Bank Class

```
import os
import sys

BASE_PATH = os.path.dirname(os.path.abspath(__file__))


class Bank:
    def __init__(self, name):
        self.name = name
        self.start_option = None

    def start(self):
        os.chdir(BASE_PATH)
        try:
            os.listdir().index("database")
        except:
            os.mkdir("database")

        print(f'''
            Welcome To {self.name} Bank :)

            What would you like us to do for you ?

                1. Open an account.
                2. Close an account.
                3. withdraw funds.
                4. Deposit funds.
                0. Quit

        ''')
        self.start_option = int(input(">>> "))
        self.handle_start_options()

    def handle_start_options(self):
        if self.start_option == 1:
            self.open_account()
            pass
        elif self.start_option == 2:
            self.close_account()
            pass
        elif self.start_option == 3:
            self.withdraw_funds()
            pass
```

```
        elif self.start_option == 4:
            self.deposit_funds()
            pass
        elif self.start_option == 0:
            self.close()
        else:
            print(f"{self.start_option} is not a valid choice !")
            self.start()

    def open_account(self):
        pass

    def close_account(self):
        pass

    def withdraw_funds(self):
        pass

    def deposit_funds(self):
        pass

    def close(self):
        sys.exit("Good Bye !")


def main():
    bank = Bank("Money")
    bank.start()


if __name__ == "__main__":
    main()
```

Account Class

```
from bank import Bank


class Account(Bank):
    '''
    This class represents a single account holder
    '''

    def __init__(self):
        super().__init__(name="Money")
        self.id = None
        self.first_name = None
        self.last_name = None
        self.phone = None
        self.email = None
        self.status = "CLOSED"

    def open_account(self):
        print('''
            Please enter the following details...
        ''')
        self.id = int(input("Account ID: >>> "))
        self.first_name = str(input("First Name: >>> "))
        self.last_name = str(input("Last Name: >>> "))
        self.phone = str(input("Phone: >>> "))
        self.email = str(input("Email: >>> "))
        self.status = "OPEN"
        self.create_account()
        self.save_account()
        print("Successfully created the account !")
        self.start()

    def create_account(self):
        pass

    def save_account(self):
        pass

    def __repr__(self):
        return (
            "Account("
            f"first_name: {self.first_name}, "
            f"last_name: {self.last_name}, "
```

```
            f"phone: {self.phone}, "
            f"email: {self.email}, "
            f"status: {self.status}"
            ")"
        )


def main():
    account = Account()
    account.start()


if __name__ == "__main__":
    main()
```

Checking Class

```
import os
import json
from account import Account

DATABASE_PATH = os.path.join(os.path.dirname(
    os.path.abspath(__file__)), "database")

try:
    with open(os.path.join(DATABASE_PATH, "checking.json"), "r") as
checking_data_base:
        ACCOUNTS = json.load(checking_data_base)
    # print(accounts)
except Exception as e:
    ACCOUNTS = []


class Checking(Account):
    '''
    Represents a checking account
     1. Deposit funds
     2. Withdraw funds
     3. Check balance
    '''

    def __init__(self) -> None:
        super().__init__()
        self.balance = float(0)

    def parse_accounts(self, account):
        return f"{account.get('id')}. [Name:
{account.get('first_name')}, Balance: {account.get('balance')}, Status:
{account.get('status')}]"

    def create_account(self):
        self.status = "OPEN"
        ACCOUNTS.append(self.to_json())

    def save_account(self):
        data = json.dumps(ACCOUNTS, indent=4)
        file = os.path.join(DATABASE_PATH, "checking.json")
        with open(file, "w") as checking_data_base:
            checking_data_base.write(data)
```

```
    def deposit_funds(self):
        '''
        Deposit funds to our account
        '''
        for account in ACCOUNTS:
            print(self.parse_accounts(account))

        id = int(input("Enter Account ID: >>> "))

        for account in ACCOUNTS:
            if account.get("id") == id:
                if account.get("status") == "OPEN":
                    amount = float(
                        input("Enter the amount you want to deposit: >>>
"))
                    float(amount)
                    account['balance'] += amount
                    self.save_account()
                    print("Funds deposited successfully !")
                    self.start()
                else:
                    raise Exception("Account is closed !")
            else:
                pass

    def withdraw_funds(self) -> None:
        '''
        Withdraw funds from our account
        '''

        for account in ACCOUNTS:
            print(self.parse_accounts(account))

        id = int(input("Enter Account ID: >>> "))

        for account in ACCOUNTS:
            if account.get("id") == id:
                if account.get("status") == "OPEN":
                    amount = float(
                        input("Enter the amount you want to withdraw:
>>> "))
                    float(amount)
                    if account["balance"] - amount >= 0:
                        account['balance'] -= amount
                        self.save_account()
                        print(f"Successfully withdrawed {amount}!")
```

```
                        self.start()
                    else:
                        raise Exception("Insufficient Funds !")
                else:
                    raise Exception("Account is closed !")
            else:
                pass

    def close_account(self):
        for account in ACCOUNTS:
            print(self.parse_accounts(account))

        id = int(input("Enter Account ID: >>> "))

        for account in ACCOUNTS:
            if account.get("id") == id:
                if account.get("status") == "OPEN":
                    account["status"] = "CLOSED"
                    self.save_account()
                    print("Successfully closed the account !")
                    self.start()
                else:
                    raise Exception("The account is already closed !")
            else:
                pass

    def to_json(self):
        '''Serializes account object type to json format'''
        if self.status == "OPEN":
            return self.__dict__
        else:
            raise Exception("Account does not exist !")

    def __repr__(self) -> str:
        return (
            "Checking("
            f"balance: {self.balance}"
            ")"
        )


def main():
    account = Checking()
    account.start()
```

```python
if __name__ == "__main__":
    main()
```

Savings Account

```
import os
import json
from account import Account

DATABASE_PATH = os.path.join(os.path.dirname(
    os.path.abspath(__file__)), "database")

try:
    with open(os.path.join(DATABASE_PATH, "savings.json"), "r") as
checking_data_base:
        ACCOUNTS = json.load(checking_data_base)
    # print(accounts)
except Exception as e:
    ACCOUNTS = []


class Savings(Account):
    '''
    Represents a savings account
       1. Limit to the number of withdrawals over a given interval
       2. Minimum balance requirements
       3. Interest rates - the saved funds earn interest over time ***
    '''

    def __init__(self):
        super().__init__()
        self.balance = float(0)
        self.minimum_balance = float(100)
        self.max_num_of_withdrawals = 3

    def parse_accounts(self, account):
        return f"{account.get('id')}. [Name: {account.get('first_name')},
{account.get('balance')}, Status: {account.get('status')}]"

    def create_account(self):
        self.status = "OPEN"
        ACCOUNTS.append(self.to_json())

    def save_account(self):
        data = json.dumps(ACCOUNTS, indent=4)
        file = os.path.join(DATABASE_PATH, "savings.json")
        with open(file, "w") as checking_data_base:
            checking_data_base.write(data)
```

```
    def deposit_funds(self):
        '''
        Deposit funds to our account
        '''
        for account in ACCOUNTS:
            print(self.parse_accounts(account))

        id = int(input("Enter Account ID: >>> "))

        for account in ACCOUNTS:
            if account.get("id") == id:
                if account.get("status") == "OPEN":
                    amount = float(
                        input("Enter the amount you want to deposit: >>> "
                    float(amount)
                    if amount >= account["minimum_balance"]:
                        account["balance"] += amount
                        self.save_account()
                        print("Funds deposited successfully !")
                        self.start()
                    else:
                        raise Exception(
                            f"Cannot deposit funds less than
{account['minimum_balance']} !")
                else:
                    raise Exception("Account is closed !")
            else:
                pass

    def withdraw_funds(self):
        '''
        Withdraw funds from our account
        '''

        for account in ACCOUNTS:
            print(self.parse_accounts(account))

        id = int(input("Enter Account ID: >>> "))

        for account in ACCOUNTS:
            if account.get("id") == id:
                if account.get("status") == "OPEN":
                    amount = float(
                        input("Enter the amount you want to withdraw: >>>
                    float(amount)
```

```
                if self.max_num_of_withdrawals > 0:
                    if account["balance"] - amount >= self.minimum_bal
                        account["balance"] -= amount
                        account["max_num_of_withdrawals"] -= 1
                        self.save_account()
                        print(f"Successfully withdrawed {amount}!")
                        self.start()
                    else:
                        raise Exception("Insufficient funds !")
                else:
                    raise Exception(
                        "You have exhausted your withdrawal tries !")
            else:
                raise Exception("Account does not exist !")
        else:
            pass

def close_account(self):
    for account in ACCOUNTS:
        print(self.parse_accounts(account))

    id = int(input("Enter Account ID: >>> "))

    for account in ACCOUNTS:
        if account.get("id") == id:
            if account.get("status") == "OPEN":
                account["status"] = "CLOSED"
                self.save_account()
                print("Successfully closed the account !")
                self.start()
            else:
                raise Exception("The account is already closed !")
        else:
            pass

def to_json(self) -> dict:
    '''Serializes account object type to json format'''
    if self.status == "OPEN":
        return self.__dict__
    else:
        raise Exception("Account does not exist !")

def __repr__(self) -> str:
    return (
        "Savings("
        f"balance: {self.balance}, "
```

```
            f"minimum_balance: {self.minimum_balance}, "
            f"max_num_of_withdrawals: {self.max_num_of_withdrawals}"
            ")"
        )


def main():
    account = Savings()
    account.start()


if __name__ == "__main__":
    main()
```