

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ В.Н.

КАРАЗІНА

Навчально-науковий інститут комп'ютерних наук та штучного
інтелекту

Кафедра інтелектуальних програмних систем і технологій

ЛАБОРАТОРНА РОБОТА № 1

на тему: «Spring boot»

дисципліна: «Архітектури і патерни проектування
інформаційних систем»

Виконала: студентка 4 курсу групи КС41

Спеціальності 122 «Комп'ютерні науки»

Манюк В.П

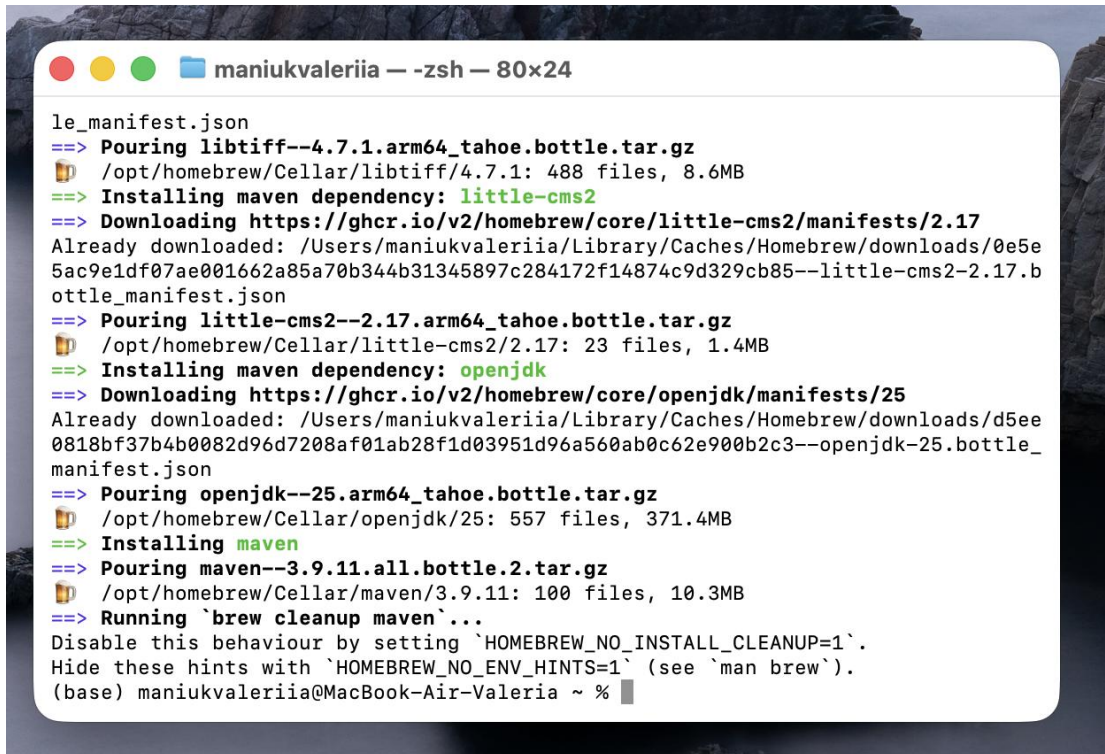
Харків

2025

Мета: Ознайомитись з технологією Spring Boot

Хід роботи

Для початку роботи було завантажено необхідні пакети для створення проєкту: `java -version`, `brew -v`, `brew install maven`, `mvn -version`



```

le_manifest.json
==> Pouring libtiff--4.7.1.arm64_tahoe.bottle.tar.gz
  /opt/homebrew/Cellar/libtiff/4.7.1: 488 files, 8.6MB
==> Installing maven dependency: little-cms2
==> Downloading https://ghcr.io/v2/homebrew/core/little-cms2/manifests/2.17
Already downloaded: /Users/maniukvaleriia/Library/Caches/Homebrew/downloads/0e5e5ac9e1df07ae001662a85a70b344b31345897c284172f14874c9d329cb85--little-cms2-2.17.bottle_manifest.json
==> Pouring little-cms2--2.17.arm64_tahoe.bottle.tar.gz
  /opt/homebrew/Cellar/little-cms2/2.17: 23 files, 1.4MB
==> Installing maven dependency: openjdk
==> Downloading https://ghcr.io/v2/homebrew/core/openjdk/manifests/25
Already downloaded: /Users/maniukvaleriia/Library/Caches/Homebrew/downloads/d5ee0818bf37b4b0082d96d7208af01ab28f1d03951d96a560ab0c62e900b2c3--openjdk-25.bottle_manifest.json
==> Pouring openjdk--25.arm64_tahoe.bottle.tar.gz
  /opt/homebrew/Cellar/openjdk/25: 557 files, 371.4MB
==> Installing maven
==> Pouring maven--3.9.11.all.bottle.2.tar.gz
  /opt/homebrew/Cellar/maven/3.9.11: 100 files, 10.3MB
==> Running `brew cleanup maven`...
Disable this behaviour by setting `HOMEBREW_NO_INSTALL_CLEANUP=1`.
Hide these hints with `HOMEBREW_NO_ENV_HINTS=1` (see `man brew`).
(base) maniukvaleriia@MacBook-Air-Valeria ~ %

```

Рисунок 1 - Початок роботи

Створено новий проєкт у середовищі VS Code за допомогою Spring Initializr. Задано назву `springbootconsoleapp`, групу `com.example`, артефакт `springbootconsoleapp`, пакет `com.example.springbootconsoleapp`. Обрано мову Java, систему збірки Maven та версію JDK 17.

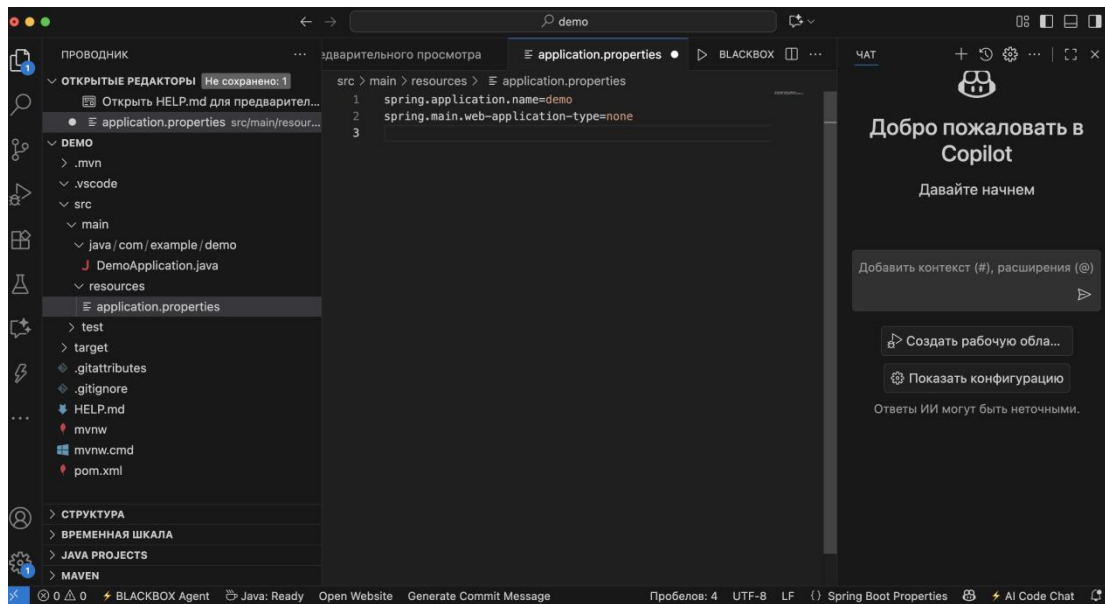


Рисунок 2 – Створення нового проєкту Spring Boot

Після генерації проєкту створено стандартну структуру Maven: каталоги `src/main/java`, `src/main/resources` та `src/test/java`. IntelliJ згенерувала початковий клас `Lab1Application` та файл конфігурації `application.properties`.

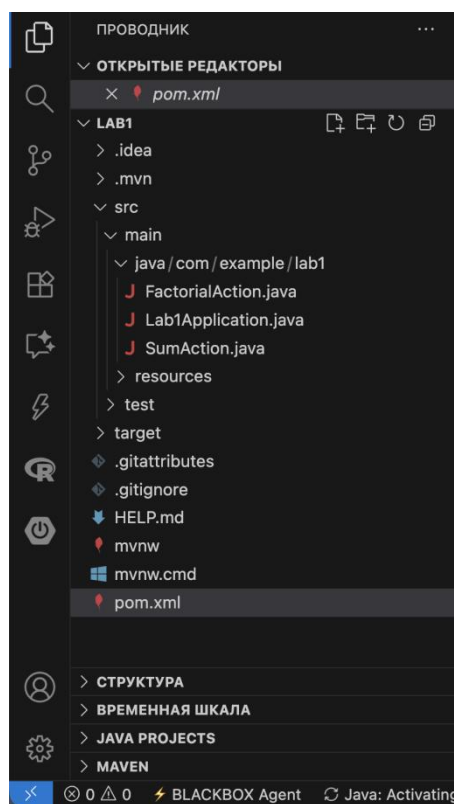
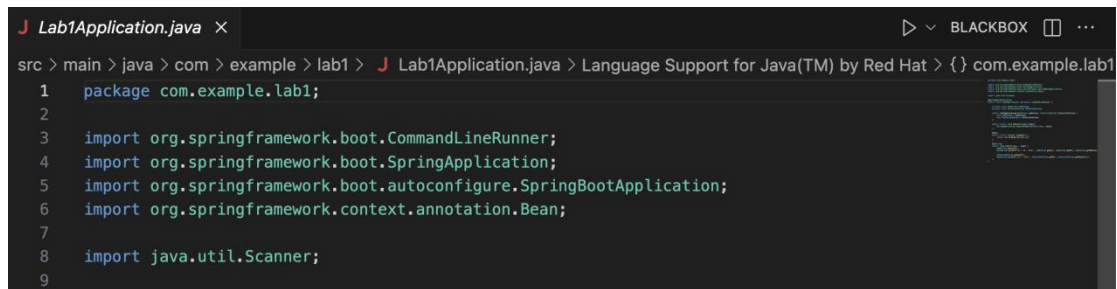


Рисунок 3 – Структура створеного проєкту



```

src > main > java > com > example > lab1 > J Lab1Application.java > Language Support for Java(TM) by Red Hat > {} com.example.lab1
1  package com.example.lab1;
2
3  import org.springframework.boot.CommandLineRunner;
4  import org.springframework.boot.SpringApplication;
5  import org.springframework.boot.autoconfigure.SpringBootApplication;
6  import org.springframework.context.annotation.Bean;
7
8  import java.util.Scanner;
9

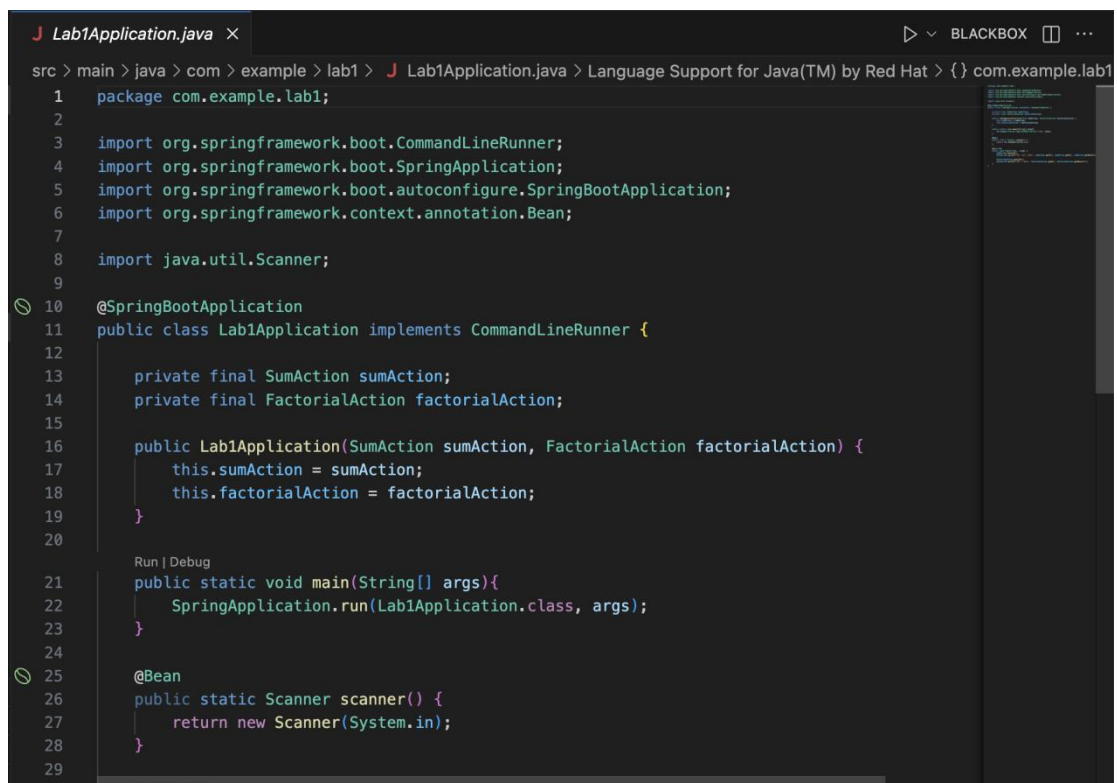
```

Рисунок 4 – Початковий клас Lab1Application

Автоматично згенерований клас Lab1Application містить метод main() із запуском Spring Boot через SpringApplication.run().

До цього класу додано реалізацію інтерфейсу CommandLineRunner, що дозволяє виконувати власний код після старту застосунку у методі run().

У цьому методі здійснюється ін'єкція залежності — об'єкт класу SumAction передається через конструктор (Dependency Injection). Далі викликається метод execute(), який зчитує вхідні дані користувача, обчислює суму введених чисел і виводить результат у КОНСОЛЬ.



```

J Lab1Application.java x
src > main > java > com > example > lab1 > J Lab1Application.java > Language Support for Java(TM) by Red Hat > {} com.example.lab1
1  package com.example.lab1;
2
3  import org.springframework.boot.CommandLineRunner;
4  import org.springframework.boot.SpringApplication;
5  import org.springframework.boot.autoconfigure.SpringBootApplication;
6  import org.springframework.context.annotation.Bean;
7
8  import java.util.Scanner;
9
10 @SpringBootApplication
11 public class Lab1Application implements CommandLineRunner {
12
13     private final SumAction sumAction;
14     private final FactorialAction factorialAction;
15
16     public Lab1Application(SumAction sumAction, FactorialAction factorialAction) {
17         this.sumAction = sumAction;
18         this.factorialAction = factorialAction;
19     }
20
21     Run | Debug
22     public static void main(String[] args){
23         SpringApplication.run(Lab1Application.class, args);
24     }
25
26     @Bean
27     public static Scanner scanner() {
28         return new Scanner(System.in);
29     }
30

```

Рисунок 5 – Модифікований клас Lab1Application

У конфігураційний файл додано параметр:

```
spring.main.web-application-type=NONE
```

Це вимикає веб-режим і дозволяє запускати застосунок як КОНСОЛЬНИЙ.

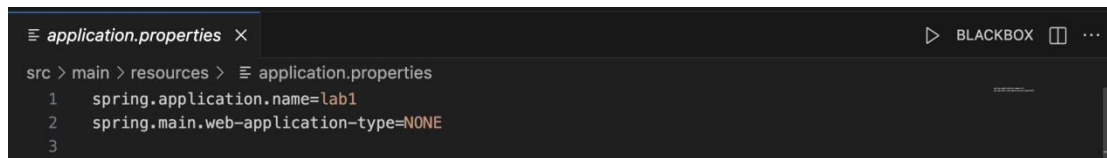


Рисунок 6 – Файл application.properties

Було створено клас SumAction, який містить поля a, b та result. Залежність Scanner впроваджено через ін'єкцію залежностей (Dependency Injection) — об'єкт Scanner передається у конструктор класу. Основна логіка розміщена в методі execute(), який зчитує два цілі числа з клавіатури, обчислює їх суму та зберігає результат у полі result. Клас має гетери для доступу до значень змінних.

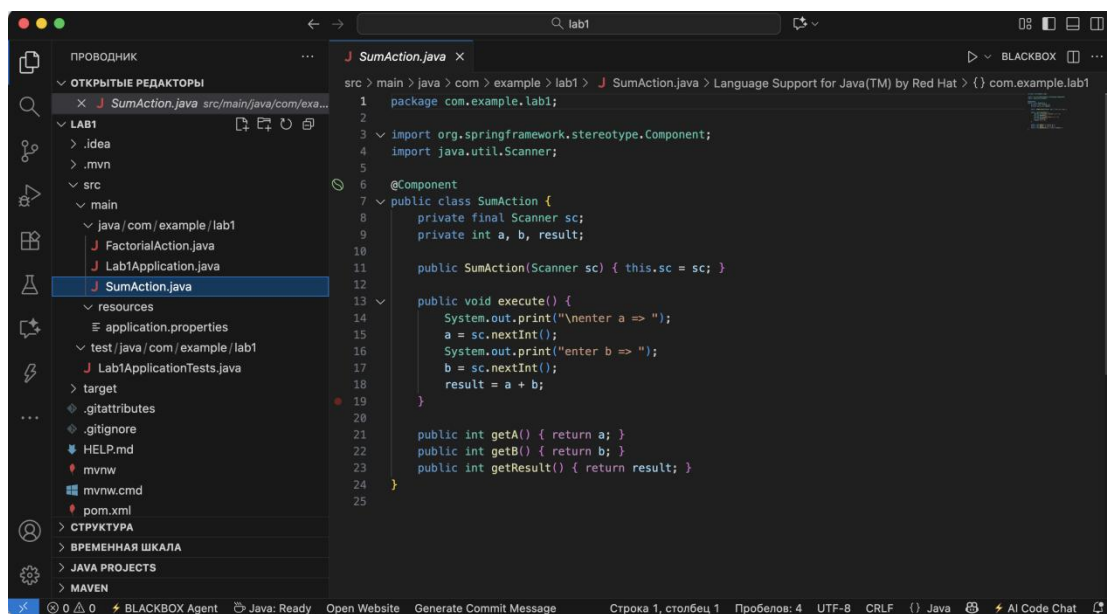


Рисунок 7 – Клас SumAction

Після запуску програми у консолі виводиться банер Spring Boot та службові повідомлення. Далі користувач вводить два числа, після чого програма обчислює їх суму та завершує роботу.

```

ТЕРМИНАЛ
[INFO] Nothing to compile - all classes are up to date.
[INFO]
[INFO] <<< spring-boot:3.5.6:run (default-cli) < test-compile @ lab1
[INFO]
[INFO]
[INFO] --- spring-boot:3.5.6:run (default-cli) @ lab1 ---
[INFO] Attaching agents: []

:: Spring Boot ::
(v3.5.6)

2025-10-13T12:12:16.670+02:00 INFO 13715 --- [lab1] [ restartedMain
n] com.example.lab1.Lab1Application : Starting Lab1Application
on using Java 25 with PID 13715 (/Users/maniukvaleriia/Desktop/lab1/
target/classes started by maniukvaleriia in /Users/maniukvaleriia/De
sktop/lab1)
2025-10-13T12:12:16.671+02:00 INFO 13715 --- [lab1] [ restartedMain
n] com.example.lab1.Lab1Application : No active profile set,
falling back to 1 default profile: "default"
2025-10-13T12:12:16.690+02:00 INFO 13715 --- [lab1] [ restartedMain
n] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defa
ults active! Set 'spring.devtools.add-properties' to 'false' to disa

```

Рисунок 8 - Результат запуску програми

У класі Lab1Application додано створення ще одного об'єкта — FactorialAction. Тепер програма виконує не тільки додавання, а й обчислення факторіала.

```

package com.example.lab1;

import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import java.util.Scanner;

@SpringBootApplication
public class Lab1Application implements CommandLineRunner {

    private final SumAction sumAction;
    private final FactorialAction factorialAction;

    public Lab1Application(SumAction sumAction, FactorialAction factorialAction) {
        this.sumAction = sumAction;
        this.factorialAction = factorialAction;
    }

    public static void main(String[] args){
        SpringApplication.run(Lab1Application.class, args);
    }

    @Bean
    public static Scanner scanner() {
        return new Scanner(System.in);
    }
}

```

Рисунок 9 – Розширений клас Lab1Application

Клас `FactorialAction`, який реалізує обчислення факторіала числа. У класі використано ін'єкцію залежностей (Dependency Injection) — об'єкт `Scanner` передається через конструктор. Метод `execute()` зчитує з клавіатури значення `n`, обчислює факторіал у циклі та зберігає результат у полі `result`. Клас має гетери `getN()` та `getResult()` для доступу до відповідних полів.

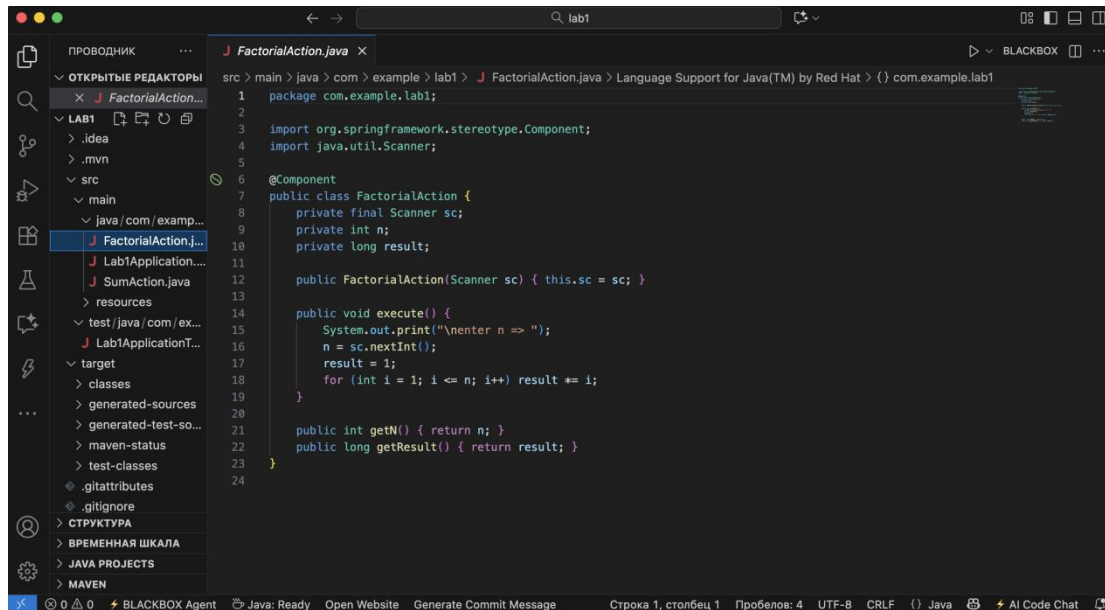


Рисунок 10 — Клас `FactorialAction` із реалізацією через ін'єкцію залежностей

```

TERМИНАЛ
n] com.example.lab1.Lab1Application : No active profile set,
falling back to 1 default profile: "default"
2025-10-13T12:12:16.690+02:00 INFO 13715 --- [lab1] [ restartedMai
n] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defa
ults active! Set 'spring.devtools.add-properties' to 'false' to disa
ble
2025-10-13T12:12:16.873+02:00 INFO 13715 --- [lab1] [ restartedMai
n] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is r
unning on port 35729
2025-10-13T12:12:16.880+02:00 INFO 13715 --- [lab1] [ restartedMai
n] com.example.lab1.Lab1Application : Started Lab1Applicatio
n in 0.357 seconds (process running for 0.505)

enter 12
enter b => 17
12 + 17 = 29

enter n => 29
29! = -7055958792655077376
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 02:20 min
[INFO] Finished at: 2025-10-13T12:14:35+02:00
[INFO] -----
(base) maniukvaleriia@MacBook-Air-Valeria lab1 %
Java: Ready Open Website Generate Commit Message Размер ин

```

Рисунок 11 - Результат запуску програми

У консолі відображено приклад роботи програми: користувач вводить числа, після чого виводиться результат додавання та факторіал введеного числа.

Контрольні запитання

1. Які переваги використання Spring Boot?

Spring Boot спрощує створення застосунків завдяки автоматичній конфігурації, вбудованому вебсерверу, готовим залежностям (starter'ам) і мінімальній кількості ручних налаштувань.

2. Які ключові компоненти Spring Boot?

Основні компоненти: Spring Boot Starters (набір готових залежностей), Spring Boot AutoConfiguration (автоматичне налаштування), Spring Boot CLI (консольний інструмент) та Actuator (моніторинг і керування).

3. Як працює Spring Boot?

Spring Boot автоматично підключає потрібні залежності й виконує конфігурацію на основі вмісту проєкту. Це дозволяє швидко запускати застосунок без тривалого налаштування.

4. Як можна увімкнути функції Spring Boot?

Функції активуються через залежності у `pom.xml` або `build.gradle`, використання анотацій (наприклад, `@SpringBootApplication`, `@EnableAutoConfiguration`) і налаштувань у файлі `application.properties` чи `application.yml`.

5. Як запускається програма для завантаження Spring?

Програма запускається через метод `main()`, у якому викликається `SpringApplication.run(...)`. Це створює і завантажує Spring Application Context.

6. Для чого потрібна анотація @SpringBootApplication?

Ця анотація поєднує три інші (`@Configuration`, `@EnableAutoConfiguration`, `@ComponentScan`) і вмикає автоматичне налаштування, конфігурацію та сканування компонентів у проєкті.

Висновки

Під час виконання лабораторної роботи було створено консольний застосунок на основі **Spring Boot** для обчислення факторіала числа. Проєкт згенеровано за допомогою **Spring Initializr**, обрано систему збірки **Maven**, мову **Java** та додано необхідні залежності. У процесі реалізації розроблено клас **FactorialService**, який виконує обчислення, та головний клас, що забезпечує запуск і відображення результатів у консолі. Результати виконання підтвердили правильну роботу застосунку. У ході роботи було закріплено навички створення консольних застосунків у Spring Boot і налаштування їх параметрів для консольного режиму.