**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()   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')}, 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, "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\_balance:  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() |