

|  |
| --- |
| Learning Material  Java Learning Program |
| Introduction to Cross-Platform Programming on Java |

|  |  |
| --- | --- |
| Abbreviations and acronyms | |
|  |  |

CONTENT

[1 INCEPTION 5](#_Toc8902275)

[1.1 PURPOSE 5](#_Toc8902276)

[1.2 SCOPE 5](#_Toc8902277)

[2 Environment and Prerequisites 6](#_Toc8902278)

[2.1 IDE 6](#_Toc8902279)

[2.2 VCS – Version control System 6](#_Toc8902280)

[2.3 Common Code Formatting Rules 6](#_Toc8902281)

[2.4 Testing 6](#_Toc8902282)

[3 Spring-Boot service 7](#_Toc8902283)

[3.1 Technologies overview 7](#_Toc8902284)

[3.1.1 Spring 7](#_Toc8902285)

[3.1.2 Spring Boot 7](#_Toc8902286)

[3.1.3 Build Tools 7](#_Toc8902287)

[3.1.4 Web Service 7](#_Toc8902288)

[3.1.5 Request and Response 8](#_Toc8902289)

[3.1.6 Web Application 8](#_Toc8902290)

[3.1.7 Tasks 8](#_Toc8902291)

[3.1.7.1 Option 1 – Copy Buttons 8](#_Toc8902292)

[3.1.7.2 Option 2 – Parallelogram Calculations 8](#_Toc8902293)

[3.1.7.3 Option 3 – Triangle Calculations 9](#_Toc8902294)

[3.1.7.4 Option 4 – Triangle Identification 9](#_Toc8902295)

[3.1.7.5 Option 5 – Simple Calculations 9](#_Toc8902296)

[3.1.7.6 Option 6 – Random Generator 10](#_Toc8902297)

[3.1.7.7 Option 7 – Addition and Multiplication 10](#_Toc8902298)

[3.1.7.8 Option 8 – Meters Convertor 10](#_Toc8902299)

[3.1.7.9 Option 9 – String Invertor 11](#_Toc8902300)

[3.1.7.10 Option 10 – Polynomic Identification 11](#_Toc8902301)

[3.1.7.11 Option 11 – Selection Method 11](#_Toc8902302)

[3.1.7.12 Option 12 – Trapezium Integration 12](#_Toc8902303)

[3.1.7.13 Option 13 – Binary Convertor 12](#_Toc8902304)

[3.1.7.14 Option 14 – Leap-year Check 12](#_Toc8902305)

[3.1.7.15 Option 15 – Day of Week Identification 13](#_Toc8902306)

[3.1.7.16 Option 16 – Compare with Random 13](#_Toc8902307)

[3.1.7.17 Option 17 – Character Inclusion 13](#_Toc8902308)

[3.1.7.18 Option 18 – Prime Check 14](#_Toc8902309)

[3.1.7.19 Option 19 – Fibonacci Calculation 14](#_Toc8902310)

[3.1.7.20 Option 20 – Inelastic Clash Simulation 14](#_Toc8902311)

[3.1.7.21 Option 21 – Uniform Rectilinear Motion 15](#_Toc8902312)

[3.1.7.22 Option 22 – Maximum Identification 15](#_Toc8902313)

[3.1.7.23 Option 23 – Triangle Calculations 15](#_Toc8902314)

[3.1.7.24 Option 24 – Vector Rate and Projection 15](#_Toc8902315)

[3.1.7.25 Option 25 – Text Shift 16](#_Toc8902316)

[3.1.7.26 Option 26 – Mass Calculation 16](#_Toc8902317)

[3.1.7.27 Option 27 – Cylinder Volume 17](#_Toc8902318)

[3.1.7.28 Option 28 – Complex Number Calculations 17](#_Toc8902319)

[3.1.7.29 Option 29 – Random Walk 17](#_Toc8902320)

[3.1.7.30 Option 30 – Division Calculations 17](#_Toc8902321)

[4 Exceptions and Error Handling 19](#_Toc8902322)

[5 Collections 20](#_Toc8902323)

[5.1 JAVA COLLECTIONS FRAMEWORK OVERVIEW 20](#_Toc8902324)

[5.2 COLLECTIONS FRAMEWORK HANDBOOK 20](#_Toc8902325)

[5.3 COLLECTIONS IN JAVA. LIST 20](#_Toc8902326)

[5.4 COLLECTIONS IN JAVA. SET 20](#_Toc8902327)

[5.5 COLLECTIONS IN JAVA. QUEUE 20](#_Toc8902328)

[5.6 COLLECTIONS IN JAVA. MAP 21](#_Toc8902329)

[5.7 HASHMAp – how to work iN JAVA 21](#_Toc8902330)

[5.8 TASK 21](#_Toc8902331)

[6 Concurrency 22](#_Toc8902332)

[6.1 Basic concepts 22](#_Toc8902333)

[6.1.1 Concurrent programming 22](#_Toc8902334)

[6.1.2 Process vs. threads 22](#_Toc8902335)

[6.1.3 Concurrency models 22](#_Toc8902336)

[6.2 JMM 22](#_Toc8902337)

[6.3 Native java concurrency 22](#_Toc8902338)

[6.3.1 Task 22](#_Toc8902339)

[6.4 java.util.concurrent 23](#_Toc8902340)

[6.4.1 Task 23](#_Toc8902341)

[6.5 Advanced: java performance monitoring 23](#_Toc8902342)

[6.5.1 Task 23](#_Toc8902343)

[7 Functional Programming in Java 8 24](#_Toc8902344)

[7.1 Functional programming in JVM 24](#_Toc8902345)

[7.2 Lambda Expressions in JAVA 24](#_Toc8902346)

[7.3 Java STREAM API 24](#_Toc8902347)

[7.4 BULK OPERATIONS 24](#_Toc8902348)

[7.5 TASK 25](#_Toc8902349)

[8 Aggregate Operations in JAVA 8 26](#_Toc8902350)

[8.1 Filter and MAP USAGE in JAVA 26](#_Toc8902351)

[8.2 Aggregate Operations 26](#_Toc8902352)

[8.3 Reduce and Collect 26](#_Toc8902353)

[8.4 Task 26](#_Toc8902354)

[9 Data Persistance 27](#_Toc8902355)

[9.1 Basic theory 27](#_Toc8902356)

[9.2 What about java? 27](#_Toc8902357)

[9.2.1 JDBC 27](#_Toc8902358)

[9.2.2 ORM 27](#_Toc8902359)

[9.3 Task 27](#_Toc8902360)

# INCEPTION

## PURPOSE

This document purpose to describe the knowledge required, but not limited, for majority of cross-platform Java projects.

## SCOPE

This document is NOT dedicated to full coverage of what’s needed to deploy production ready project on Java or to what’s needed to be learned to become successful Java developer. The scope of work is about giving the direction for learning path to start career. Treat it as a brief overview to the most trending and demanding technologies in cross-platform Java stack.

Materials will be provided in the form of hand-on labs, so that there will be an ability to touch and practice things that has just been learned.

# Environment and Prerequisites

## IDE

Your primary tool will be IntelliJ IDEA, Community Edition:

<https://www.jetbrains.com/idea>

It’s easy to use, have built-in autocompletion for most of the features and provides great integration with Spring.

## VCS – Version control System

If you unaware of how Version Control systems work – that’s fine: we are going to use Git and it is very easy to jump in:

<https://githowto.com/ru>

Please maximize your console usage of it at least for the first time to get proper understanding of how it works. You can switch to IDE integration later on.

For deep knowledge – please refer to Pro Git book:

<https://git-scm.com/book/en/v2>

We will use GitHub as a server of this VCS – please register yourself there and create your projects in that space:

[https://github.com](https://github.com/)

## Common Code Formatting Rules

We recommend using of Google Code Formatting rules:

<https://google.github.io/styleguide/javaguide.html>

Oracle’s are pretty old and were designed for small monitor resolution.

Please integrate it with your codebase using CheckStyle:

<http://checkstyle.sourceforge.net/google_style.html>

Guide for wiring it to Groovy:

<https://docs.gradle.org/current/userguide/checkstyle_plugin.html>

## Testing

It’s very important to cover your code with tests. This is one of the best practices for software development. We will care about Unit Tests a lot.

Please read about it and get proper understanding of why and how to write it:

<http://softwaretestingfundamentals.com/unit-testing/>

Please check out F.I.R.S.T. principle for writing good unit tests:

<https://dzone.com/articles/writing-your-first-unit-tests>

Here is how to measure your code coverage inside IDE:

<https://www.jetbrains.com/help/idea/viewing-code-coverage-results.html>

# Spring-Boot service

## Technologies overview

### Spring

Spring is one of the core technologies on top of Java stack. It allows a lot of things including but not limited to:

* Declaring dependencies and relations between components
* Managing objects lifecycle from initiation to destruction
* Multiple configurations of object properties in different states of its life

There is a nice description of Spring available on the official site:

<https://docs.spring.io/spring/docs/current/spring-framework-reference/overview.html>

Please go through it to get basic understanding of why we need it.

Here is the list of quick-start guides available:

<https://spring.io/guides>

### Spring Boot

Spring Boot can be treated as engine or accelerator to create your production-ready Spring-based application with minimal effort and code:

<https://spring.io/projects/spring-boot>

Here is an “Initializer” allowing you to generate the application with Spring-Boot engine in few seconds:

[https://start.spring.io](https://start.spring.io/)

### Build Tools

To launch something, you usually need to build it. The same is true about the code. We will touch one of the technologies to build it, called Gradle. Here is a brief intro to that technology:

<https://docs.gradle.org/current/userguide/getting_started.html>

And the basic Java-based project user guide:

<https://guides.gradle.org/building-java-libraries/>

### Web Service

There are a lot of ways to describe the Web Service term. It’s safe to say that it is a thing to serve the relation between client and service with some defined API on top of some protocol. These protocols and APIs could be of various types and we will only briefly touch the one that is called REST API.

Here is a nice intro to it:

[https://www.restapitutorial.com](https://www.restapitutorial.com/)

And vision of proper REST API design from Martin Fowler:

<https://martinfowler.com/articles/richardsonMaturityModel.html>

### Request and Response

If we talk about RESTful Services – the most common types of requests are HTTP GET and POST. Please read the common description of both methods:

<https://www.w3schools.com/tags/ref_httpmethods.asp>

When we talk about the structure of response – it’s mostly JSON based, here is an example of such structure:

<https://json.org/example.html>

### Web Application

Now we will try to build and launch our first application based on the above terms and technologies. You will be able to see how easy and fast to create and launch it based on the above stack.

Here is Spring Boot quick-start for building your first lightweight RESTful Web Service:

<https://spring.io/guides/gs/rest-service/>

### Tasks

1. Create and run locally simple web/REST service using any open-source example on Java stack: Spring/maven/gradle/Jersey/Spring MVC.

2. Create GET endpoint with input parameters as URL query params, performing calculation based on your Option, and returning results in JSON format.

**Common description**

Optionally, UI is basically an HTML form that passes request field with form parameters to the Web service.

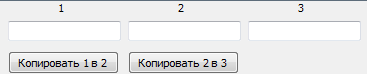
Each service should have

* Request parameters should be transferred as URL parameters for GET and JSON object for POST
* Response should be JSON-based object

#### Option 1 – Copy Buttons

Flow:

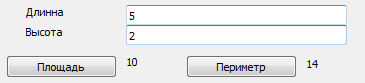
* **When:** User inputs value to 1st input field
* **Then:** Presses “Копировать 1 в 2” button
* **And:** Value appears in 2nd input field
* **Then:** User presses “Копировать 2 в 3” button
* **And:** Value appears in 3rd input



#### Option 2 – Parallelogram Calculations

Flow:

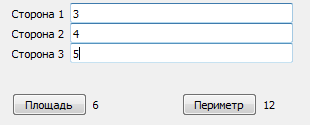
* **When:** User inputs value to “Длина” input field
* **And**: User inputs value to “Высота” input field
* **Then:** Presses “Площадь” button
* **And:** Value appears in “Площадь” text field
* **Then:** User presses “Периметр” button
* **And:** Value appears in “Периметр” text field



#### Option 3 – Triangle Calculations

Flow:

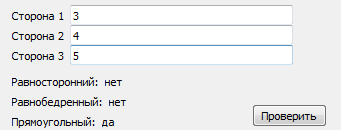
* **When:** User inputs value to “Сторона 1” input field
* **And**: User inputs value to “Сторона 2” input field
* **And**: User inputs value to “Сторона 3” input field
* **Then:** Presses “Площадь” button
* **And:** Value appears in “Площадь” text field
* **Then:** User presses “Периметр” button
* **And:** Value appears in “Периметр” text field



#### Option 4 – Triangle Identification

Flow:

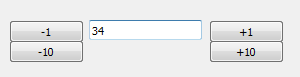
* **When:** User inputs value to “Сторона 1” input field
* **And**: User inputs value to “Сторона 2” input field
* **And**: User inputs value to “Сторона 3” input field
* **Then:** Presses “Check” button
* **And:** «Да/Нет» value appears in “Равносторонний” text field
* **And:** «Да/Нет» value appears in “Равнобедренный” text field
* **And:** «Да/Нет» value appears in “Прямоугольный” text field



#### Option 5 – Simple Calculations

Flow:

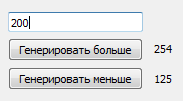
* **When:** User inputs “X” value to input field
* **And**: Presses “-1” button
* **Then**: Input field is changed to “X-1”
* **And**: Presses “-10” button
* **Then**: Input field is changed to “X-11”
* **And**: Presses “+1” button
* **Then**: Input field is changed to “X-10”
* **And**: Presses “+10” button
* **Then**: Input field is changed to “X”



#### Option 6 – Random Generator

Flow:

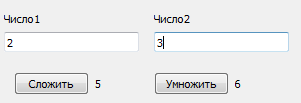
* **When:** User inputs “X” value to input field
* **Then**: Presses “Генерировать больше” button
* **And**: Value greater than “X” appears in “Больше” text field
* **Then**: Presses “Генерировать меньше” button
* **And**: Value less than “X” appears in “Меньше” text field



#### Option 7 – Addition and Multiplication

Flow:

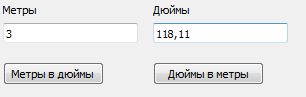
* **When:** User inputs value to “Число 1” input field
* **And**: User inputs value to “Число 2” input field
* **Then:** Presses “Сложить” button
* **And:** Value appears in “Сумма” text field
* **Then:** User presses “Умножить” button
* **And:** Value appears in “Произведение” text field



#### Option 8 – Meters Convertor

Flow:

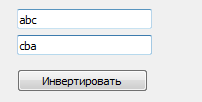
* **When:** User inputs value to “Метры” input field
* **Then:** Presses “Метры в дюймы” button
* **And**: Value appears in “Дюймы” input field
* **When:** User inputs value to “Дюймы” input field
* **Then:** Presses “Дюймы в метры” button
* **And**: Value appears in “Метры” input field



#### Option 9 – String Invertor

Flow:

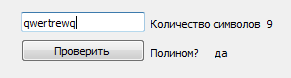
* **When:** User inputs value to 1st input field
* **Then**: Presses “Инвертировать” button
* **And**: Inverted value appears in 2nd input field
* **When:** User inputs value to 2nd input field
* **Then**: Presses “Инвертировать” button
* **And**: Inverted value appears in 1st input field



#### Option 10 – Polynomic Identification

Flow:

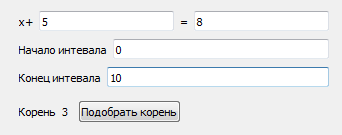
* **When:** User inputs value to input field
* **Then:** Presses “Проверить” button
* **And:** Number of symbols appears in “Количество символов” text field
* **And:** «Да/Нет» value appears in “Полином” text field



#### Option 11 – Selection Method

Flow:

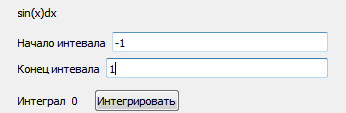
* **When:** User inputs value to 1st input field
* **And**: User inputs value to 2nd input field
* **And**: User inputs value to “Начало интервала” input field
* **And**: User inputs value to “Конец интервала” input field
* **Then:** Presses “Подобрать корень” button
* **And:** Value appears in “Корень” text field



#### Option 12 – Trapezium Integration

Flow:

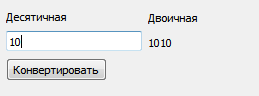
* **When**: User inputs value to “Начало интервала” input field
* **And**: User inputs value to “Конец интервала” input field
* **Then:** Presses “Интегрировать” button
* **And:** Value appears in “Интеграл” text field



#### Option 13 – Binary Convertor

Flow:

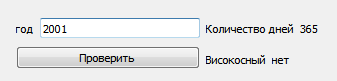
* **When:** User inputs value to “Десятичная” input field
* **Then:** Presses “Конвертировать” button
* **And:** Value appears in “Двоичная” text field



#### Option 14 – Leap-year Check

Flow:

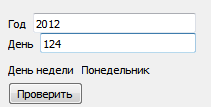
* **When:** User inputs value to “Год” input field
* **Then:** Presses “Проверить” button
* **And:** Value appears in “Количество дней” text field
* **And:** «Да/Нет» value appears in “Високосный” text field



#### Option 15 – Day of Week Identification

Flow:

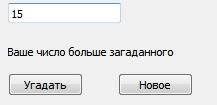
* **When:** User inputs value to “Год” input field
* **And**: User inputs value to “День” input field
* **Then:** Presses “Проверить” button
* **And:** Value appears in “День недели” text field



#### Option 16 – Compare with Random

Flow:

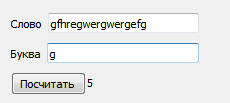
* **When:** User inputs value to input field
* **Then:** Presses “Угадать” button
* **And:** Either “Ваше число больше загаданного” or “Ваше число меньше загаданного” value appears in text field
* **Then:** User presses “Новое” button
* **And:** Either “Ваше число больше загаданного” or “Ваше число меньше загаданного” value appears in text field



#### Option 17 – Character Inclusion

Flow:

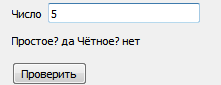
* **When:** User inputs value to “Слово” input field
* **And**: User inputs value to “Буква” input field
* **Then:** Presses “Посчитать” button
* **And:** Value appears in text field



#### Option 18 – Prime Check

Flow:

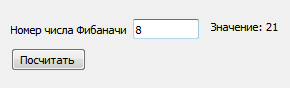
* **When:** User inputs value to “Число” input field
* **Then:** Presses “Проверить” button
* **And:** «Да/Нет» value appears in “Простое” text field
* **And:** «Да/Нет» value appears in “Четное” text field



#### Option 19 – Fibonacci Calculation

Flow:

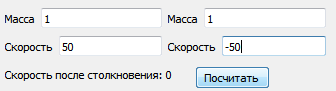
* **When:** User inputs value to input field
* **Then:** Presses “Посчитать” button
* **And:** Value appears in text field



#### Option 20 – Inelastic Clash Simulation

Flow:

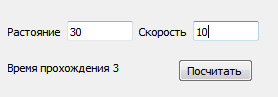
* **When:** User inputs value to 1st “Масса” input field
* **And**: User inputs value to 1st “Скорость” input field
* **And:** User inputs value to 2nd “Масса” input field
* **And**: User inputs value to 2nd “Скорость” input field
* **Then:** Presses “Посчитать” button
* **And:** Value appears in “Скорость после столкновения” text field



#### Option 21 – Uniform Rectilinear Motion

Flow:

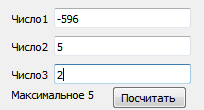
* **When:** User inputs value to “Расстояние” input field
* **And**: User inputs value to “Скорость” input field
* **Then:** Presses “Посчитать” button
* **And:** Value appears in “Время прохождения” text field



#### Option 22 – Maximum Identification

Flow:

* **When:** User inputs value to “Число 1” input field
* **And**: User inputs value to “Число 2” input field
* **And**: User inputs value to “Число 3” input field
* **Then:** Presses “Посчитать” button
* **And:** Max value appears in “Максимальное” text field



#### Option 23 – Triangle Calculations

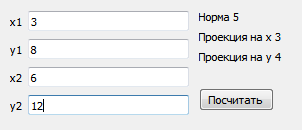
Flow:

* **When:** User inputs value to “Сторона 1” input field
* **And**: User inputs value to “Сторона 2” input field
* **And**: User inputs value to “Сторона 3” input field
* **Then:** Presses “Площадь” button
* **And:** Value appears in “Площадь” text field
* **Then:** User presses “Периметр” button
* **And:** Value appears in “Периметр” text field

#### Option 24 – Vector Rate and Projection

Flow:

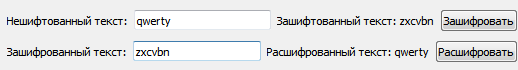
* **When:** User inputs value to “x1” input field
* **And**: User inputs value to “y1” input field
* **And**: User inputs value to “x2” input field
* **And**: User inputs value to “y2” input field
* **Then:** Presses “Посчитать” button
* **And:** Value appears in “Норма” text field
* **And:** Value appears in “Проекция на x” text field
* **And:** Value appears in “Проекция на y” text field



#### Option 25 – Text Shift

Flow:

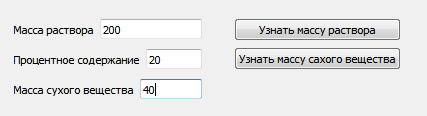
* **When:** User inputs value to “Нешифтованный текст” input field
* **Then:** Presses “Зашифтовать” button
* **And:** Value appears in “Зашифтованный” text field
* **When:** User inputs value to “Зашифтованный текст” input field
* **Then:** Presses “Расшифтовать” button
* **And:** Value appears in “Расшифтованный текст” text field



#### Option 26 – Mass Calculation

Flow:

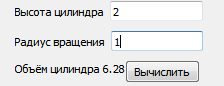
* **When:** User inputs value to “Масса раствора” input field
* **And**: User inputs value to “Процентное содержание” input field
* **Then:** Presses “Узнать массу сухого вещества” button
* **And:** Value appears in “Масса сухого вещество” text field
* **When:** User inputs value to “Масса сухого вещества” input field
* **And**: User inputs value to “Процентное содержание” input field
* **Then:** Presses “Узнать массу раствора” button
* **And:** Value appears in “Масса раствора” text field



#### Option 27 – Cylinder Volume

Flow:

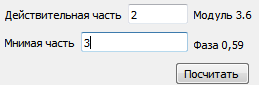
* **When:** User inputs value to “Высота цилиндра” input field
* **And**: User inputs value to “Радиус вращения” input field
* **Then:** Presses “Вычислить” button
* **And:** Value appears in “Объем цилиндра” text field



#### Option 28 – Complex Number Calculations

Flow:

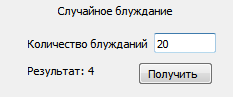
* **When:** User inputs value to “Действительная часть” input field
* **And**: User inputs value to “Мнимая часть” input field
* **Then:** Presses “Посчитать” button
* **And:** Value appears in “Модуль” text field
* **And:** Value appears in “Фаза” text field



#### Option 29 – Random Walk

Flow:

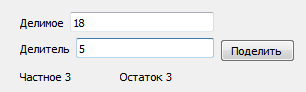
* **When:** User inputs value to “Количество блужданий” input field
* **Then:** Presses “Получить” button
* **And:** Value appears in “Результат” text field



#### Option 30 – Division Calculations

Flow:

* **When:** User inputs value to “Делимое” input field
* **And**: User inputs value to “Делитель” input field
* **Then:** Presses “Поделить” button
* **And:** Value appears in “Частное” text field
* **And:** Value appears in “Остаток” text field



# Exceptions and Error Handling

## Error Codes and LOGGING ERRORS

Exceptions in web-services:

There are 5 types of HTTP response codes:

1xx - info codes

2xx - successful codes

3xx - redirect codes

4xx - client error codes

5xx - server error codes

More info at <https://en.wikipedia.org/wiki/List_of_HTTP_status_codes>

We should map our inner exceptions to 4xx and 5xx error codes in the following way.

If exception caused by lack of input parameters, or nonexistent endpoint was been called, the 4xx error code should be in response.

If exception caused by some inner issues (like NPE, DB connection issues, etc.), the 5xx error code should be sent.

Not all of inner issues should be shown to client, some of them can be handled without any notification (but it should be logged anyway). For example, if client missed some parameters in request or uses inappropriate ones, we can use default parameters instead, if it's possible.

As well, all error situation should be logged with appropriate log level (ERROR), while non-error cases that uses default parameters instead of inappropriate could be logged with WARNING log level, to help you track such cases.

## TASKS

1. Add validation for the input parameters returning 400 HTTP error in case of empty or incorrect params provided

2. Add internal errors handling returning 500 HTTP error code in case of internal service exception/error

3. Add logging of all process steps and errors (use different log levels for errors and debug messages)

4. Add unit test for your service checking happy path and exceptional cases

# Collections

## JAVA COLLECTIONS FRAMEWORK OVERVIEW

The Java collections framework is a set of classes and interfaces that implement commonly reusable collection data structures.

<https://en.wikipedia.org/wiki/Java_collections_framework>

Collections in Java. Introduction.

<http://www.seostella.com/ru/article/2012/08/07/kollekcii-collections-v-java-vvedenie.html>

<http://tutorials.jenkov.com/java-collections/overview.html>

## COLLECTIONS FRAMEWORK HANDBOOK

Java Collection Framework is a hierarchy of interfaces and their implementations, which is part of the JDK and allows the developer to use many data structures from the "box".

<https://habr.com/ru/post/237043/>

<http://java-course.ru/begin/collections_01/>

## COLLECTIONS IN JAVA. LIST

The Java List interface, java.util.List, represents an ordered sequence of objects. The elements contained in a Java List can be inserted, accessed, iterated and removed according to the order in which they appear internally in the Java List. The ordering of the elements is why this data structure is called a List.

<http://tutorials.jenkov.com/java-collections/list.html>

## COLLECTIONS IN JAVA. SET

The Java Set interface, java.util.Set, represents a collection of objects where each object in the Set is unique. In other words, the same object cannot occur more than once in a Java Set.

<http://tutorials.jenkov.com/java-collections/set.html>

## COLLECTIONS IN JAVA. QUEUE

The Java Queue interface, java.util.Queue represents a data structure designed to have elements inserted at the end of the queue, and elements removed from the beginning of the queue. This is similar to how a queue in a supermarket works.

<http://tutorials.jenkov.com/java-collections/queue.html>

## COLLECTIONS IN JAVA. MAP

The Java Map interface, java.util.Map, represents a mapping between a key and a value. More specifically, a Java Map can store pairs of keys and values. Each key is linked to a specific value. Once stored in a Map, you can later look up the value using just the key.

<http://tutorials.jenkov.com/java-collections/map.html>

## HASHMAp – how to work iN JAVA

Original: <https://www.geeksforgeeks.org/internal-working-of-hashmap-java/>

Habr.com: <https://habr.com/ru/post/421179/>

## TASK

Add simple cache for your service as in-memory Map where key is an input parameter and value is calculation results. This Map (cache) should be stored as a separate class (bean). Use dependencies injection mechanism of Spring Framework (autowiring) for adding cache to your service.

Web service should check cache to get calculation results from it first, if there are no calculation results for input parameter yet, do the calculation and put it into the cache before returning results in response.

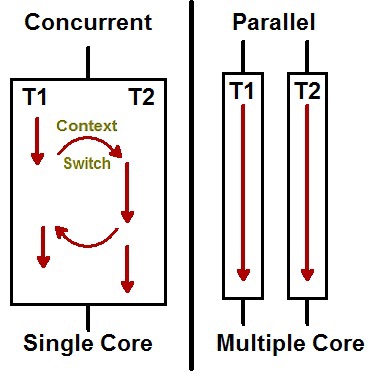
# Concurrency

## Basic concepts

### Concurrent programming

<https://en.wikipedia.org/wiki/Concurrent_computing>

Parallelism vs. Concurrency



### Process vs. threads

Threads are an inescapable feature of the Java language, and they can simplify the development of complex systems by turning complicated asynchronous code into simpler straight-line code. In addition, threads are the easiest way to tap the computing power of multiprocessor systems. And, as processor counts increase, exploiting concurrency effectively will only become more important.

### Concurrency models

<http://tutorials.jenkov.com/java-concurrency/concurrency-models.html>

## JMM

<https://medium.com/platform-engineer/understanding-java-memory-model-1d0863f6d973>

<http://tutorials.jenkov.com/java-concurrency/java-memory-model.html>

## Native java concurrency

<http://tutorials.jenkov.com/java-concurrency/creating-and-starting-threads.html>

<https://www.ibm.com/developerworks/java/library/j-5things15/index.html?ca=drs->

### Task

## java.util.concurrent

Java 5 added a new Java package to the Java platform, the [java.util.concurrent](https://docs.oracle.com/javase/8/docs/api/index.html?java/util/concurrent/package-summary.html) package. This package contains a set of classes that makes it easier to develop concurrent (multithreaded) applications in Java. Before this package was added, you would have to program your utility classes yourself.

The java.util.concurrent contains too many features. It’s recommended to understand at least following utilities in this package:

* Executor
* ExecutorService
* ScheduledExecutorService
* Future
* CountDownLatch
* CyclicBarrier
* Semaphore
* ThreadFactory
* BlockingQueue
* DelayQueue
* Locks
* Phaser

Details: <https://www.baeldung.com/java-util-concurrent>

### Task

## Advanced: java performance monitoring

<https://www.ibm.com/developerworks/java/library/j-5things7/index.html?ca=drs->

<https://www.ibm.com/developerworks/java/library/j-5things8/index.html?ca=drs->

<http://tutorials.jenkov.com/java-performance/index.html>

## Task

1. Add a new service (Counter) which calculates number of your main service calls and returns the number when requested. Counter should be implemented as a separate thread safe bean with synchronized access.

2. Create high load test for your main service using JMeter, Postman or any other tool. The load test should make thousands of calls in short period of time. Then make sure that Counter correctly calculates number of service calls done by the load test.

# Functional Programming in Java 8

## Functional programming in JVM

In common words functional programming is a different way of programming/coding comparing to traditional way of writing code (which is also called imperative programming), when instead of describing steps to perform required operation (describing how to do it) we do number of functions calls to achieve required result (describing what to do). Functional programming provides some benefits (same as some drawbacks which are usually skipped introducing functional programming, so we will skip this here as well), which can be found in detail here:

<https://dzone.com/articles/functional-programming-jvm>

## Lambda Expressions in JAVA

There were several features introduced in Java 8 to support functional programming in Java. Among most important of them was lambda expressions. Lambdas in Java help easily define anonymous functions or create functional interfaces to reuse once written function, and overall in can make your code look more laconic in compact in terms of lines of code. Very simple introduction to lambdas in Java:

<https://medium.freecodecamp.org/learn-these-4-things-and-working-with-lambda-expressions-b0ab36e0fffc>

Official Oracle documentation can be found here:

<https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html>

## Java STREAM API

A bunch of functions were introduced in Java 8 in java.util.stream package aimed to simplify your work with sequences of elements. In particular, stream() method was added to Collection interface, so we can create stream from any collection and use new way to process collections (as alternative to traditional loops used). You can find simplified introduction to Java Streams here:

<https://www.baeldung.com/java-8-streams-introduction>

More detailed overview of Stream API functions can be found here:

<http://tutorials.jenkov.com/java-functional-programming/streams.html>

## BULK OPERATIONS

Bulk operations are operations which helps us to apply same manipulation(s) to a sequence of elements, or stream. Java 8 Stream API can help us to simplify writing bulk operations in Java. Overview on how Java 8 new features can be used for bulk operation can be found here:

<https://zeroturnaround.com/rebellabs/java-8-revealed-lambdas-default-methods-and-bulk-data-operations/>

## TASK

In scope of this practical task we need to try functional programming approach using Java 8 new features:

* Update initial service created in first practical task to have ability to process sequence (stream) of input parameters. Stream API and lambdas should be used for this.
* Add new POST method which should get sequence of input parameters (they can be passed as JSON data in request body, or as simple CSV file), validate them and call new service for processing bulk data. Result can be returned as JSON or CSV file.

# Aggregate Operations in JAVA 8

## Filter and MAP USAGE in JAVA

Map(), filter() and foreach() functions are among most useful when we are doing bulk operations on streams. Common overview of these functions and how they can be used for bulk operations:

<https://jrebel.com/rebellabs/java-8-explained-applying-lambdas-to-java-collections/>

## Aggregate Operations

Having a collection or a stream of data, you may need to do some calculation over the data, for example, calculate a sum or average of the integer element of your collection. Such type of operations called aggregation operations. Please see more info on aggregate operation and how they are used in Java 8 here:

<https://docs.oracle.com/javase/tutorial/collections/streams/index.html>

## Reduce and Collect

Aggregate operation which making some calculation over collection of data also called reduction, since it reduces a big amount of data into some particular smaller values (at least in terms of memory used). There are number of standard aggregate functions in Java8 you can use, but if you need something more specific to do with your stream, you can try reduce() and collect() functions. For more information on them please refer to:

<https://docs.oracle.com/javase/tutorial/collections/streams/reduction.html>

## Task

In previous task we did bulk operation for a stream of input data. In this task we need to apply some statistics calculation using aggregation/reduction. Calculated statistic should be added into POST response data (either JSON or CSV file). Statistics should include:

* Total amount of input parameters (or pairs/triples of parameters) provided
* Total amount of input parameters (or pairs/triples of parameters) which produced incorrect result (were not valid)
* Maximum and minimum values from results calculated
* Result which was returned most times (most popular)

# Data Persistance

## Basic theory

Very important to store result of your operations or intermediate actions apart from application for higher durability: you don’t want to lose all the data if application goes down. The common approach to persist it to database. Currently there are a lot of databases and they are different in terms of data structure and operations.

It’s crucial to understand whether you need SQL or NoSQL database, should it support transactions or not.

It all spins around CAP theorem, so make sure you understand what it is about:

<https://en.wikipedia.org/wiki/CAP_theorem>

Please also check basic materials to understand transaction features for SQL databases:

<https://en.wikipedia.org/wiki/ACID_(computer_science)>

And take a look at why NoSQL corresponds to something called BASE

<https://ru.wikipedia.org/wiki/NoSQL>

Now when you have general understanding of data storage – let’s pick one DB for the upcoming task.

## What about java?

It’s very nice that we have so many options to store the data, but how to do it with Java? Let’s see what tools we have in our toolbox

### JDBC

Java has a nice interface to interfere with DB inside your application and it’s called Java DataBase Connectivity. It basically allows your java program to use the DB driver that you specify to connect to it and offers a nice set of classes and methods to operate with it:

<https://www.tutorialspoint.com/jdbc/jdbc-introduction.htm>

### ORM

ORM or Object Relational Mapping is a common description for the frameworks allowing you to care less about writing queries and care more about specifying object relations to get DB structure wired to your Java classes. There are a lot of frameworks allowing you to do that, most used from them are:

Hibernate, Spring Data, MyBatis.

Here is an easy get-started guide to understands how Hibernate works and it will basically give you an understanding of how ORM frameworks work in general:

<https://www.baeldung.com/hibernate-ogm>

## Task

We are going to wire persistence layer for storage of inbound request data and outbound response. Feel free to pick whatever database that will satisfy your needs but be ready to explain your choice.

For NoSQL option there is a nice DB called MongoDB:

[https://www.mongodb.com](https://www.mongodb.com/)

They usually have a lot of free training, so you can consider learning that in future:

[https://university.mongodb.com](https://university.mongodb.com/)

There is a nice and lightweight option for SQL DB: MySQL

<https://www.mysql.com/downloads/>

Checkout MySQL tutorial to jump straight to development and get nice base ground of its features: [http://www.mysqltutorial.org](http://www.mysqltutorial.org/)

To practice your SQL skills, use this resource: [http://www.sql-ex.ru](http://www.sql-ex.ru/)

Adjust your web service with ability to store all calculation results of the service in database or file using one of standard persistence framework (Spring Data/Hibernate/MyBatis).

# Asyncroneous Calls

## Asynchronous calls

In some cases services on BE side, or remote calls to some third party service can take a while to complete, which is blocking all further calculation steps until previous step results are received with standard synchronous calculation approach. But what if you actually do not need to wait for long-lasting call results to proceed? Non-blocking asynchronous calls can help with it. Java provides several ways for non-blocking asynchronous calls, please see the review here: <https://www.baeldung.com/java-asynchronous-programming>

## ASYnc CALLS IN JAVA8

Among newly provided features introduced in Java 8, the one which drastically improved our ability to work with asynchronous calls was CompletableFuture. More information:  
<https://www.deadcoderising.com/java8-writing-asynchronous-code-with-completablefuture/>

## TASKS

Improve your web-service by adding asynchronous call for main calculation service and providing REST call response containing auto-generated processID (random number, which should be used as a key to store calculation results in database) without waiting for the calculation results. All calculation results should be written into database (storage) within asynchronously called service. Add a rest endpoint to get calculation results from database by processID provided.

| REVISION HISTORY | | | | | |
| --- | --- | --- | --- | --- | --- |
| Ver. | Description of Change | Author | Date | Approved | |
| Name | Effective Date |
| 0.1 | Draft template preparation | Andrei Finski | 4-Mar-2019 |  |  |
| 0.2 | Filled tasks options from BSUIR legacy | Andrei Finski | 24-Mar-2019 |  |  |
| 0.2.1 | Add description to Exception and Error handling | Aliaksei Sychou | 29-Mar-2019 |  |  |
| 0.3 | Functional Programming in Java 8 added | Dzmitry Marchankau | 29-Mar-2019 |  |  |
| 0.4 | Aggregate operations in Java 8 added | Dzmitry Marchankau | 09-Apr-2019 |  |  |
| 0.5 | Filled Persistance section | Andrei Finski | 12-May-19 |  |  |
| 0.6 | Added Async calls section Updated tasks descriptions | Dzmitry Marchankau | 21-Jan-20 |  |  |