

Test Coverage Report

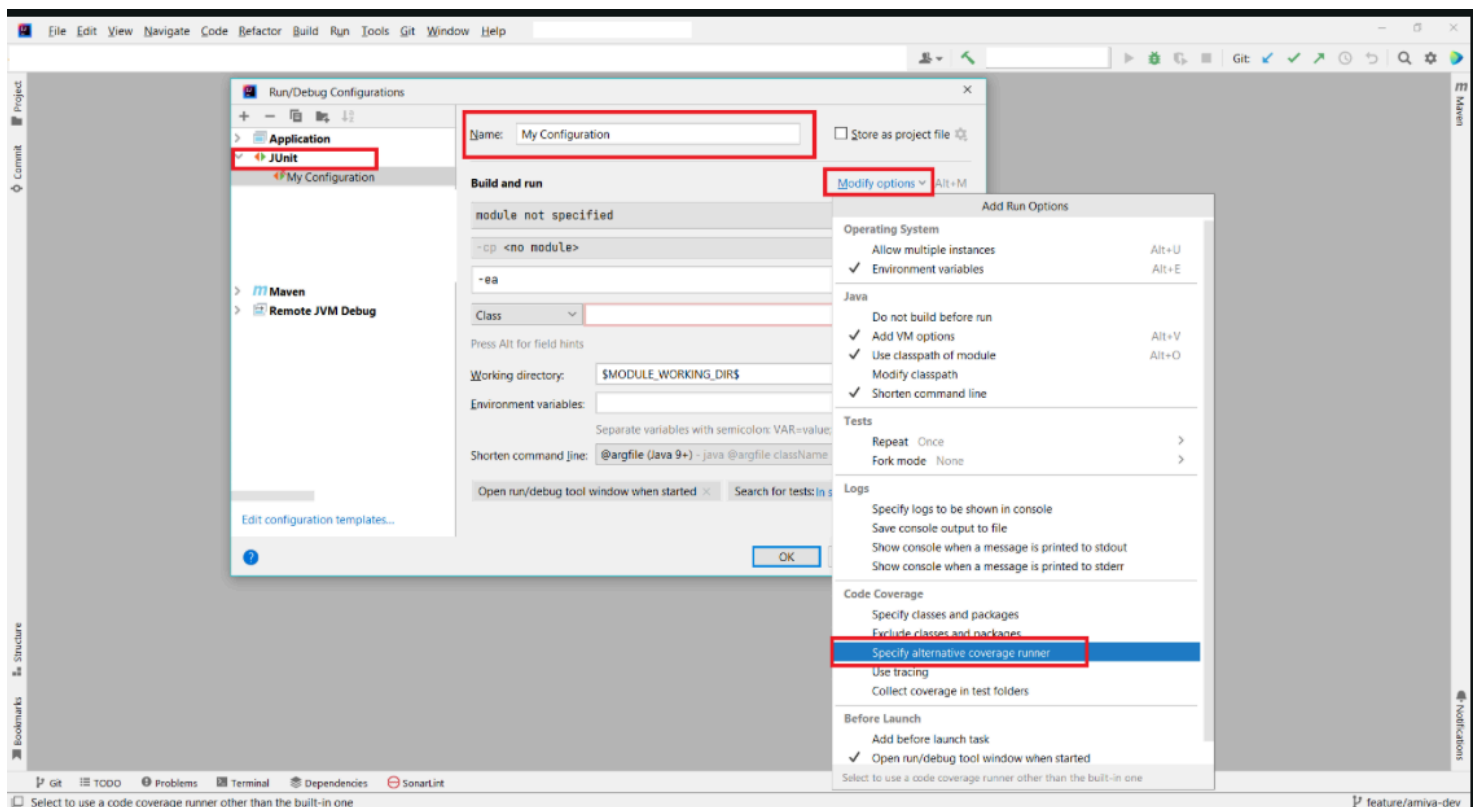
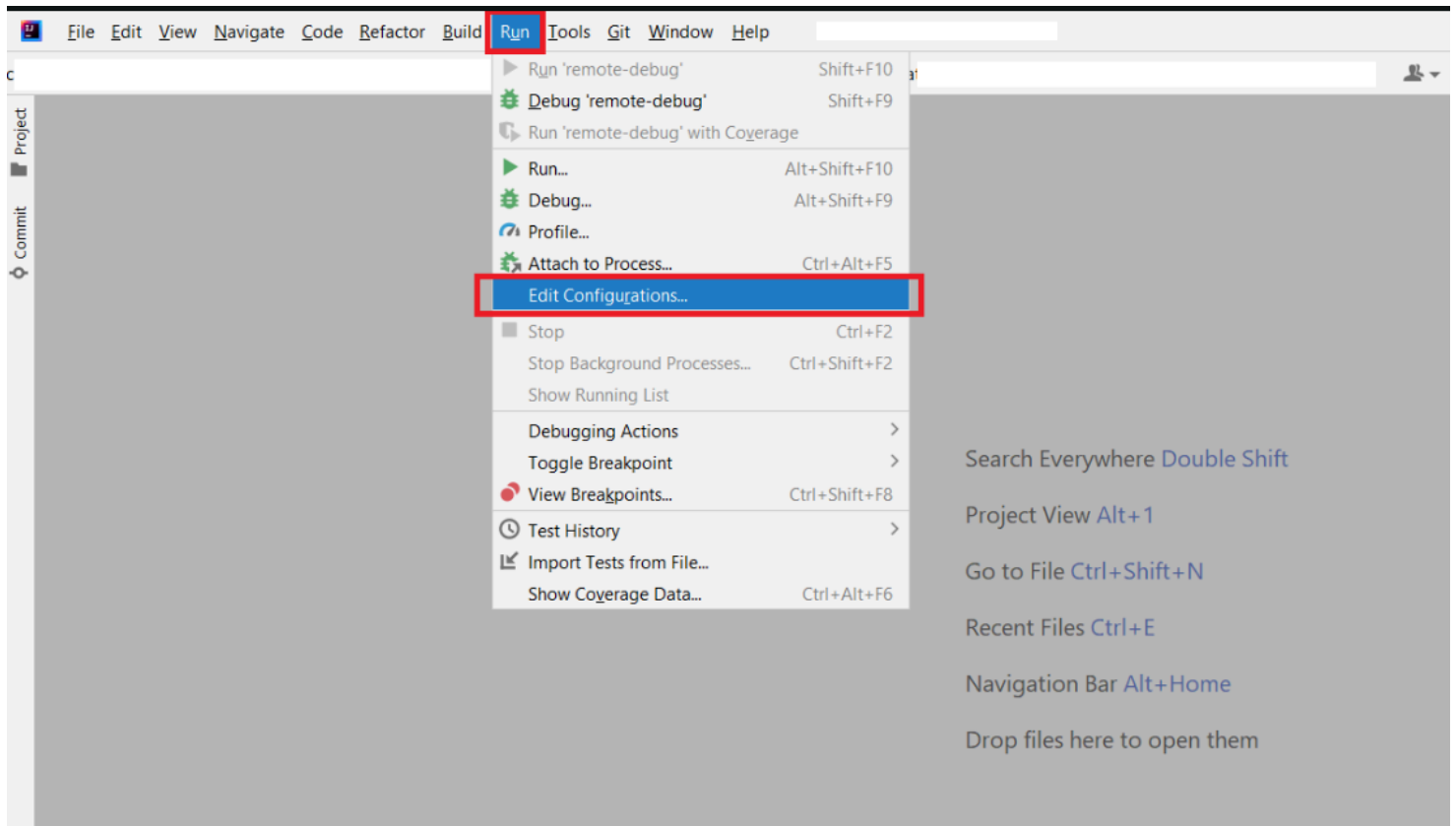
Introduction:

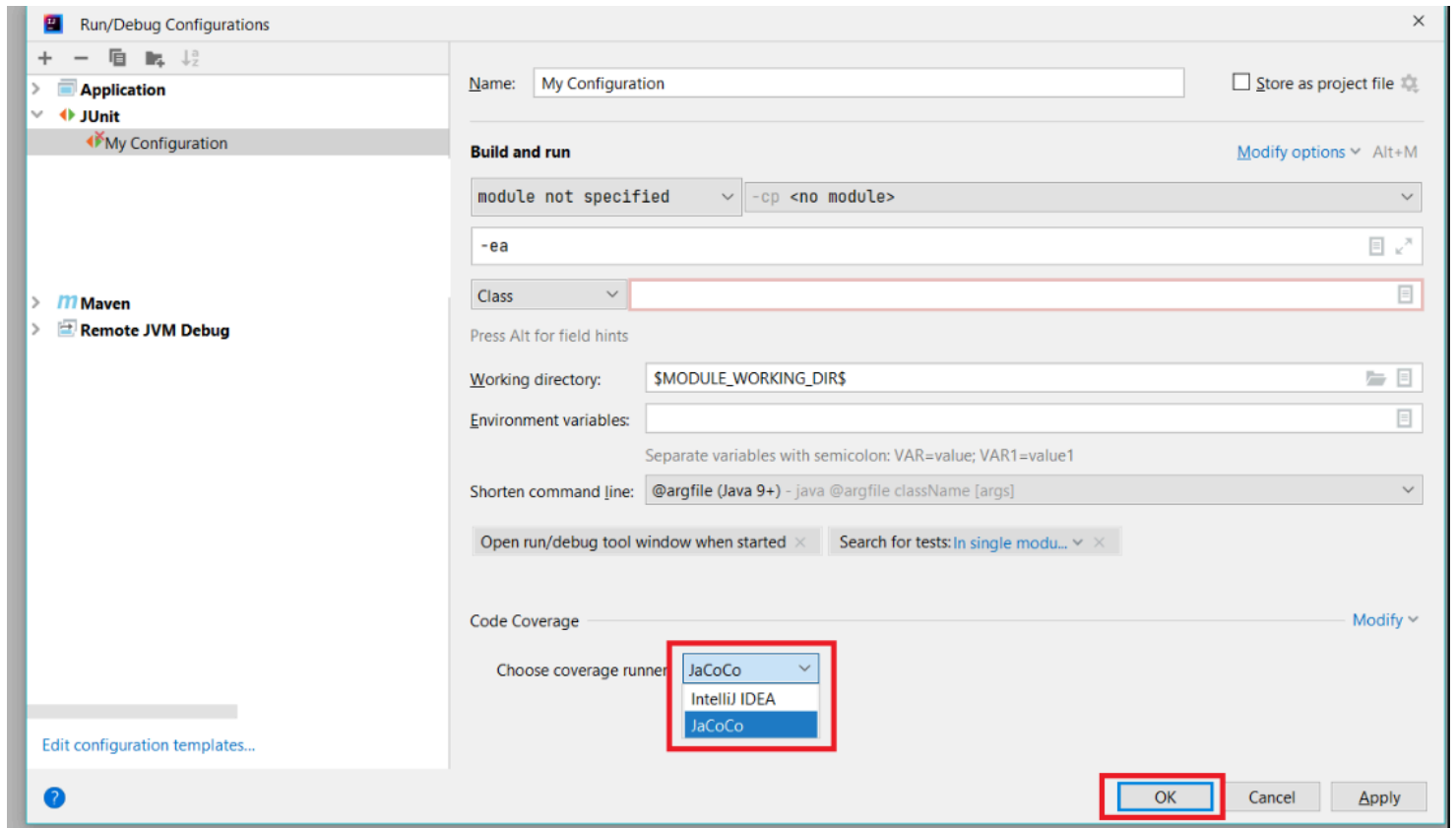
- Test Coverage for the Library Management System is the best way to ensure the validity of all critical aspects of the system through comprehensive analysis, as it helps to identify the tested, as well as the untested parts, by providing a detailed calculation of various elements : classes, methods, lines and branches .
- In this report, we used JaCoCo, an open-source tool compatible with JUnit 5; it offers detailed overview into the code set case coverage within several aspects: classes, methods, lines and branches .
- this report has different purposes :
 1. It helps testers spot defects early in the system life cycle .
 2. It provides a specific calculation of hoe much the code we did cover .
 3. It shows the test cases of low to no relevance to our project.

The following section will present the coverage Test cases in a more detailed way; it includes the representation of the Test Coverage, plus a structured explanation of every coverage metric .

Testing Infrastructure:

- JUnit 5 is a unit-testing framework in the java ecosystem. It contains a number of features that supports Java 8 and above, as well as many different styles of testing . However, IntelliJ supports JUnit 5 by default(it is simply done by Right Click->Run).
- As you can see in our code, we have used @BeforeEach, @Test, and assertions (assertEquals, assertFalse, assertTrue...) . These annotations are extended models for writing tests in JUnit 5 .
- JaCoCo (Java Code Coverage) is an open-source code coverage library for java . The library is often used to measure the coverage of unit or integration tests .
- Implementation : go to Run>Edit Configuration>modify options> specify alternative coverage runner>select JaCoCo from the Choose coverage runner





Coverage Summary:

Coverage Tests in 'Prototype.test' x				
Element	Class, %	Method, %	Line, %	Branch, %
all	50% (7/14)	42% (36/85)	31% (95/304)	16% (13/77)
Main	0% (0/1)	0% (0/1)	0% (0/8)	0% (0/2)
view	25% (2/8)	11% (2/18)	8% (14/168)	1% (1/59)
View	100% (1/1)	25% (1/4)	24% (6/25)	6% (1/16)
ReportingMenu	0% (0/1)	0% (0/2)	0% (0/15)	0% (0/4)
ReportingInfo	0% (0/1)	0% (0/2)	0% (0/4)	100% (0/0)
MainMenu	100% (1/1)	50% (1/2)	40% (8/20)	0% (0/5)
CustomerMenu	0% (0/1)	0% (0/2)	0% (0/16)	0% (0/4)
CustomerInfo	0% (0/1)	0% (0/2)	0% (0/27)	0% (0/8)
BookMenu	0% (0/1)	0% (0/2)	0% (0/16)	0% (0/4)
BookInfo	0% (0/1)	0% (0/2)	0% (0/45)	0% (0/18)
model	100% (3/3)	45% (17/37)	58% (40/68)	100% (0/0)
Customer	100% (1/1)	41% (5/12)	56% (13/23)	100% (0/0)
BookCopy	100% (1/1)	41% (5/12)	52% (11/21)	100% (0/0)
Book	100% (1/1)	53% (7/13)	66% (16/24)	100% (0/0)
exceptions	100% (1/1)	100% (1/1)	100% (2/2)	100% (0/0)
BorrowingNotNullException	100% (1/1)	100% (1/1)	100% (2/2)	100% (0/0)
controller	100% (1/1)	57% (16/28)	67% (39/58)	75% (12/16)
Controller	100% (1/1)	57% (16/28)	67% (39/58)	75% (12/16)

Here is a visual representation of the coverage metrics for each module and class within the Library Management System project .

The columns represent the several test cases related to classes, methods, lines, and branches .

element	Class%	Method%	Line%	Branch%
The packages (e.g:view,model, exceptions, controller) the classes of each package (e.g:View,MainMenu,PhysicalBook, Controller...)	Percentage of classes that have been tested == number of tested classes / the total number of classes in that package or class	Percentage of methods that have been tested == Number of tested methods / The total number of methods	Percentage of lines that have been tested == Number of tested lines / The total number of lines	Percentage of branches that have been tested == Number of tested branches / The total number of branches

— Interpretation:

● *Package : view*

1. View:

- This class is fully covered by tests, but only 25% of methods are covered.

```

7  public abstract class View {
8      private final String PROMPT_TO_EXIT = "Press 'q' to " + (this instanceof MainMenu ? "quit." : "back.");
9      private final String CURSOR = "\n> ";
10
11      protected Controller controller;
12      protected String name;
13      protected View prev;
14
15      public View(Controller controller, View prev) {
16          this.controller = controller;
17          this.prev = prev;
18      }
19      public abstract void show();

```

2. ReportingMenu+ReportingInfo:

- The classes, methods, lines are entirely not covered by tests.

3. MainMenu:

- The class is fully covered, the methods are half tested, and only 40% of lines are tested.

```

11 public MainMenu(Controller controller) {
12     super(controller, prev: null);
13
14     this.name = "Main Menu";
15
16     this.options = new ArrayList<>();
17     this.options.add("Book Menu");
18     this.options.add("Customer Menu");
19     this.options.add("Reporting Menu");
20     this.options.add("Settings");
21 }
22
23 @Override
24 public void show() {
25     System.out.println("Welcome to the Library Management System!\nPlease type a number or press enter.");
26     String input = super.promptMenu(this.options);
27
28     switch (input.charAt(0)) {
29         case '0':
30             controller.setMenu(new BookMenu(controller, prev: this));
31             break;
32         case '1':
33             controller.setMenu(new CustomerMenu(controller, prev: this));
34             break;
35         case '2':
36             controller.setMenu(new ReportingMenu(controller, prev: this));
37             break;
38         case '3':
39             super.promptAndExit(s: "A setting menu will be displayed");
40             this.show();
41         default:
42             break;
43     }
44 }

```

- *Package: model*
 - The classes, as well as the branches are fully covered, however, averagely, 45% of the methods and 58% of the lines are covered in total.
- *Package: Controller*
 - The class Controller was fully tested, 57% of methods were successfully executed, 67% of lines were tested.

—> we can say that these different percentages, concerning the Test Case Coverage are due to the use of classes when it comes to testing :

For example , in the controllerTest :

- **the Controller class:** it was fully covered and as methods: getBooks() / getPhysicalBooks() / getCustomers() / deleteBook() / getId() / addBook() , however searchBook(), searchCustomer() were not used while testing, that is why the methods and lines of the Controller class were not fully covered.
- **Book class:** the class was fully covered because it was tested and used in so many other classes such as the Controller, PhysicalBook, ControllerTest, but not all the methods were used, we can see that the constructor was primarily used .

- **Customer class:** it is also fully tested, but we only used as methods: the constructor, plus the getBorrowedList(), and this is what explains the 41%of methods tested .
- **BookCopy class:** 100% test covered as class, but only 41% methods tested because only the constructor, the getters and setBorrower() were used in the testing process.

Conclusion

The test Coverage Report for the Library Management System provides a structured analysis of the tests covering the code . These results highlight several areas that require some improvements to ensure an effective, yet efficient system performance .