Spring Boot Dependency Injection Case Study: Online Bookstore Inventory Management

Objective:

Build a simplified backend service for managing books in an online bookstore, applying Spring Boot's Dependency Injection (DI) features.

Scenario Overview:

Create a microservice that manages book inventory: adding, retrieving, and searching books.

Core Requirements:

1. Add a new book to inventory
2. Get all books
3. Search for a book by ID

Data Model:

```
public class Book {
    private Long id;
    private String title;
    private String author;
    private Double price;
}
```

Technical Requirements:

- Use Spring stereotypes: @Component, @Service, @Repository
- Use constructor-based dependency injection with @Autowired
- Organize code into layers (Controller, Service, Repository)

Task Breakdown:

1. Book Model Class:

```
public class Book {
    private Long id;
```

```
    private String title;
    private String author;
    private Double price;
    // Constructors, Getters, Setters
}
```

2. BookRepository (Simulated In-Memory Storage):

```
@Repository
public class BookRepository {
    private Map<Long, Book> bookMap = new HashMap<>();
    public void save(Book book) { bookMap.put(book.getId(), book); }
    public Collection<Book> findAll() { return bookMap.values(); }
    public Optional<Book> findById(Long id) { return Optional.ofNullable(bookMap.get(id)); }
}
```

3. BookService (Injected with Repository):

```
@Service
public class BookService {
    private final BookRepository bookRepository;
    @Autowired
    public BookService(BookRepository bookRepository) {
        this.bookRepository = bookRepository;
    }
    public void addBook(Book book) { bookRepository.save(book); }
    public Collection<Book> getAllBooks() { return bookRepository.findAll(); }
    public Book getBookById(Long id) {
        return bookRepository.findById(id).orElseThrow(() -> new RuntimeException("Book not found"));
    }
}
```

4. BookController (Injected with Service):

```
@RestController
@RequestMapping("/api/books")
```

```
public class BookController {
    private final BookService bookService;
    @Autowired
    public BookController(BookService bookService) {
        this.bookService = bookService;
    }
    @PostMapping
    public ResponseEntity<String> addBook(@RequestBody Book book) {
        bookService.addBook(book);
        return ResponseEntity.ok("Book added");
    }
    @GetMapping
    public Collection<Book> getAllBooks() { return bookService.getAllBooks(); }
    @GetMapping("/{id}")
    public Book getBookById(@PathVariable Long id) {
        return bookService.getBookById(id);
    }
}
```

5. Main Application Class:

```
@SpringBootApplication
public class BookstoreApplication {
    public static void main(String[] args) {
        SpringApplication.run(BookstoreApplication.class, args);
    }
}
```

Learning Goals:

- Apply Dependency Injection in Spring Boot
- Understand constructor vs field injection
- Build modular, testable applications with Spring Boot