

Язык Java

Осенний семестр 2023, магистратура, 1 курс

Лекция 6: Spring Boot

Содержание

1. Фреймворк Spring и его основные компоненты
2. Spring Boot
3. Создание проектов Spring с помощью Spring Initializr
4. Проект "Hello, World!" на Spring
5. IoC-контейнер Spring
6. Работа с конфигурационными параметрами
7. Профили

Spring

- Универсальный фреймворк (каркас) для создания современных корпоративных приложений
- Бесплатный фреймворк с открытым кодом
 - Ссылка на гитхаб: <https://github.com/spring-projects/spring-framework>
- Официальная страница проекта: <https://spring.io/projects/spring-framework>

Основные компоненты фреймворка Spring

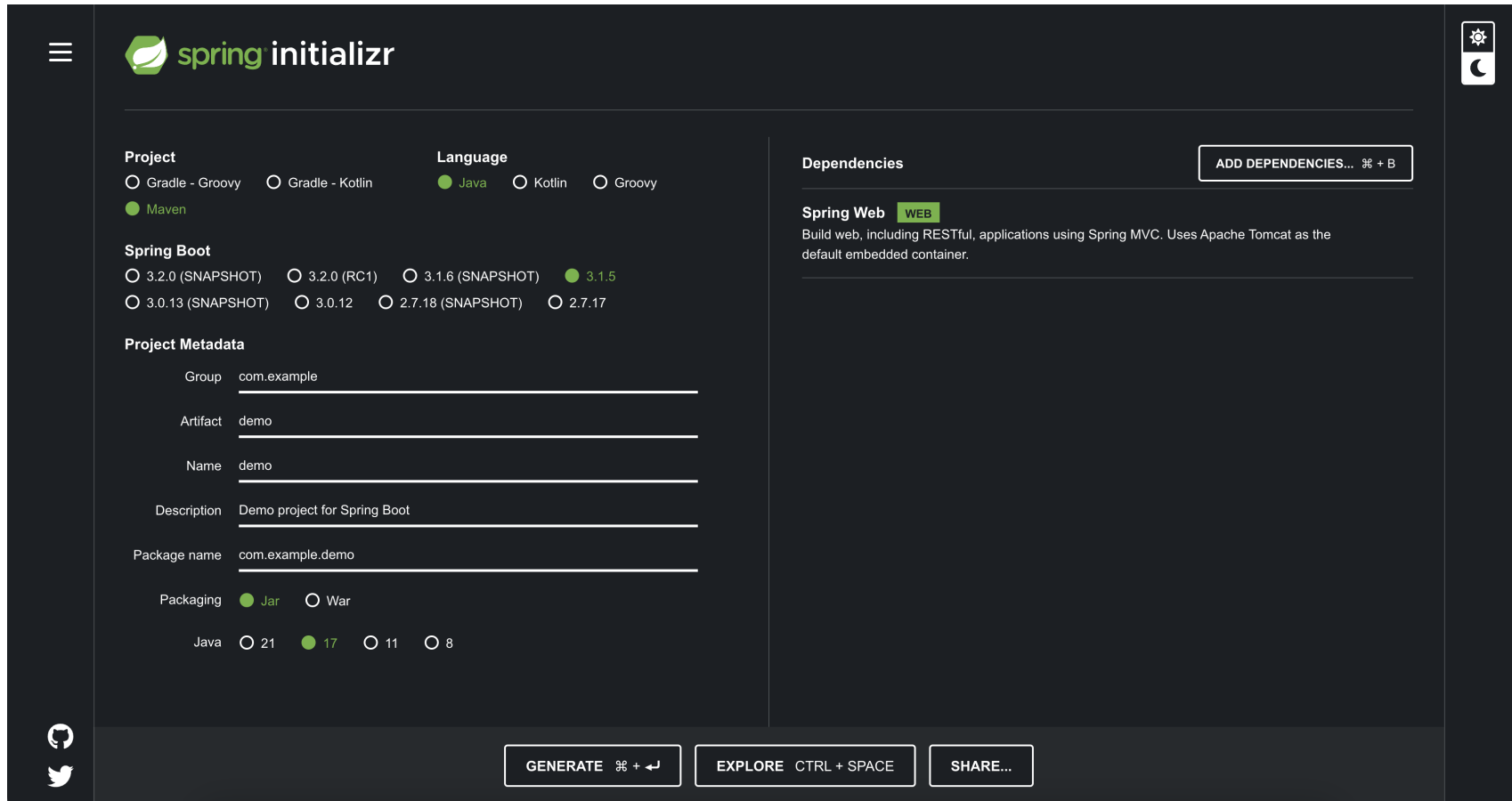
- **IoC-контейнер** (IoC - инверсия управления)
- **Spring MVC** (создание веб-приложений)
- **Spring Data** (работа с базами данных)
- **Spring Security** (управление аутентификацией и авторизацией)
- **Spring Test** (создание юнит- и интеграционных тестов)

Spring Boot

- Расширение фреймворка Spring
- Упрощает создание приложений за счет уменьшения затрат на конфигурирование и написание шаблонного кода для инициализации приложения
- Главные фичи:
 - "Стартеры" (Maven-зависимости) для простоты управления зависимостями
 - Встроенный веб-сервер (Apache Tomcat по умолчанию)
 - Автоматическая конфигурация

Spring Initializr

Ссылка: <https://start.spring.io/>



The screenshot shows the Spring Initializr web application interface. The header includes the Spring logo and the text "spring initializr". The main content area is divided into several sections:

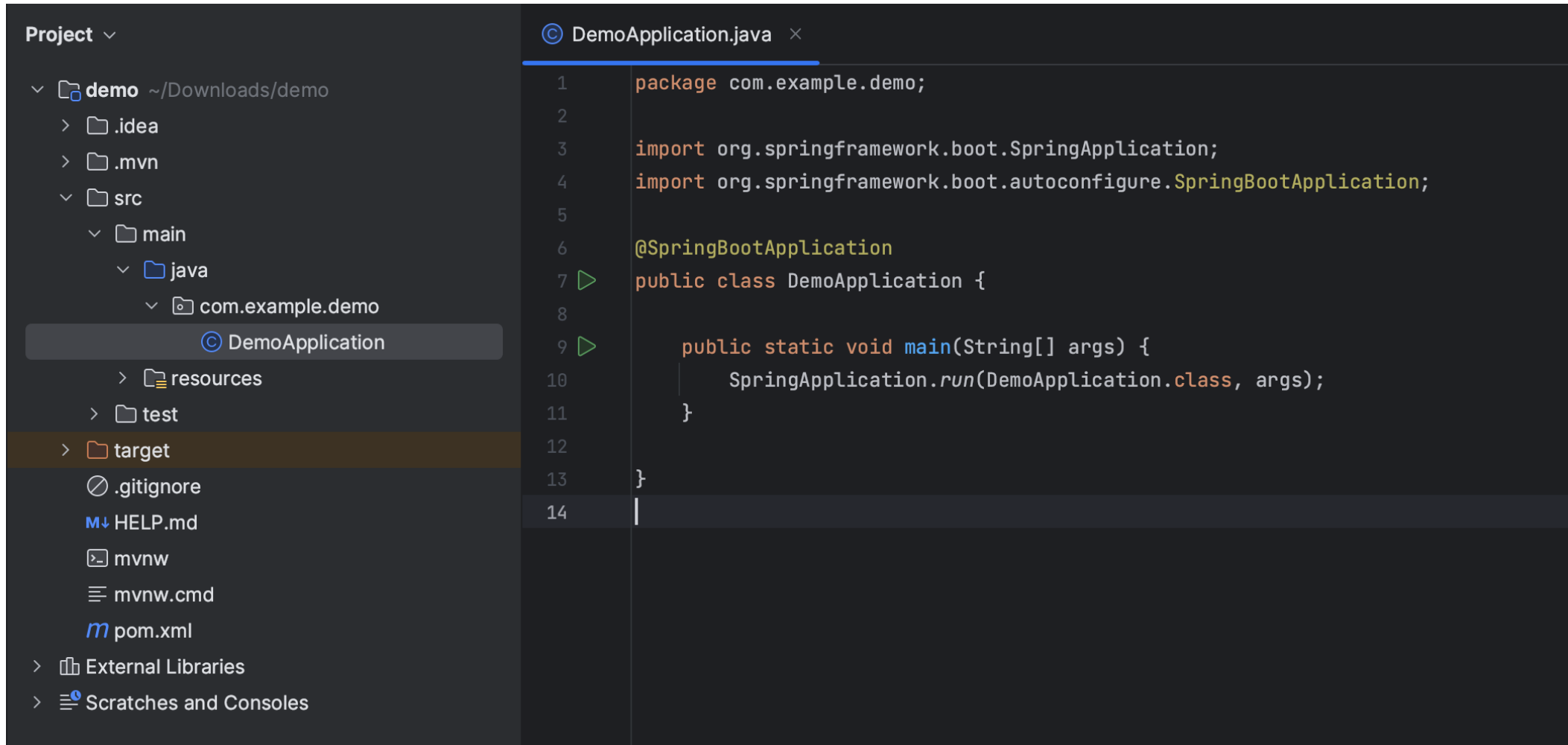
- Project:** Radio buttons for "Gradle - Groovy", "Gradle - Kotlin", "Java" (selected), "Kotlin", and "Groovy". Below this, "Maven" is also selected.
- Spring Boot:** Radio buttons for "3.2.0 (SNAPSHOT)", "3.2.0 (RC1)", "3.1.6 (SNAPSHOT)", "3.1.5" (selected), "3.0.13 (SNAPSHOT)", "3.0.12", "2.7.18 (SNAPSHOT)", and "2.7.17".
- Project Metadata:** Text input fields for "Group" (com.example), "Artifact" (demo), "Name" (demo), "Description" (Demo project for Spring Boot), and "Package name" (com.example.demo). Below these, "Packaging" has radio buttons for "Jar" (selected) and "War". At the bottom, "Java" has radio buttons for "21", "17" (selected), "11", and "8".
- Dependencies:** A section titled "Spring Web" with a "WEB" tag. Below it, a description reads: "Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container." A button "ADD DEPENDENCIES... ⌘ + B" is located at the top right of this section.

At the bottom of the interface, there are three buttons: "GENERATE ⌘ + ↵", "EXPLORE CTRL + SPACE", and "SHARE...".

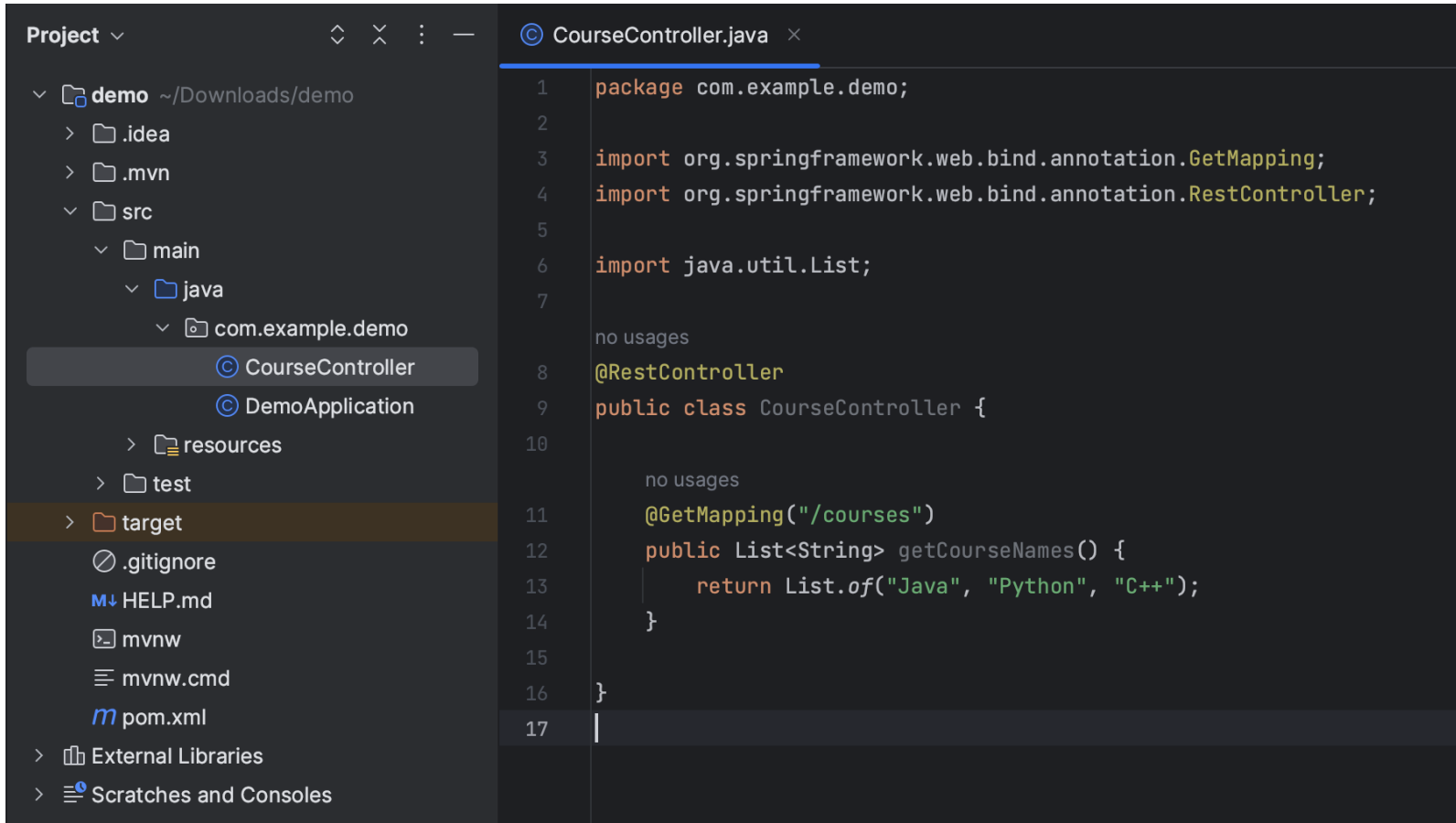
Демо-проект Spring Boot: pom.xml

```
m pom.xml (demo) x
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
4   <modelVersion>4.0.0</modelVersion>
5   <parent>
6     <groupId>org.springframework.boot</groupId>
7     <artifactId>spring-boot-starter-parent</artifactId>
8     <version>3.1.5</version>
9     <relativePath/> <!-- lookup parent from repository -->
10  </parent>
11  <groupId>com.example</groupId>
12  <artifactId>demo</artifactId>
13  <version>0.0.1-SNAPSHOT</version>
14  <name>demo</name>
15  <description>Demo project for Spring Boot</description>
16  <properties>
17    <java.version>17</java.version>
18  </properties>
19  <dependencies>
20    <dependency>
21      <groupId>org.springframework.boot</groupId>
22      <artifactId>spring-boot-starter-web</artifactId>
23    </dependency>
24
25    <dependency>
26      <groupId>org.springframework.boot</groupId>
27      <artifactId>spring-boot-starter-test</artifactId>
28      <scope>test</scope>
29    </dependency>
30  </dependencies>
31
32  <build>
33    <plugins>
34      <plugin>
35        <groupId>org.springframework.boot</groupId>
36        <artifactId>spring-boot-maven-plugin</artifactId>
37      </plugin>
38    </plugins>
39  </build>
40
41 </project>
```

Демо-проект Spring Boot: метод main

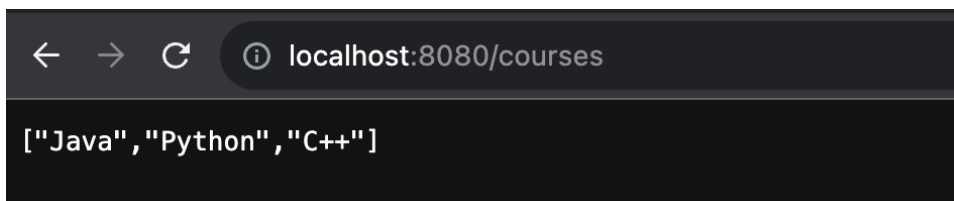


Пример: REST-контроллер



The screenshot shows an IDE with a project explorer on the left and a code editor on the right. The project explorer shows a project named 'demo' with a directory structure including 'src/main/java/com.example.demo'. The file 'CourseController.java' is selected. The code editor shows the following Java code:

```
1 package com.example.demo;
2
3 import org.springframework.web.bind.annotation.GetMapping;
4 import org.springframework.web.bind.annotation.RestController;
5
6 import java.util.List;
7
8 no usages
9 @RestController
10 public class CourseController {
11
12     no usages
13     @GetMapping("/courses")
14     public List<String> getCourseNames() {
15         return List.of("Java", "Python", "C++");
16     }
17 }
```



The screenshot shows a web browser with the address bar displaying 'localhost:8080/courses'. The response body shows the following JSON array:

```
["Java","Python","C++"]
```

Пример: сервис

```
1 package com.example.demo;
2
3 import org.springframework.web.bind.annotation.GetMapping;
4 import org.springframework.web.bind.annotation.PostMapping;
5 import org.springframework.web.bind.annotation.RequestBody;
6 import org.springframework.web.bind.annotation.RestController;
7
8 import java.util.List;
9
10 no usages
11 @RestController
12 public class CourseController {
13
14     2 usages
15     private final CourseService courseService = new CourseService();
16
17     no usages
18     @GetMapping("/courses")
19     public List<String> getCourses() {
20         return courseService.getCourses();
21     }
22
23     no usages
24     @PostMapping("/course")
25     public void addCourse(@RequestBody String course) {
26         courseService.addCourse(course);
27     }
28 }
```

```
1 package com.example.demo;
2
3 import org.springframework.stereotype.Service;
4
5 import java.util.ArrayList;
6 import java.util.List;
7
8 2 usages
9 @Service
10 public class CourseService {
11
12     2 usages
13     private final List<String> courses = new ArrayList<>();
14
15     1 usage
16     public List<String> getCourses() {
17         return courses;
18     }
19
20     1 usage
21     public void addCourse(String course) {
22         courses.add(course);
23     }
24 }
```

Вопрос: какие недостатки есть у такой реализации?

IoC-контейнер

Инверсия управления (inversion of control) – делегирование фреймворку (контейнеру) управления (в частности, создания) объектов

Внедрение зависимостей (dependency injection) через конструктор:

```
1  package com.example.demo;
2
3  > import ...
9
   no usages
10  @RestController
11  public class CourseController {
12
   3 usages
13      private final CourseService courseService;
14
   no usages
15  > public CourseController(CourseService courseService) { this.courseService = courseService; }
18
   no usages
19      @GetMapping("/courses")
20  > public List<String> getCourses() { return courseService.getCourses(); }
23
   no usages
24      @PostMapping("/course")
25  > public void addCourse(@RequestBody String course) { courseService.addCourse(course); }
28
29  }
```

Аннотация @Autowired

Внедрение зависимости через поле

```
1 package com.example.demo;
2
3 > import ...
4
5 no usages
6
7 @RestController
8 public class CourseController {
9
10     1 usage
11     @Autowired
12     private CourseService courseService;
13
14     no usages
15     @GetMapping("/courses")
16     > public List<String> getCourses() { return courseService.getCourses(); }
17
18 }
19
20
21
22
23
```

Внедрение зависимости через сеттер

```
1 package com.example.demo;
2
3 > import ...
4
5 no usages
6
7 @RestController
8 public class CourseController {
9
10     2 usages
11     private CourseService courseService;
12
13     no usages
14     @GetMapping("/courses")
15     > public List<String> getCourses() { return courseService.getCourses(); }
16
17     no usages
18     @Autowired
19     private void setCourseService(CourseService courseService) {
20         System.out.println("Inject course service");
21         this.courseService = courseService;
22     }
23
24 }
25
26
27
```

Упражнение 1

Как Вы думаете, корректен ли следующий код?

```
interface CourseService {  
    List<String> getCourses();  
}  
  
@Service  
class CourseServiceImpl implements CourseService {  
    public List<String> getCourses() { return List.of("Java", "Python", "C++"); }  
}  
  
@RestController  
class CourseController {  
    @Autowired  
    private CourseService courseService;  
  
    @GetMapping("/courses")  
    public List<String> getCourses() { return courseService.getCourses(); }  
}
```

Упражнение 2

А что насчет следующего кода?

```
interface CourseService {
    List<String> getCourses();
}

@Service
class CourseServiceImpl1 implements CourseService {
    public List<String> getCourses() { return List.of("Java", "Python", "C++"); }
}

@Service
class CourseServiceImpl2 implements CourseService {
    public List<String> getCourses() { return List.of("Algebra", "Geometry"); }
}

@RestController
class CourseController {
    @Autowired
    private CourseService courseService;

    @GetMapping("/courses")
    public List<String> getCourses() { return courseService.getCourses(); }
}
```

Аннотация `@Primary`

- В случае конфликта (если тип для внедряемой зависимости не может быть однозначно определен) будет выбран тип с аннотацией `@Primary` (если такой имеется).
- В противном случае будет сгенерировано исключение `NoUniqueBeanDefinitionException`

```
interface CourseService {
    List<String> getCourses();
}

@Service
@Primary
class CourseServiceImpl1 implements CourseService {
    public List<String> getCourses() { return List.of("Java", "Python", "C++"); }
}

@Service
class CourseServiceImpl2 implements CourseService {
    public List<String> getCourses() { return List.of("Algebra", "Geometry"); }
}

@RestController
class CourseController {
    @Autowired
    private CourseService courseService; // CourseServiceImpl1

    @GetMapping("/courses")
    public List<String> getCourses() { return courseService.getCourses(); }
}
```

Аннотация `@Qualifier`

Позволяет внедрять зависимости по названию

```
interface CourseService {
    List<String> getCourses();
}

@Service
@Qualifier("programmingCourses")
class CourseServiceImpl1 implements CourseService {
    public List<String> getCourses() { return List.of("Java", "Python", "C++"); }
}

@Service
@Qualifier("mathCourses")
class CourseServiceImpl2 implements CourseService {
    public List<String> getCourses() { return List.of("Algebra", "Geometry"); }
}

@RestController
class CourseController {
    @Autowired
    @Qualifier("programmingCourses")
    private CourseService courseService; // CourseServiceImpl1

    @GetMapping("/courses")
    public List<String> getCourses() { return courseService.getCourses(); }
}
```


Интерфейс `ApplicationContext`

- Интерфейс для работы с IoC-контейнером Spring
- Объекты (зависимости), которые управляются IoC-контейнером, называются **бинами** (Spring Bean).

```
package com.example.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;

@SpringBootApplication
public class DemoApplication {

    public static void main(String[] args) {
        ApplicationContext context = SpringApplication.run(DemoApplication.class, args);
        int count = context.getBeanDefinitionCount(); // 138
        CourseService service = context.getBean(CourseService.class); // CourseServiceImpl
    }
}
```

Конфигурация бинов на основе аннотаций

Аннотации для классов

- `@Component` (автоматическое создание и конфигурация бина)
- Частные случаи `@Component` с указанием роли класса:
 - `@Service` (бизнес-логика приложения)
 - `@Controller`, `@RestController` (обработка запросов)
 - `@Repository` (доступ к базам данных)
 - `@Configuration` (класс-контейнер с определением бинов)

Конфигурация бинов Java-кодом

- `@Configuration` - аннотация для класса, указывающая на то, что класс содержит бины, управляемые IoC-контейнером Spring
- `@Bean` - аннотация для метода, указывающая на то, что метод создает объект (бин), которым будет управлять IoC-контейнер Spring

```
public class CourseService {  
    private final List<String> courses;  
  
    public void initialize() {  
        courses = List.of("Java", "Python");  
    }  
  
    public List<String> getCourses() {  
        return courses;  
    }  
}
```

```
@Configuration  
public class AppConfig {  
    @Bean  
    public CourseService courseService() {  
        CourseService service = new CourseService();  
        service.initialize();  
        return service;  
    }  
}
```

Области видимости бинов (scopes)

- **Singleton** (область видимости по умолчанию): IoC-контейнер создает ровно один экземпляр объекта и помещает его в кэш

```
@Service
@Scope("singleton") // указывать необязательно
class CourseService {}

@RestController
class CourseController {
    @Autowired
    CourseService service1;

    @Autowired
    CourseService service2;

    public boolean compare() {
        return service1 == service2; // вернет true
    }
}
```

- **Prototype:** IoC-контейнер создает новый экземпляр бина каждый раз при использовании бина

```
@Service
@Scope("prototype")
class CourseService {}

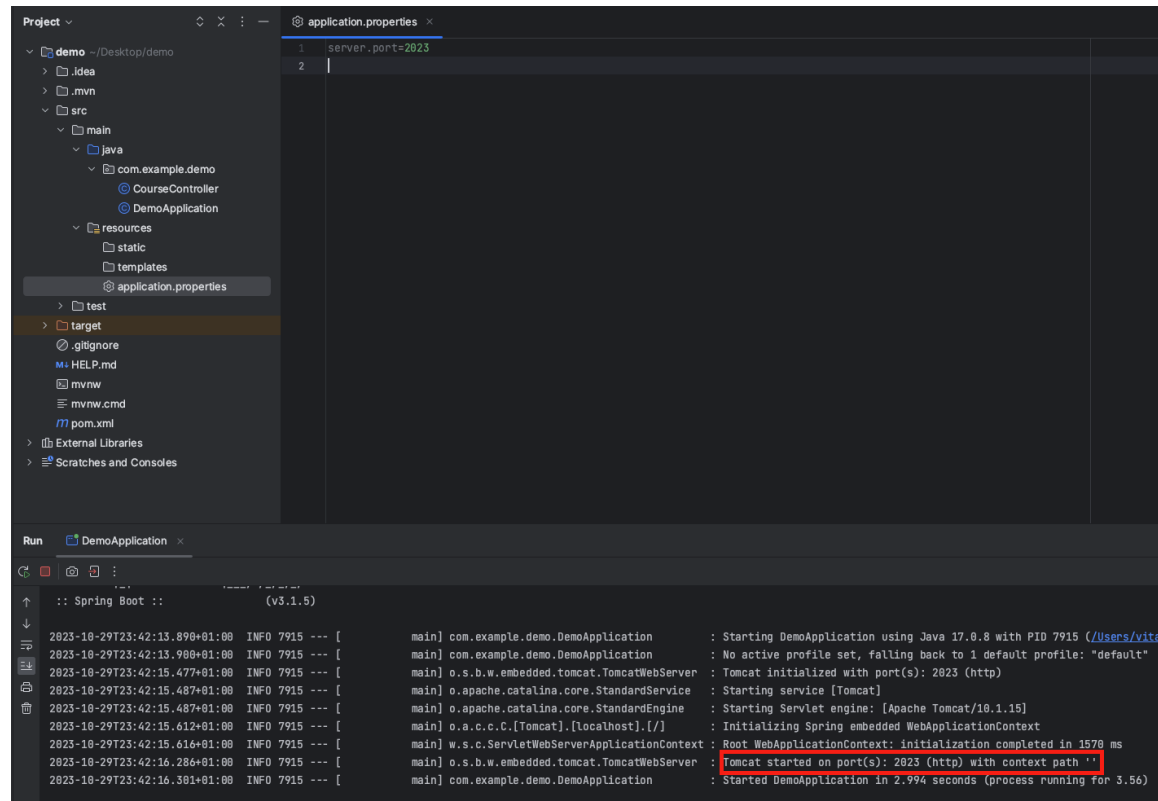
@RestController
class CourseController {
    @Autowired
    CourseService service1;

    @Autowired
    CourseService service2;

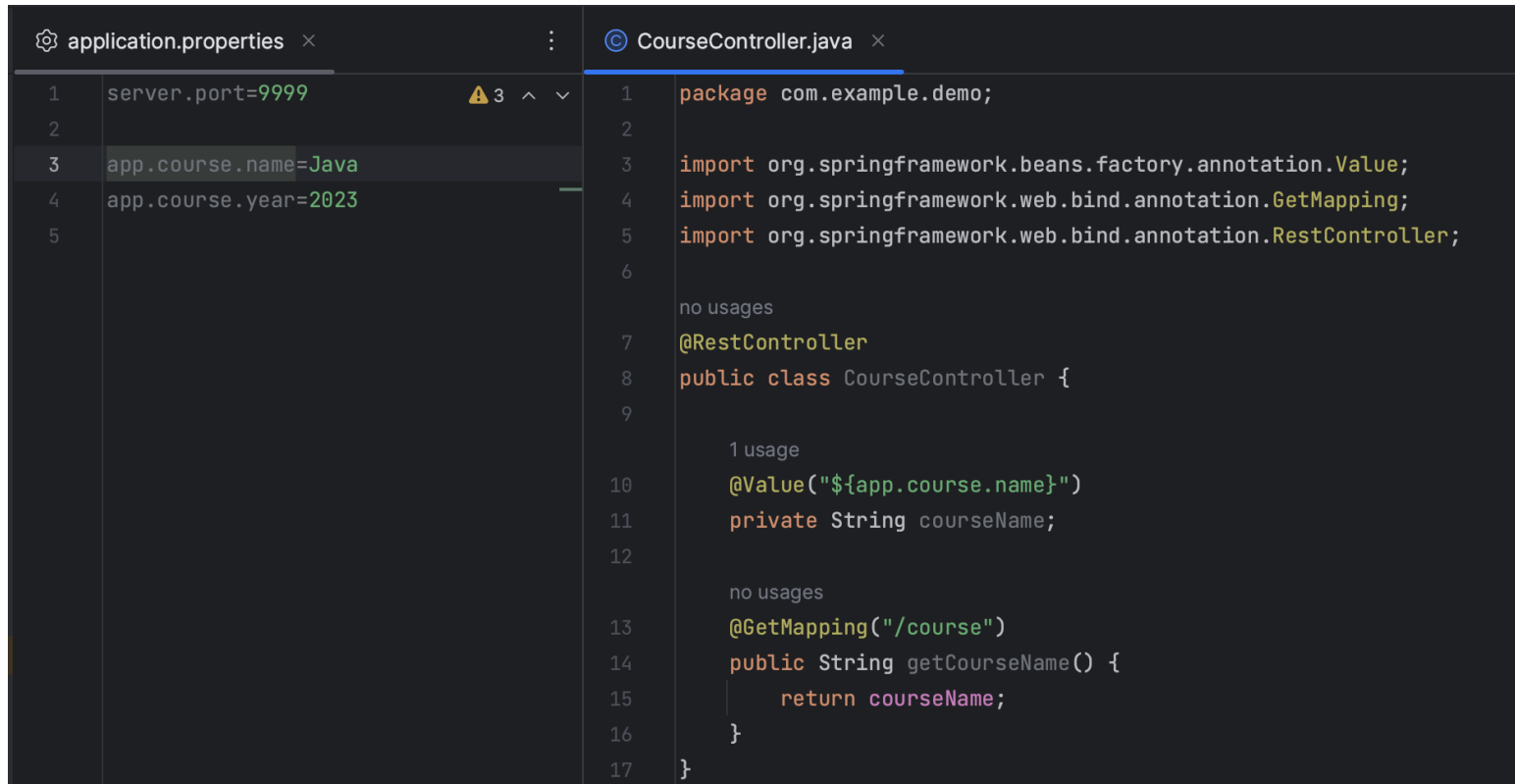
    public boolean compare() {
        return service1 == service2; // вернет false
    }
}
```

Файл с конфигурационными параметрами

- По умолчанию параметры записываются в файл `src/main/resources/application.properties`
- Вместо `application.properties` можно использовать файл `application.yml`



Добавление собственных конфигурационных параметров

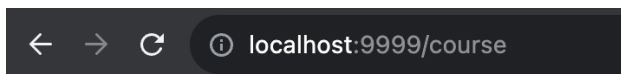


The screenshot shows an IDE with two tabs. The left tab, 'application.properties', contains the following configuration:

```
1 server.port=9999
2
3 app.course.name=Java
4 app.course.year=2023
5
```

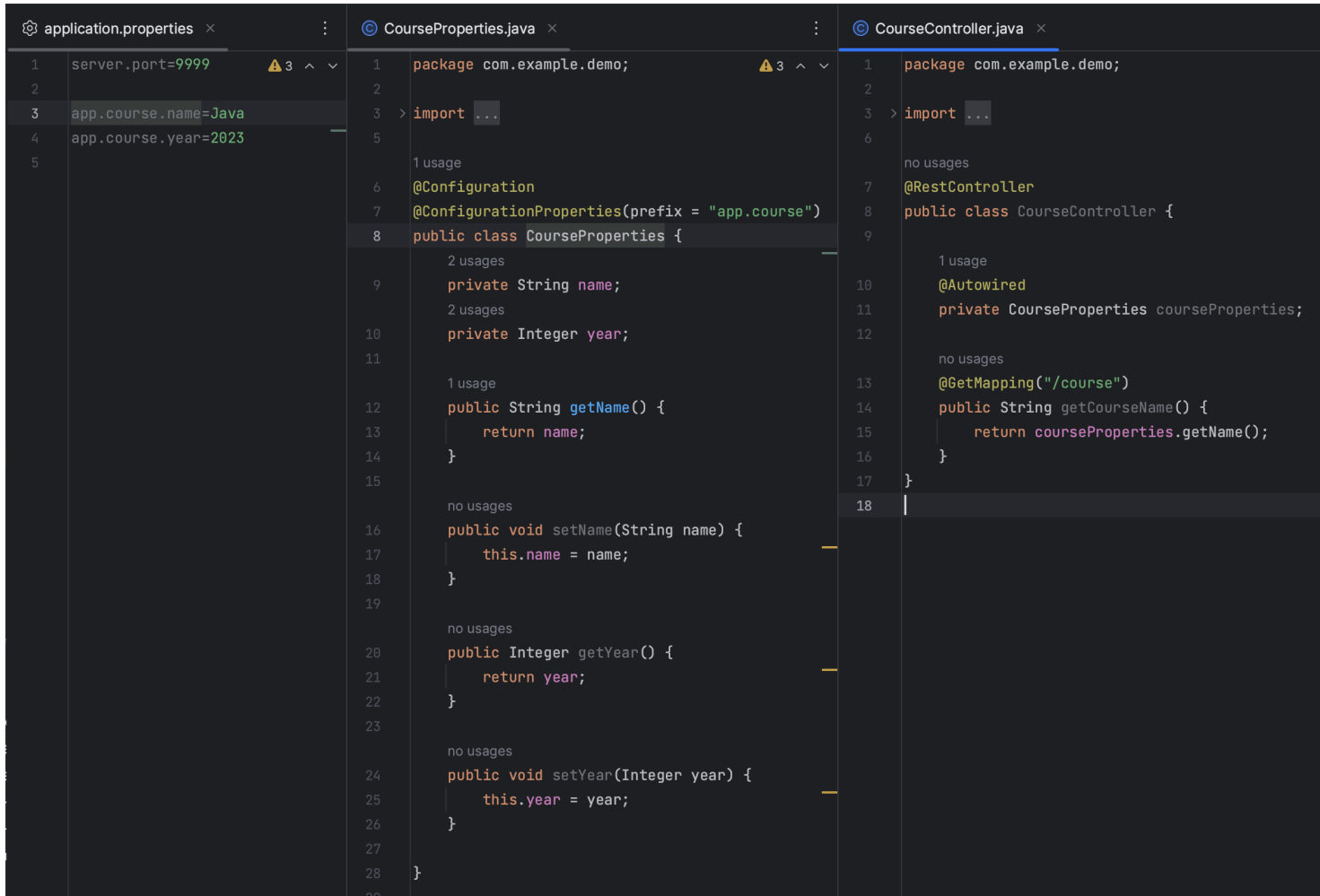
The right tab, 'CourseController.java', contains the following Java code:

```
1 package com.example.demo;
2
3 import org.springframework.beans.factory.annotation.Value;
4 import org.springframework.web.bind.annotation.GetMapping;
5 import org.springframework.web.bind.annotation.RestController;
6
7 no usages
8 @RestController
9 public class CourseController {
10
11     1 usage
12     @Value("${app.course.name}")
13     private String courseName;
14
15     no usages
16     @GetMapping("/course")
17     public String getCourseName() {
18         return courseName;
19     }
20 }
```



Java

Создание класса для хранения конфигурационных параметров



The screenshot displays an IDE with three open files. The first file, `application.properties`, contains configuration properties for a server and a course. The second file, `CourseProperties.java`, is a Java class that uses Spring's `@Configuration` and `@ConfigurationProperties` annotations to manage these properties. The third file, `CourseController.java`, is a REST controller that uses `@RestController` and `@GetMapping` annotations to provide an API endpoint for the course name.

```
application.properties
1 server.port=9999
2
3 app.course.name=Java
4 app.course.year=2023
5

CourseProperties.java
1 package com.example.demo;
2
3 import org.springframework.boot.context.properties.ConfigurationProperties;
4
5
6 @Configuration
7 @ConfigurationProperties(prefix = "app.course")
8 public class CourseProperties {
9     private String name;
10    private Integer year;
11
12    public String getName() {
13        return name;
14    }
15
16    public void setName(String name) {
17        this.name = name;
18    }
19
20    public Integer getYear() {
21        return year;
22    }
23
24    public void setYear(Integer year) {
25        this.year = year;
26    }
27
28 }
29

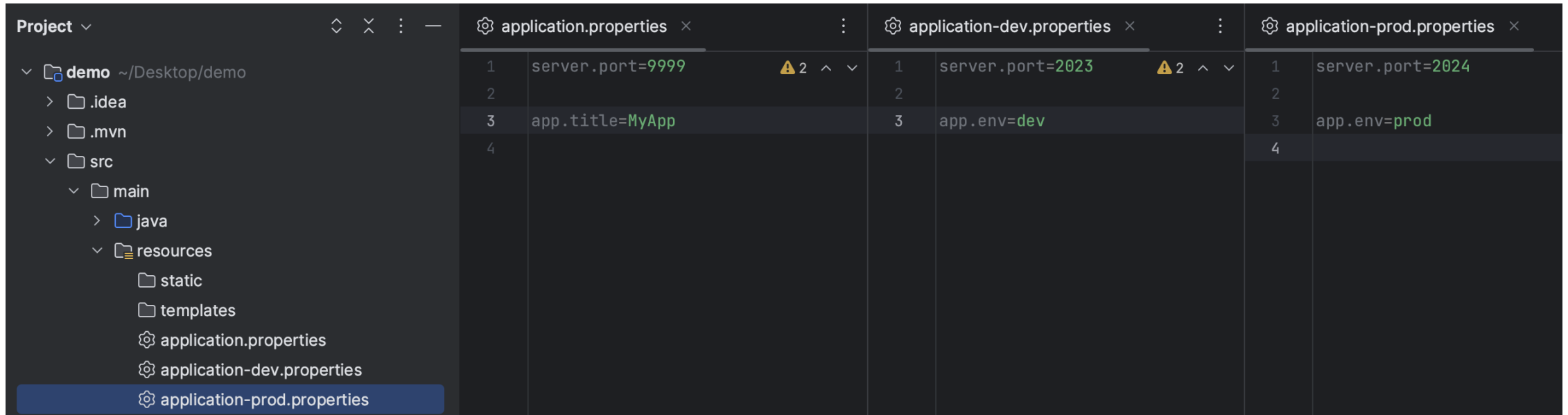
CourseController.java
1 package com.example.demo;
2
3 import org.springframework.web.bind.annotation.GetMapping;
4
5
6 @RestController
7 public class CourseController {
8
9     private CourseProperties courseProperties;
10
11    @GetMapping("/course")
12    public String getCourseName() {
13        return courseProperties.getName();
14    }
15
16 }
17
18
```


Профили Spring

- Позволяют иметь несколько различных конфигураций в рамках одного приложения
- Пример: конфигурации для различных этапов разработки ПО:
 - dev (среда разработчика)
 - test (среда тестировщика)
 - uat (среда для приемо-сдаточных испытаний)
 - prod (среда эксплуатации)

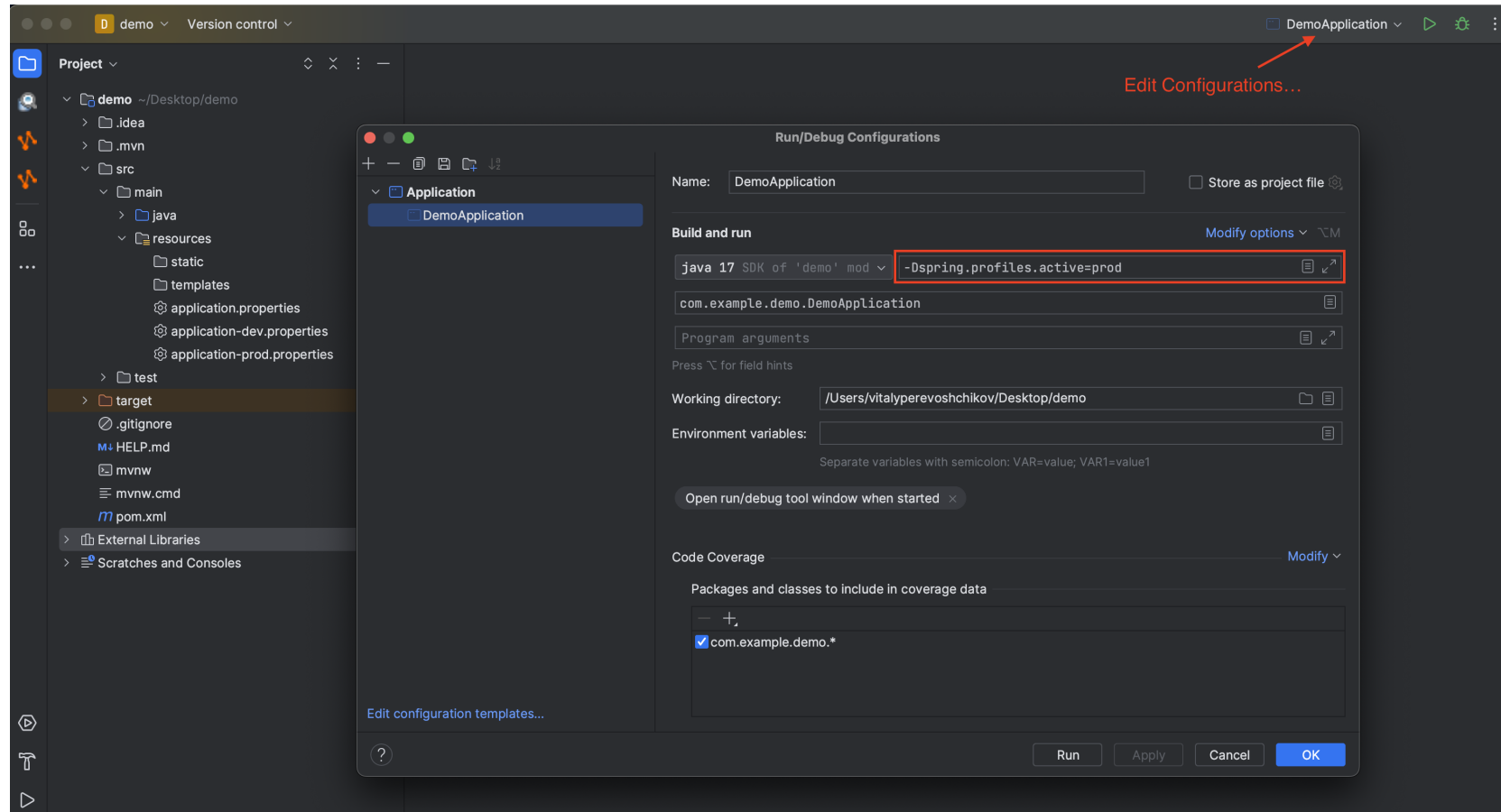
Конфигурационные параметры для профиля

Записываются в файл `src/main/resources/application-{название профиля}.properties`



Указание активного профиля

- как параметр JVM



Аннотация @Profile

DevController.java	ProdController.java
<pre>1 package com.example.demo; 2 3 import org.springframework.beans.factory.annotation.Value; 4 import org.springframework.context.annotation.Profile; 5 import org.springframework.web.bind.annotation.GetMapping; 6 import org.springframework.web.bind.annotation.RestController; 7 8 @RestController 9 @Profile("dev") 10 public class DevController { 11 12 1 usage 13 @Value("\${spring.profiles.active}") 14 private String activeProfile; 15 16 no usages 17 @GetMapping("/dev") 18 public String getActiveProfile() { 19 return activeProfile; 20 } 21 }</pre>	<pre>1 package com.example.demo; 2 3 > import ... 4 5 no usages 6 7 @RestController 8 @Profile("prod") 9 10 public class ProdController { 11 12 1 usage 13 @Value("\${spring.profiles.active}") 14 private String activeProfile; 15 16 no usages 17 @GetMapping("/prod") 18 public String getActiveProfile() { 19 return activeProfile; 20 } 21 }</pre>