*[The Test Case ID should be unique. In addition, the name of each Test Case should reflect the intent of the test case, ideally expressed as a Boolean condition.]*

**1 - testLoanParametersNotNull()**

Description:

Checks the detail of a new loan to ensure that the parameters are not empty. If the parameters are not empty then the program will set the book, borrower, borrowDate and dueDate to the loan. The state of the book will be set to ‘PENDING’.

Pre-conditions:

private final IBook book;

private final IMember borrower;

private Date borrowDate;

private Date dueDate;

this.book = book;

this.borrower = borrower;

this.borrowDate = borrowDate;

this.dueDate = returnDate;

state = ELoanState.PENDING;

Post-conditions:

assertNotNull(actualBook);

assertNotNull(actualBorrower);

assertNotNull(borrowDate);

assertNotNull(dueDate);

Data required:

[Identify the type of data required for this Test Case.]

**2 - testLoanParametersAreNull()**

Description:

Checks the detail of a new loan to ensure that the parameters are empty. This will throw an exception to the user as the parameters are empty for a new loan to be created.

Pre-conditions:

private final IBook book;

private final IMember borrower;

private Date borrowDate;

private Date dueDate;

Post-conditions:

assertNull(book);

assertNull(borrower);

assertNull(borrowDate);

assertNull(dueDate);

throw new IllegalArgumentException("Error: Parameters are empty.");

Data required:

[Identify the type of data required for this Test Case.]

**3 - testLoanCurrentStateOfLoan()**

Description:

Checks the current state of a loan is PENDING before taking out a new loan. This test confirms the current loan state is PENDING.

Pre-conditions:

private ELoanState state;

state = ELoanState.PENDING;

Post-conditions:

state = ELoanState.PENDING;

Data required:

[Identify the type of data required for this Test Case.]

**4 - testLoanParameterValues()**

Description:

This test checks the parameters of a loan. The parameters actualBook, actualBorrower, actualBorrowDate and actualDueDate should all contain values.

Pre-conditions:

private final IBook book;

private final IMember borrower;

private Date borrowDate;

private Date dueDate;

IBook actualBook = loan.getBook();

IMember actualBorrower = loan.getBorrower();

Date actualBorrowDate = borrowDate;

Date actualDueDate = dueDate;

Post-conditions:

assertEquals(book, actualBook);

assertEquals(borrower, actualBorrower);

assertEquals(borrowDate, actualBorrowDate);

assertEquals(dueDate, actualDueDate);

Data required:

[Identify the type of data required for this Test Case.]

**5 - testCommitWithPositiveID()**

Description:

Check the loans startState, then commit the loan with a positive ID and compare the loans endState. This test should show the loan start as PENDING and end as CURRENT without throwing an exception.

Pre-conditions:

int id = 5;

loan.commit(id);

int actualId = loan.getID();

ELoanState endState = loan.getCurrentState();

Post-conditions:

assertEquals(ELoanState.PENDING, startState);

assertEquals(ELoanState.CURRENT, endState);

assertEquals(id, actualId);

Data required:

[Identify the type of data required for this Test Case.]

**6 - testCommitWithNegativeID()**

Description:

Try to commit a loan with a negative loan ID. This will throw an exception as the loan ID must be positive.

Pre-conditions:

int id = -5;

loan.commit(id);

int actualId = loan.getID();

Post-conditions:

assertEquals(id, actualId);

Data required:

[Identify the type of data required for this Test Case.]

**7 - testCompleteState()**

Description:

Checks the current state of the loan is in COMPLETE state.

Pre-conditions:

loan.commit(5);

loan.complete();

Post-conditions:

assertEquals(ELoanState.COMPLETE, loan.getCurrentState());

Data required:

[Identify the type of data required for this Test Case.]

**8 - testIsOverDue()**

Description:

Checks if the module returns OVERDUE state.

Pre-conditions:

boolean state = loan.isOverDue();

Post-conditions:

assertTrue(state);

Data required:

[Identify the type of data required for this Test Case.]

**9 - testCheckOverDueError()**

Description:

Throws an exception if the current state of the loan is not CURRENT or OVERDUE. Then checks the current state is COMPLETE.

Pre-conditions:

loan.complete();

loan.checkOverDue(currentDate);

Post-conditions:

assertEquals(ELoanState.COMPLETE, loan.getCurrentState());

Data required:

[Identify the type of data required for this Test Case.]

**10 - testCheckOverDue()**

Description:

Throws an exception if the current state of the loan is not CURRENT or OVERDUE. Then checks the current state is COMPLETE.

Pre-conditions:

loan.isOverDue();

loan.checkOverDue(currentDate);

Post-conditions:

assertEquals(ELoanState.COMPLETE, loan.getCurrentState());

Data required:

[Identify the type of data required for this Test Case.]

**11 - testGetBorrower()**

Description:

Confirms the borrower of a book so that it can be returned to the user.

Pre-conditions:

IMember actualBorrower = loan.getBorrower();

Post-conditions:

assertEquals(borrower, actualBorrower);

Data required:

[Identify the type of data required for this Test Case.]

**12 - testGetBook()**

Description:

Confirms the details of a book so that it can be returned to the user.

Pre-conditions:

IBook actualBook = loan.getBook();

Post-conditions:

assertEquals(book, actualBook);

Data required:

[Identify the type of data required for this Test Case.]

**13 - testGetID()**

Description:

Checks the current value of ID to confirm the method can return a value.

Pre-conditions:

int actualID = loan.getID();

Post-conditions:

assertEquals(0, actualID);

Data required:

[Identify the type of data required for this Test Case.]

**14 - testGetCurrentState()**

Description:

Goes through each state of a typical loan and checks to see if the state changes.

Pre-conditions:

loan.commit(5);

loan.checkOverDue(currentDate);

loan.complete();

Post-conditions:

assertEquals(ELoanState.PENDING, loan.getCurrentState());

assertEquals(ELoanState.CURRENT, loan.getCurrentState());

assertEquals(ELoanState.OVERDUE, loan.getCurrentState());

assertEquals(ELoanState.COMPLETE, loan.getCurrentState());

Data required:

[Identify the type of data required for this Test Case.]

**15 - testToStringNotNull()**

Description:

Confirms that if details of a loan (author, title, borrowerFirstName, borrowerLastName, borrowDate and dueDate) can be returned via toString method.

Pre-conditions:

int id = 0;

String author = "John Doe";

String title = "Test Book";

String borrowerFirstName = "Evan";

String borrowerLastName = "Watkins";

String bd = DateFormat.getDateInstance().format(borrowDate);

String dd = DateFormat.getDateInstance().format(dueDate);

String string = String.format("Loan ID: %d\n"

+ "Author: %s\n"

+ "Title of Book: %s\n"

+ "Borrower's Name: %s %s\n"

+ "Date Borrowed: %s\n"

+ "Due Date: %s",

id, author,title,borrowerFirstName, borrowerLastName, bd, dd);

Post-conditions:

assertNotNull(string);

Data required:

[Identify the type of data required for this Test Case.]