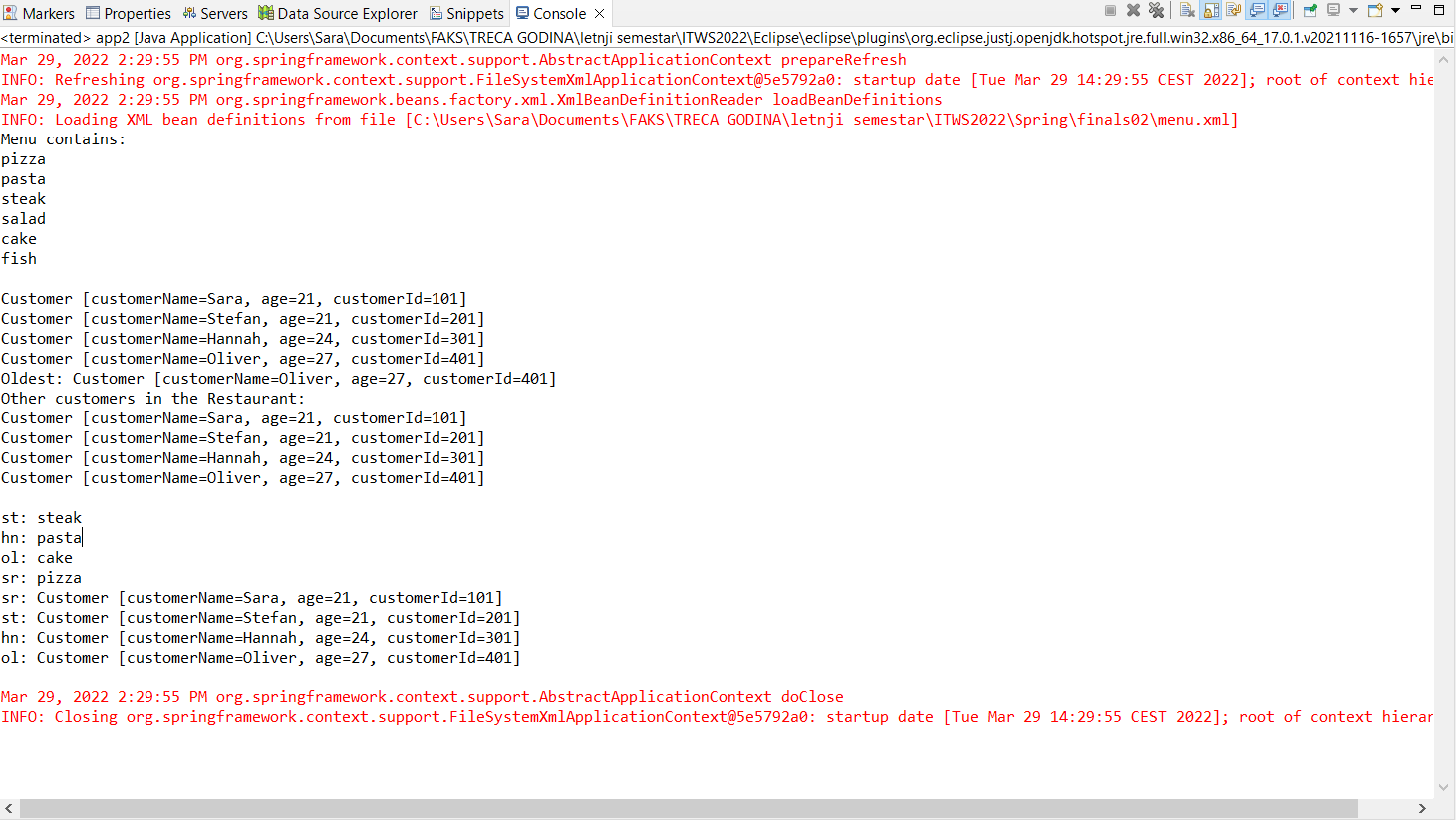
### Menu.java



### App2.java



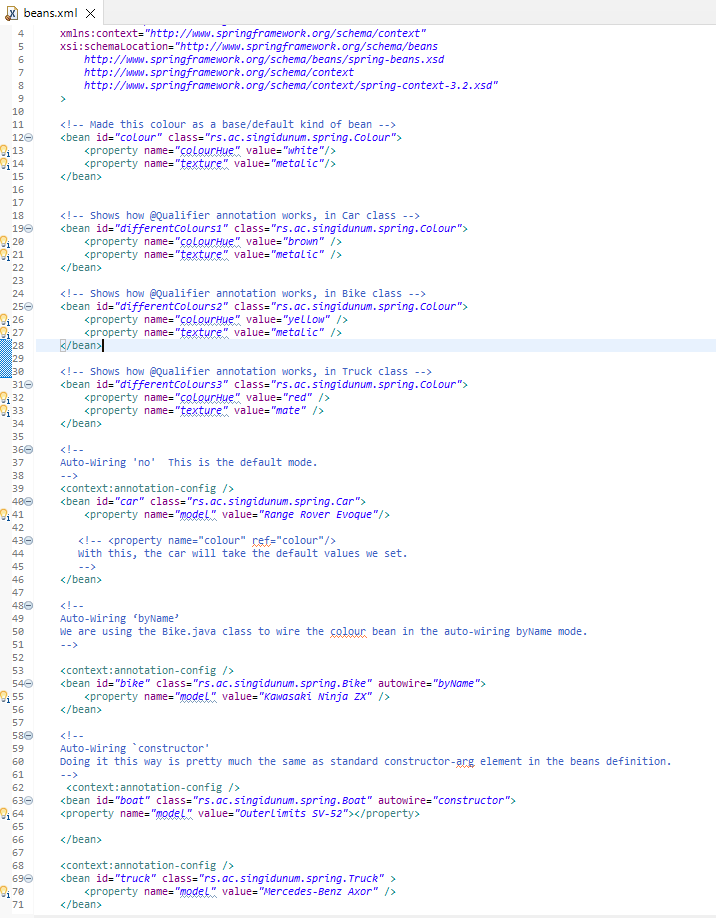
### App2\_Console



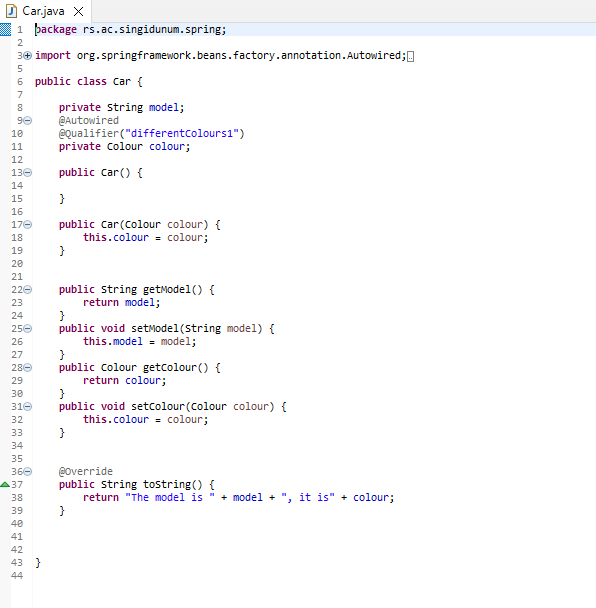
## Project #3

The idea/theme for this project was a kind of vehicle list, of different types of vehicles and their colours. Just to note, pretty much all the classes were quite similar in base idea so we will only show Car.java.

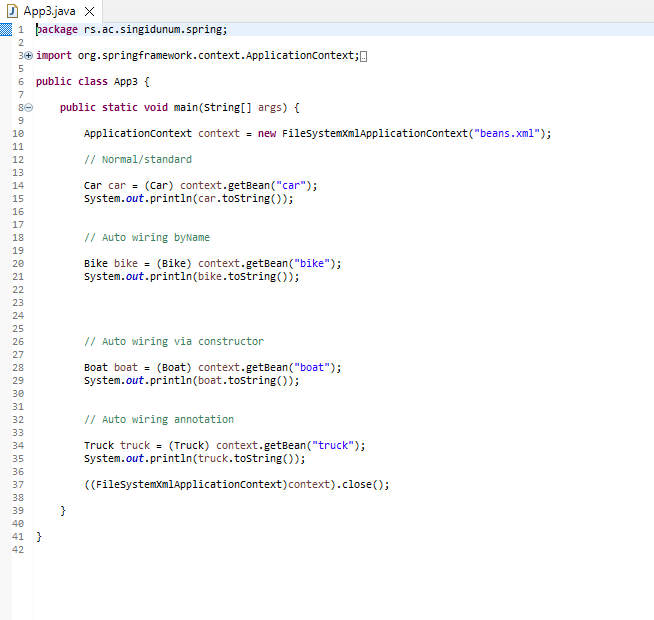
### beans.xml



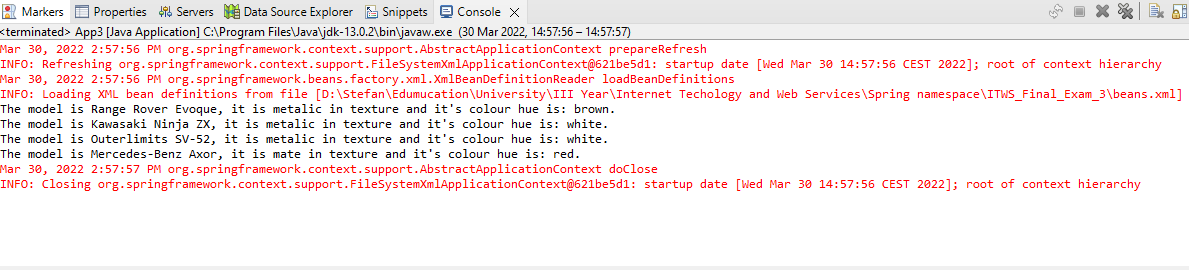
### Car.java



### App3.java



### App3\_Console



# Conclusion & Future upgrades

When it comes to each project, there are multiple possible upgrades or changes we could implement:

* For the first one, we could add a way for the server to have a longer track record of each user. For example, saving the days logs into a local .txt file of some kind.   
  Also adding more users would make it feel less primitive in nature.
* For the second one, showing the how much each customers items cost would be more similar in nature to a check/bill of some kind.
* For the third one, besides adding more vehicles and types to the list, maybe instead it could focus on just one manufacturer and their vehicles they make.   
  So that then it could work as some kind of storage/stock application.

In hindsight, maybe the three projects could have been melded into a single one. Where the first would monitor the online users who are browsing the *web* page. With the third one serving as some kind of show like page, and lastly the second one as a kind of checkout.

Of course, there are likely more upgrades or changes that could be implemented but I cannot think of them as of this moment.