Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут ім. Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

ЛАБОРАТОРНА РОБОТА № 2

з дисципліни «Розроблення застосунків з використанням Spring Framework»
на тему «Створення та зв’язування компонентів»

ВИКОНАВ:
студент III курсу ФІОТ
групи ІО-25
Мамченко Д.О.
Залікова №2518
Скоробогатов І.В.
Залікова №2527
Сокольчук О.С.
Залікова №2529

Київ – 2024

Тема: Створення та зв’язування компонентів.

Код програми:

StubBookRepository.java

```
@Repository
public class StubBookRepository implements BookRepository {

 private final List<Book> books = new ArrayList<>();

 @Override
 public List<Book> findAll() {
   return books;
 }

 @Override
 public List<Book> findByTitleContaining(String titlePrefixes) {
   List<String> titlePrefixesList = List.of(titlePrefixes.trim().split("[,.; ]"));
   List<Book> foundBooks = new ArrayList<>();
   if (!titlePrefixesList.isEmpty()) {
     for (Book book : books) {
       boolean matches = titlePrefixesList.stream()
           .anyMatch(
               prefix ->
book.getTitle().toLowerCase().contains(prefix.toLowerCase().trim()));
       if (matches) {
         foundBooks.add(book);
       }
     }
   }
   return foundBooks;
 }

 @Override
 public List<Book> findByAuthorContaining(String authorPrefixes) {
   List<String> authorPrefixesList = List.of(authorPrefixes.trim().split("[,.; ]"));
   List<Book> foundBooks = new ArrayList<>();
   if (!authorPrefixesList.isEmpty()) {
     for (Book book : books) {
       boolean matches = authorPrefixesList.stream()
           .anyMatch(
               prefix ->
book.getAuthor().toLowerCase().contains(prefix.toLowerCase().trim()));
       if (matches) {
         foundBooks.add(book);
       }
     }
   }
   return foundBooks;
```

```
 }

 @Override
 public List<Book> findByKeywordsContaining(String keywords) {
   List<String> keywordsList = List.of(keywords.trim().split("[,.; ]"));
   List<Book> foundBooks = new ArrayList<>();
   if (!keywordsList.isEmpty()) {
     for (Book book : books) {
       boolean matches = keywordsList.stream()
           .anyMatch(prefix -> book.getKeywords().stream()
               .anyMatch(keyword ->
keyword.toLowerCase().contains(prefix.toLowerCase().trim())));
       if (matches) {
         foundBooks.add(book);
       }
     }
   }
   return foundBooks;
 }

 @Override
 public Optional<Book> findById(Long id) {
   return books.stream().filter(book -> book.getId().equals(id)).findFirst();
 }

 @Override
 public Book save(Book book) {
   if (book.getId() == null) {
     book.setId((long) ((Math.random() * (Long.MAX_VALUE -
1_000_000_000_000_000_000L)) +
         1_000_000_000_000_000_000L));
   } else {
     Optional<Book> repoBook = findById(book.getId());
     repoBook.ifPresent(books::remove);
   }
   books.add(book);
   return book;
 }

 @Override
 public void deleteById(Long id) {
   books.removeIf(book -> book.getId().equals(id));
 }
}
```

BookServiceImpl.java

```
@Service
@RequiredArgsConstructor
public class BookServiceImpl implements BookService {

 private final BookRepository bookRepository;
```

```
 public List<Book> findAll() {
   return bookRepository.findAll();
 }

 @Override
 public List<Book> findAllBySearchRequest(SearchRequest searchRequest) {
   Set<Book> books = new HashSet<>(

bookRepository.findByTitleContaining(searchRequest.getSearchRequestPrefixes()));
   if (searchRequest.isSearchByAuthor()) {

books.addAll(bookRepository.findByAuthorContaining(searchRequest.getSearchRequestPre
fixes()));
   }
   if (searchRequest.isSearchByKeywords()) {
     books.addAll(

bookRepository.findByKeywordsContaining(searchRequest.getSearchRequestPrefixes()));
   }
   return books.stream().toList();
 }

 @Override
 public Book findById(Long id) {
   return bookRepository.findById(id).orElse(null);
 }

 @Override
 public Book save(BookDto bookDto) {
   Book book = Book.builder()
       .title(bookDto.getTitle())
       .author(bookDto.getAuthor())
       .year(bookDto.getYear())
       .keywords(List.of(bookDto.getKeywords().split(",")))
       .build();
   return bookRepository.save(book);
 }

 @Override
 public Book save(Long id, BookDto bookDto) {
   Book book = Book.builder()
       .id(id)
       .title(bookDto.getTitle())
       .author(bookDto.getAuthor())
       .year(bookDto.getYear())
       .keywords(List.of(bookDto.getKeywords().split(",")))
       .build();
   return bookRepository.save(book);
 }

 @Override
 public void deleteById(Long id) {
   bookRepository.deleteById(id);
```

```
 }
}
```

Book.java

```
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class Book {

 private Long id;
 private String title;
 private String author;
 private Integer year;
 private List<String> keywords;
}
```

BookDto.java

```
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class BookDto {

 private String title;
 private String author;
 private Integer year;
 private String keywords;
}
```

SearchRequest.java

```
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class SearchRequest {

 private String searchRequestPrefixes;
 private boolean searchByAuthor;
 private boolean searchByKeywords;
}
```

LibraryController.java

```
@Controller
@RequiredArgsConstructor
public class LibraryController {
 private final BookService bookService;
```

```
 @GetMapping
 public String index(Model model) {
   model.addAttribute("books", bookService.findAll());
   model.addAttribute("searchRequest", new SearchRequest());
   return "index";
 }

 @PostMapping("/searchBooks")
 public String searchBooks(@ModelAttribute SearchRequest searchRequest, Model model) {
   List<Book> books = bookService.findAllBySearchRequest(searchRequest);
   model.addAttribute("books", books);
   model.addAttribute("searchRequest", searchRequest);
   return "index";
 }
}
```

AdminController.java

```
@Controller
@RequestMapping("/admin")
@RequiredArgsConstructor
public class AdminController {
 private final BookService bookService;

 @GetMapping
 public String index(Model model) {
   model.addAttribute("books", bookService.findAll());
   model.addAttribute("searchRequest", new SearchRequest());
   model.addAttribute("bookToAdd", new BookDto());
   return "admin";
 }

 @PostMapping("/searchBooks")
 public String searchBooks(@ModelAttribute SearchRequest searchRequest, Model model) {
   List<Book> books = bookService.findAllBySearchRequest(searchRequest);
   model.addAttribute("books", books);
   model.addAttribute("searchRequest", new SearchRequest());
   model.addAttribute("bookToAdd", new BookDto());
   return "admin";
 }

 @PostMapping("/addBook")
 public String addBook(@ModelAttribute BookDto bookDto) {
   bookService.save(bookDto);
   return "redirect:/admin";
 }

 @PostMapping("/deleteBook/{id}")
 public String deleteBook(@PathVariable Long id) {
   bookService.deleteById(id);
   return "redirect:/admin";
 }
```

```
 @GetMapping("/changeBook")
 public String changeBookPage(@RequestParam Long id, Model model) {
   Book book = bookService.findById(id);
   model.addAttribute("bookToChange", BookDto.builder()
        .title(book.getTitle())
        .author(book.getAuthor())
        .year(book.getYear())
        .keywords(String.join(",", book.getKeywords()))
        .build());
   return "book-change";
 }

 @PostMapping("/changeBook/{id}")
 public String changeBook(@PathVariable Long id, @ModelAttribute BookDto bookDto) {
   bookService.save(id, bookDto);
   return "redirect:/admin";
 }
}
```

DataLoader.java

```
@Component
@RequiredArgsConstructor
public class DataLoader implements CommandLineRunner {

 private final BookService bookService;

 @Override
 public void run(String... args) throws Exception {
   List<BookDto> books = List.of(
        new BookDto("To Kill a Mockingbird", "Harper Lee", 1960, "classic,fiction"),
        new BookDto("1984", "George Orwell", 1949, "dystopian,fiction"),
        new BookDto("Pride and Prejudice", "Jane Austen", 1813, "romance,classic"),
        new BookDto("The Great Gatsby", "F. Scott Fitzgerald", 1925, "classic,fiction"),
        new BookDto("Moby Dick", "Herman Melville", 1851, "adventure,classic"),
        new BookDto("War and Peace", "Leo Tolstoy", 1869, "historical,classic"),
        new BookDto("The Catcher in the Rye", "J.D. Salinger", 1951, "classic,fiction"),
        new BookDto("The Hobbit", "J.R.R. Tolkien", 1937, "fantasy,adventure"),
        new BookDto("Crime and Punishment", "Fyodor Dostoevsky", 1866,
"classic,philosophical"),
        new BookDto("The Brothers Karamazov", "Fyodor Dostoevsky", 1880,
"classic,philosophical"),
        new BookDto("Brave New World", "Aldous Huxley", 1932, "dystopian,fiction"),
        new BookDto("The Lord of the Rings", "J.R.R. Tolkien", 1954, "fantasy,adventure"),
        new BookDto("Jane Eyre", "Charlotte Bronte", 1847, "romance,classic"),
        new BookDto("Wuthering Heights", "Emily Bronte", 1847, "romance,classic"),
        new BookDto("The Divine Comedy", "Dante Alighieri", 1320, "classic,poetry")
   );

   books.forEach(bookService::save);
 }
}
```

Опис:

Ця програма реалізує просту бібліотечну систему для збереження, пошуку та управління книгами. Вона складається з кількох компонентів: репозиторію для зберігання книг (стаб), сервісного шару для обробки логіки, контролерів для обробки запитів (головний та для адміністрації), моделей для представлення даних і класу для завантаження початкових даних.

StubBookRepository – це клас-репозиторій, який імітує сховище книг, зберігаючи їх у списку books.

BookServiceImpl – сервіс, який містить бізнес-логіку для роботи з книгами. Пошук книг за запитом, який може містити заголовок, автора або ключові слова. Збереження нових або оновлених книг на основі BookDto. Видалення книги по id.

Book – модель що представляє собою книгу.
BookDto – DTO для книги.

SearchRequest – модель для обробки запитів пошуку книг. Містить поля для введення користувача (префікси для пошуку) та параметри, що вказують, чи здійснювати пошук за автором або ключовими словами.

LibraryController – контролер для обробки запитів від користувачів до бібліотеки.

AdminController – контролер для адміністративних дій, таких як додавання, зміна та видалення книг.

DataLoader – прелоуд клас, який завантажує початкові дані (список книг) у систему під час запуску програми.

Результат виконання роботи:

Library Catalog

Books

Search Clear ☐ Search by Author ☐ Search by Keywords Find

Title	Author	Year	Keywords
To Kill a Mockingbird	Harper Lee	1960	classic fiction
1984	George Orwell	1949	dystopian fiction
Pride and Prejudice	Jane Austen	1813	romance classic
The Great Gatsby	F. Scott Fitzgerald	1925	classic fiction
Moby Dick	Herman Melville	1851	adventure classic
War and Peace	Leo Tolstoy	1869	historical classic
The Catcher in the Rye	J.D. Salinger	1951	classic fiction
The Hobbit	J.R.R. Tolkien	1937	fantasy adventure
Crime and Punishment	Fyodor Dostoevsky	1866	classic philosophical
The Brothers Karamazov	Fyodor Dostoevsky	1880	classic philosophical
Brave New World	Aldous Huxley	1932	dystopian fiction
The Lord of the Rings	J.R.R. Tolkien	1954	fantasy adventure
Jane Eyre	Charlotte Bronte	1847	romance classic
Wuthering Heights	Emily Bronte	1847	romance classic

Library Catalog

Books

great Clear ☐ Search by Author ☐ Search by Keywords Find

Title	Author	Year	Keywords
The Great Gatsby	F. Scott Fitzgerald	1925	classic fiction

Library Catalog

Books

Orwell Clear ☑ Search by Author ☐ Search by Keywords Find

Title	Author	Year	Keywords
1984	George Orwell	1949	dystopian fiction

Library Catalog

Books

fiction adventure | Clear | Search by Author | Search by Keywords | Find

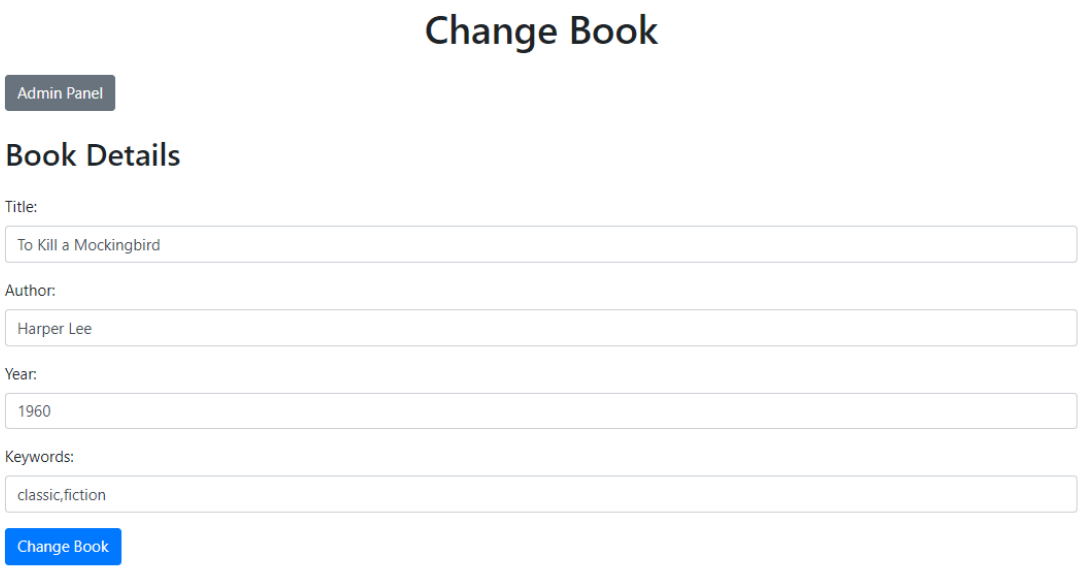
Title	Author	Year	Keywords
The Hobbit	J.R.R. Tolkien	1937	fantasy adventure
Brave New World	Aldous Huxley	1932	dystopian fiction
To Kill a Mockingbird	Harper Lee	1960	classic fiction
The Catcher in the Rye	J.D. Salinger	1951	classic fiction
The Lord of the Rings	J.R.R. Tolkien	1954	fantasy adventure
Moby Dick	Herman Melville	1851	adventure classic
The Great Gatsby	F. Scott Fitzgerald	1925	classic fiction
1984	George Orwell	1949	dystopian fiction

Admin Panel

Homepage

Books

Title	Author	Year	Keywords	Actions
To Kill a Mockingbird	Harper Lee	1960	classic fiction	Change book Delete book
1984	George Orwell	1949	dystopian fiction	Change book Delete book
Pride and Prejudice	Jane Austen	1813	romance classic	Change book Delete book
The Great Gatsby	F. Scott Fitzgerald	1925	classic fiction	Change book Delete book
Moby Dick	Herman Melville	1851	adventure classic	Change book Delete book
War and Peace	Leo Tolstoy	1869	historical classic	Change book Delete book
The Catcher in the Rye	J.D. Salinger	1951	classic fiction	Change book Delete book
The Hobbit	J.R.R. Tolkien	1937	fantasy adventure	Change book Delete book
Crime and Punishment	Fyodor Dostoevsky	1866	classic philosophical	Change book Delete book
The Brothers Karamazov	Fyodor Dostoevsky	1880	classic philosophical	Change book Delete book
Brave New World	Aldous Huxley	1932	dystopian fiction	Change book Delete book



Відповіді на контрольні питання:

1. **Inversion of Control (IoC)** – це принцип, коли управління створенням та життєвим циклом об'єктів передається зовнішній системі (наприклад IoC-контейнеру). Тобто, замість того, щоб клас сам створював свої залежності, вони передаються йому ззовні.
   IoC container – це інструмент, який керує створенням об'єктів та їхніми залежностями згідно з принципом IoC. У Spring таким контейнером є ApplicationContext, який автоматично створює, конфігурує та управляє бінами.
   Dependency Injection (DI) – це конкретна реалізація IoC, яка передбачає передачу залежностей класу ззовні, замість того, щоб клас сам створював їх. В Spring DI здійснюється за допомогою конструкторів, сетерів або через поля.
   Dependency inversion principle – принцип SOLID, який стверджує, що модулі високого рівня не повинні залежати від модулів низького рівня, а обидва повинні залежати від абстракцій. Абстракції не повинні залежати від деталей, але деталі повинні залежати від абстракцій.

2. **@Component** – це анотація на класі, яка вказує IoC-контейнеру Spring створити бін цього класу і керувати його життєвим циклом. Вона використовується для автоматичного сканування компонентів через механізм сканування пакетів.
   @Bean – це анотація на методі, яка вказує, що результат цього методу має бути зареєстрований як бін в Spring-контейнері. Її використовують для більш тонкого налаштування та створення бінів в класах, які помічені анотацією @Configuration.

3. Ін'єкція залежностей напряму у поляне є ідеальною практикою.
   Недоступність для тестування: У випадку ін'єкції через поле ви не можете легко протестувати клас, тому що залежності не передаються через конструктор або сетери.
   Недоступність для фінальних полів: Поля, що оголошені як final, не можуть бути ін'єктовані таким способом.
   Ускладнення налагодження: Непрозорість процесу ін'єкції може ускладнювати розуміння того, як і коли залежність ініціалізується.

4. **Ін'єкція через конструктор:**
   Рекомендується для обов'язкових залежностей.
   Забезпечує імм'ютабельність класу (final).
   Ін'єкція через сетери:
   Використовується для необов'язкових залежностей.
   Коли залежність може змінюватися після створення об'єкта.

5. **Singleton:**
   Коли бін має бути створений один раз і використовуватися протягом усього часу роботи програми. Використовується для компонентів, які є станозалежними, таких як служби або контролери.
   Prototype:
   Використовується, коли потрібні нові екземпляри бінів для кожного запиту. Корисно для компонентів з коротким життєвим циклом або коли кожен користувач має отримувати окремий екземпляр об'єкта.

6. Spring дозволяє створювати циркулярні залежності, але це може призвести до проблем у випадку ін'єкції через конструктор. При використанні сетерів або полів Spring може коректно обробляти такі

залежності завдяки механізму proxy. У випадку ін'єкції через конструктор циркулярні залежності спричиняють помилки.

7. У Spring може бути кілька реалізацій одного інтерфейсу, але якщо цей інтерфейс використовується для ін'єкції, потрібно уточнити, яку саме реалізацію ви хочете інжектувати. Це можна зробити за допомогою анотації **@Qualifier** для вказання конкретного біна, або **@Primary**, щоб позначити основну реалізацію, яка буде інжектуватися за замовчуванням.

8. Так, бін може мати кілька методів з анотацією @Autowired одночасно.

9. Ні, бін не може мати кілька конструкторів з анотацією @Autowired одночасно. Spring не зможе визначити, який конструктор використовувати для ін'єкції.