# Coding Cheat Sheet: Deepdive into Spring Boot

This reading provides a reference list of code you'll encounter as you learn and use the Spring framework in Java. Understanding these concepts will help you write and debug Java programs that utilize the Spring framework. Let's explore the following Java coding concepts:

- Creating a Spring Boot application
- Managing a Spring Boot application
- REST API versioning
- Sending and receiving data using REST API parameters

Keep this summary reading available as a reference as you progress through your course, and refer to this reading as you begin coding with Java after this course!

# Creating a Spring Boot application

Spring Boot simplifies Java application development by streamlining configuration and setup.

| Description | Example |
| --- | --- |
| **Creating a simple controller:**<br>A controller handles web requests in Spring Boot. The `BookController` class defines an endpoint that returns book information. | `package com.example.bookfinder.controller;`<br><br>`import org.springframework.web.bind.annotation.GetMapping;`<br>`import org.springframework.web.bind.annotation.RestController;`<br><br>`@RestController`<br>`public class BookController {`<br><br>`  @GetMapping("/book")`<br>`  public String getBook() {`<br>`  return "Effective Java by Joshua Bloch";`<br>`  }`<br>`}` |
| **Using getters and setters:**<br>In Java, getters and setters are used to access the private variables of a class. Let's create a simple `Book` class with some properties. | `package com.example.bookfinder.model;`<br><br>`public class Book {`<br><br>`  private String title;`<br>`  private String author;`<br><br>`  public Book(String title, String author) {`<br>`  this.title = title;`<br>`  this.author = author;`<br>`  }`<br><br>`  public String getTitle() {`<br>`  return title;`<br>`  }`<br><br>`  public void setTitle(String title) {`<br>`  this.title = title;`<br>`  }`<br><br>`  public String getAuthor() {`<br>`  return author;`<br>`  }`<br><br>`  public void setAuthor(String author) {`<br>`  this.author = author;`<br>`  }`<br>`}` |

| Description | Example |
| --- | --- |
| **Logging in Spring Boot:** Logging helps track application behavior. The `Logger` is used to log messages in a Spring Boot application. Here, `LoggerFactory` creates a logger for `BookController`. | `package com.example.bookfinder.controller;`<br><br>`import org.slf4j.Logger;`<br>`import org.slf4j.LoggerFactory;`<br>`import org.springframework.web.bind.annotation.GetMapping;`<br>`import org.springframework.web.bind.annotation.RestController;`<br><br>`@RestController`<br>`public class BookController {`<br><br>`private static final Logger logger = LoggerFactory.getLogger(BookController.class);`<br><br>`@GetMapping("/book")`<br>`public String getBook() {`<br>`logger.info("getBook() method called");`<br>`return "Effective Java by Joshua Bloch";`<br>`}`<br>`}` |

# Managing a Spring Boot application

| Description | Example |
| --- | --- |
| **Using Maven:** Ensure your `pom.xml` has the following configuration under the `<build>` tag. | `<build>`<br>`<plugins>`<br>`<plugin>`<br>`<groupId>org.springframework.boot</groupId>`<br>`<artifactId>spring-boot-maven-plugin</artifactId>`<br>`</plugin>`<br>`</plugins>`<br>`</build>` |
| **Running Maven package command:** This will create a JAR file in the `target` directory. | `mvn clean package` |
| **Versioning in JAR** Specify the version in `pom.xml` to track changes and updates. | `<version>1.0.0-SNAPSHOT</version>` |
| **Adding dependencies in Maven:** Add dependencies inside the `<dependencies>` section of `pom.xml`. | `<dependencies>`<br>`<dependency>`<br>`<groupId>org.springframework.boot</groupId>`<br>`<artifactId>spring-boot-starter-web</artifactId>`<br>`</dependency>`<br>`</dependencies>` |

| Description | Example |
| --- | --- |
| | |
| **Exception handling - Custom exception:** Define a custom exception class for better error handling. | `public class CustomException extends RuntimeException {`<br>`  public CustomException(String message) {`<br>`  super(message);`<br>`  }`<br>`}` |
| **Exception handling - Using `@ExceptionHandler`:** Handle exceptions at the controller level. | `@RestController`<br>`public class MyController {`<br>`  @GetMapping("/example")`<br>`  public String example() {`<br>`  throw new CustomException("This is a custom exception message.");`<br>`  }`<br><br>`  @ExceptionHandler(CustomException.class)`<br>`  public ResponseEntity<String> handleCustomException(CustomException ex) {`<br>`  return new ResponseEntity<>(ex.getMessage(), HttpStatus.BAD_REQUEST);`<br>`  }`<br>`}` |
| **Exception handling - Using `@ControllerAdvice`:** Handle exceptions globally across all controllers. | `@ControllerAdvice`<br>`public class GlobalExceptionHandler {`<br>`  @ExceptionHandler(CustomException.class)`<br>`  public ResponseEntity<String> handleCustomException(CustomException ex) {`<br>`  return new ResponseEntity<>(ex.getMessage(), HttpStatus.BAD_REQUEST);`<br>`  }`<br><br>`  @ExceptionHandler(Exception.class)`<br>`  public ResponseEntity<String> handleGeneralException(Exception ex) {`<br>`  return new ResponseEntity<>("An error occurred: " + ex.getMessage(), HttpStatus.INTERNAL_SERVER_ERROR);`<br>`  }`<br>`}` |
| **Application properties in YAML:** Convert application properties to YAML format. | `server:`<br>`  port: 9090` |

# REST API versioning

| Description | Example |
| --- | --- |
| **Defining a simple REST controller (Version 1):**<br>Create a Spring Boot REST controller to handle API requests for version 1. | `@RestController`<br>`public class GreetingControllerV1 {`<br>`  @GetMapping("/api/v1/greeting")`<br>`  public String getGreetingV1() {`<br>`  return "Hello, World! - v1";`<br>`  }`<br>`}` |
| **Defining a simple REST controller (Version 2):**<br>Create another controller to handle version 2 of the API. | `@RestController`<br>`public class GreetingControllerV2 {`<br>`  @GetMapping("/api/v2/greeting")`<br>`  public String getGreetingV2() {`<br>`  return "Hello, Universe! - v2";`<br>`  }`<br>`}` |
| **Running the Spring Boot application:**<br>Start the application using Maven. | `mvn spring-boot:run` |
| **Accessing API endpoints:**<br>Use a browser or a tool like `curl` to test the API. | http://localhost:8080/api/v1/greeting<br>http://localhost:8080/api/v2/greeting |

# Sending and recieving data using REST API parameters

| Description | Example |
| --- | --- |
| **Sending data with a GET request:**<br>A GET request retrieves data and can include parameters in the URL. | `@RestController`<br>`@RequestMapping("/api")`<br>`public class DataController {`<br>`  @GetMapping("/data")`<br>`  public String getData(@RequestParam(value = "name", defaultValue = "World") String name) {`<br>`  return "Hello, " + name + "!";`<br>`  }`<br>`}` |

| Description | Example |
| --- | --- |
| | |
| **Sending data with a POST request:**<br>A POST request is used to create new resources. | `@RestController`<br>`@RequestMapping("/api")`<br>`public class UserController {`<br>`  @PostMapping("/users")`<br>`  public String createUser(@RequestBody User user) {`<br>`  return "User created with name: " + user.getName();`<br>`  }`<br>`}` |
| **Sending data with a PUT request:**<br>A PUT request updates an existing resource. | `@RestController`<br>`@RequestMapping("/api")`<br>`public class ProductController {`<br>`  @PutMapping("/products/{id}")`<br>`  public String updateProduct(@PathVariable Long id, @RequestBody Product product) {`<br>`  return "Product with ID " + id + " updated to name: " + product.getName();`<br>`  }`<br>`}` |
| **Sending data with a DELETE request:**<br>A DELETE request removes a resource by its ID. | `@RestController`<br>`@RequestMapping("/api")`<br>`public class OrderController {`<br>`  @DeleteMapping("/orders/{id}")`<br>`  public String deleteOrder(@PathVariable Long id) {`<br>`  return "Order with ID " + id + " has been deleted.";`<br>`  }`<br>`}` |
| **Cumulative example - Managing books:**<br>A simple API to manage a collection of books. | `@RestController`<br>`@RequestMapping("/api")`<br>`public class BookController {`<br>`  private Map<Long, Book> bookRepository = new HashMap<>();`<br>`  @GetMapping("/books/{id}")`<br>`  public Book getBook(@PathVariable Long id) {`<br>`  return bookRepository.get(id);`<br>`  }`<br>`  @PostMapping("/books")`<br>`  public String addBook(@RequestBody Book book) {`<br>`  bookRepository.put(book.getId(), book);`<br>`  return "Book added with ID: " + book.getId();`<br>`  }`<br>`  @PutMapping("/books/{id}")`<br>`  public String updateBook(@PathVariable Long id, @RequestBody Book book) {`<br>`  if (!bookRepository.containsKey(id)) {`<br>`  return "Book not found!";`<br>`  }`<br>`  bookRepository.put(id, book);`<br>`  return "Book updated with ID: " + id;`<br>`  }`<br>`  @DeleteMapping("/books/{id}")`<br>`  public String deleteBook(@PathVariable Long id) {` |

| Description | Example |
| --- | --- |
| | ```            bookRepository.remove(id);<br>            return "Book deleted with ID: " + id;<br>        }<br>    }``` |
| **Testing the API:**<br>Sample curl commands for sending data. | ```# GET request with query parameters<br>curl -X GET "http://localhost:8080/api/products?category=electronics&sort=price"<br><br># POST request with JSON body<br>curl -X POST "http://localhost:8080/api/users" \<br>     -H "Content-Type: application/json" \<br>     -d '{"name": "John Doe", "email": "john.doe@example.com"}'``` |

# Author(s)

Ramanujam Srinivasan
Lavanya Thiruvali Sunderarajan