**Test01 - testBook:**

Description: The “testBook” test checks the “Book(String author, String title, String callNumber, int id)” constructor functions correctly.

Pre-conditions: Variables must be initiated.

Post-conditions: Creates an instance of a Book.

Data required: Book(String author, String title, String callNumber, int id)

**Test02 - testBorrow:**

Description: The “testBorrow” test checks the “borrow(ILoan loan)” method functions correctly.

Pre-conditions: [List conditions that must be true before this Test Case can start.]

Post-conditions: Associates the loan with the book.

Sets the state of the book as ON\_LOAN.

Data required: borrow(ILoan loan)

**Test03- testBorrowThrowsRuntimeExceptionNotAvailable:**

Description: The “testBorrowThrowsRuntimeExceptionNotAvailable” test checks whether the RuntimeException is throw in “borrow(ILoan)”, when the state is not AVAILABLE.

Pre-conditions: The book’s state is not currently AVAILABLE.

Post-conditions: A RuntimeException is thrown.

Data required: borrow(ILoan loan)

**Test04 - testGetLoan:**

Description: The “testGetLoan” test checks the “getLoan()” method functions correctly.

Pre-conditions: The book’s state is currently ON\_LOAN.

Post-conditions: Returns the Loan associated with the book.

Data required: getLoan()

**Test05 - testReturnBookTrue:**

Description: The “testReturnBookTrue” test checks whether the state of the book is changed from ON\_LOAN to DAMAGED by “returnBook()”.

Pre-conditions: The book is damaged.

Post-conditions: Sets the state of the book as DAMAGED.

Data required: returnBook()

**Test06 - testReturnBookFalse:**

Description: The “testReturnBookFalse” test checks whether the state of the book is changed from ON\_LOAN to AVAILABLE by “returnBook()”.

Pre-conditions: The book is not damaged.

Post-conditions: Sets the state of the book as AVAILABLE.

Data required: returnBook()

**Test07 - testReturnBookThrowsRuntimeExceptionNotOnLoan:**

Description: The “testReturnBookThrowsRuntimeExceptionNotLoan” test checks whether the RuntimeException is throw in “returnBook()”, when the state is not ON\_LOAN.

Pre-conditions The book’s state is not currently ON\_LOAN.

Post-conditions: A RuntimeException is thrown.

Data required: returnBook()

**Test08 - testReturnBookThrowsRuntimeExceptionNotLost:**

Description: The “testReturnBookThrowsRuntimeExceptionNotLoan” test checks whether the RuntimeException is throw in “returnBook()”, when the state is not LOST.

Pre-conditions: The book’s state is not currently LOST.

Post-conditions: A RuntimeException is thrown.

Data required: returnBook()

**Test09 - testLose:**

Description: The “testLose” test checks the “lose()” method functions correctly.

Pre-conditions: The book’s state is currently ON\_LOAN.

Post-conditions: Sets the state of the book as LOST.

Data required: lose()

**Test10 - testLoseThrowsRuntimeException:**

Description: The “testLoseThrowsRuntimeException” test checks whether the RuntimeException is throw in “lose()”, when the state is not ON\_LOAN.

Pre-conditions: The book’s state is not currently ON\_LOAN.

Post-conditions: A RuntimeException is thrown.

Data required: lose()

**Test11 - testRepair:**

Description: The “testRepair” test checks the “repair()” method functions correctly.

Pre-conditions: The book’s state is not currently DAMAGED.

Post-conditions: Sets the state of the book as AVAILABLE.

Data required: repair()

**Test12 - testRepairThrowsRuntimeExceptionNotDamaged:**

Description: The “testRepairThrowsRuntimeExceptionNotDamaged” test checks whether the RuntimeException is throw in “repair()”, when the state is not DAMAGED.

Pre-conditions: The book’s state is not currently DAMAGED.

Post-conditions: A RuntimeException is thrown.

Data required: repair()

**Test13 - testDispose:**

Description: The “testDispose” test checks the “dispose()” method functions correctly.

Pre-conditions: The book’s state is either currently AVAILABLE, DAMAGED or LOST.

Post-conditions: Sets the state of the book as DISPOSED.

Data required: dispose()

**Test14 - testDisposeThrowsRuntimeExceptionNotAvailable:**

Description: The “testDisposeBookThrowsRuntimeExceptionNotAvailable” test checks whether the RuntimeException is throw in “dispose()”, when the state is not AVAILABLE.

Pre-conditions: The book’s state is not currently AVAILABLE.

Post-conditions: A RuntimeException is thrown.

Data required: dispose()

**Test15 - testDisposeThrowsRuntimeExceptionNotDamaged:**

Description: The “testDisposeBookThrowsRuntimeExceptionNotDamaged” test checks whether the RuntimeException is throw in “dispose()”, when the state is not DAMAGED.

Pre-conditions: The book’s state is not currently DAMAGED.

Post-conditions: A RuntimeException is thrown.

Data required: dispose()

**Test16 - testDisposeThrowsRuntimeExceptionNotLost:**

Description: The “testDisposeBookThrowsRuntimeExceptionNotLost” test checks whether the RuntimeException is throw in “dispose()”, when the state is not LOST.

Pre-conditions: The book’s state is not currently LOST.

Post-conditions: A RuntimeException is thrown.

Data required: dispose()

**Test17 - testGetState:**

Description: The “testGetState” test checks the “getState()” method functions correctly.

Pre-conditions: A book must exist.

The book must have a state.

Post-conditions: Returns the book’s current state.

Data required: getState()

**Test18- testGetAuthor:**

Description: The “testGetAuthor” test checks the “getAuthor()” method functions correctly.

Pre-conditions: A book must exist.

The book must have a valid author.

Post-conditions: Returns the book’s author.

Data required: getAuthor()

**Test19 - testGetAuthorThrowsIllegalArgumentExceptionNull:**

Description: The “testGetAuthorThrowsIllegalArgumentExceptionNull” test checks whether the IllegalArgumentException is throw in the “Book(String author, String title, String callNumber, int id)” constructor.

Pre-conditions: The book’s author is null.

Post-conditions: An IllegalArgumentException is thrown.

Data required: Book(String author, String title, String callNumber, int id)

**Test20 - testGetAuthorThrowsIllegalArgumentExceptionBlank:**

Description: The “testGetAuthorThrowsIllegalArgumentExceptionBlank” test checks whether the IllegalArgumentException is throw in the “Book(String author, String title, String callNumber, int id)” constructor.

Pre-conditions: The book’s author is blank.

Post-conditions: An IllegalArgumentException is thrown.

Data required: Book(String author, String title, String callNumber, int id)

**Test21 - testGetTitle:**

Description: The “testGetTitle” test checks the “getTitle()” method functions correctly.

Pre-conditions: A book must exist.

The book must have a valid author.

Post-conditions: Returns the book’s title.

Data required: getTitle()

**Test22 - testGetTitleThrowsIllegalArgumentExceptionNull:**

Description: The “testGetTitleThrowsIllegalArgumentExceptionNull” test checks whether the IllegalArgumentException is throw in the “Book(String author, String title, String callNumber, int id)” constructor.

Pre-conditions: The book’s title is null.

Post-conditions: An IllegalArgumentException is thrown.

Data required: Book(String author, String title, String callNumber, int id)

**Test23 - testGetTitleThrowsIllegalArgumentExceptionBlank:**

Description: The “testGetTitleThrowsIllegalArgumentExceptionBlank” test checks whether the IllegalArgumentException is throw in the “Book(String author, String title, String callNumber, int id)” constructor.

Pre-conditions: The book’s title is blank.

Post-conditions: An IllegalArgumentException is thrown.

Data required: Book(String author, String title, String callNumber, int id)

**Test24 - testGetCallNumber:**

Description: The “testGetCallNumber” test checks the “getCallNumber()” method functions correctly.

Pre-conditions: A book must exist.

The book must have a valid call number.

Post-conditions: Returns the book’s call number.

Data required: getCallNumber()

**Test25 - testGetCallNumberThrowsIllegalArgumentExceptionNull:**

Description: The “testGetCallNumberThrowsIllegalArgumentExceptionNull” test checks whether the IllegalArgumentException is throw in the “Book(String author, String title, String callNumber, int id)” constructor.

Pre-conditions: The book’s call number is null.

Post-conditions: An IllegalArgumentException is thrown.

Data required: Book(String author, String title, String callNumber, int id)

**Test26 - testGetCallNumberThrowsIllegalArgumentExceptionBlank:**

Description: The “testGetCallNumberThrowsIllegalArgumentExceptionBlank” test checks whether the IllegalArgumentException is throw in the “Book(String author, String title, String callNumber, int id)” constructor.

Pre-conditions: The book’s call number is blank.

Post-conditions: An IllegalArgumentException is thrown.

Data required: Book(String author, String title, String callNumber, int id)

**Test27 - testGetId:**

Description: The “testGetId” test checks the “getId()” method functions correctly.

Pre-conditions: A book must exist.

The book must have a valid id.

Post-conditions: Returns the book’s id.

Data required: getId()

**Test28 - testGetIdThrowsIllegalArgumentExceptionLessThanZero:**

Description: The “testGetIdThrowsIllegalArgumentExceptionLessThanZero” test checks whether the IllegalArgumentException is throw in the “Book(String author, String title, String callNumber, int id)” constructor.

Pre-conditions: A book ID is less than zero.

Post-conditions: An IllegalArgumentException is thrown.

Data required: Book(String author, String title, String callNumber, int id)

**Test29 - testGetIdThrowsIllegalArgumentExceptionIsZero:**

Description: The “testGetIdThrowsIllegalArgumentExceptionIsZero” test checks whether the IllegalArgumentException is throw in the “Book(String author, String title, String callNumber, int id)” constructor.

Pre-conditions: A book ID is 0.

Post-conditions: An IllegalArgumentException is thrown.

Data required: Book(String author, String title, String callNumber, int id)