Simulation of RFID Smart Shopping System in the Supermarket

1. **Introduction**

This is the simulation of innovative system that could be implemented in the supermarkets all over the world. Similar RFID system already exists in a lot of supermarkets in digitally advanced countries, but this system is a little bit different.

Instead of using personal cards, clients would use baskets that have RFID scanner in themselves, and every instance of article would have its own RFID number, even if there are more same articles, but every instance should have own RFID number, for easier (re)scanning.

Client can take article out and put it back on the shelf, RFID scanner will automatically rescan an article, because it will find an article in list that is given to specific basket which is currently in use.

Also, the client can put in the basket an article that is from totally different shop, scanner will not recognize it, and it won’t be appended to the total price for that basket.

On the Cash desk there is also RFID scanner that should scan the id number of every Shopping basket, and by scanning if its active, it will just throw the bill to the client. So Client just need to pay to the Cash Worker for the bill and wait to change.

This the great advantage and optimization for waisitng time in shopping because it will make faster circulation of waiting queues at the Cash desks, and by that it will enourmously reduce time for waiting, and make Cash Worker job easier, faster, with less mistakes.

This can by both interface for sensors and simulation of algorithm on the server and database system, with some space and time optimizations and more concurrent programmmig.

Application is programmed as Spring application with few types of Dependency Injection and one example of Autowired injection in to the object.

1. **Implementation**

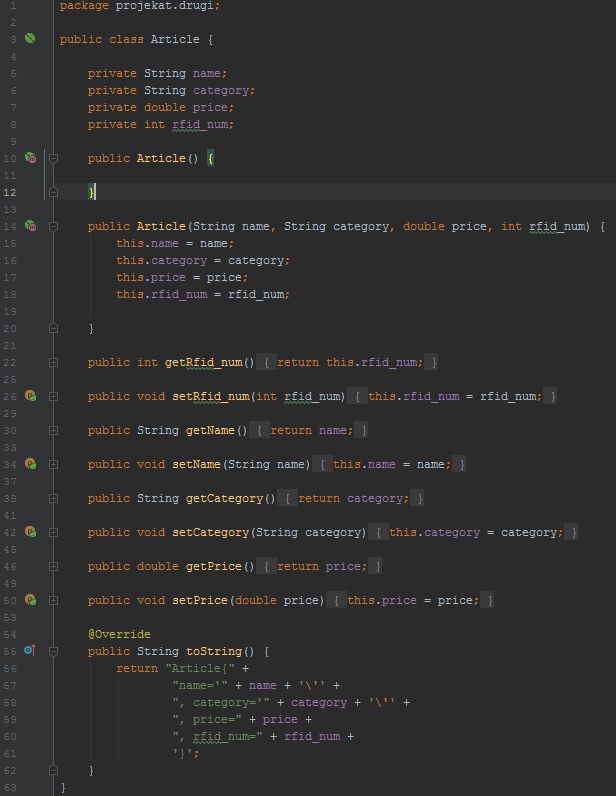
As we said this is application made in popular Spring framework for Java.

There are 6 Java classes, one of them is main class with main method and it is ***MarketServer.java***.

It is not real client-server application, instead of that is simulation, because in the real life, server won’t actually ask for every moment what client want, instead of that client will by his own decision buy or take back what he wants, and go in which he wants category.

We will begin with simple classes, and at the end we will see the main class MarketServer.

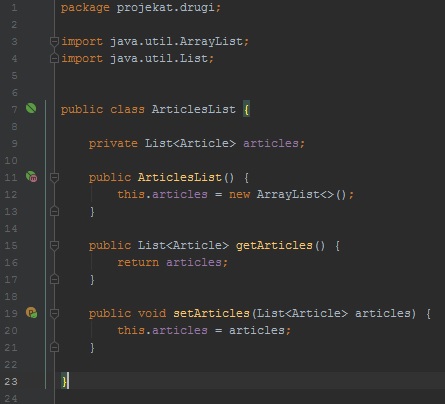
**2.1 Article.java**

****

This class represent an implementation for making Article objects. Every Article has name, category, price and RFID number. We will see later examples for object injection in ***spring.xml*** file.

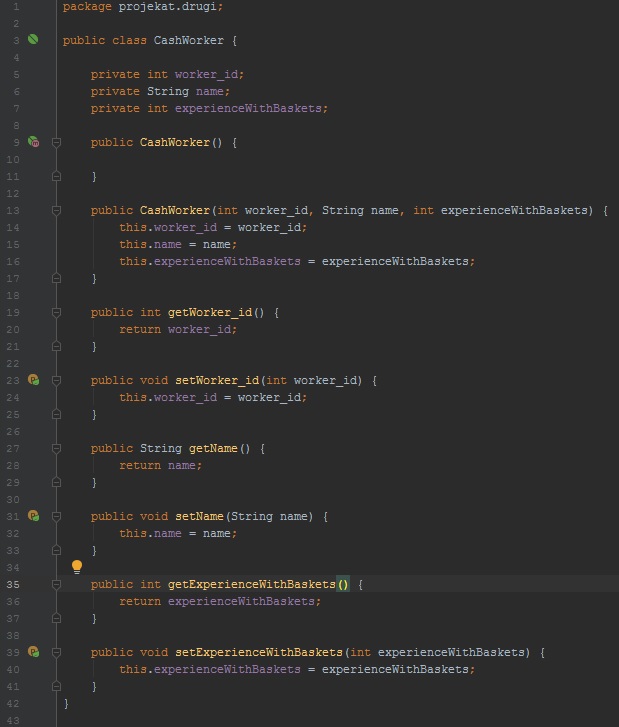
We will have list of Articles both for database simulation via Hash map and also for specific basket list which user has taken.

* 1. **ArticleList.java**

****

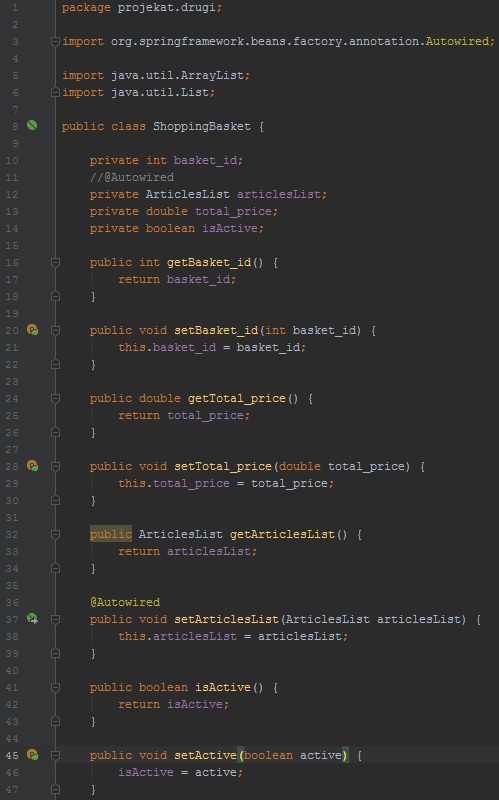
This class is just for purpose of project requirements made, to show an example of Autowiring object of ArticleList in ShoppingBasket object via setter method. In this class we have only defined a list of Articles with constructor, getter and setter.

* 1. **CashWorker.java**

****

This class represent one Cash Worker which will work on one Cash Desk to scan and charging bill. Every worker has its id number, name and experience represented with number of baskets scanned. We will see an example with injecting one object in another for these objects. In the class, there are only constructors, getter and setters.

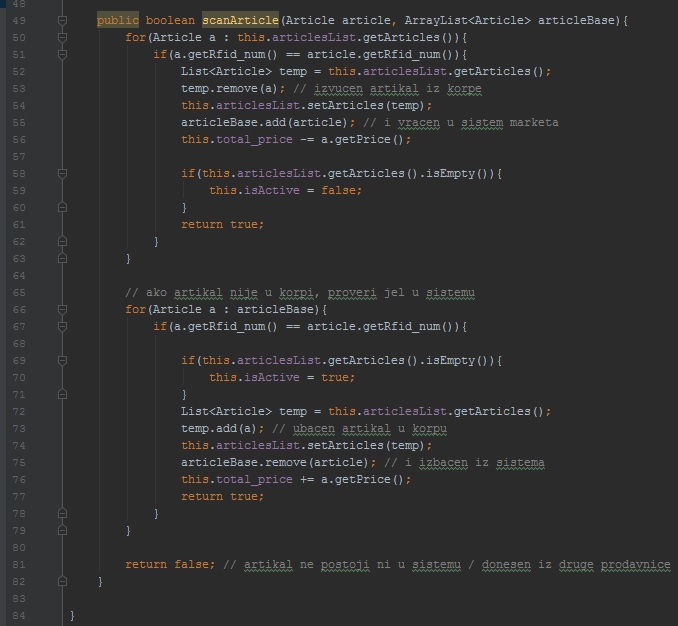
* 1. **ShoppingBasket.java**

****

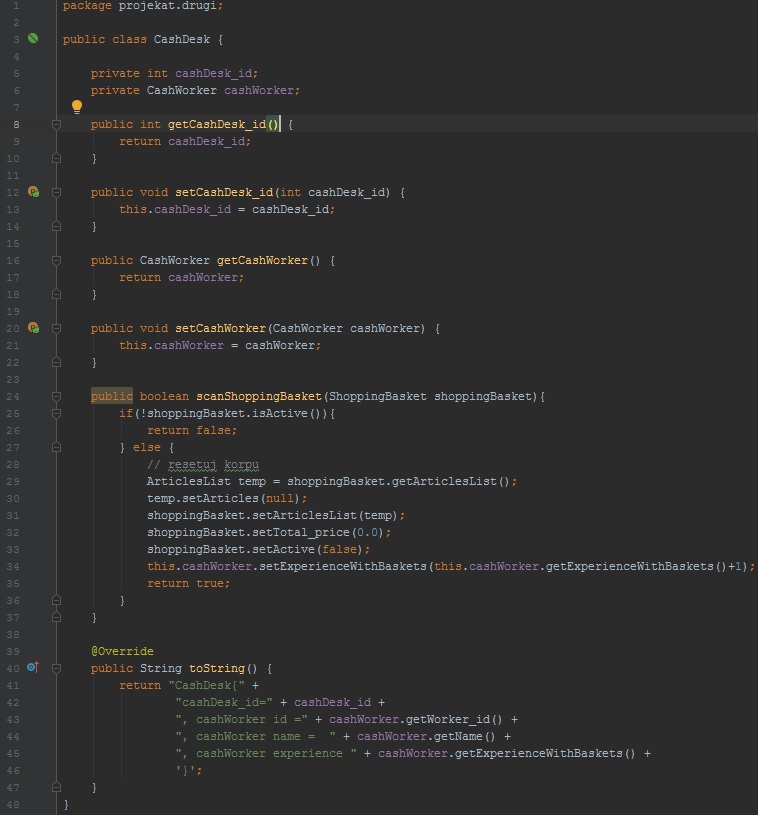
***scanArticle()*** is one of the most important methods in this class and generally, by the way we don’t have too many methods in whole application. ***scanArticle()*** method simulate scanner on the basket. There are 3 cases generally:

1. In first loop we check if article is taked out from the basket and in that case we should rescan it, so remove from basket list and add back to system list.
2. In second loop we check if article is put in the basket, so we shuld scan it, add to basket list and remove from system list.
3. If article is not in the system and in the basket so it is from another shop and ignore it

This class represent an implementation for concrete Shopping Basket object. Every basket has its id number, object Article List which is Autowired, total price for all articles, and isActive attribute for deciding if the basket has any articles for buying or not. In the first picture we can only see getters and setters and setter which will automatically wire bean for Article List that is at the beginning independent bean.

****

* 1. **CashDesk.java**

****

CashDesk class will build concrete desk object. This object has its own id and CashWorker object.

It doesn’t have constructors because we use setter injection for making Cash Desk objects and object injection with separate or inner bean injecting Cash Worker bean in Cash Desk object as property, and there also other getters and setters.

Also we have one more important method which simulate scanning of basket. There are 2 cases:

1. Basket is not active so there is nothing in it and Cash Worker can’t make the bill.
2. Basket is active so we reset all attributes in the basket, gain Worker experience and make bill after that.

We have finished with all side classes which are used for making concrete objects in our application. Below we will see implementation of Market Server with simulation of database as Hash Map with categories as keys and ArrayLists of Articles as values. Also we will see definitions of beans in spring.xml configuration file for Spring framework, and all 6 methods of Dependency Injection. After all those definitions we will see simulation of shopping by one client.

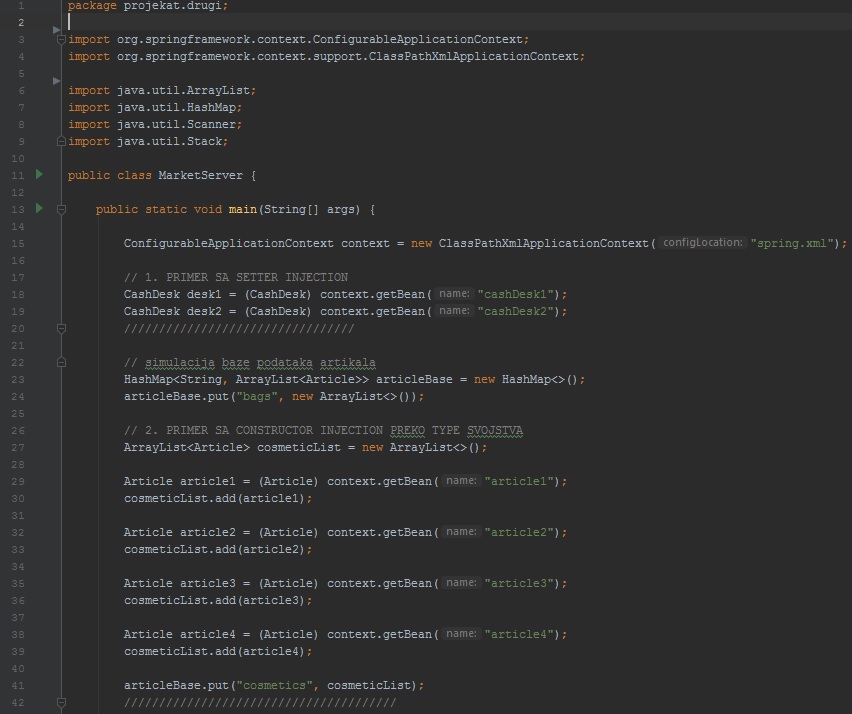
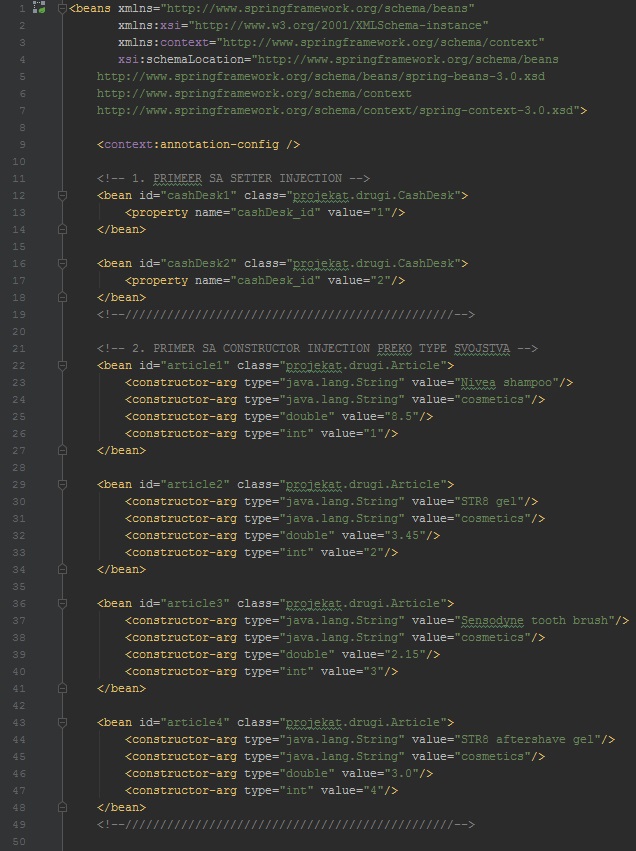
* 1. **MarketServer.java**

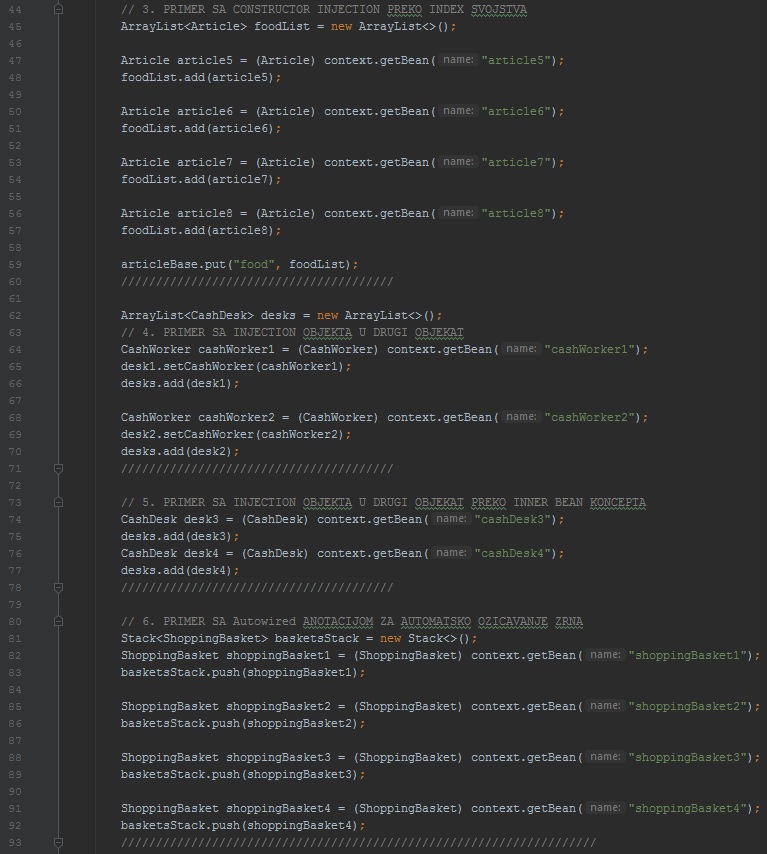
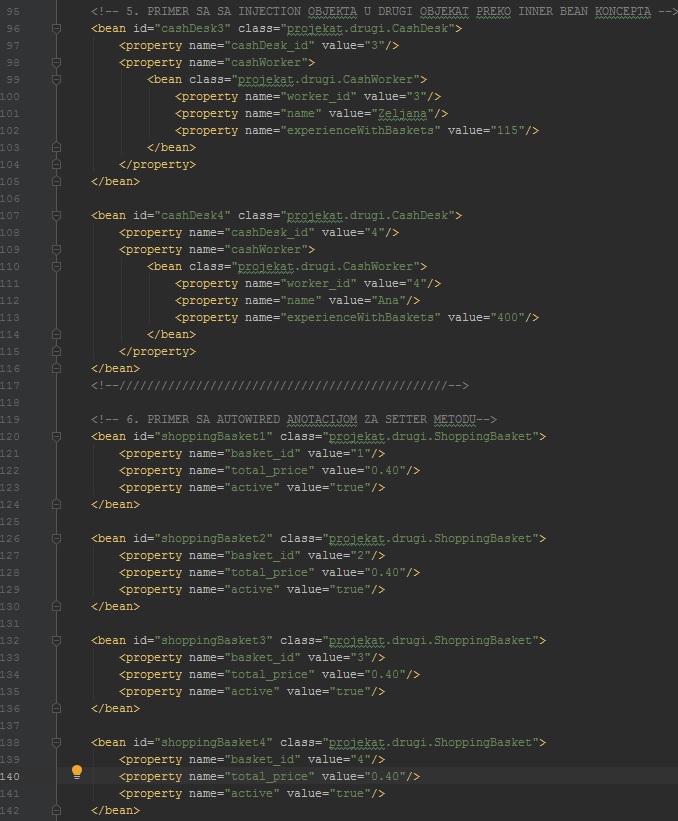
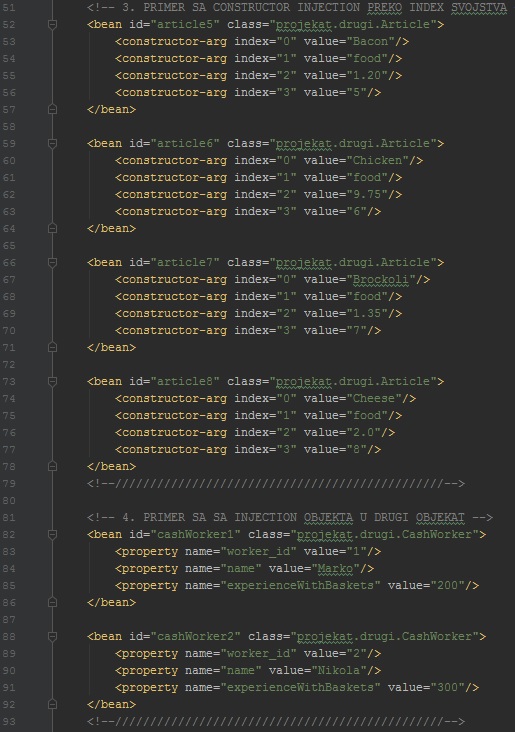
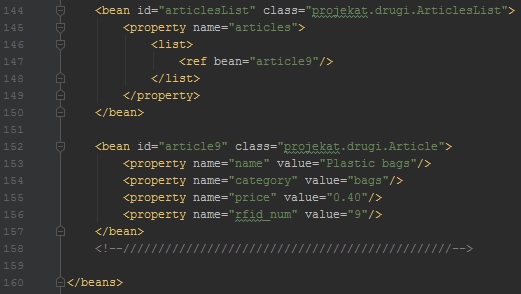
Below is an implementation of MarketServer class with main method on the left side and on the right side there is xml configuration file for Spring with application context and beans.

We have created Configurable Application Context that takes all necessary data for application context in xml file. At the beginning we have simulation of the database with Hash Map that stores categories as keys and articles in lists as values.

First example presents setter injection where we set attributes via properties in xml file.

Second example shows Constructor Injection via **constructor-arg** tags with type attribute where we put type of the argument. So we will get instances of these objects by calling ***context.getBean()*** method, cast in specific object type and save. While getting instances of articles, we will immediately add them into the list and at the end put list in the hash map.



Next is **third example** for Dependency Injection with objection injection and index attribute in constructor-arg tags where we put numbers as indexes of constructor arguments and we will use that to get beans for other articles. **Fourth example** is injecting one object into another via setter method in CashDesk class. **Fifth example** is the same injection but with inner bean principle so we have nested bean as property in desk beans nad we only need to get bean from xml file and we have totally initialized object. **Sixth example** explains how injecting one object into another can be done automatically. We can choose which autowiring method we want. We can assign annotation to setter, constructor and variable. In this example it was autowired by setter. So when we call ***getBean()*** method, it will automatically autowire object ArticleList to Shopping Basket by setter via property tag.

Now we are finished with Spring dependency injection and analyzing xml file. Below is shown a simulation of client coming in the supermarket and looking to buy something.

**3.Functionalities**

**4.Conclusion**