```python
"""
CS241 Team Activity 09
Written by Chad Macbeth
"""

class BalanceError(Exception):

    ### Create a BalanceError exception and store the overage amount
    def __init__(self, message, overage):
        super().__init__(message)
        self.overage = overage

    ### Display error message with overage info
    def display(self):
        print("{} (Overage: {})" .format(self,self.overage))


class OutOfChecksError(Exception):

    ### Create OutOfChecksError exception
    def __init__(self, message):
        super().__init__(message)

class CheckingAccount:

    ### Initialize a Checking Account and verify the starting balance
    def __init__(self, starting_balance, num_checks):
        if starting_balance < 0:
            raise BalanceError("Initial Balance Cannot Be Negative", starting_balance)
        self.balance = starting_balance
        self.num_checks = num_checks

    ### Deposit into the account.  Ensure that the deposit amount is valid
    def deposit(self, amount):
        if amount < 0:
            raise ValueError("Cannot Deposit Negative Value")
        self.balance += amount

    ### Attempt to process a check.  The amount must be valid and not exceed the balance
    ### A check must also be available.
    def write_check(self, amount):
        if amount < 0:
            raise ValueError("Cannot Write a Check for a Negative Value")
        """  Handled by Property below
        if self.balance - amount < 0:
            raise BalanceError("Insufficient Funds for Check")
        """
        if self.num_checks <= 0:
            raise OutOfChecksError("No More Checks")
        self.balance -= amount
        self.num_checks -= 1

    ### Display account information
    def display(self):
        print("Balance: {}  Number Checks: {}" .format(self.balance, self.num_checks))

    ### Apply for credit line ... not implemented yet
    def apply_for_credit(self, amount):
        raise NotImplementedError("Credit Application Not Available Today")

    @property
    def balance(self):
        return self._balance

    @balance.setter
    def balance(self, balance):
        if balance < 0:
            raise BalanceError("Insufficient Funds", balance)
        self._balance = balance

def display_menu():
    """
    Displays the available commands.
    """
    print()
    print("Commands:")
    print("  quit - Quit")
    print("  new - Create new account")
    print("  display - Display account information")
    print("  deposit - Desposit money")
    print("  check - Write a check")


def main():
    """
    Used to test the CheckingAccount class.
    """
    acc = None
    command = ""

    while command != "quit":
        display_menu()
        command = input("Enter a command: ")

        if command == "new":
            balance = float(input("Starting balance: "))
            num_checks = int(input("Numbers of checks: "))

            try:
                acc = CheckingAccount(balance, num_checks)
            except BalanceError as e:
                e.display()

        elif command == "display":
            acc.display()
        elif command == "deposit":
            amount = float(input("Amount: "))

            try:
                acc.deposit(amount)
            except ValueError as e:
                print(e)

        elif command == "check":
            amount = float(input("Amount: "))

            try:
                acc.write_check(amount)
            except BalanceError as e:
                e.display()
            except ValueError as e:
                print(e)
            except OutOfChecksError as e:
                print(e)
                response = input("Do you want to buy more checks? ")
                if response == "y":
                    try:
                        acc.balance -= 5
                        acc.num_checks += 25
                    except BalanceError as e:
                        e.display()

```

```python
        elif command == "credit":
            amount = float(input("Amount: "))
            try:
                acc.apply_for_credit(amount)
            except NotImplementedError as e:
                print(e)



if __name__ == "__main__":
    main()

```