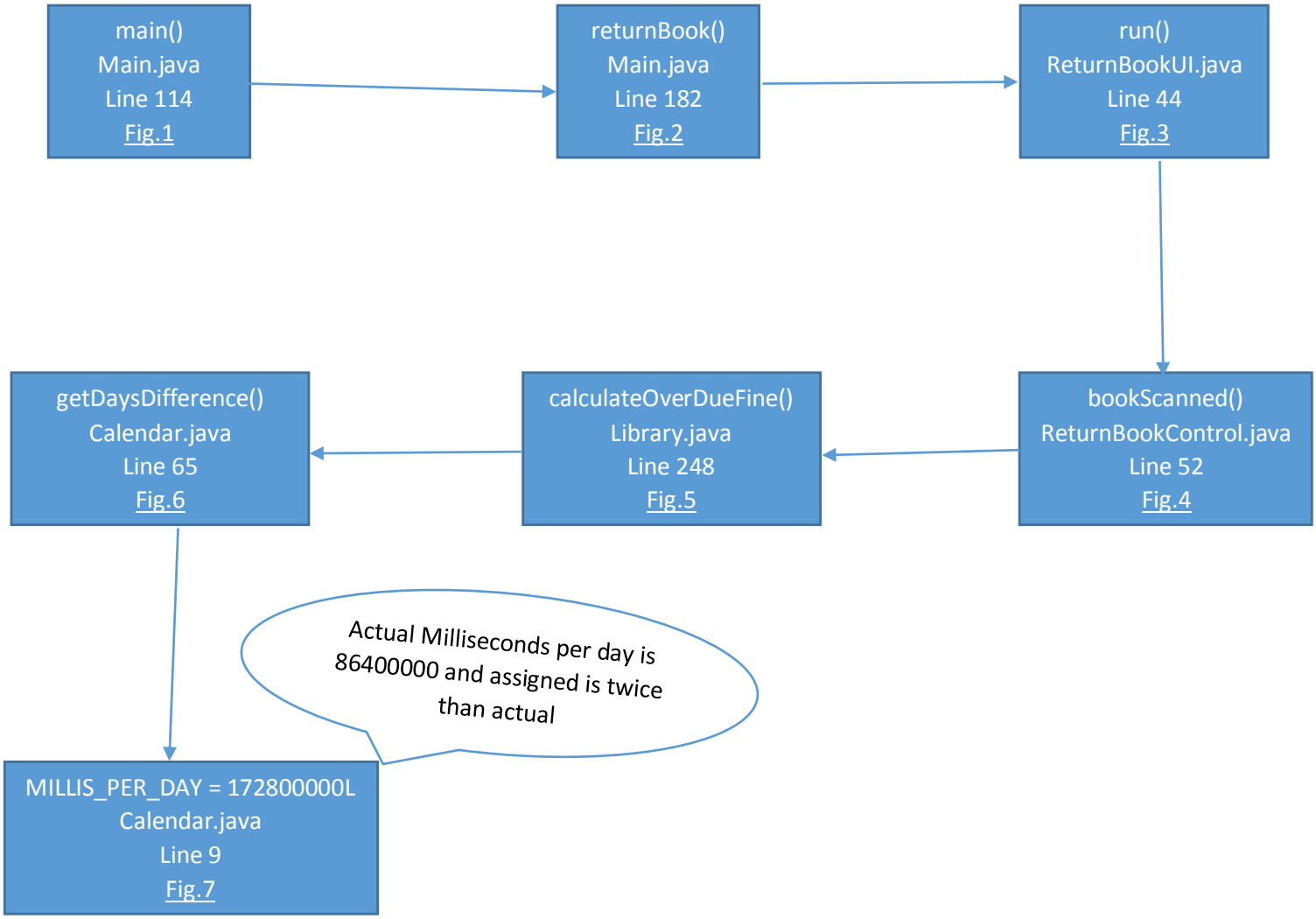
Table of Contents

Bug Tracing 1

Note: - Fig.* is hyperlink




```
113             case "R":
114                 returnBook();
115                 libraryHelper.saveLibrary(library);
116                 break;
```

Fig.1

```
180     private static void returnBook() {
181         IReturnBookControl returnBookControl = new ReturnBookControl(library);
182         new ReturnBookUI(returnBookControl).run();
183     }
```

Fig.2

```
23  public void run() {
24      showOutput( object: "Return Book Use Case UI");
25      SimpleDateFormat dateFormat = new SimpleDateFormat( pattern: "dd/MM/yyyy");
26      Date currentDate = Calendar.getInstance().getDate();
27      String dateString = dateFormat.format(currentDate);
28      showOutput( object: dateString + "\n");
29
30      while (true) {
31
32          switch (uiState) {
33
34          case INITIALISED:
35              break;
36
37          case READY:
38              String bookStr = getUserInput( prompt: "Scan Book (<enter> completes): ");
39              if (bookStr.length() == 0) {
40                  returnBookControl.scanningComplete();
41              } else {
42                  try {
43                      int bookId = Integer.valueOf(bookStr).intValue();
44                      returnBookControl.bookScanned(bookId);
45                  } catch (NumberFormatException e) {
46                      showOutput( object: "Invalid bookId");
47                  }
48              }
49              break;
```

Activate Windows

Fig.3

```
36  public void bookScanned(int bookId) {                                    ⚠ 2
37      if (!controlState.equals(ControlStateConstants.READY)) {
38          throw new RuntimeException("ReturnBookControl: cannot call bookScanned except in READY state");
39      }
40      IBook currentBook = library.getBookById(bookId);
41      if (currentBook == null) {
42          returnBookUI.display( object: "Invalid Book Id");
43          return;
44      }
45      if (!currentBook.isOnLoan()) {
46          returnBookUI.display( object: "Book has not been borrowed");
47          return;
48      }
49      currentLoan = library.getCurrentLoanByBookId(bookId);
50      double overDueFine = 0.0;
51      if (currentLoan.isOverDue()) {
52          overDueFine = library.calculateOverDueFine(currentLoan);
53          currentLoan.getPatron().incurFine(overDueFine);
54      }
55      returnBookUI.display( object: "Inspecting");
56      returnBookUI.display(currentBook);
57      returnBookUI.display(currentLoan);
58
59      if (currentLoan.isOverDue()) {
60          returnBookUI.display(String.format("\nOverdue fine : $%.2f", overDueFine));
61      }
62      returnBookUI.setState(IReturnBookUI.UIStateConstants.INSPECTING);
63      controlState = ControlStateConstants.INSPECTING;
```

Activate Windows

Fig.4

```
244 @   public double calculateOverDueFine(ILoan loan) {
245             double fine = 0.0;
246             if (loan.isOverDue()) {
247                 Date dueDate = loan.getDueDate();
248                 long daysOverDue = Calendar.getInstance().getDaysDifference(dueDate);
249                 fine = daysOverDue * FINE_PER_DAY;
250             }
251             return fine;
252         }
```

Fig.5

```
63 @    public synchronized long getDaysDifference(Date targetDate) {
64             long diffMilliseconds = getDate().getTime() - targetDate.getTime();
65             long diffDays = diffMilliseconds / MILLIS_PER_DAY;
66             return diffDays;
67         }
68
69     }
```

Fig.6

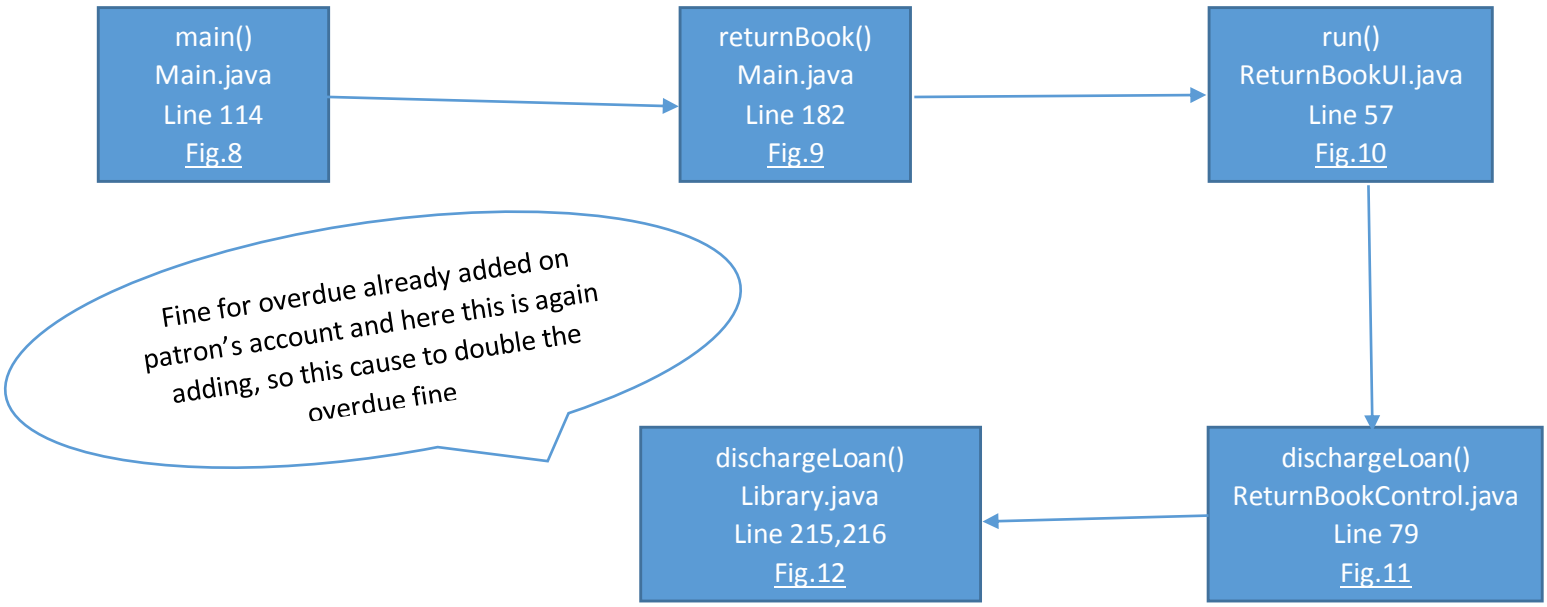
```
5   public class Calendar implements ICalendar {
6
7       private static ICalendar self;
8       private static java.util.Calendar calendar;
9       private static final long MILLIS_PER_DAY = 172800000L;
```

Fig.7

Bug Tracing 2

This bug cannot be see in return book menu option but it happens which we go to pay fine. This bug adding overdue fine second time which already added.

Note: - Fig.* is hyperlink




```
113                 case "R":
114                     returnBook();
115                     libraryHelper.saveLibrary(library);
116                     break;
```

Fig.8

```
180     private static void returnBook() {
181         IReturnBookControl returnBookControl = new ReturnBookControl(library);
182         new ReturnBookUI(returnBookControl).run();
183     }
```

Fig.9

```
51            case INSPECTING:
52                String answer = getUserInput( prompt: "Is book damaged? (Y/N): ");
53                boolean isDamaged = false;
54                if (answer.toUpperCase().equals("Y")) {
55                    isDamaged = true;
56                }
57                returnBookControl.dischargeLoan(isDamaged);
58
```

Fig.10

```
75    public void dischargeLoan(boolean isDamaged) {
76        if (!controlState.equals(ControlStateConstants.INSPECTING)) {
77            throw new RuntimeException("ReturnBookControl: cannot call dischargeLoan except in INSPECTING state");
78        }
79        library.dischargeLoan(currentLoan, isDamaged);
80        currentLoan = null;
81        returnBookUI.setState(IReturnBookUI.UIStateConstants.READY);
82        controlState = ControlStateConstants.READY;
83    }
```

Fig.11

```
210       @Override
211       public void dischargeLoan(ILoan loan, boolean isDamaged) {
212           IPatron patron = loan.getPatron();
213           IBook book = loan.getBook();
214
215           double overDueFine = calculateOverDueFine(loan);
216           patron.incurFine(overDueFine);
217
218           Integer bookId = book.getId();
219           if (isDamaged) {
220               patron.incurFine(DAMAGE_FEE);
221               damagedBooks.put(bookId, book);
222           }
223           loan.discharge(isDamaged);
224
225           currentLoans.remove(bookId);
226           setPatronBorrowingRestrictions(patron);
227       }
```

Fig.12