Spring Boot

Rapid Application Development

Spring Boot

- Spring Boot is an open-source Java-based framework developed by Pivotal (now part of VMware) that simplifies the process of building and deploying production-ready applications.
- It is built on top of the Spring Framework that provides the RAD (Rapid Application Development) feature to the Spring Framework.
- Spring Boot is widely used in the Java development community due to its simplicity, productivity, and the ability to quickly create robust and scalable applications

Features

- **Convention over Configuration:**
  Allowing developers to get started quickly with minimal setup.
- **Embedded Web Server:**
  Run your application as a stand-alone JAR file.
- **Auto-Configuration:**
  Intelligently guesses and configures common application settings based on the dependencies in the classpath. No requirement for XML configuration.
- **Standalone:**
  Do not require a complex deployment process, making it easy to package and distribute applications as JAR files.
- **Microservices Support:**
  Provides features such as embedded service discovery (using Eureka), centralized configuration management (using Spring Cloud Config), and more.

Features (cont…)

- **Dependency Management:**

  Starter templates include commonly used dependencies for various tasks, such as web development, data access, and messaging.
- **Spring Boot Starters:**

  Set of convenient dependency descriptors to simplify our Maven configuration.

  for web application, you can include the spring-boot-starter-web starter, which will bring in all the necessary dependencies for web development.
- **Spring Boot Actuator:**

  Production-ready features for monitoring and managing applications.

First Spring Boot App

Using Spring Initializer

start.spring.io

spring initializr

Project

- ○ Gradle - Groovy
- ○ Gradle - Kotlin
- ● Maven

Language

- ● Java
- ○ Kotlin
- ○ Groovy

Spring Boot

- ○ 3.3.0 (SNAPSHOT)
- ○ 3.3.0 (M1)
- ○ 3.2.3 (SNAPSHOT)
- ● 3.2.2
- ○ 3.1.9 (SNAPSHOT)
- ○ 3.1.8

Project Metadata

Group com.rit

Artifact SBDemo1

Name SBDemo1

Dep

No d

Dependencies ADD DEPENDENCIES... CTRL + B

Spring Web WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Name SBDemo1

Description Demo project for Spring Boot

Package name com.rit

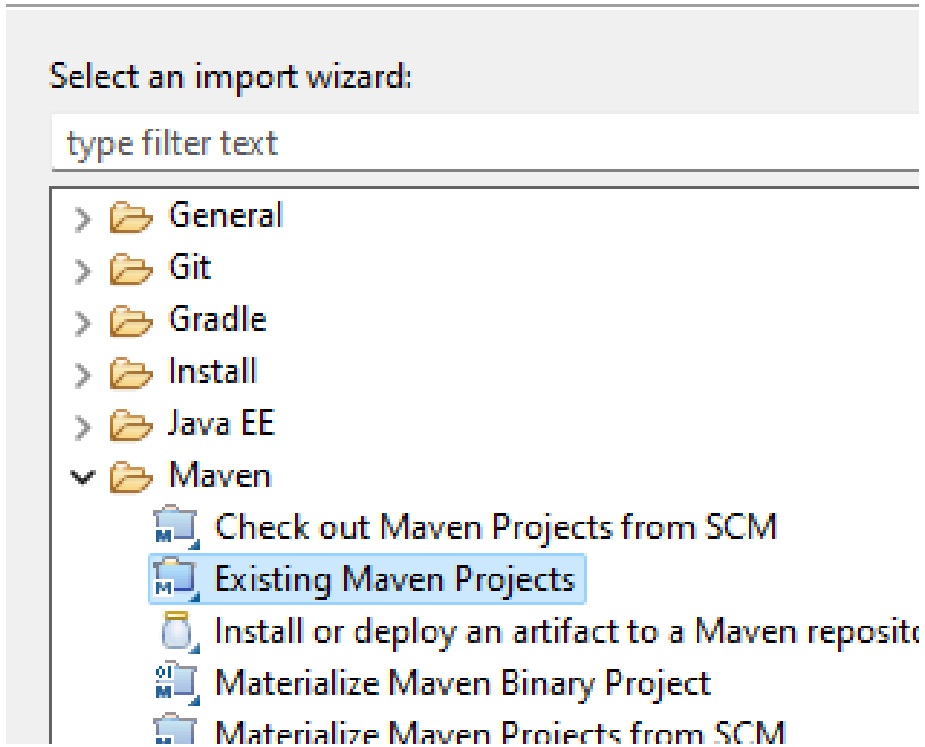
Packaging ● Jar ○ War

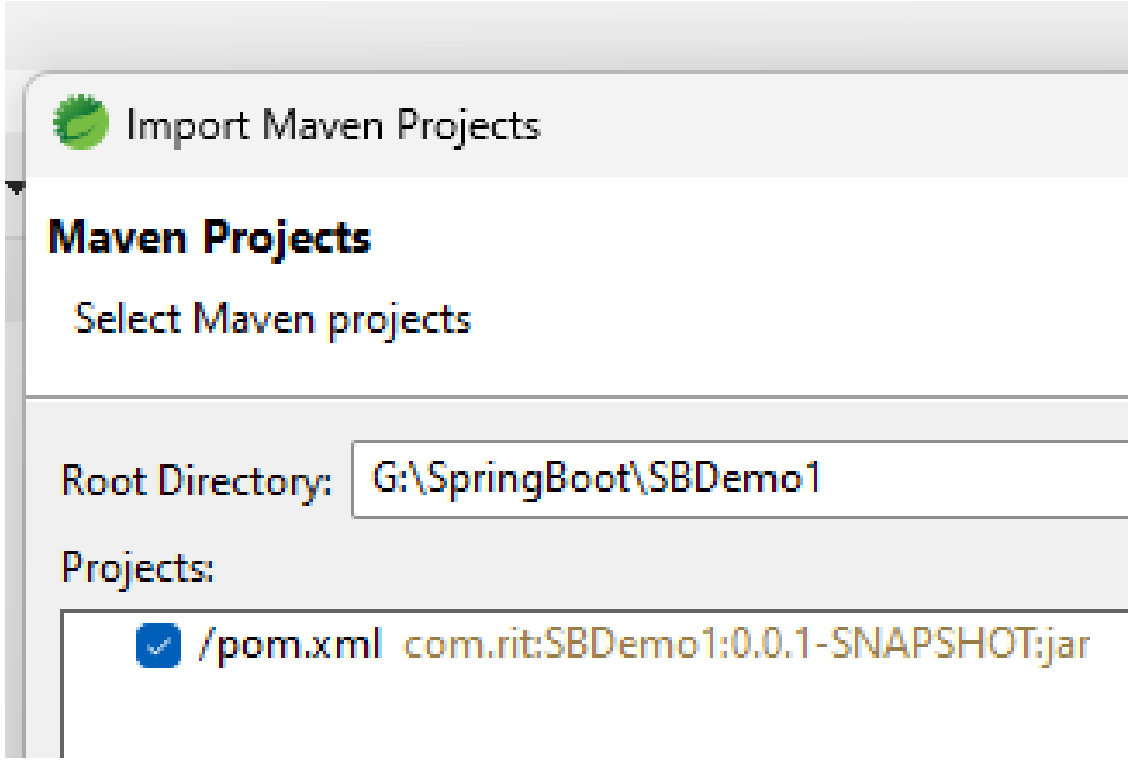
Java ○ 21 ● 17

GENERATE CTRL + ⏎ EX

Project Development: step by step

- Download and Extract the project from spring initializer
- Import -> Existing Maven Project in STS





Package Ex... | Project Exp...

- SBDemo1 [boot] [devtools]
  - src/main/java
    - com.rit
      - SbDemo1Application.java
  - src/main/resources
  - src/test/java
  - JRE System Library [JavaSE-16]
  - Maven Dependencies
  - src
  - target
  - HELP.md
  - mvnw
  - mvnw.cmd
  - pom.xml

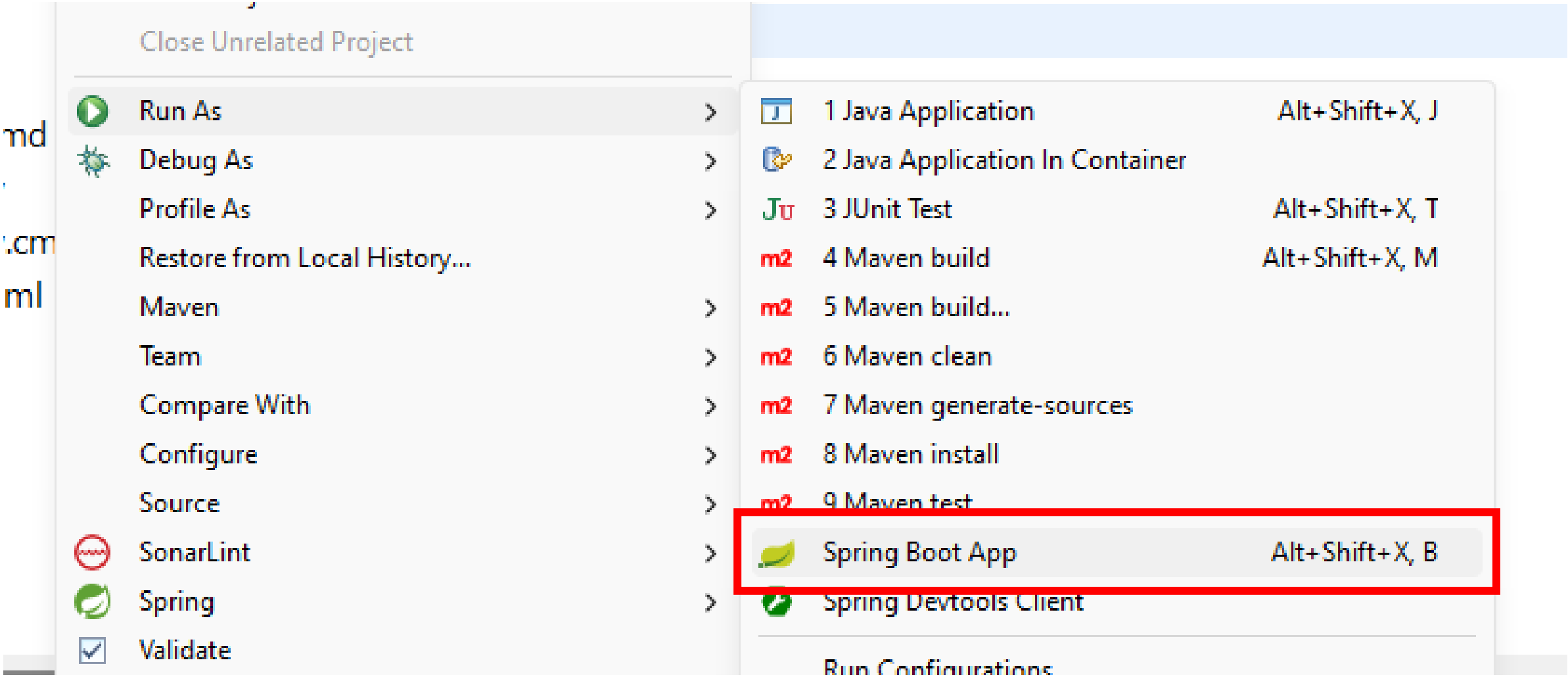
SbDemo1Application.java

```java
package com.rit;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
public class SbDemo1Application {

    public static void main(String[] args) {
        SpringApplication.run(SbDemo1Application.class, args);

        System.out.println("Welcome to Spring Boot");
    }

}
```

@SpringBootApplication

- A single @SpringBootApplication annotation is used to enable the following annotations:

- @Configuration: It allows us to register extra beans in the context or import additional configuration classes.
- @ComponentScan: It scans the package where the application is located.
- @EnableAutoConfiguration: It enables the Spring Boot auto-configuration mechanism.


Close Unrelated Project
Run As
Debug As
Profile As
Restore from Local History...
Maven
Team
Compare With
Configure
Source
SonarLint
Spring
Validate
1 Java Application
Alt+Shift+X, J
2 Java Application In Container
3 JUnit Test
Alt+Shift+X, T
4 Maven build
Alt+Shift+X, M
5 Maven build...
6 Maven clean
7 Maven generate-sources
8 Maven install
9 Maven test
Spring Boot App
Alt+Shift+X, B
Spring Devtools Client
Run Configurations

Problems | Javadoc | Declaration | Console | SonarLint On-The-Fly

SBDemo1 - SbDemo1Application [Spring Boot App] E:\Softwares\sts-4.12.1.RELEASE\plugins\org.eclipse.justj.op

```
2024-01-23T22:06:07.467+05:30  INFO 2724 --- [
main] w.s.c.ServletWebServerApplicationContext :
Root WebApplicationContext: initialization
completed in 880 ms
2024-01-23T22:06:07.849+05:30  INFO 2724 --- [
main] o.s.b.w.embedded.tomcat.TomcatWebServer  :
Tomcat started on port 8080 (http) with context
path ''
2024-01-23T22:06:07.858+05:30  INFO 2724 --- [
main] com.rit.SBDemo1.SbDemo1Application       :
Started SbDemo1Application in 1.795 seconds
(process running for 2.589)
Welcome to Spring Boot
```

First Controller

@RestController

Package Ex... | Project Exp...

- SBDemo1 [boot] [devtools]
  - src/main/java
    - com.rit
    - com.rit.p1intro
      - P1Controller.java
  - src/main/resources
  - src/test/java
  - JRE System Library [JavaSE-16]
  - Maven Dependencies
  - src
  - target
  - HELP.md
  - mvnw
  - mvnw.cmd
  - pom.xml

P1Controller.java

```java
1  package com.rit.p1intro;
2
3  import org.springframework.web.bind.annotation.GetMapping;
7
8  @RestController
9  @RequestMapping("p1")
10 public class P1Controller {
11
12     @RequestMapping(value = "/m1", method = RequestMethod.GET)
13     public String method1() {
14         return "Welcome to Method 1";
15     }
16
17     @GetMapping("/m2")
18     public String method2() {
19         return "Welcome to Method 2";
20     }
```

GET http://localhost:8080/p1/m1 Send

Body 200 OK 114 ms 183 B Save Response

Pretty Text

```
1  Welcome to Method 1
```

GET http://localhost:8080/p1/m2 Send

Body 200 OK 6 ms 183 B Save Response

Pretty Text

```
1  Welcome to Method 2
```

DevTools Dependency

- By adding spring boot dev tools dependency in the POM
- Avoid restarting the application every time after making changes.


```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
    <optional>true</optional>
</dependency>
```

Mappings

Package Ex... | Project Exp...

- SBDemo1 [boot] [devtools]
  - src/main/java
    - com.rit
    - com.rit.p1intro
    - com.rit.p2mapping
      - P2Controller.java
  - src/main/resources
  - src/test/java
  - JRE System Library [JavaSE-16]
  - Maven Dependencies
  - src
  - target
  - HELP.md
  - mvnw
  - mvnw.cmd
  - pom.xml

P2Controller.java

```java
3  import org.springframework.web.bind.annotation.DeleteMap
9
10 @RestController
11 @RequestMapping("p2")
12 public class P2Controller {
13
14     @GetMapping("/m1")
15     public String method1() { return "Get Method"; }
16
17     @PostMapping("/m2")
18     public String method2() { return "Post Method"; }
19
20     @PutMapping("/m3")
21     public String method3() { return "Put Method"; }
22
23     @DeleteMapping("/m4")
24     public String method4() { return "Delete Method"; }
25 }
```

GET http://localhost:8080/p2/m1 Send

Body 200 OK 99 ms 174 B Save Response

Pretty Text

```
1  Get Method
```

POST http://localhost:8080/p2/m2 Send

Body 200 OK 9 ms 175 B Save Response

Pretty Text

```
1  Post Method
```

PUT http://localhost:8080/p2/m3 Send

Body 200 OK 8 ms 174 B Save Response

Pretty Text

```
1  Put Method
```

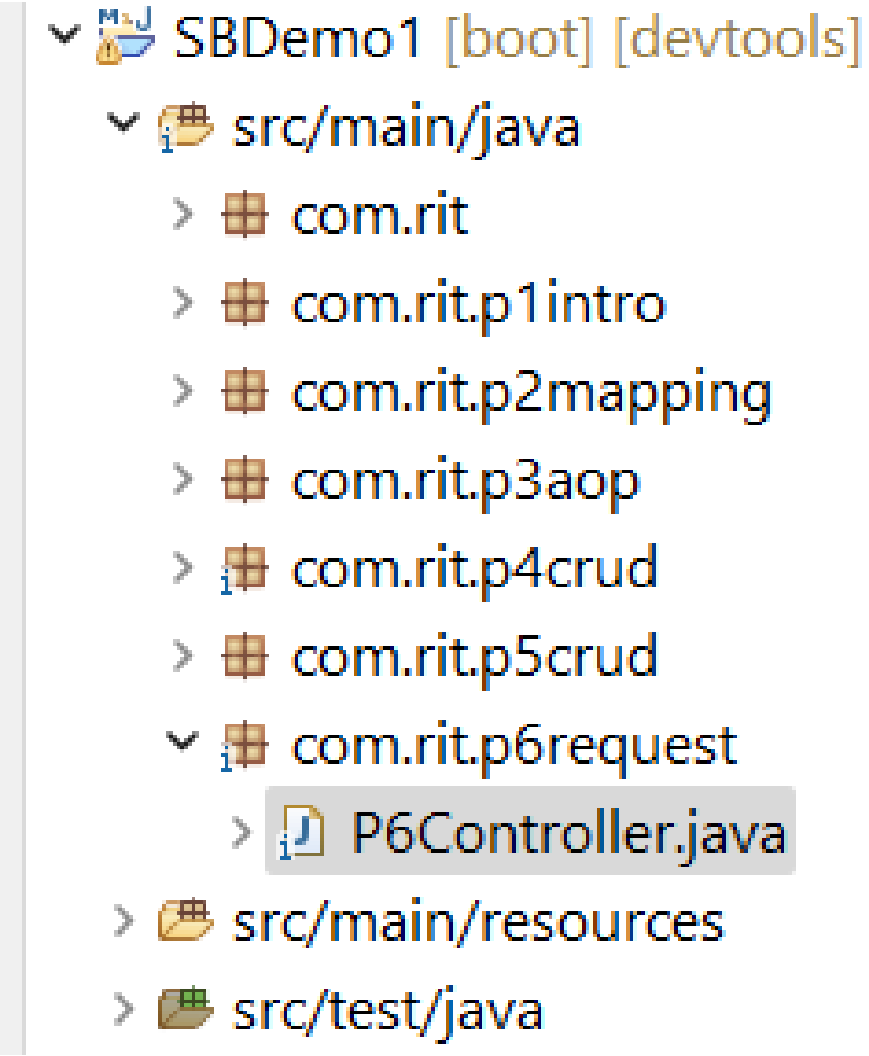
DELETE http://localhost:8080/p2/m4 Send

Body 200 OK 8 ms 177 B Save Response

Pretty Text

```
1  Delete Method
```

Request Methods



```java
//@RestController
@Controller
@ResponseBody
@RequestMapping("p6")
public class P6Controller {

    @GetMapping("/m1/{roll}/{name}/{mark}")
    public String method1(@PathVariable("roll") int a,
            @PathVariable String name, @PathVariable double mark) {
        return "Roll: "+a+" Name : "+name+" Mark : "+mark;
    }

    @GetMapping("/m2")
    public String method2(@RequestParam("roll") int a,
            @RequestParam String name, @RequestParam double mark) {
        return "Roll: "+a+" Name : "+name+" Mark : "+mark;
    }

    @GetMapping("/m3/{mark}")
    public String method3(@RequestHeader("roll") int a,
            @RequestHeader String name, @PathVariable double mark) {
        return "Roll: "+a+" Name : "+name+" Mark : "+mark;
    }
}
```

@PathVariable

http://localhost:8080/p6/m1/101/ram/45.00

Raw Preview Visualize Text

Roll: 101 Name : ram Mark : 45.0

```
@GetMapping("/m1/{roll}/{name}/{mark}")
public String method1(@PathVariable("roll") int a,
        @PathVariable String name, @PathVariable double mark) {
    return "Roll: "+a+" Name : "+name+" Mark : "+mark;
}
```

@RequestParam

http://localhost:8080/p6/m2?roll=101&name=Raja&mark=87.55

Pretty Raw Preview Visualize Text

```
1   Roll: 101 Name : Raja Mark : 87.55
```

```
@GetMapping("/m2")
public String method2(@RequestParam("roll") int a,
        @RequestParam String name, @RequestParam double mark) {
    return "Roll: "+a+" Name : "+name+" Mark : "+mark;
}
```

RequestHeader

GET http://localhost:8080/p6/m3/44.55

Params Auth Headers (8) Body Pre-req. Tests

☑	roll	102
☑	name	Anand

Body 200 OK 18 ms 2

Pretty Raw Preview Visualize Text

```
1   Roll: 102 Name : Anand Mark : 44.55
```

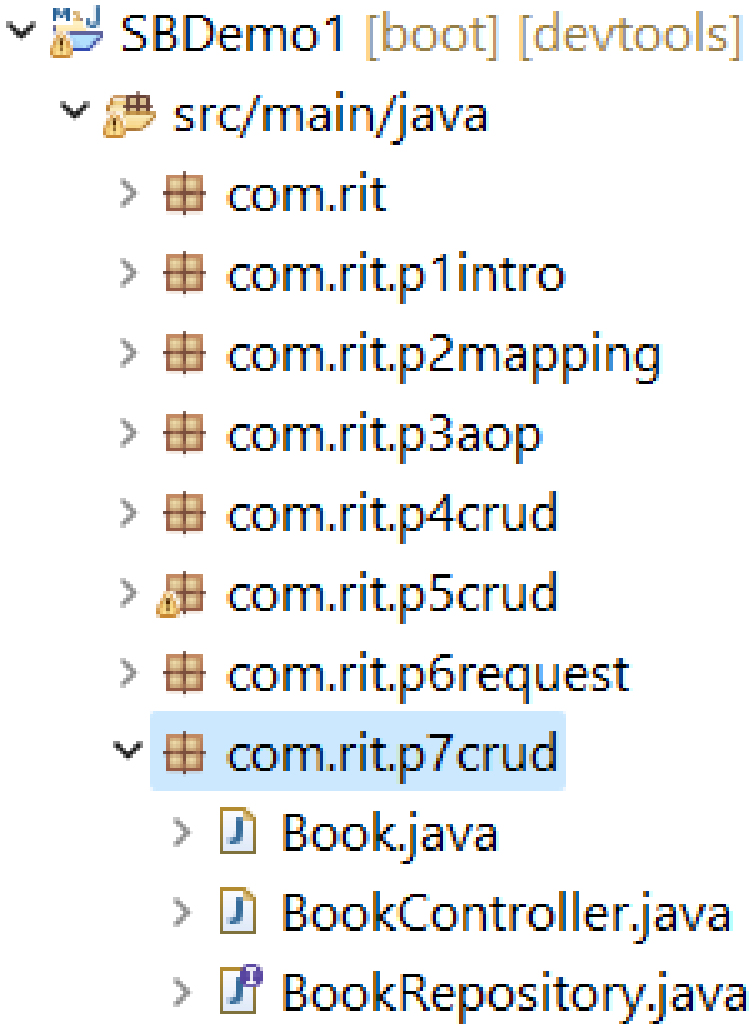
```java
@GetMapping("/m3/{mark}")
public String method3(@RequestHeader("roll") int a,
        @RequestHeader String name, @PathVariable double mark) {
    return "Roll: "+a+" Name : "+name+" Mark : "+mark;
}
```

@RestController

@RestController can be considered as a combination of @Controller and @ResponseBody annotations.

```
//@RestController
@Controller
@ResponseBody
@RequestMapping("p6")
public class P6Controller {
```

Using Stream



Book.java

```java
@Entity
public class Book {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long bookid;
    private String name;
    private String author;
    private String category;
    private String isbn;
    private Double price;

    //Constructor, Getter, Setter & toString

```

BookController.java

```java
@RestController
@RequestMapping("book")
public class BookController {

    @Autowired
    private BookRepository repo;

    public List<Book> getBooks(){
        return (List<Book>) repo.findAll();
    }

    @GetMapping()
    public List<Book> viewBooks() {
        return getBooks();
    }
```

*BookController.java

```
29
30      @PostMapping
31      public Book addBook(@RequestBody Book book) {
32          return repo.save(book);
33      }
34
35      @GetMapping("/author/{author}")
36      public List<Book> viewBookByAuthor(@PathVariable String author) {
37          return getBooks()
38                  .stream()
39                  .filter(book -> book.getAuthor().equals(author))
40                  .collect(Collectors.toList());
41      }
```

BookController.java

```java
    @GetMapping("/category/{category}")
    public List<Book> viewBookByCategory(@PathVariable String category) {
        return getBooks().stream()
                .filter(book -> book.getCategory().equals(category))
                .collect(Collectors.toList());
    }

    @GetMapping("/isbn/{isbn}")
    public Book viewBookByIsbn(@PathVariable String isbn) {
        return getBooks().stream()
                .filter(book -> book.getIsbn().equals(isbn))
                .findAny()
                .orElse(new Book());
    }
```

BookController.java

```
                .findAny()
                .orElse(new Book());
    }

    @GetMapping("/price/{price}")
    public List<Book> viewBookByPrice(@PathVariable double price) {
        return getBooks().stream()
                .filter(book -> book.getPrice() >= price)
                .toList();
    }

}
```

Thank you