**Test01 – testConstructorAllParamsOK:**

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 – testConstructorBadParamAuthorIsNull:**

Description:

The “testConstructorBadParamAuthorIsNull” 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)

**Test03 - testConstructorBadParamAuthorIsBlank:**

Description:

The “testConstructorBadParamAuthorIsBlank” 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)

**Test04 - testConstructorBadParamTitleIsNull:**

Description:

The “testConstructorBadParamTitleIsNull” 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)

**Test05 - testConstructorBadParamTitleIsBlank:**

Description:

The “testConstructorBadParamTitleIsBlank” 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)

**Test06 - testConstructorBadParamCallNumberIsNull:**

Description:

The “testConstructorBadParamCallNumberIsNull” 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)

**Test07 - testConstructorBadParamCallNumberIsBlank:**

Description:

The “testConstructorBadParamCallNumberIsBlank” 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)

**Test08 - testConstructorBadParamIdLessThanZero:**

Description:

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

Pre-conditions:

Book ID is less than zero.

Post-conditions:

An IllegalArgumentException is thrown.

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

**Test09 - testConstructorBadParamIdEqualsZero:**

Description:

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

Pre-conditions:

Book ID is 0.

Post-conditions:

An IllegalArgumentException is thrown.

Data required:

Book(String author, String title, String callNumber, int id)

**Test10 - testBorrow:**

Description:

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

Pre-conditions:

The book’s state is currently AVAILABLE.

Post-conditions:

Associates the loan with the book.

Sets the state of the book as ON\_LOAN.

Data required:

borrow(ILoan loan)

**Test11 - testBorrowNotAvailable:**

Description:

The “testBorrowNotAvailable” 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)

**Test12 - 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()

**Test13 - 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()

**Test14 - 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()

**Test15 - testReturnBookNotOnLoan:**

Description:

The “testReturnBookNotOnLoan” 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()

**Test16 - testReturnBookNotLost:**

Description:

The “testReturnBookNotLost” 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()

**Test17 - 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()

**Test18 - 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()

**Test19 - 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()

**Test20 - testRepairBookIsNotDamaged:**

Description:

The “testRepairBookIsNotDamaged” 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()

**Test21 - testDisposeWhenBookIsAvailable:**

Description:

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

Pre-conditions:

The book’s state is currently AVAILABLE.

Post-conditions:

A RuntimeException is thrown.

Data required:

dispose()

**Test22 - testDisposeWhenBookIsDamaged:**

Description:

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

Pre-conditions:

The book’s state is currently DAMAGED.

Post-conditions:

A RuntimeException is thrown.

Data required:

dispose()

**Test23 - testDisposeWhenBookIsLost:**

Description:

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

Pre-conditions:

The book’s state is currently LOST.

Post-conditions:

A RuntimeException is thrown.

Data required:

dispose()

**Test24 - testDisposeWhenBookIsOnLoan:**

Description:

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

Pre-conditions:

The book’s state is currently LOST.

Post-conditions:

A RuntimeException is thrown.

Data required:

dispose()

**Test25 - 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()

**Test26- 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()

**Test27 - 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()

**Test28 - 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()

**Test29 - 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()