Test Case ID: FUNC-01

Test Case Title: Create a New Loan RequestObjective: Verify that a borrower can successfully create a new loan request with correct parameters.Preconditions:

- The smart contract is deployed.
- The borrower account is available and has sufficient balance.

Test Steps:

1. Step 1: Borrower calls `requestLoan` with valid parameters (amount, term, purpose, repayment frequency).
    - Expected Result: Transaction is successful and emits a `LoanRequested` event.
1. Step 2: Retrieve the loan data using `getLoanBasic(0)`.
    - Expected Result: Loan ID is 0, borrower address matches, lender is zero address, amount and term match input, status is "Requested".
1. Step 3: Retrieve the loan count using `getLoanCount()`.
    - Expected Result: Loan count is 1.

Postconditions:

- A new loan request exists in the contract with the correct parameters.

Pass Criteria:

- All expected results are met for each step.

Fail Criteria:

- Any step fails, or the loan data does not match the input.

Notes:

- This test ensures the basic loan request functionality works

Test Case ID: FUNC-02

Test Case Title: Accept a Loan RequestObjective: Verify that a lender can accept a loan request and the contract updates the state correctly.Preconditions:

- A loan request exists (see FUNC-01).
- Lender account is available.

Test Steps:

1. Step 1: Lender calls `acceptLoan(0)`.
    - Expected Result: Transaction is successful and emits a `LoanAccepted` event.
1. Step 2: Retrieve the loan data using `getLoanBasic(0)`.
    - Expected Result: Lender address is updated, status is "Accepted", other parameters remain unchanged.
1. Step 3: Retrieve the loan count.
    - Expected Result: Loan count remains 1.

Postconditions:

- The loan is marked as accepted and the lender is recorded.

Pass Criteria:

- All expected results are met for each step.

Fail Criteria:

- Any step fails, or the loan state is not updated correctly.

Notes:

- Ensures only a lender can accept a loan and the state transitions properly.

Test Case ID: TRAN-01

Test Case Title: Collateral DepositObjective: Verify that the borrower can deposit collateral and the contract updates balances and state.Preconditions:

- Loan is accepted (see FUNC-02).
- Borrower has sufficient balance.

Test Steps:

1. Step 1: Borrower calls `depositCollateral(0)` with the required collateral amount.
   - Expected Result: Transaction is successful and emits a `CollateralDeposited` event.
1. Step 2: Retrieve the loan data using `getLoanBasic(0)`.
   - Expected Result: Status is "CollateralDeposited".
1. Step 3: Check contract and borrower balances.
   - Expected Result: Contract balance increases by collateral amount, borrower balance decreases by at least the collateral amount.

Postconditions:

- Collateral is held by the contract and the loan status is updated.

Pass Criteria:

- All expected results are met for each step.

Fail Criteria:

- Any step fails, or balances/state are not updated correctly.

Notes:

- Gas costs are accounted for in the balance check.

Test Case ID: TRAN-02

Test Case Title: Multiple Loan RequestsObjective: Verify that multiple loans can be created and tracked independently.Preconditions:

- Contract is deployed.
- Borrower has sufficient balance.

Test Steps:

1. Step 1: Borrower creates the first loan request.
    - Expected Result: Loan ID is 0, event emitted.
1. Step 2: Borrower creates a second loan request with different parameters.
    - Expected Result: Loan ID is 1, event emitted.
1. Step 3: Retrieve both loans and the loan count.
    - Expected Result: Loan count is 2, both loans have correct parameters and IDs.

Postconditions:

- Both loans exist and are tracked separately.

Pass Criteria:

- All expected results are met for each step.

Fail Criteria:

- Any step fails, or loans are not tracked independently.

Notes:

- Ensures the contract can handle multiple loans.

Test Case ID: SEC-01

Test Case Title: Collateral Deposit Access ControlObjective: Ensure only the borrower can deposit collateral and only the correct amount is accepted.Preconditions:

- Loan is accepted (see FUNC-02).

Test Steps:

1. Step 1: Attacker (not borrower) tries to deposit collateral.
    - Expected Result: Transaction reverts with an error.
1. Step 2: Lender tries to deposit collateral.
    - Expected Result: Transaction reverts with an error.
1. Step 3: Third party tries to deposit collateral.
    - Expected Result: Transaction reverts with an error.
1. Step 4: Borrower tries to deposit an incorrect amount.
    - Expected Result: Transaction reverts with an error.
1. Step 5: Check loan status.
    - Expected Result: Status remains **\"Accepted\".**

Postconditions:

- Only the borrower can deposit the exact collateral amount.

Pass Criteria:

- All unauthorized attempts fail, and the loan status does not change.

Fail Criteria:

- Any unauthorized deposit succeeds, or status changes incorrectly.

Notes:

- Ensures strict access control for collateral deposits.

Test Case ID: SEC-02

Test Case Title: Prevent Invalid Loan OperationsObjective: Ensure invalid loan operations are rejected and state remains consistent.Preconditions:

- Contract is deployed.

Test Steps:

1. Step 1: Lender tries to accept a non-existent loan.
    - Expected Result: Transaction reverts with an error.
1. Step 2: Borrower creates a loan and lender accepts it.
    - Expected Result: Both transactions succeed.
1. Step 3: Lender tries to accept the same loan again.
    - Expected Result: Transaction reverts with an error.
1. Step 4: Third party tries to accept the loan.
    - Expected Result: Transaction reverts with an error.
1. Step 5: Check loan state.
    - Expected Result: Lender remains unchanged, status remains **\"Accepted\".**

Postconditions:

- Invalid operations are rejected, and the loan state is consistent.

Pass Criteria:

- All invalid attempts fail, and the loan state is correct.

Fail Criteria:

- Any invalid operation succeeds, or state is inconsistent.

Notes:

- Ensures contract robustness against invalid actions.