# Java Basic Web: Phonebook

Problems for exercises and homework for the ["Technology Fundamentals" course @ SoftUni](https://softuni.bg/courses/technology-fundamentals)

## Problem

You have been tasked to create a simple **Phonebook** application. The application should hold **contacts**, which are the main app **entity**.

The functionality of the application should support:

* **Listing contacts**



* **Add Contact**

****

****

## Overview

### Requirements

* **Spring** framework (**Spring MVC** + **Spring Boot** + **Spring Data**)
* **Thymeleaf** view engine

### Data Model

The Contact entity holds **2 properties**:

* name – non-empty text
* number – non-empty text

### Project Skeletons

You will be given the applications' skeletons, which holds about **90%** of the logic. You’ll be given some **files**. The files will have **partially implemented logic**, so you’ll need to write some code for the application to **function properly**.

The application’s views will be given to you fully implemented. You only need to include them in your business logic.

Everything that has been given to you inside the skeleton is **correctly implemented** and if you write your code **correctly**, the application should work just fine. You are free to change anything in the Skeleton on your account.

## Preparation

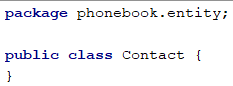
Start XAMPP and MySQL through it. Start Intellij and **import** the skeleton.

## Contact Entity

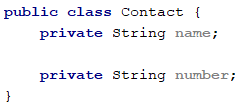
It's time to create our first entity. In the src/main/java/phonebook **package** you can see few packages that **define our project**. A **package** is a **folder** **containing** **Java** files. The one we are interested in is the "entity" package. Inside, create a **new** **java class** called "Contact":

****

The file should look like this:



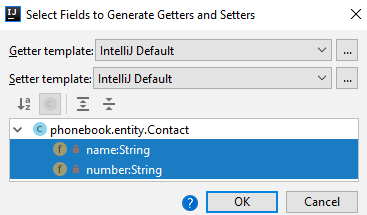
We need to define our contact entity. Create the following private fields:



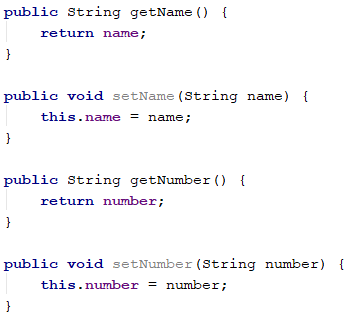
Let's create [getters and setters](http://java.about.com/od/workingwithobjects/a/accessormutator.htm) for our fields. You should already be familiar with them. If you are curious why are we doing that, you can read more [here](https://www.tutorialspoint.com/java/java_encapsulation.htm). There is a **simple way to create them** in **IntelliJ Idea**. If you press "**Alt + Insert**", you should see that context menu:



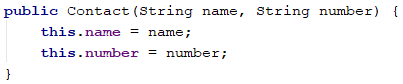
Choosing the "**Getter and Setter**" option will **open new window**. You should select **all private fields** from there:

****

When you **click** "**OK**", you should **receive this code**:

****

It's the time to **create our constructor**:

****

We will **use** **this** **constructor** to **create tasks** easily. However, we need to create another **empty** **constructor** for **Spring**:



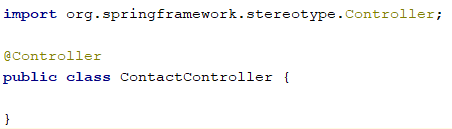
And this is pretty much everything. Our Contact entity is ready.

## Contact Controller

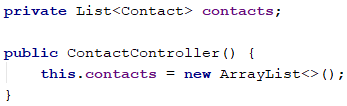
We have reached the point in which we can create our **controller**. In the "controller" package create a new class called "ConctactController":



Add the following annotation:



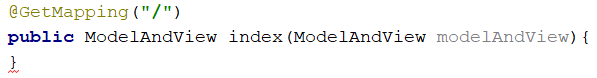
This class will **list** and **save** contacts. That means that it will use **routes**. In order to let **Spring,** know that this class will be controller, we need to use the "@Controller" annotation. This annotation also gives us **access** to **requests** and gives us the ability to respond to them. Now, we need to store contacts somewhere. We will use a **list**, later in the course we will learn how to store information in **database**.



We need to **initialize** our list, so we are going to do it through the **constructor**.

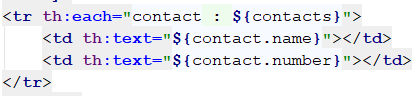
### Listing All Tasks

First, we need to list all articles to the given route. Go to the ContactControllerand create method index.

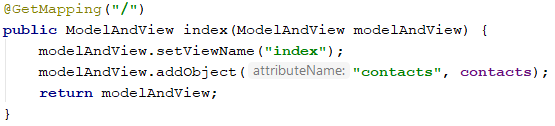


The "@GetMapping" annotation tells **Spring** that this method **cannot be called** if the user wants to **submit data**. It should be **only used** for **viewing data**. ModelAndView allows us to pass all the information required by Spring MVC in one return:

We have only one view which we will use in our application, called "index". If you look the view, you will find:



Which means that our view expects a collection called "contacts", so we have to add contactsto the modelAndViewand return it as a result:

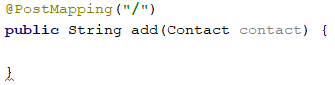


We are ready with the listing all contacts.

### Adding Contact

As we have mentioned, we have only one view. If you go back to the view and take a look once again, you will find a **form**:

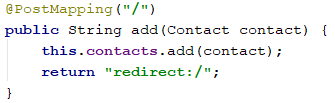
  
Thanks to the form, we can process post requests.



**Spring** will automatically map the data into contact object, if we've created it correctly.

With "@PostMapping" annotation, we told **Spring** that **this method expects data** that it needs to **autofill in** contact **object**. The annotation handles "POST" **request** that are usually what the **HTML forms** are using as a "method" of the **request**. In summary, **this method** will be called **when users submit the data**.

Finally, we need to **add the contact in our list** and give a **response** to the user.



We redirect the user back to index page as response.

With that we finished our **Java Phonebook**. Feel free to **build on your project even further**. ☺