

70094 16

Creation

Submitters

zz3823

sf23

Zhang Zhang

Shihan Fu

Emarking

Final Tests**TestSummary.txt: 1/1****Zhang Zhang - zz3823:v5**

1: Final Tests: Summary for zz3823 of v5

2: -----

3:

4: Public Tests:

5: Compiles: 1 / 1 +

6: Tests Pass: 1 / 1

7: Coverage and Style Checks: 1 / 1

8:

9: Git Repo: git@gitlab.doc.ic.ac.uk:lab2324_spring/SE_Design_Ex6_zz3823.git

10: Commit ID: 4df6d

Formatting doesn't actually pass Google's guidelines? Should use 2 spaces indentation.

17/20

Good implementation!

Since you're controlling what `searchFor` returns, you only need to checks that the queries are correct (using expectations). You also need to check (once) that the correct results is propagated from `searchFor`.

`catalogue` must be a mandatory parameter for the builder: the query without an underlying library doesn't make sense and is useless.

When implementing singleton, make the constructor private.

See comments below.

Final Tests BookSearchQueryTest.java: 1/3 Zhang Zhang - zz3823:v5

```

1: package ic.doc;
2:
3: import ic.doc.catalogues.LibraryCatalogue;
4: import org.jmock.Expectations;
5: import org.jmock.integration.junit4.JUnitRuleMockery;
6: import org.junit.Rule;
7: import org.junit.Test;
8:
9: import java.util.Arrays;
10: import java.util.List;
11:
12: import static org.hamcrest.CoreMatchers.is;
13: import static org.junit.Assert.assertThat;
14: import static org.junit.Assert.assertTrue;
15:
16: public class BookSearchQueryTest {
17:
18:     @Rule public JUnitRuleMockery context = new JUnitRuleMockery();
19:
20:     private final LibraryCatalogue catalogue = context.mock(LibraryCatalogue.class);
21:
22:     @Test
23:     public void searchesForBooksInLibraryCatalogueByAuthorSurname() {
24:         context.checking(
25:             new Expectations() {
26:                 {
27:                     oneOf(catalogue).searchFor(with(any(String.class)));
28:                     will(
29:                         returnValue(
30:                             Arrays.asList(
31:                                 new Book("A Tale of Two Cities", "Charles Dickens",
1859),
32:                                 new Book("Oliver Twist", "Charles Dickens", 1838)));
33:                 }
34:             });
35:
36:         BookSearchQuery query =
37:             new BookSearchQueryBuilder().withLastName("dickens")
38:             .libraryCatalogue(catalogue).build();
39:         List<Book> books = query.execute();
40:
41:         assertThat(books.size(), is(2));
42:         assertTrue(books.get(0).matchesAuthor("dickens"));
43:     }
44:
45:     @Test
46:     public void searchesForBooksInLibraryCatalogueByAuthorFirstname() {
47:         context.checking(
48:             new Expectations() {
49:                 {
50:                     oneOf(catalogue).searchFor(with(any(String.class)));
51:                     will(
52:                         returnValue(
53:                             Arrays.asList(
54:                                 new Book("Pride and Prejudice", "Jane Austen", 1813),
55:                                 new Book("Sense and Sensibility", "Jane Austen",
1811)));
56:                 }
57:             });
58:
59:         BookSearchQuery query =
60:             new BookSearchQueryBuilder().withFirstName("Jane")
61:             .libraryCatalogue(catalogue).build();
62:         List<Book> books = query.execute();
63:     }

```

Final Tests BookSearchQueryTest.java: 2/3 Zhang Zhang - zz3823:v5

```

62:     assertThat(books.size(), is(2));
63:     assertTrue(books.get(0).matchesAuthor("Austen"));
64: }
65:
66: @Test
67: public void searchesForBooksInLibraryCatalogueByTitle() {
68:     context.checking(
69:         new Expectations() {
70:             {
71:                 oneOf(catalogue).searchFor(with(any(String.class)));
72:                 will(
73:                     returnValue(
74:                         List.of(new Book("A Tale of Two Cities", "Charles
Dickens", 1859)));
75:             }
76:         });
77:
78:         BookSearchQuery query =
79:             new BookSearchQueryBuilder().withTitle("Two Cities")
80:             .libraryCatalogue(catalogue).build();
81:         List<Book> books = query.execute();
82:
83:         assertThat(books.size(), is(1));
84:         assertTrue(books.get(0).matchesAuthor("dickens"));
85:     }
86:
87:     @Test
88:     public void searchesForBooksInLibraryCatalogueBeforeGivenPublicationYear() {
89:         context.checking(
90:             new Expectations() {
91:                 {
92:                     oneOf(catalogue).searchFor(with(any(String.class)));
93:                     will(
94:                         returnValue(
95:                             Arrays.asList(
96:                                 new Book("Hamlet", "William Shakespeare", 1603),
97:                                 new Book("The Tempest", "William Shakespeare", 1611)));
98:                 }
99:             });
100:
101:         BookSearchQuery query =
102:             new
BookSearchQueryBuilder().publishedBefore(1700).libraryCatalogue(catalogue).build();
103:         List<Book> books = query.execute();
104:
105:         assertThat(books.size(), is(2));
106:         assertTrue(books.get(0).matchesAuthor("Shakespeare"));
107:     }
108:
109:     @Test
110:     public void searchesForBooksInLibraryCatalogueAfterGivenPublicationYear() {
111:         context.checking(
112:             new Expectations() {
113:                 {
114:                     oneOf(catalogue).searchFor(with(any(String.class)));
115:                     will(
116:                         returnValue(List.of(new Book("Lord of the Flies", "William
Golding", 1954)));
117:                 }
118:             });
119:
120:         BookSearchQuery query =
121:             new
BookSearchQueryBuilder().publishedAfter(1950).libraryCatalogue(catalogue).build();
122:         List<Book> books = query.execute();
123:     }

```

Final Tests BookSearchQueryTest.java: 3/3 Zhang Zhang - zz3823:v5

```

123:     assertThat(books.size(), is(1));
124:     assertTrue(books.get(0).matchesAuthor("Golding"));
125: }
126:
127: @Test
128: public void searchesForBooksInLibraryCatalogueWithCombinationOfParameters() {
129:     context.checking(
130:         new Expectations() {
131:             {
132:                 oneOf(catalogue).searchFor(with(any(String.class)));
133:                 will(returnValue(List.of(new Book("Oliver Twist", "Charles Dickens", 1838))));
134:             }
135:         });
136:
137:     BookSearchQuery query =
138:         new BookSearchQueryBuilder()
139:             .withLastName("dickens")
140:             .publishedBefore(1840)
141:             .libraryCatalogue(catalogue)
142:             .build();
143:     List<Book> books = query.execute();
144:
145:     assertThat(books.size(), is(1));
146:     assertTrue(books.get(0).matchesAuthor("charles dickens"));
147: }
148:
149: @Test
150: public void searchesForBooksInLibraryCatalogueWithCombinationOfTitleAndOtherParameters() {
151:     context.checking(
152:         new Expectations() {
153:             {
154:                 oneOf(catalogue).searchFor(with(any(String.class)));
155:                 will(
156:                     returnValue(
157:                         Arrays.asList(
158:                             new Book("Great Expectations", "Charles Dickens", 1861),
159:                             new Book("The Mystery of Edwin Drood", "Charles Dickens", 1870),
160:                             new Book("The Old Curiosity Shop", "Charles Dickens", 1841)
161:                         )
162:                     );
163:             }
164:
165:             BookSearchQuery query =
166:                 new BookSearchQueryBuilder()
167:                     .withTitle("of")
168:                     .publishedAfter(1800)
169:                     .publishedBefore(2000)
170:                     .libraryCatalogue(catalogue)
171:                     .build();
172:             List<Book> books = query.execute();
173:
174:             assertThat(books.size(), is(3));
175:             assertTrue(books.get(0).matchesAuthor("charles dickens"));
176:         }
177:     }

```

Final Tests LibraryCatalogue.java: 1/1 Zhang Zhang - zz3823:v5

```

1: package ic.doc.catalogues;
2:
3: import ic.doc.Book;
4:
5: import java.util.Collection;
6: import java.util.List;
7:
8: public interface LibraryCatalogue {
9:     List<Book> searchFor(String query);
10:
11:     Collection<Book> allTheBooks();
12: }

```

'allTheBooks()' method was initially private, so shouldn't be extracted into the interface.

```

1: package ic.doc.catalogues;
2:
3: import static ic.doc.catalogues.QueryParser.firstNameFrom;
4: import static ic.doc.catalogues.QueryParser.lastNameFrom;
5: import static ic.doc.catalogues.QueryParser.publishedAfterFrom;
6: import static ic.doc.catalogues.QueryParser.publishedBeforeFrom;
7: import static ic.doc.catalogues.QueryParser.titleFrom;
8:
9: import ic.doc.Book;
10: import java.util.Arrays;
11: import java.util.Collection;
12: import java.util.List;
13: import java.util.stream.Collectors;
14:
15: public class BritishLibraryCatalogue implements LibraryCatalogue {
16:
17:     private static BritishLibraryCatalogue instance;
18:
19:     // imagine that each new instance of this object uses more than 500MB of RAM
20:
21:     private final Collection<Book> catalogue = allTheBooks();
22:
23:     // Private constructor to prevent instantiation.
24:     private BritishLibraryCatalogue() {
25:         System.out.println("Memory Usage: 500MB...");
26:     }
27:
28:     // Static method to get the instance of the class.
29:     public static synchronized BritishLibraryCatalogue getInstance() {
30:         if (instance == null) {
31:             instance = new BritishLibraryCatalogue();
32:         }
33:         return instance;
34:     }
35:
36:     @Override
37:     public List<Book> searchFor(String query) {
38:         return catalogue.stream()
39:             .filter(book -> book.matchesAuthor(lastNameFrom(query)))
40:             .filter(book -> book.matchesAuthor(firstNameFrom(query)))
41:             .filter(book -> book.matchesTitle(titleFrom(query)))
42:             .filter(book -> book.publishedSince(publishedAfterFrom(query)))
43:             .filter(book -> book.publishedBefore(publishedBeforeFrom(query)))
44:             .collect(Collectors.toList());
45:     }
46:
47:     @Override
48:     public Collection<Book> allTheBooks() {
49:
50:         return Arrays.asList(
51:             new Book("A Tale of Two Cities", "Charles Dickens", 1859),
52:             new Book("Pride and Prejudice", "Jane Austen", 1813),
53:             new Book("Pride and Prejudice", "Jane Austen", 1813),
54:             new Book("The Picture of Dorian Gray", "Oscar Wilde", 1890),
55:             new Book("Oliver Twist", "Charles Dickens", 1838),
56:             new Book("Frankenstein", "Mary Shelley", 1817),
57:             new Book("Brave New World", "Aldous Huxley", 1932),
58:             new Book("Lord of the Flies", "William Golding", 1954),
59:             new Book("Hamlet", "William Shakespeare", 1603),
60:             new Book("The Life and Opinions of Tristram Shandy, Gentleman",
61: "Laurence Sterne", 1759));
62:
63:         // and so on... Imagine that this list is very large and therefore uses a
64: lot of memory.
65:     }

```

```

65: }

```

It's still possible to construct your class without using `getInstance`. Also need to make the constructor private.

```

1: package ic.doc;
2:
3: import ic.doc.catalogues.BritishLibraryCatalogue;
4: import ic.doc.catalogues.LibraryCatalogue;
5:
6: public class BookSearchQueryBuilder {
7:
8:     private String name1;
9:     private String name2;
10:    private String title;
11:    private Integer date1;
12:    private Integer date2;
13:    private LibraryCatalogue catalogue = BritishLibraryCatalogue.getInstance();
14:
15:    public BookSearchQueryBuilder withFirstName(String name1) {
16:        this.name1 = name1;
17:        return this;
18:    }
19:
20:    public BookSearchQueryBuilder withLastName(String name2) {
21:        this.name2 = name2;
22:        return this;
23:    }
24:
25:    public BookSearchQueryBuilder withTitle(String title) {
26:        this.title = title;
27:        return this;
28:    }
29:
30:    public BookSearchQueryBuilder publishedAfter(Integer date1) {
31:        this.date1 = date1;
32:        return this;
33:    }
34:
35:    public BookSearchQueryBuilder publishedBefore(Integer date2) {
36:        this.date2 = date2;
37:        return this;
38:    }
39:
40:    public BookSearchQueryBuilder libraryCatalogue(LibraryCatalogue catalogue) {
41:        this.catalogue = catalogue;
42:        return this;
43:    }
44:
45:    public BookSearchQuery build() {
46:        return new BookSearchQuery(name1, name2, title, date1, date2, catalogue);
47:    }
48: }

```

Unnecessary coupling and a dangerous default.

Make 'catalogue' a mandatory parameter in the constructor (and maybe add a factory method as well).

```

1: package ic.doc;
2:
3: import ic.doc.catalogues.LibraryCatalogue;
4: import java.util.List;
5:
6: public class BookSearchQuery {
7:
8:     private final String name1;
9:     private final String name2;
10:    private final String title;
11:    private final Integer date1;
12:    private final Integer date2;
13:    private final LibraryCatalogue catalogue;
14:
15:    public BookSearchQuery(
16:        String p1, String p2, String p3, Integer p4, Integer p5, LibraryCatalogue
17:        catalogue) {
18:        this.name1 = p1;
19:        this.name2 = p2;
20:        this.title = p3;
21:        this.date1 = p4;
22:        this.date2 = p5;
23:        this.catalogue = catalogue;
24:    }
25:
26:    public List<Book> execute() {
27:        StringBuilder query = new StringBuilder();
28:        if (name1 != null) {
29:            query.append("FIRSTNAME=").append(name1).append(" ");
30:        }
31:        if (name2 != null) {
32:            query.append("LASTNAME=").append(name2).append(" ");
33:        }
34:        if (title != null) {
35:            query.append("TITLECONTAINS(").append(title).append(") ");
36:        }
37:        if (date1 != null) {
38:            query.append("PUBLISHEDAFTER(").append(date1).append(") ");
39:        }
40:        if (date2 != null) {
41:            query.append("PUBLISHEDBEFORE(").append(date2).append(") ");
42:        }
43:        return catalogue.searchFor(query.toString());
44:    }

```

Final Tests

testResults.txt: 1/1

Zhang Zhang - zz3823:v5

```
1: ----- Test Output -----
2: Running LabTS build... (Wed 21 Feb 23:08:07 UTC 2024)
3:
4: Submission summary...
5: You made 6 commits
6:   - 6908581 feat: Introduce builder to improve query construction [3 files changed, 60 insertions, 8 deletions]
7:   - 806bda8 feat: Introduce Singleton to ensure only one instance of BritishLibraryCatalogue is created [3 files changed, 17 insertions, 7 deletions]
8:   - 2b8a182 feat: Introduce Dependence Inversion to reduce coupling between BookSearchQuery and BritishLibraryCatalogue [5 files changed, 29 insertions, 6 deletions]
9:   - 1d23dfa feat: Introduce mock setup to test BookSearchQuery in isolation [2 files changed, 113 insertions, 24 deletions]
10:  - 5c1adcd style: minor changes [2 files changed, 3 insertions, 4 deletions]
11:  - 4df6db6 refactor: Put allTheBooks into the interface LibraryCatalogue [2 files changed, 6 insertions, 1 deletion]
12:
13: Preparing...
14:
15: BUILD SUCCESSFUL in 622ms
16:
17: Compiling...
18: BUILD SUCCESSFUL in 4s
19:
20: Running tests...
21:
22: ic.doc.BookTest > supportsPublicationDataQuery PASSED
23:
24: ic.doc.BookTest > supportsCaseInsensitiveTitleQuery PASSED
25:
26: ic.doc.BookTest > convertsToFormattedStringOfTitleAndAuthor PASSED
27:
28: ic.doc.BookTest > supportsCaseInsensitiveAuthorQuery PASSED
29:
30: ic.doc.BookSearchQueryTest > searchesForBooksInLibraryCatalogueWithCombinationOfParameters PASSED
31:
32: ic.doc.BookSearchQueryTest > searchesForBooksInLibraryCatalogueByAuthorFirstname PASSED
33:
34: ic.doc.BookSearchQueryTest > searchesForBooksInLibraryCatalogueByTitle PASSED
35:
36: ic.doc.BookSearchQueryTest > searchesForBooksInLibraryCatalogueAfterGivenPublicationYear PASSED
37:
38: ic.doc.BookSearchQueryTest > searchesForBooksInLibraryCatalogueWithCombinationOfTitleAndOtherParameters PASSED
39:
40: ic.doc.BookSearchQueryTest > searchesForBooksInLibraryCatalogueBeforeGivenPublicationYear PASSED
41:
42: ic.doc.BookSearchQueryTest > searchesForBooksInLibraryCatalogueByAuthorSurname PASSED
43:
44: BUILD SUCCESSFUL in 1s
45:
46: Checking test coverage and code style...
47: BUILD SUCCESSFUL in 3s
48: Finished auto test. (Wed 21 Feb 23:08:25 UTC 2024)
49:
50: ----- Test Errors -----
51:
```