VICTORIA UNIVERSITY

(NIT5150)

Advanced Object Oriented Programming

Assessment 3:

Final Assignment

Prepared By:

Kok Thong Ong - s8153363

Date: 15th September 2024

Introduction

Objective

In the healthcare sector, the accurate management of patient and procedure data is critical for ensuring high-quality patient care and administrative efficiency. This project aims to develop a Python-based application designed to streamline the management of patient records and medical procedures. The primary objectives are to:

1. **Store and Manage Patient Details**: Implement a system to capture and manage comprehensive patient information, including personal details and emergency contact data.
2. **Record and Track Medical Procedures**: Develop functionality to document medical procedures performed on patients, detailing procedure names, dates, practitioners, and associated costs.
3. **Calculate Total Charges**: Provide tools to calculate and display the total charges for all procedures conducted on a patient, enhancing billing accuracy and financial tracking.

Background

The healthcare industry relies heavily on precise and accessible data management systems to ensure continuity of care and meet regulatory requirements. Managing patient information effectively helps in maintaining up-to-date records, which is essential for both patient care and compliance. Furthermore, tracking medical procedures and their associated costs is vital for accurate billing and comprehensive medical record-keeping.

The proposed system addresses these needs by implementing two core classes in Python: Patient and Procedure. The Patient class is designed to manage and update personal and emergency contact information, while the Procedure class handles details about medical procedures and associated charges. The system also includes functionalities for data input, file management, and data display, allowing for an integrated approach to patient and procedure management.

Case Study 1: Healthcare Records

Class Diagrams

Patient Class Diagram

Patient
- first_name: str - middle_name: str - last_name: str - address: str - city: str - state: str - postal_code: str - contact_number: str - emergency_contact_name: str - emergency_contact_number: str
+ __init__(self, first_name, middle_name, last_name, address, city, state, postal_code, contact_number, emergency_contact_name, emergency_contact_number) + get_full_name(): str + get_address(): str + get_contact_number(): str + get_emergency_contact(): str + set_address(): None + set_emergency_contact(...): None

Procedure Class Diagram

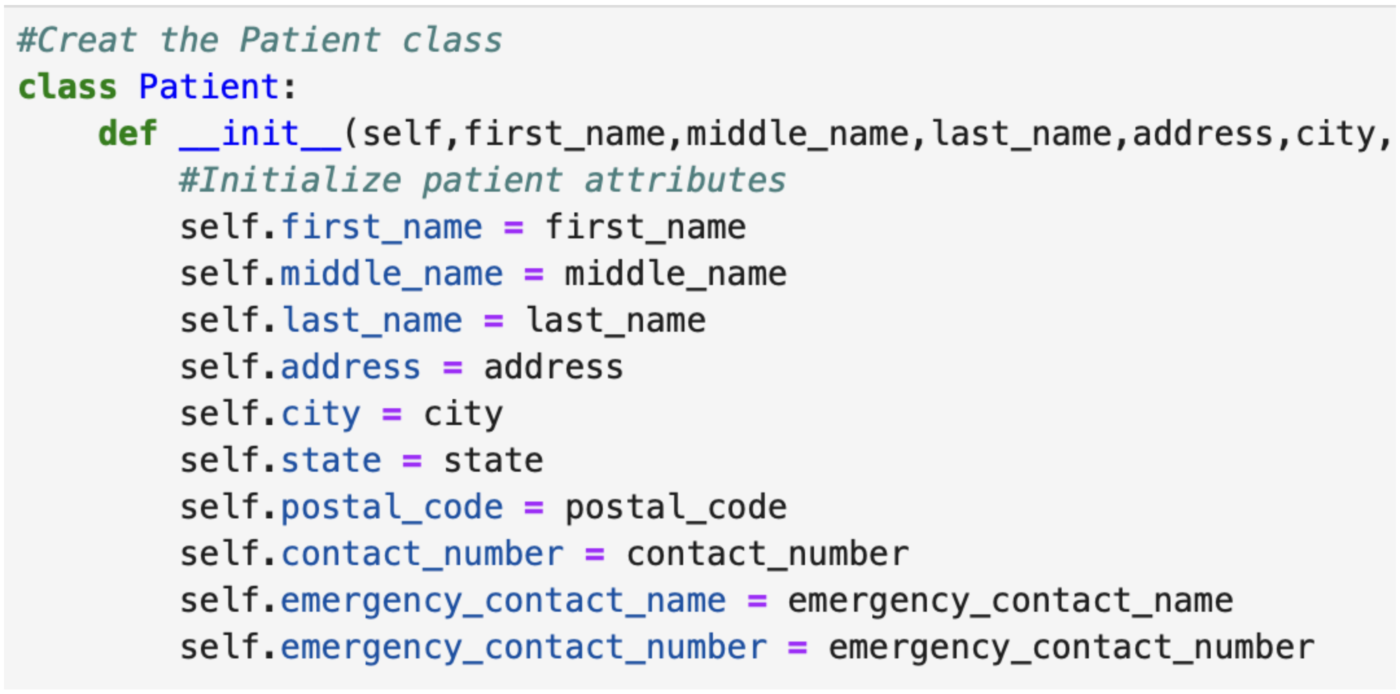
Procedure
- Name: str - Date: str - Practitional: str - Charges: float
+ __init__(self, procedure_name, date, pratitioner, charges) + get_procedure_name(): str + get_date(): str + get_practitioner(): str + get_charges(): float + set_procedure_name(...): None + set_date(...): None + set_practitioner(...): None + set_charges(...): None

Class Diagram Integration

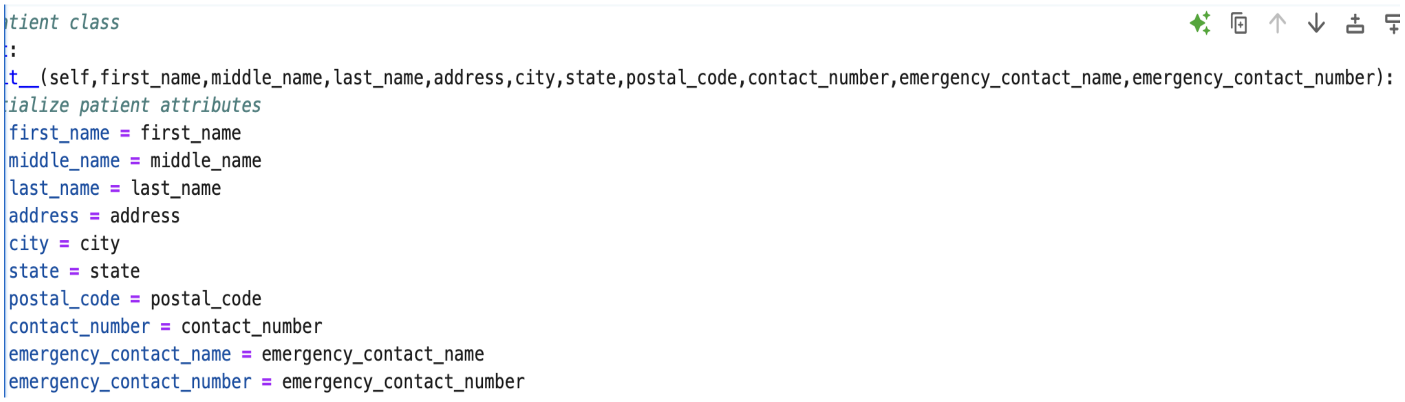
Patient	Procedure
- first_name: str - middle_name: str - last_name: str - address: str - city: str - state: str - postal_code: str - contact_number: str - emergency_contact_name: str - emergency_contact_number: str	- Name: str - Date: str - Practitional: str - Charges: float
+ __init__(self, first_name, middle_name, last_name, address, city, state, postal_code, contact_number, emergency_contact_name, emergency_contact_number) + get_full_name(): str + get_address(): str + get_contact_number(): str + get_emergency_contact(): str + set_address(): None + set_emergency_contact(...): None	+ __init__(self, procedure_name, date, pratitioner, charges) + get_procedure_name(): str + get_date(): str + get_practitioner(): str + get_charges(): float + set_procedure_name(...): None + set_date(...): None + set_practitioner(...): None + set_charges(...): None

Program Implementation

Patient Class



```
#Creat the Patient class
class Patient:
    def __init__(self,first_name,middle_name,last_name,address,city,
        #Initialize patient attributes
        self.first_name = first_name
        self.middle_name = middle_name
        self.last_name = last_name
        self.address = address
        self.city = city
        self.state = state
        self.postal_code = postal_code
        self.contact_number = contact_number
        self.emergency_contact_name = emergency_contact_name
        self.emergency_contact_number = emergency_contact_number
```



```
tient class
:
t__(self,first_name,middle_name,last_name,address,city,state,postal_code,contact_number,emergency_contact_name,emergency_contact_number):
ialize patient attributes
first_name = first_name
middle_name = middle_name
last_name = last_name
address = address
city = city
state = state
postal_code = postal_code
contact_number = contact_number
emergency_contact_name = emergency_contact_name
emergency_contact_number = emergency_contact_number
```

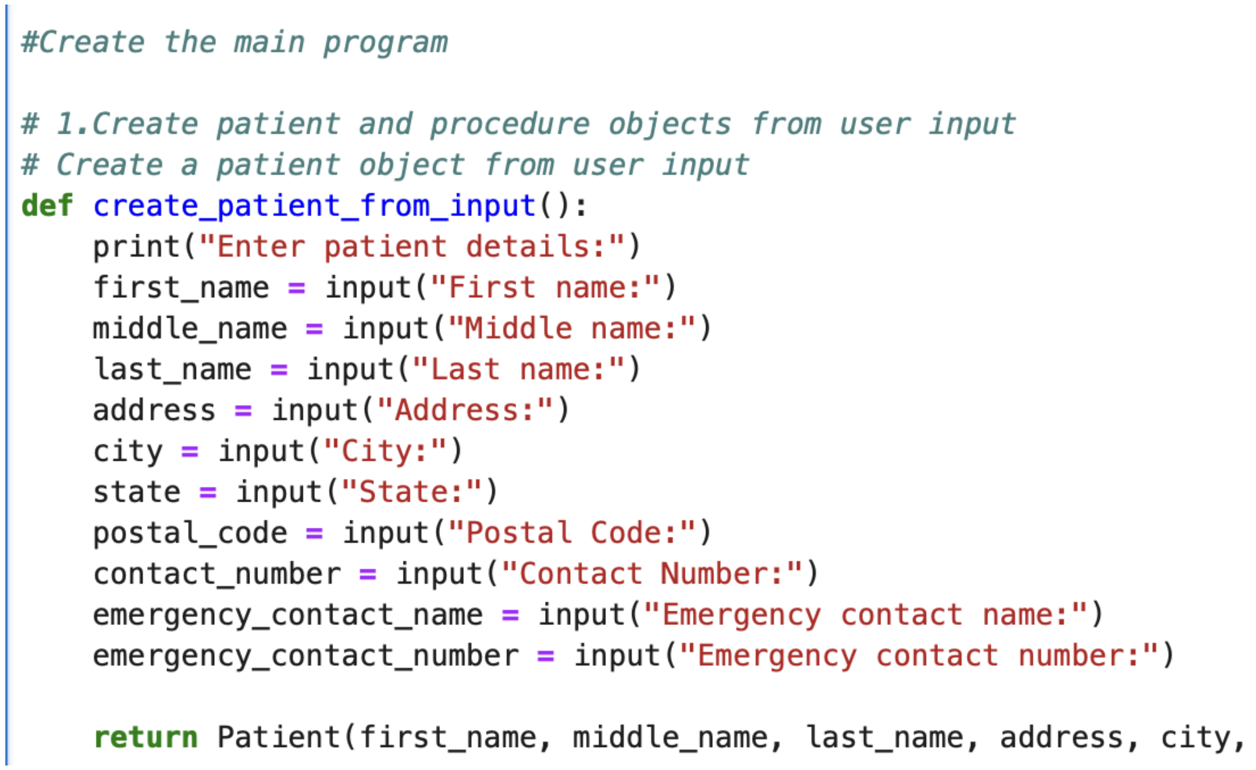
(The picture above is just to show the remaining codes of Patient Class)

```python
# Getter methods for each attribute(Patient class)

    def get_full_name(self):
        return f"{self.first_name} {self.middle_name} {self.last_name}"

    def get_address(self):
        return f"{self.address}, {self.city}, {self.state}, {self.postal_code}"

    def get_contact_number(self):
        return self.contact_number

    def get_emergency_contact(self):
        return f"{self.emergency_contact_name}: {self.emergency_contact_number}"

#Setter methods for each attribute (Patient class)

    def set_address(self,address,city,state,postal_code):
        self.address = address
        self.city = city
        self.state = state
        self.postal_code = postal_code

    def set_emergency_contact(self,name,number):
        self.emergency_contact_name = name
        self.emergency_contact_number = number
```

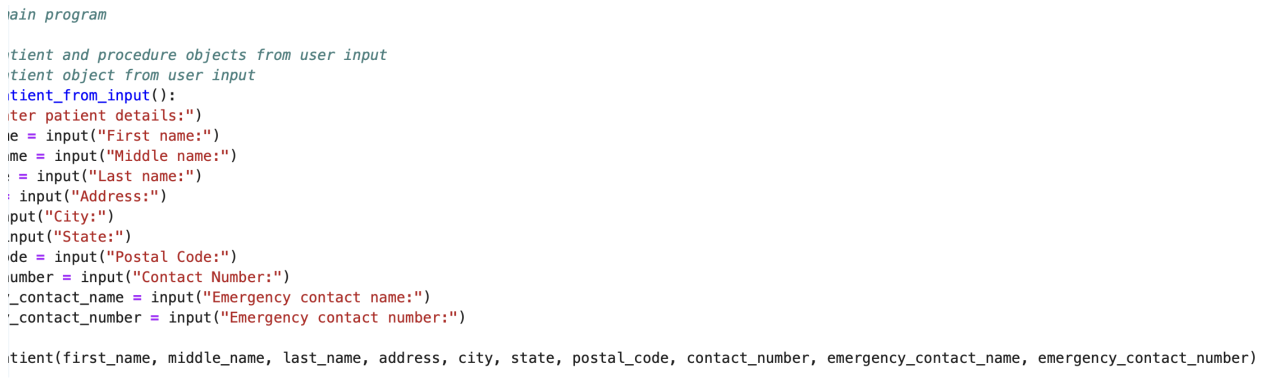
Procedure Class

```
#Create the class for Procedure

class Procedure:
    def __init__(self,procedure_name,date,practitioner,charges):
        self.procedure_name = procedure_name
        self.date = date
        self.practitioner = practitioner
        self.charges = charges

#Getter methods for each attribute(Procedure class)

    def get_procedure_name(self):
        return self.procedure_name

    def get_date(self):
        return self.date

    def get_practitioner(self):
        return self.practitioner

    def get_charges(self):
        return self.charges

#Setter methods for each attribute(Procedure class)

    def set_procedure_name(self,name):
        self.procedure_name = name

    def set_date(self,date):
        self.date = date

    def set_practitioner(self,name):
        self.practitioner = name

    def set_charges(self,charges):
        self.charges = charges
```

Main Program



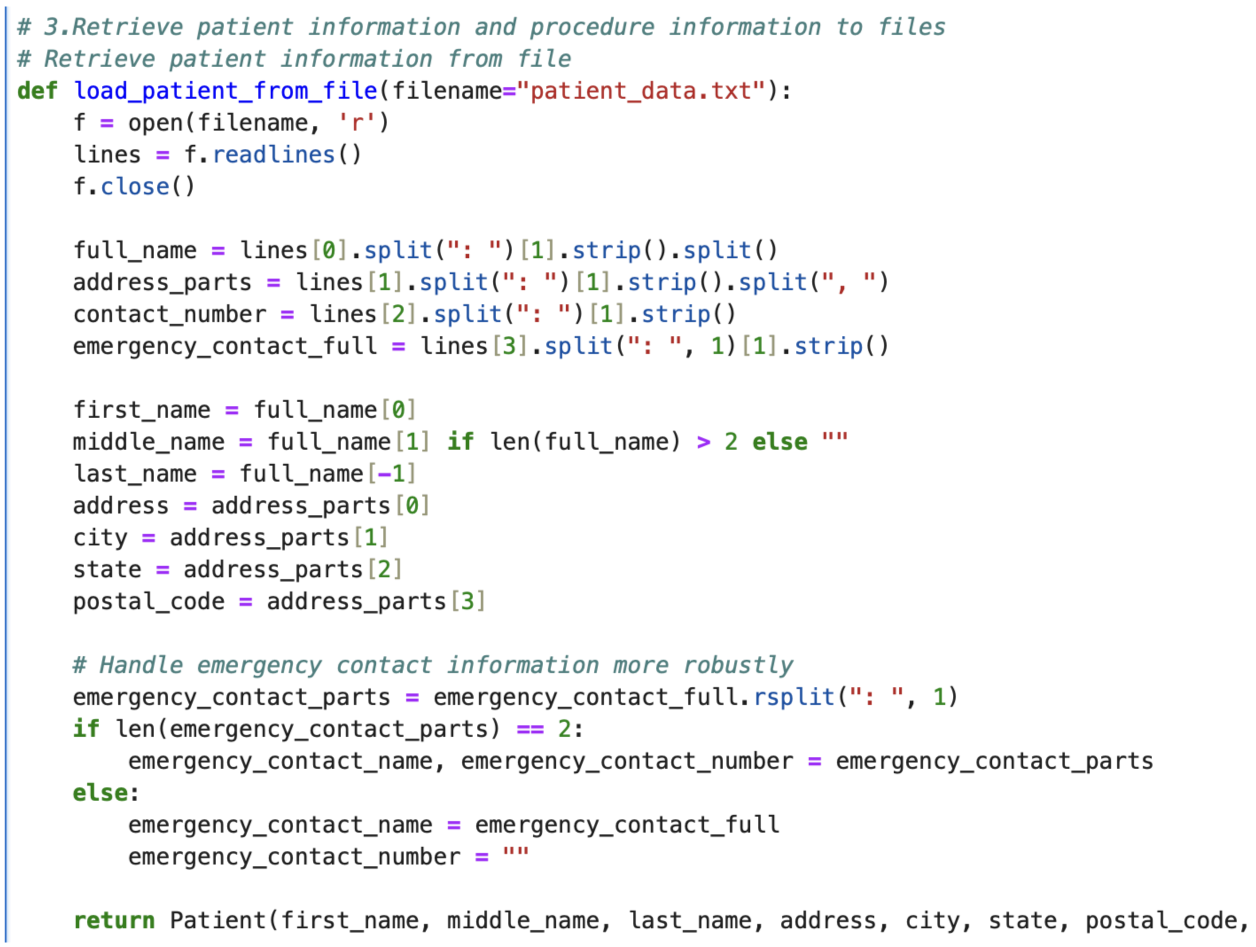
```
#Create the main program

# 1.Create patient and procedure objects from user input
# Create a patient object from user input
def create_patient_from_input():
    print("Enter patient details:")
    first_name = input("First name:")
    middle_name = input("Middle name:")
    last_name = input("Last name:")
    address = input("Address:")
    city = input("City:")
    state = input("State:")
    postal_code = input("Postal Code:")
    contact_number = input("Contact Number:")
    emergency_contact_name = input("Emergency contact name:")
    emergency_contact_number = input("Emergency contact number:")

    return Patient(first_name, middle_name, last_name, address, city,
```



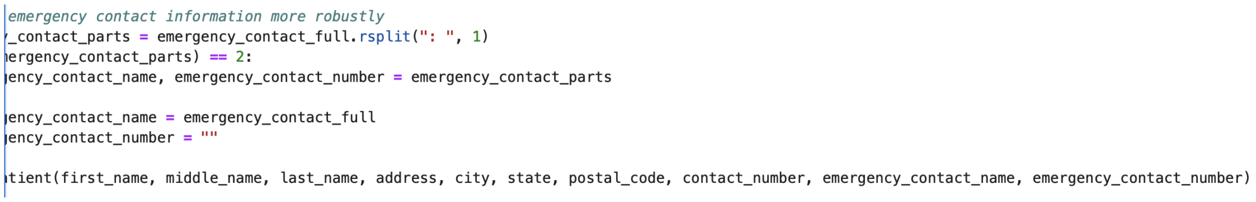
```
ain program

tient and procedure objects from user input
tient object from user input
tient_from_input():
ter patient details:")
e = input("First name:")
me = input("Middle name:")
= input("Last name:")
input("Address:")
put("City:")
nput("State:")
de = input("Postal Code:")
umber = input("Contact Number:")
_contact_name = input("Emergency contact name:")
_contact_number = input("Emergency contact number:")

tient(first_name, middle_name, last_name, address, city, state, postal_code, contact_number, emergency_contact_name, emergency_contact_number)
```

(The picture above is just to show the remaining codes)

```python
# Create procedure objects from user input
def create_procedure_from_input():
    print("Enter procedure details:")
    name = input("Procedure name:")
    date = input("Date:")
    practitioner = input("Practitioner:")
    charges = float(input("Charges:"))

    return Procedure(name, date, practitioner, charges)

# 2.Store patient and procedure information to text files
# Store patient information to text file
def save_patient_to_file(patient, filename="patient_data.txt"):
    f = open(filename, 'w')
    f.write(f"Full Name: {patient.get_full_name()}\n")
    f.write(f"Address: {patient.get_address()}\n")
    f.write(f"Contact Number: {patient.get_contact_number()}\n")
    f.write(f"Emergency Contact: {patient.get_emergency_contact()}\n")
    f.close()
    print(f"Patient data saved to {filename}")

# Store procedure information to text files
def save_procedure_to_file(procedure, filename="procedure.txt"):
    f = open(filename, 'w')
    f.write(f"Name: {procedure.get_procedure_name()}\n")
    f.write(f"Date: {procedure.get_date()}\n")
    f.write(f"Practitioner: {procedure.get_practitioner()}\n")
    f.write(f"Charges: {procedure.get_charges()}\n")
    f.close()
    print(f"Procedure data saved to {filename}")
```



```python
# 3.Retrieve patient information and procedure information to files
# Retrieve patient information from file
def load_patient_from_file(filename="patient_data.txt"):
    f = open(filename, 'r')
    lines = f.readlines()
    f.close()

    full_name = lines[0].split(": ")[1].strip().split()
    address_parts = lines[1].split(": ")[1].strip().split(", ")
    contact_number = lines[2].split(": ")[1].strip()
    emergency_contact_full = lines[3].split(": ", 1)[1].strip()

    first_name = full_name[0]
    middle_name = full_name[1] if len(full_name) > 2 else ""
    last_name = full_name[-1]
    address = address_parts[0]
    city = address_parts[1]
    state = address_parts[2]
    postal_code = address_parts[3]

    # Handle emergency contact information more robustly
    emergency_contact_parts = emergency_contact_full.rsplit(": ", 1)
    if len(emergency_contact_parts) == 2:
        emergency_contact_name, emergency_contact_number = emergency_contact_parts
    else:
        emergency_contact_name = emergency_contact_full
        emergency_contact_number = ""

    return Patient(first_name, middle_name, last_name, address, city, state, postal_code,
```



```python
emergency contact information more robustly
_contact_parts = emergency_contact_full.rsplit(": ", 1)
ergency_contact_parts) == 2:
ency_contact_name, emergency_contact_number = emergency_contact_parts

ency_contact_name = emergency_contact_full
ency_contact_number = ""

tient(first_name, middle_name, last_name, address, city, state, postal_code, contact_number, emergency_contact_name, emergency_contact_number)
```

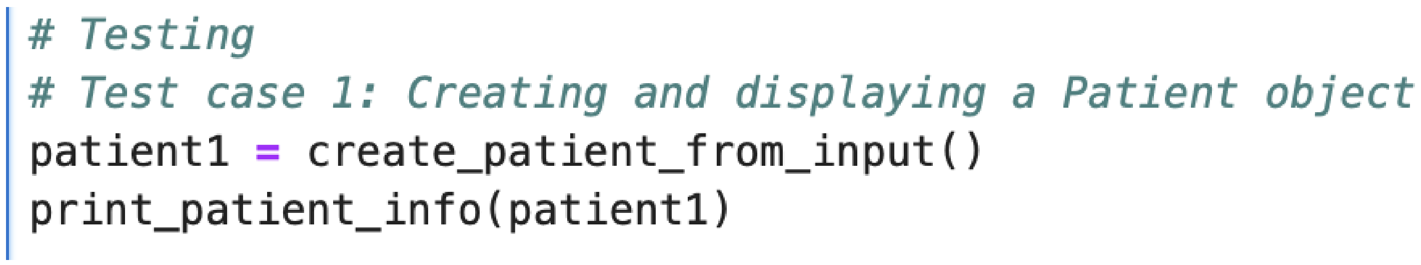
(The picture above is just to show the remaining codes)

```python
#Retrieve procedure information to file
def load_procedure_from_file(filename="procedure.txt"):
    f = open(filename,"r") #Open file for reading mode
    lines = f.readlines()
    f.close() #Close the file

    # Extract information from lines
    name = lines[0].split(": ")[1].strip()
    date = lines[1].split(": ")[1].strip()
    practitioner = lines[2].split(": ")[1].strip()
    charges = float(lines[3].split(": ")[1].strip())

    return Procedure(name, date, practitioner, charges)

# 4.Print patient and procedure information
# Function to print patient information
def print_patient_info(patient):
    print("Patient Information:")
    print(f"Full Name: {patient.get_full_name()}")
    print(f"Address: {patient.get_address()}")
    print(f"Contact Number: {patient.get_contact_number()}")
    print(f"Emergency Contact: {patient.get_emergency_contact()}")


# Function to print procedure information
def print_procedure_info(procedure):
    print("Procedure Information:")
    print(f"Name: {procedure.get_procedure_name()}")
    print(f"Date: {procedure.get_date()}")
    print(f"Practitioner: {procedure.get_practitioner()}")
    print(f"Charges: ${procedure.get_charges():.2f}")
```

```python
# 5.Print procedures taken by specific patient
def print_procedures(procedures):
    print("\nProcedures for this patient:")
    i = 1  # Display starts with Procedure 1 instead of 0
    for procedure in procedures:
        print(f"\nProcedure {i}:")
        print(f"Name: {procedure.get_procedure_name()}")
        print(f"Date: {procedure.get_date()}")
        print(f"Practitioner: {procedure.get_practitioner()}")
        print(f"Charges: ${procedure.get_charges():.2f}")
        i = i + 1  # Increment of number counting for procedures

# 6. Calculate and print total charges of procedures
# Function to calculate and print total charges for procedures
def calculate_and_print_total_charges(procedures):
    total_charges = 0  # Charge the patient from 0 buck

    for procedure in procedures: #forloop all the charges from a specific patient
        total_charges = total_charges + procedure.get_charges()
        #Add charges one by one from beginning to the end

    # Print the total charges in two decimal places
    print(f"\nTotal Charges: ${total_charges:.2f}")
```

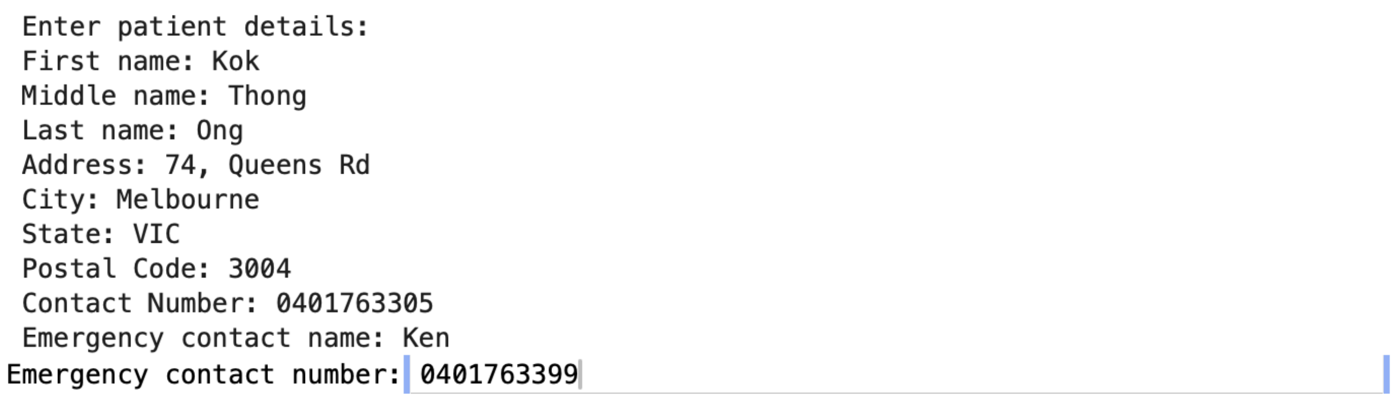
Testing

Test Case 1: Creating and displaying a Patient object



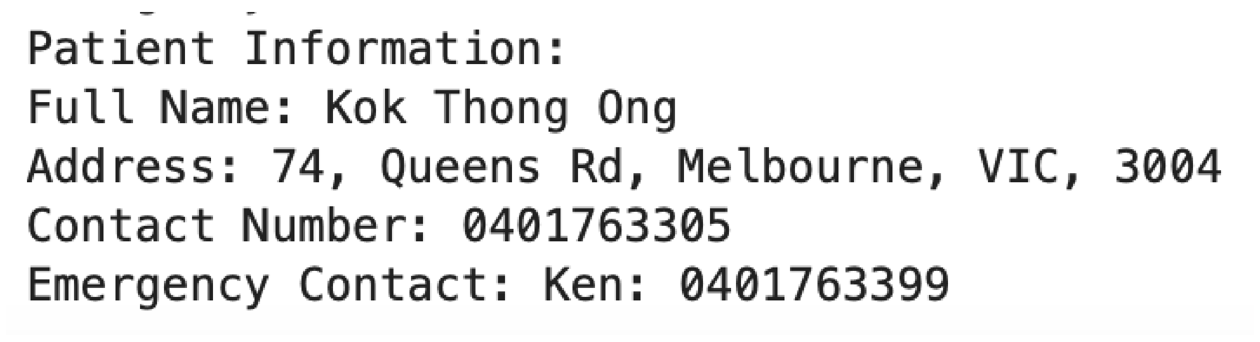
```
# Testing
# Test case 1: Creating and displaying a Patient object
patient1 = create_patient_from_input()
print_patient_info(patient1)
```

Input



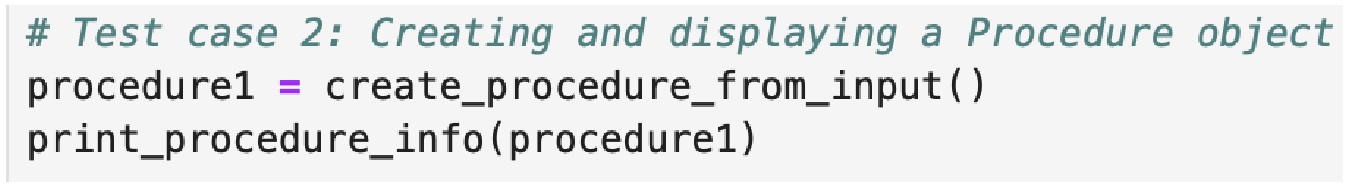
```
 Enter patient details:
 First name: Kok
 Middle name: Thong
 Last name: Ong
 Address: 74, Queens Rd
 City: Melbourne
 State: VIC
 Postal Code: 3004
 Contact Number: 0401763305
 Emergency contact name: Ken
Emergency contact number: 0401763399
```

Output



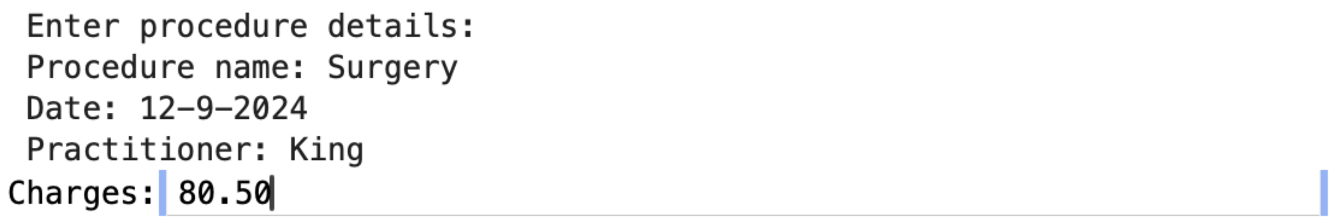
```
Patient Information:
Full Name: Kok Thong Ong
Address: 74, Queens Rd, Melbourne, VIC, 3004
Contact Number: 0401763305
Emergency Contact: Ken: 0401763399
```

Test Case 2: Creating and displaying Procedure objects



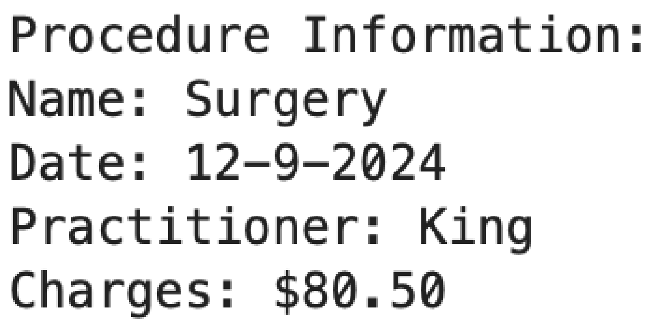
```
# Test case 2: Creating and displaying a Procedure object
procedure1 = create_procedure_from_input()
print_procedure_info(procedure1)
```

Input



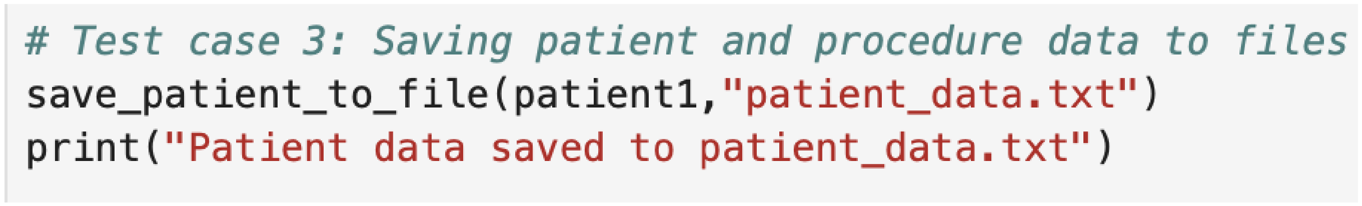
```
 Enter procedure details:
 Procedure name: Surgery
 Date: 12-9-2024
 Practitioner: King
Charges: 80.50
```

Output



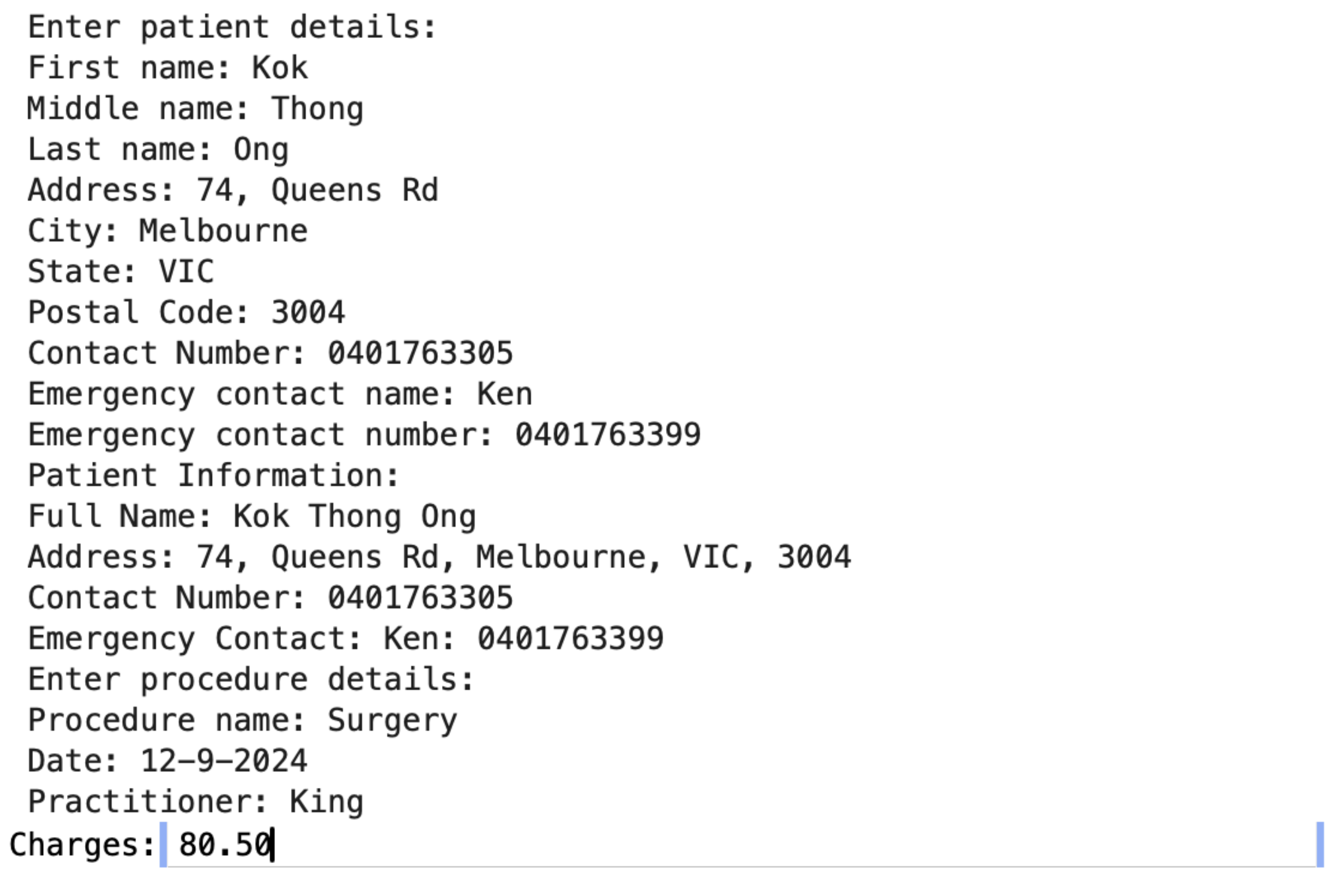
```
Procedure Information:
Name: Surgery
Date: 12-9-2024
Practitioner: King
Charges: $80.50
```

Test Case 3: Saving patient and procedure data to files



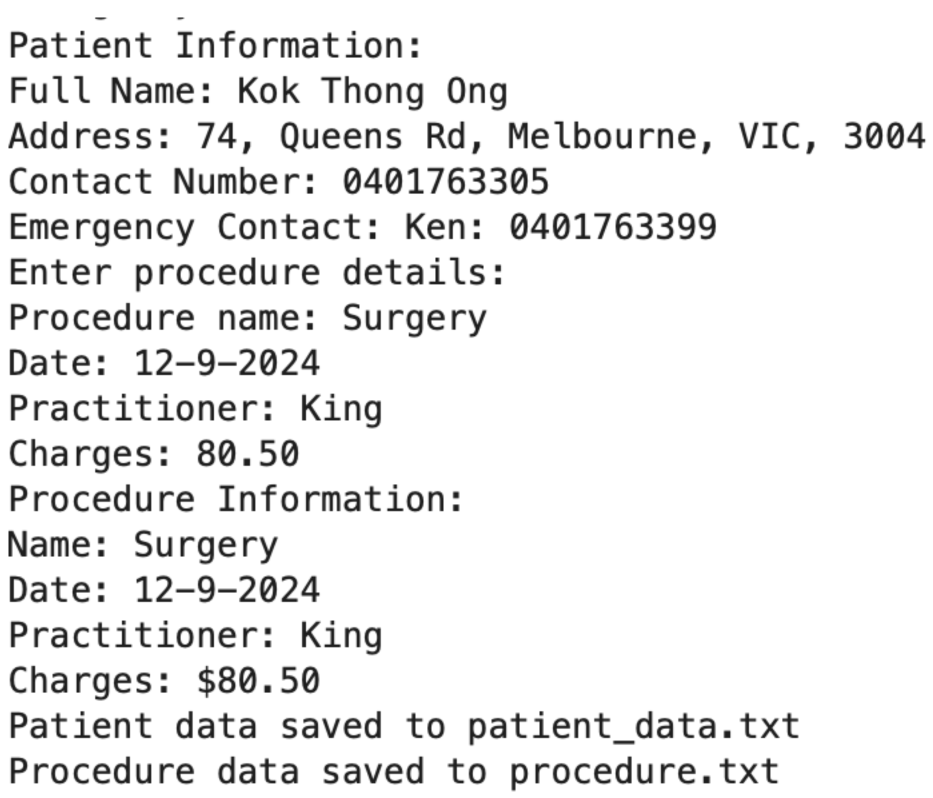
```python
# Test case 3: Saving patient and procedure data to files
save_patient_to_file(patient1,"patient_data.txt")
print("Patient data saved to patient_data.txt")
```

Input



```
 Enter patient details:
 First name: Kok
 Middle name: Thong
 Last name: Ong
 Address: 74, Queens Rd
 City: Melbourne
 State: VIC
 Postal Code: 3004
 Contact Number: 0401763305
 Emergency contact name: Ken
 Emergency contact number: 0401763399
 Patient Information:
 Full Name: Kok Thong Ong
 Address: 74, Queens Rd, Melbourne, VIC, 3004
 Contact Number: 0401763305
 Emergency Contact: Ken: 0401763399
 Enter procedure details:
 Procedure name: Surgery
 Date: 12-9-2024
 Practitioner: King
Charges: 80.50
```

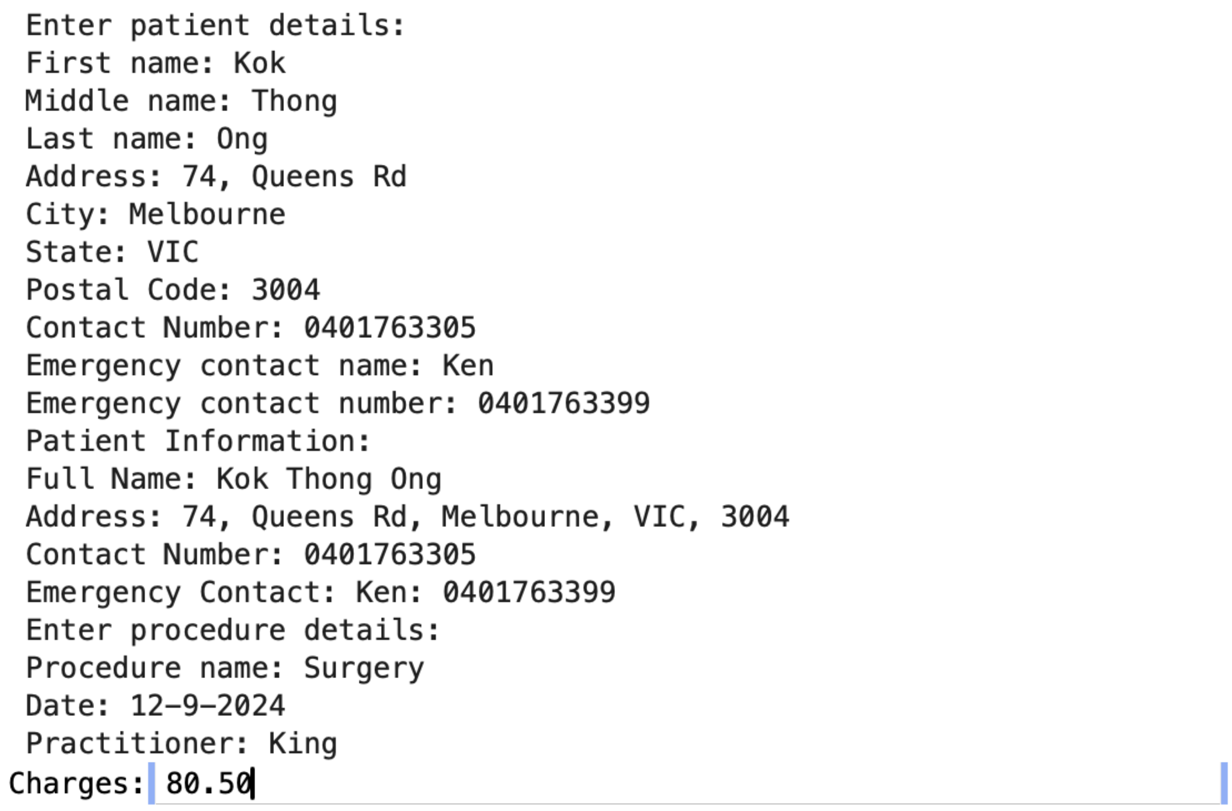
Output



```
Patient Information:
Full Name: Kok Thong Ong
Address: 74, Queens Rd, Melbourne, VIC, 3004
Contact Number: 0401763305
Emergency Contact: Ken: 0401763399
Enter procedure details:
Procedure name: Surgery
Date: 12-9-2024
Practitioner: King
Charges: 80.50
Procedure Information:
Name: Surgery
Date: 12-9-2024
Practitioner: King
Charges: $80.50
Patient data saved to patient_data.txt
Procedure data saved to procedure.txt
```

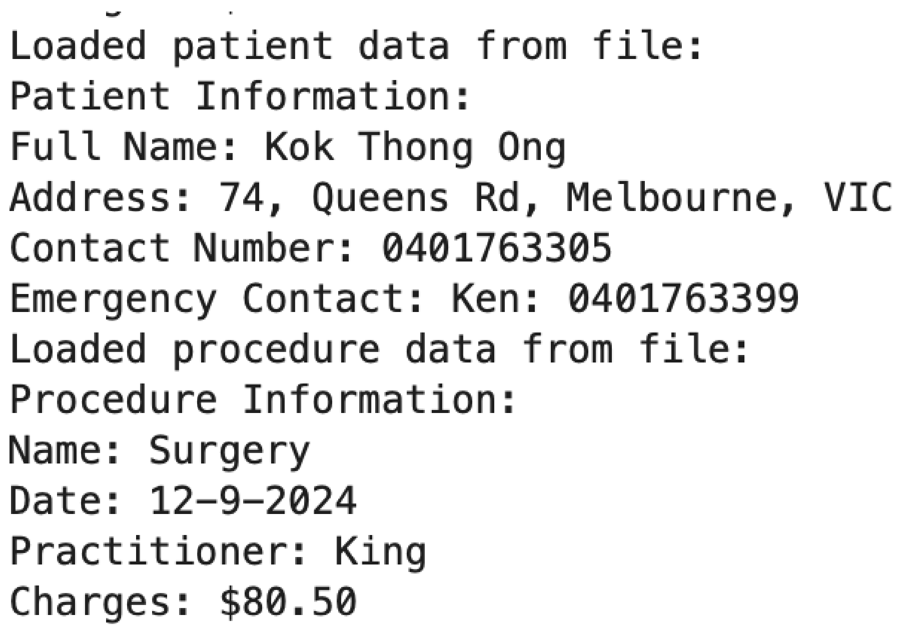
Test Case 4: Loading patient and procedure data from files

Input



```
 Enter patient details:
 First name: Kok
 Middle name: Thong
 Last name: Ong
 Address: 74, Queens Rd
 City: Melbourne
 State: VIC
 Postal Code: 3004
 Contact Number: 0401763305
 Emergency contact name: Ken
 Emergency contact number: 0401763399
 Patient Information:
 Full Name: Kok Thong Ong
 Address: 74, Queens Rd, Melbourne, VIC, 3004
 Contact Number: 0401763305
 Emergency Contact: Ken: 0401763399
 Enter procedure details:
 Procedure name: Surgery
 Date: 12-9-2024
 Practitioner: King
Charges: 80.50
```

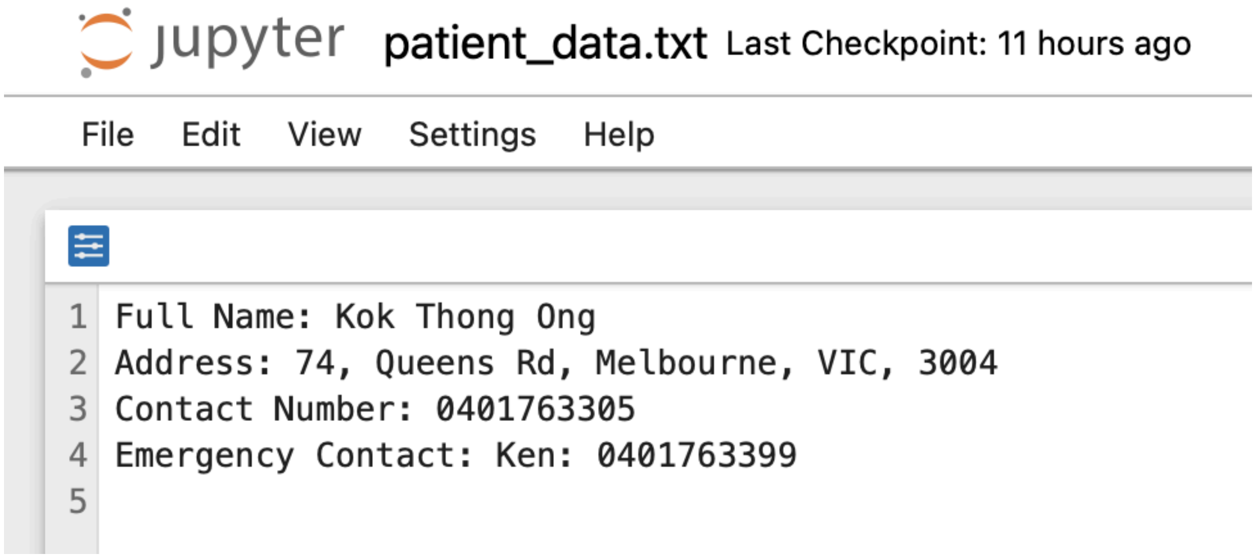
Output(Jupyter Notebook)



```
Loaded patient data from file:
Patient Information:
Full Name: Kok Thong Ong
Address: 74, Queens Rd, Melbourne, VIC
Contact Number: 0401763305
Emergency Contact: Ken: 0401763399
Loaded procedure data from file:
Procedure Information:
Name: Surgery
Date: 12-9-2024
Practitioner: King
Charges: $80.50
```

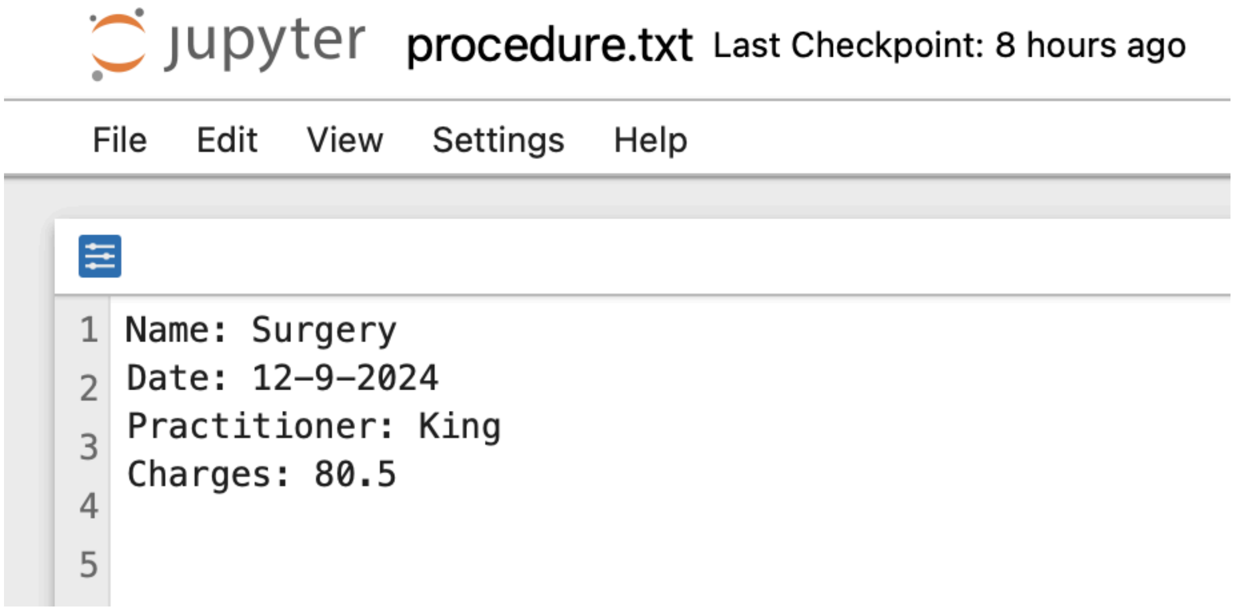
Output(Files)

Patient_data.txt


jupyter
patient_data.txt Last Checkpoint: 11 hours ago
File Edit View Settings Help

```
Full Name: Kok Thong Ong
Address: 74, Queens Rd, Melbourne, VIC, 3004
Contact Number: 0401763305
Emergency Contact: Ken: 0401763399

```

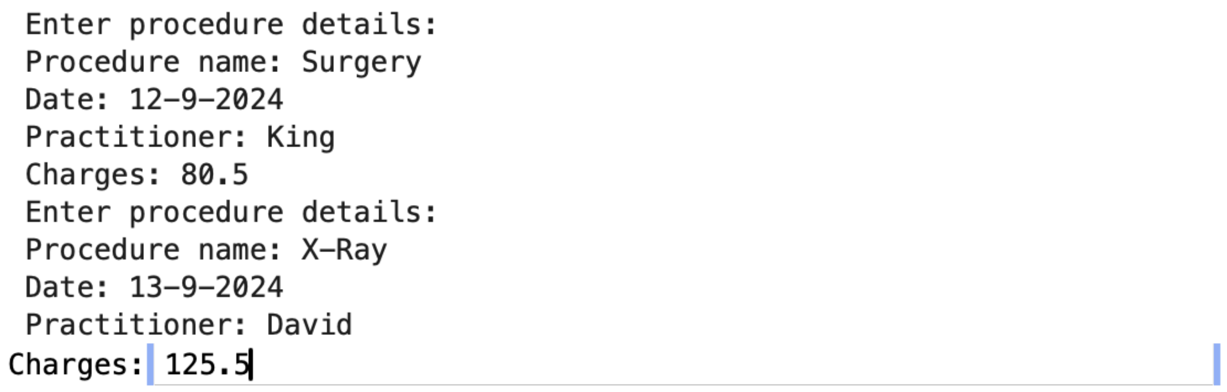
Procedure.txt


jupyter
procedure.txt Last Checkpoint: 8 hours ago
File Edit View Settings Help

```
Name: Surgery
Date: 12-9-2024
Practitioner: King
Charges: 80.5

```

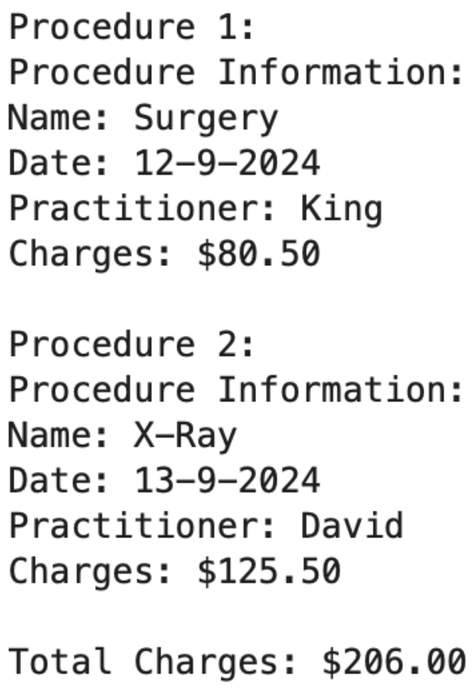
Test Case 5: Calculating and displaying total charges

Input



```
 Enter procedure details:
 Procedure name: Surgery
 Date: 12-9-2024
 Practitioner: King
 Charges: 80.5
 Enter procedure details:
 Procedure name: X-Ray
 Date: 13-9-2024
 Practitioner: David
Charges: 125.5
```

Output



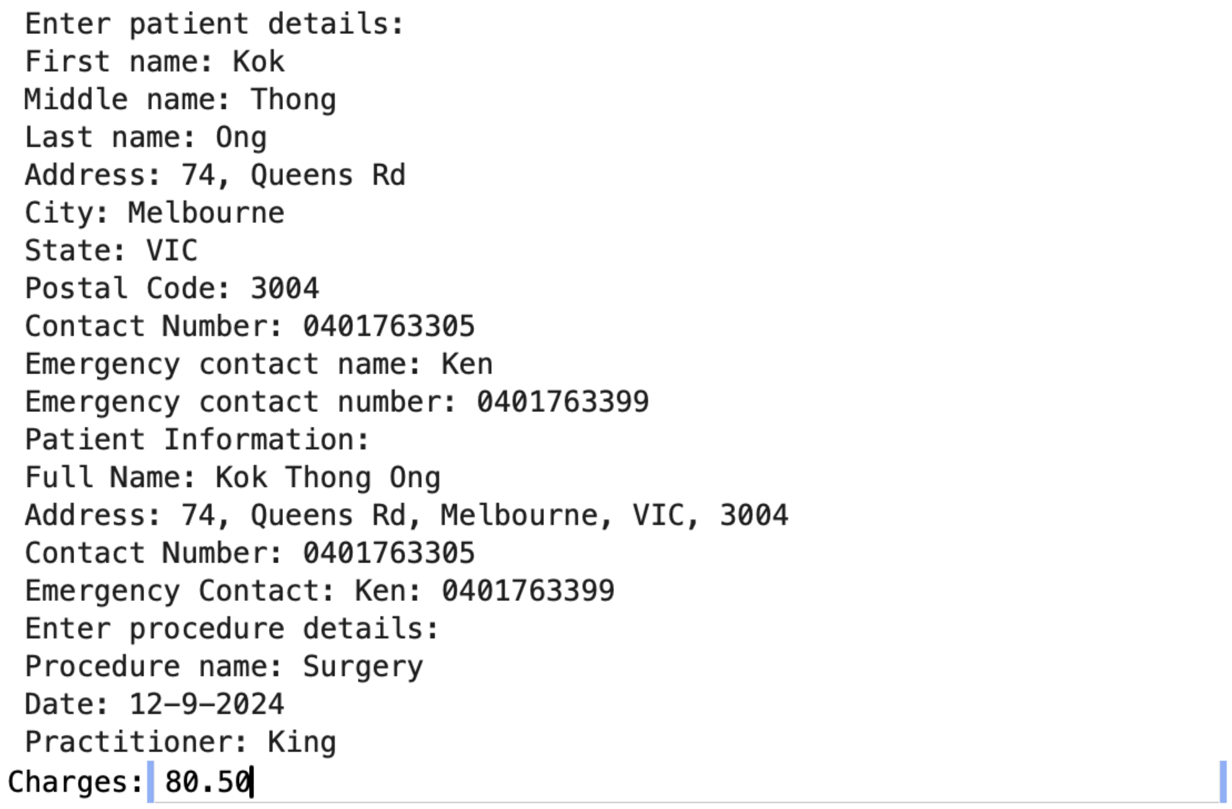
```
Procedure 1:
Procedure Information:
Name: Surgery
Date: 12-9-2024
Practitioner: King
Charges: $80.50

Procedure 2:
Procedure Information:
Name: X-Ray
Date: 13-9-2024
Practitioner: David
Charges: $125.50

Total Charges: $206.00
```

User Guide

Instructions for Running the Program

1. **Execution**:
   - Open your Jupyter Notebook or Python environment where your code is saved.
   - Ensure that all necessary code cells are executed in order, starting from the top of the notebook.

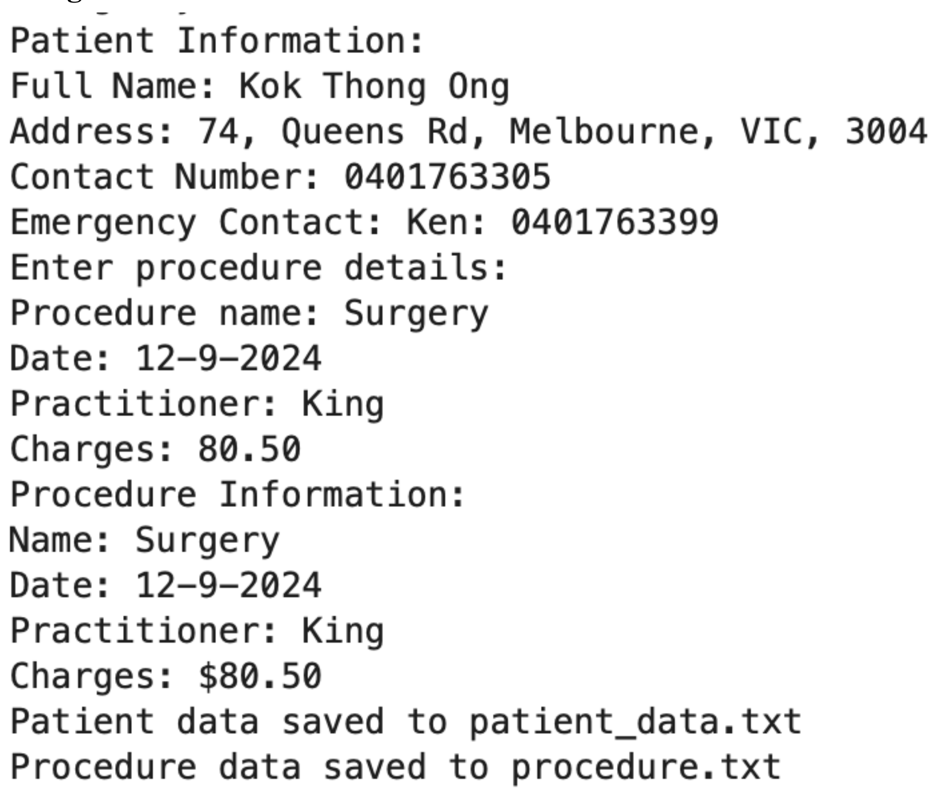
2. **Input Data**:



```
Enter patient details:
First name: Kok
Middle name: Thong
Last name: Ong
Address: 74, Queens Rd
City: Melbourne
State: VIC
Postal Code: 3004
Contact Number: 0401763305
Emergency contact name: Ken
Emergency contact number: 0401763399
Patient Information:
Full Name: Kok Thong Ong
Address: 74, Queens Rd, Melbourne, VIC, 3004
Contact Number: 0401763305
Emergency Contact: Ken: 0401763399
Enter procedure details:
Procedure name: Surgery
Date: 12-9-2024
Practitioner: King
Charges: 80.50
```

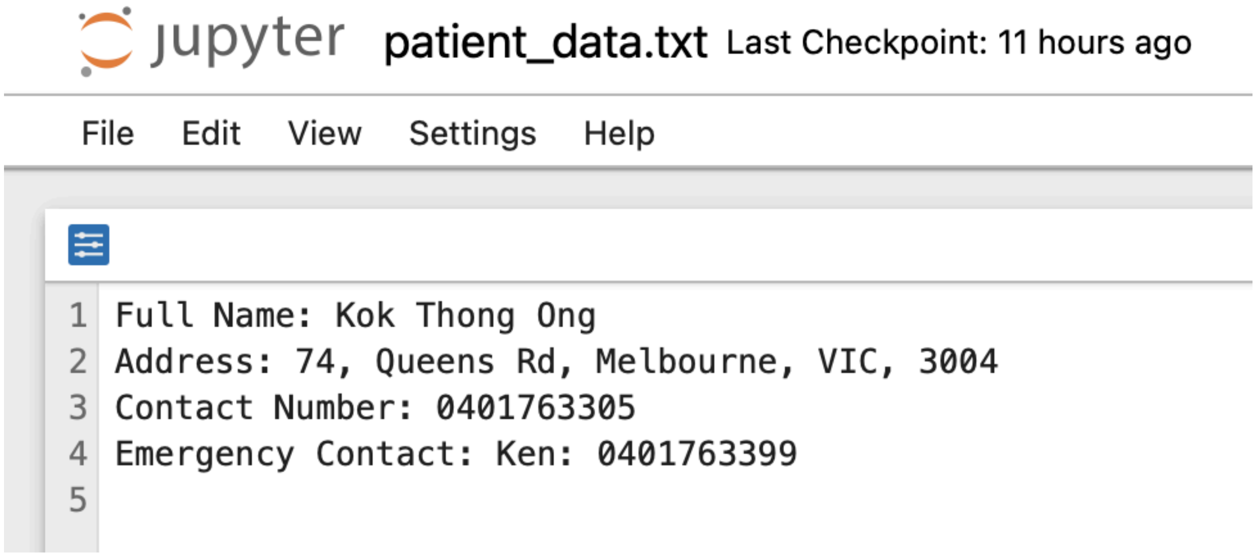
   - When prompted by the program, enter the required information for patients and procedures. This includes names, dates, practitioners, and charges.
   - Input values when the program requests them, such as "Procedure name:", "Date:", "Practitioner:", and "Charges:".

3. **Saving Data to Files**:

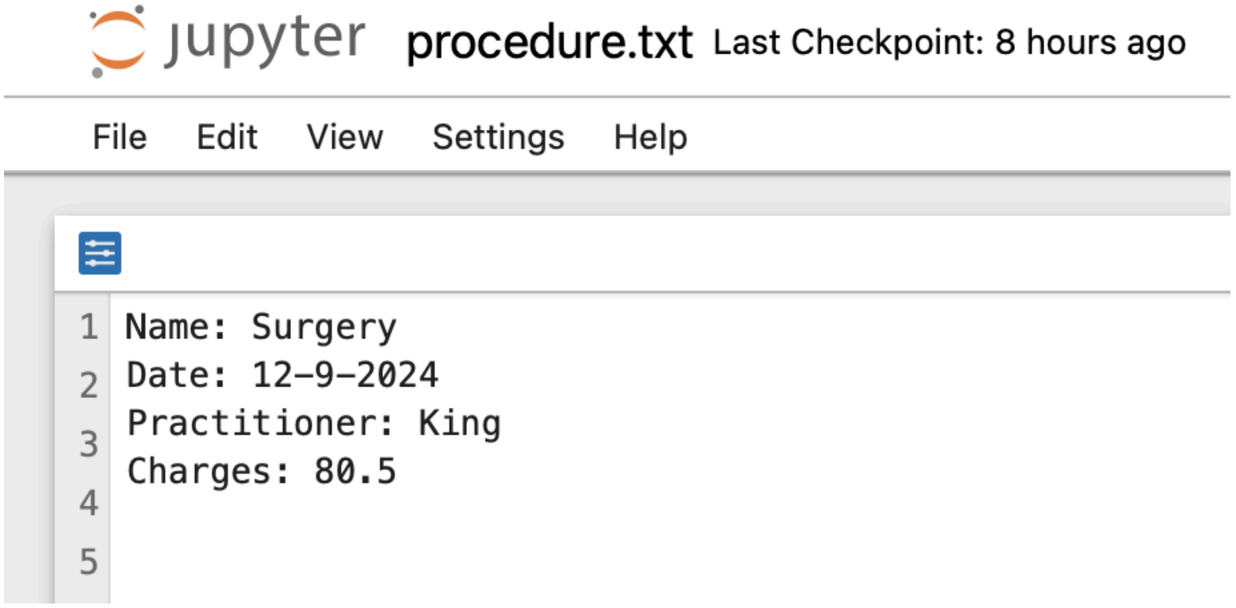

```
Patient Information:
Full Name: Kok Thong Ong
Address: 74, Queens Rd, Melbourne, VIC, 3004
Contact Number: 0401763305
Emergency Contact: Ken: 0401763399
Enter procedure details:
Procedure name: Surgery
Date: 12-9-2024
Practitioner: King
Charges: 80.50
Procedure Information:
Name: Surgery
Date: 12-9-2024
Practitioner: King
Charges: $80.50
Patient data saved to patient_data.txt
Procedure data saved to procedure.txt
```

- The program will automatically save the patient and procedure data to text files named patient_data.txt and procedure.txt respectively.
- Ensure that you have write permissions in the directory where the program is running.

4. **Loading Data from Files**:



```
Full Name: Kok Thong Ong
Address: 74, Queens Rd, Melbourne, VIC, 3004
Contact Number: 0401763305
Emergency Contact: Ken: 0401763399
```



```
Name: Surgery
Date: 12-9-2024
Practitioner: King
Charges: 80.5
```

- To test loading data, ensure that patient_data.txt and procedure.txt contain valid data.
- The program will read these files and reconstruct the patient and procedure objects. This is done using the load_patient_from_file and load_procedure_from_file functions.

Design Decision

Rational for Class Design

Patient Class:

- **Attributes:**
  - **First Name, Middle Name, Last Name:** Captures full names in parts to allow for flexible name formatting and searching.
  - **Address:** Stored as a single string to keep all address details together, simplifying data management.
  - **Contact Number:** Includes fields for phone number and email, which are essential for communication and record-keeping.
  - **Emergency Name and Contact:** Includes the emergency contact's name and phone number, which are crucial for situations where immediate contact is necessary.

  Rationale: These attributes were chosen to cover all necessary aspects of patient information, ensuring comprehensive record-keeping and facilitating efficient data retrieval and management. The inclusion of emergency contact information helps in promptly addressing urgent situations where quick communication with a designated person is needed.

- **Methods:**
  - *Getters and Setters:* These methods provide controlled access to the patient's attributes, ensuring data integrity and allowing validation checks before modifying attributes.
  - *Example:* get_full_name() method combines first, middle, and last names into a single string for easy display.

  Rationale: Methods were included to encapsulate patient data, provide easy access and modification, and maintain data consistency.

Procedure Class:

- **Attributes:**
  - *Procedure Name:* Identifies the procedure performed.
  - *Date:* Records when the procedure took place.
  - *Practitioner:* Stores the name of the person performing the procedure.
  - *Charges:* Indicates the cost associated with the procedure.

Rationale: These attributes are crucial for tracking and managing medical procedures, providing a clear record of each procedure's details and costs.

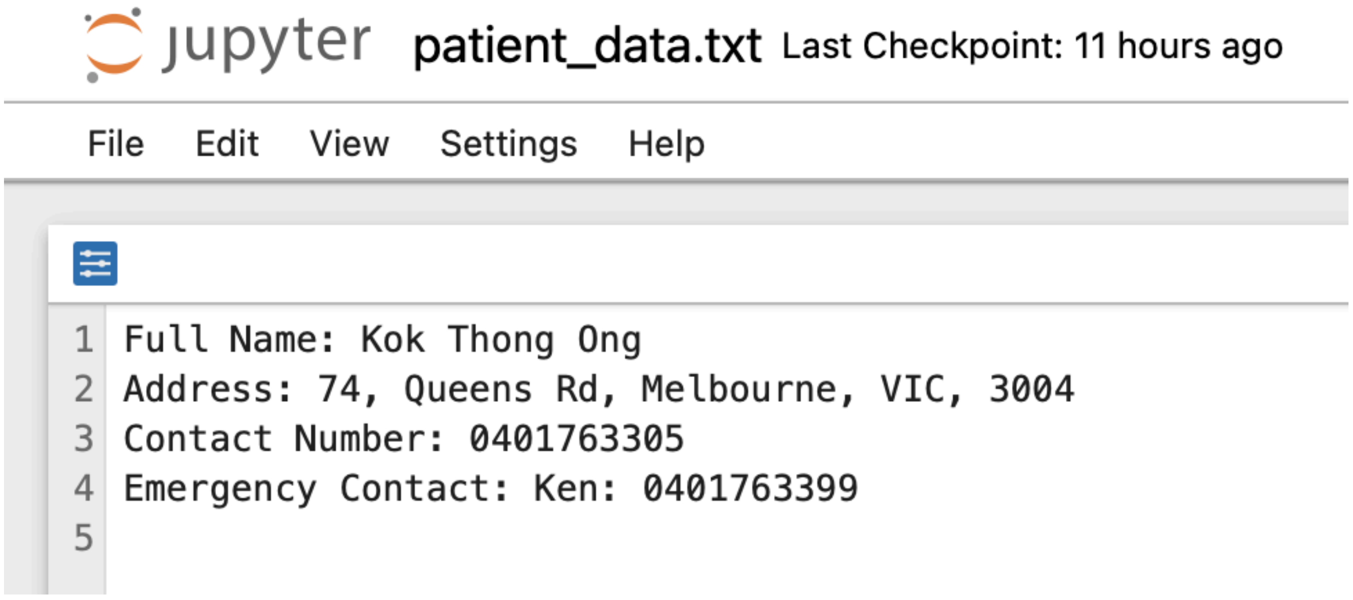
- **Methods:**
  - *Getters and Setters:* These methods ensure that procedure details are accessed and modified correctly, with validation as needed.
  - *Example***:** The set_charges() method updates the cost of the procedure, allowing for a new value to be set and stored in the charges attribute. This method ensures that the procedure's cost can be modified as needed.

Rationale: Methods in this class help manage and display procedure data accurately and effectively.

File Format

Patient File:

- **Format:** Plain text file with each patient's details organized with labels.

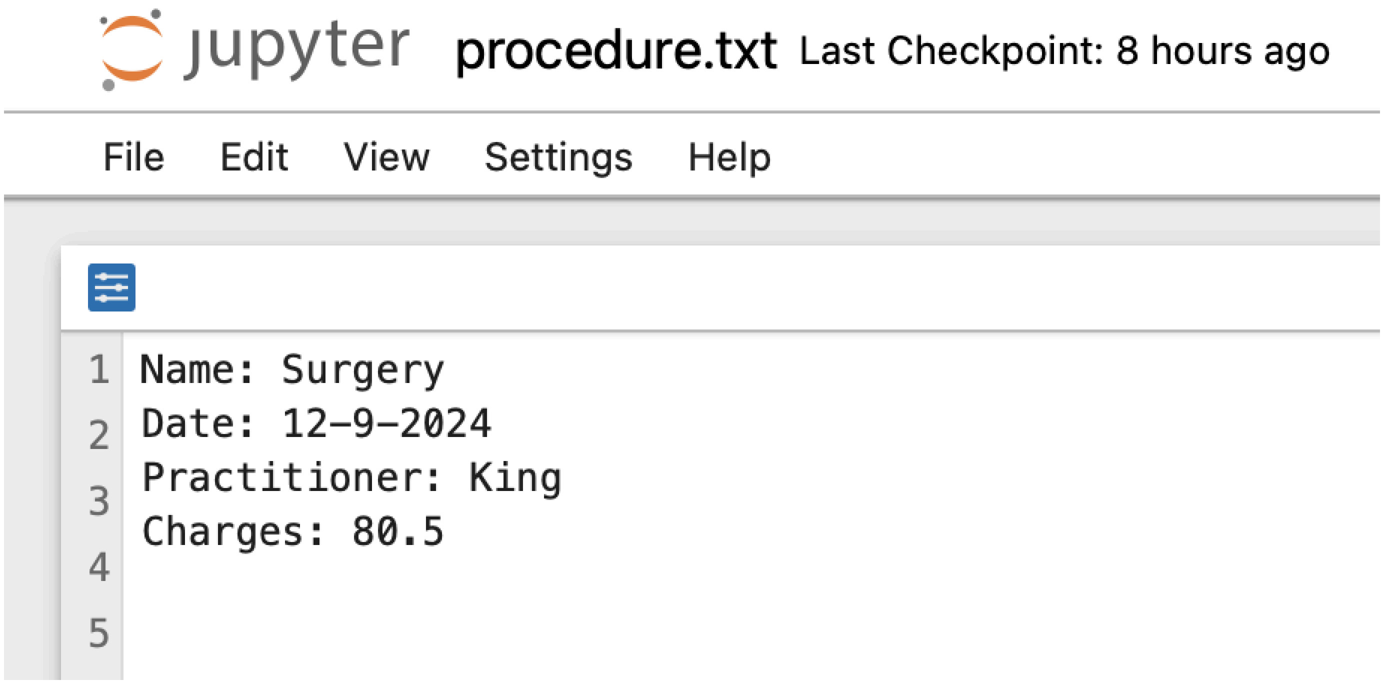


```
Full Name: Kok Thong Ong
Address: 74, Queens Rd, Melbourne, VIC, 3004
Contact Number: 0401763305
Emergency Contact: Ken: 0401763399

```

Description: Each piece of information is stored on a separate line with a descriptive label for easy reading and writing.

Procedure File:

- **Format:** Plain text file with each procedure's details organized similarly.



```
Name: Surgery
Date: 12-9-2024
Practitioner: King
Charges: 80.5
```

Description: The file format mirrors the patient file structure, using labels for clarity.

Appendices

- **Complete sources code:** README file has been submitted together to the dropbox.
- **Sample Outputs:** All related outputs are shown in the section above.
- **Class Diagrams:** All UML diagrams are provided in the sectin above.

Case Study 2: Employee and Customer Management System

Class Definitions

Employee Class with getter, setter methods

```python
# Create Employee Class (Parent Class)
class Employee:
    def __init__(self, name, emp_number): # Constructor
        self.name = name
        self.emp_number = emp_number

    # Getter methods for Employee Class
    def get_name(self):
        return self.name

    def get_emp_number(self):
        return self.emp_number

    # Setter methods for Employee Class
    def set_name(self, name):
        self.name = name

    def set_emp_number(self, emp_number):
        self.emp_number = emp_number

    # Display info for Employee Class
    def display_info(self):
        return f"Employee: {self.name}, ID: {self.emp_number}"
```

ProductionWorker Class (Child class of Employee class)

```python
# Create ProductionWorker Class
class ProductionWorker(Employee): #Child class of Employee Class
    def __init__(self, name, emp_number, shift_number, hourly_pay_rate):
        super().__init__(name, emp_number)
        self.shift_number = shift_number
        self.hourly_pay_rate = hourly_pay_rate

    # Getter methods for ProductionWorker Class
    def get_shift_number(self):
        return self.shift_number

    def get_hourly_pay_rate(self):
        return self.hourly_pay_rate

    # Setter methods for ProductionWorker Class
    def set_shift_number(self):
        self.shift_number = shift_number

    def set_hourly_pay_rate(self):
        self.hourly_pay_rate

    # Method overriding
    def display_info(self):
        return f"{super().display_info()}, Shift: {self.shift_number}, Hourly Rate: ${self.hourly_pay_rate:.2f} "
```

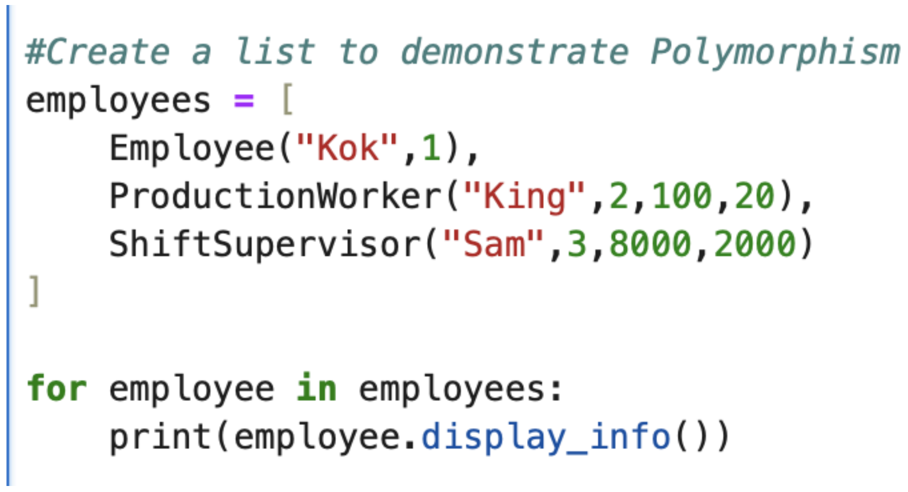
The codes above show i**nheritence relationships** between Employee (parent class) and ProductionWorker (child class). It also shows **method overriding** in the display_info() method.

Create ShiftSupervisor Class (Child class of Employee class)

```python
# Create Shiftsupervisor Class
class ShiftSupervisor(Employee): #Child Class of Employee Class
    def __init__(self, name, emp_number, annual_salary, annual_bonus):
        super().__init__(name, emp_number)
        self.annual_salary = annual_salary
        self.annual_bonus = annual_bonus

    #Getter methods
    def get_annual_salary(self):
        return self.annual_salary

    def get_annual_bonus(self):
        return self.annual_bonus

    # Method overriding
    def display_info(self):
        return f"{super().display_info()}, Annual Salary: ${self.annual_salary}, Annual Bonus: ${self.annual_bonus}"
```

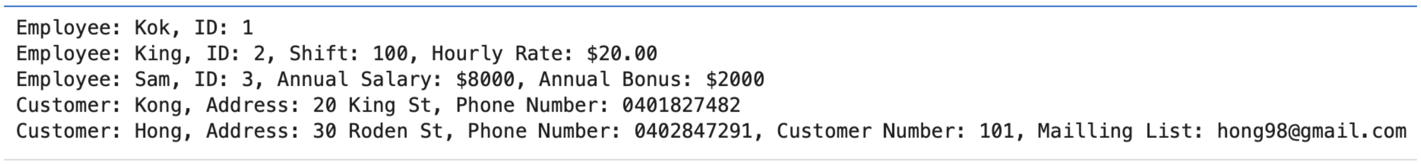
The codes above show **inheritence relationships** between Employee (parent class) and Shiftsupervisor (child class). It also shows **method overriding** in the display_info() method.

Implement Polymorphism



```python
#Create a list to demonstrate Polymorphism
employees = [
    Employee("Kok",1),
    ProductionWorker("King",2,100,20),
    ShiftSupervisor("Sam",3,8000,2000)
]

for employee in employees:
    print(employee.display_info())
```

Output



```
Employee: Kok, ID: 1
Employee: King, ID: 2, Shift: 100, Hourly Rate: $20.00
Employee: Sam, ID: 3, Annual Salary: $8000, Annual Bonus: $2000
Customer: Kong, Address: 20 King St, Phone Number: 0401827482
Customer: Hong, Address: 30 Roden St, Phone Number: 0402847291, Customer Number: 101, Mailling List: hong98@gmail.com
```

Polymorphism is demonstrated by using a base class reference to call methods of subclasses. A list of Employee objects that includes both ProductionWorker and ShiftSupervisor is created, and use base class methods to interact with them.

Programs

ProductionWorker Program

```
# Create the Main Program
#ProductionWorker Program
def production_worker_program():
    name = input("Enter production worker's name:")
    emp_number = input("Enter employee number:")
    shift_number = int(input("Enter shift number(1 for day, 2 for night):"))
    hourly_pay_rate = float(input("Enter hourly pay rate:"))

     #Create an instance for ProductionWorker Program
    worker = ProductionWorker(name, emp_number, shift_number, hourly_pay_rate)

    print("\n Production Worker Information:")
    print(f"Name: {worker.get_name()}")
    print(f"Employee Number: {worker.get_emp_number()}")
    print(f"Shift: {worker.get_shift_number()}")
    print(f"Hourly Pay Rate: ${worker.get_hourly_pay_rate()}")

production_worker_program()
```

Description: The ProductionWorker program is designed to create and display information about a production worker. The ProductionWorker class is a subclass of Employee and includes additional attributes for shift number and hourly pay rate.

ShiftSupervisor program

```python
#ShiftSuperviror Program
def shift_supervisor_program():
    name = input("Enter shift supervisor's name:")
    emp_number = input("Enter employee number")
    annual_salary = float(input("Enter annual salary:"))
    annual_bonus = float(input("Enter annual bonus:"))

    supervisor = ShiftSupervisor(name, emp_number, annual_salary, annual_bonus)

    print("\n Shift Supervisor Information")
    print(f"Name: {supervisor.get_name()}")
    print(f"Employee Number: {supervisor.get_emp_number()}")
    print(f"Annual Salary: {supervisor.get_annual_salary()}")
    print(f"Annual Bonus: {supervisor.get_annual_bonus()}")

shift_supervisor_program()
```

Description: The ShiftSupervisor program is intended to create and display information about a shift supervisor. The ShiftSupervisor class is also a subclass of Employee and includes additional attributes for annual salary and annual bonus.

Customer Program

```python
# Create the Customer Program
def customer_program():
    name = input("Enter customer's name:")
    address = input("Enter customer's address:")
    phone_number = input("Enter customer's phone number:")
    customer_number = input("Enter customer number:")
    mailing_list = input("Add to mailing list?(yes/no)").lower()

    #Create instance for Customer Program
    customer = Customer(name, address, phone_number, customer_number, mailing_list)

    print("\n Customer Information:")
    print(f"Name: {customer.get_name()}")
    print(f"Address: {customer.get_address()}")
    print(f"Phone Number: {customer.get_phone_number()}")
    print(f"Customer Number: {customer.get_customer_number()}")
    print(f"On Mailing List: {"Yes" if customer.get_mailing_list() else "No"}")

customer_program()
```

Description: The Customer program is meant to create and display information about a customer. The Customer class extends the Person class and includes additional attributes for customer number and mailing list.

Documentation

Code Comments

All comments have been added throughout the codes to explain the functionality of each section and provided clarity on how the classes and methods are implemented.

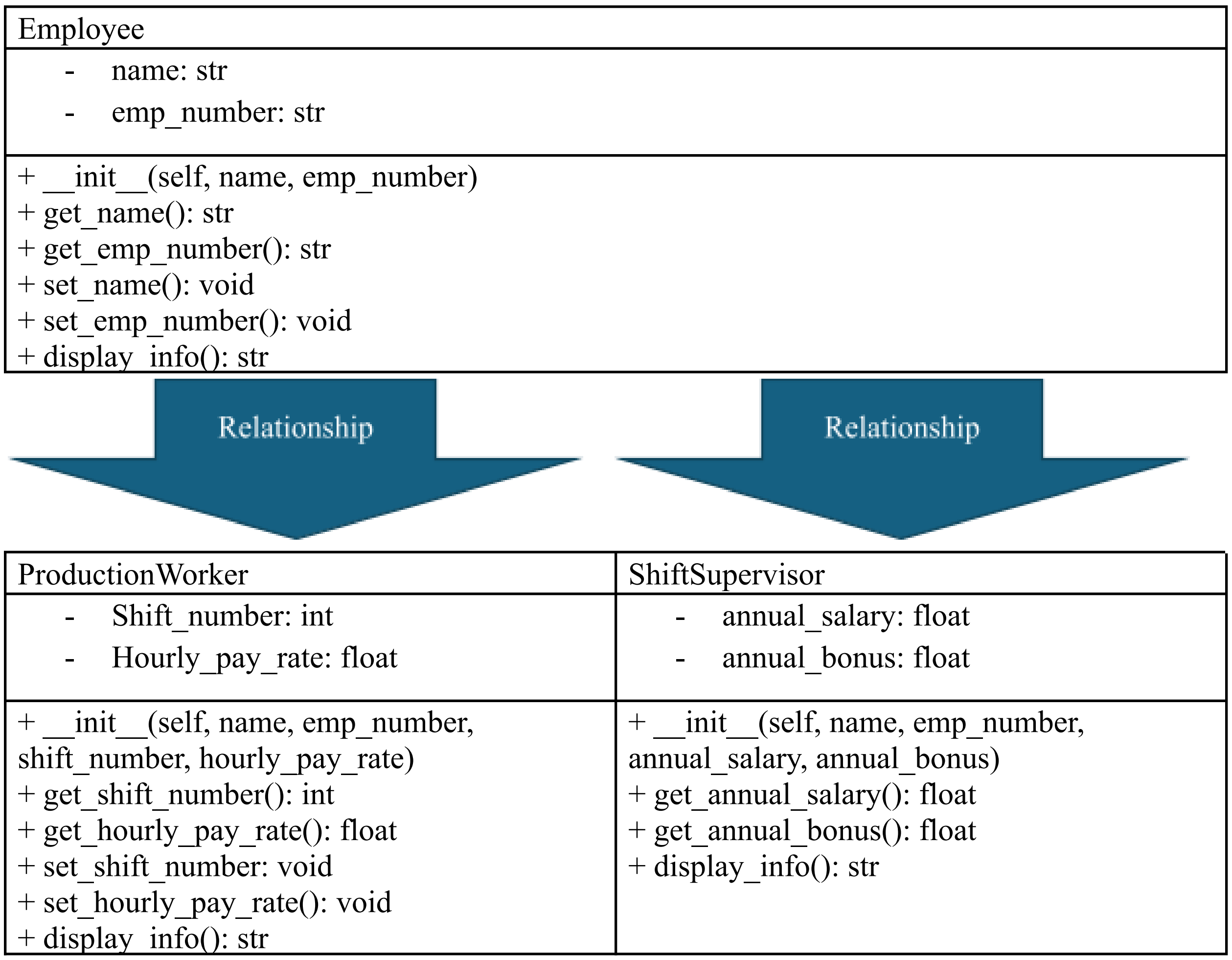
User Guide

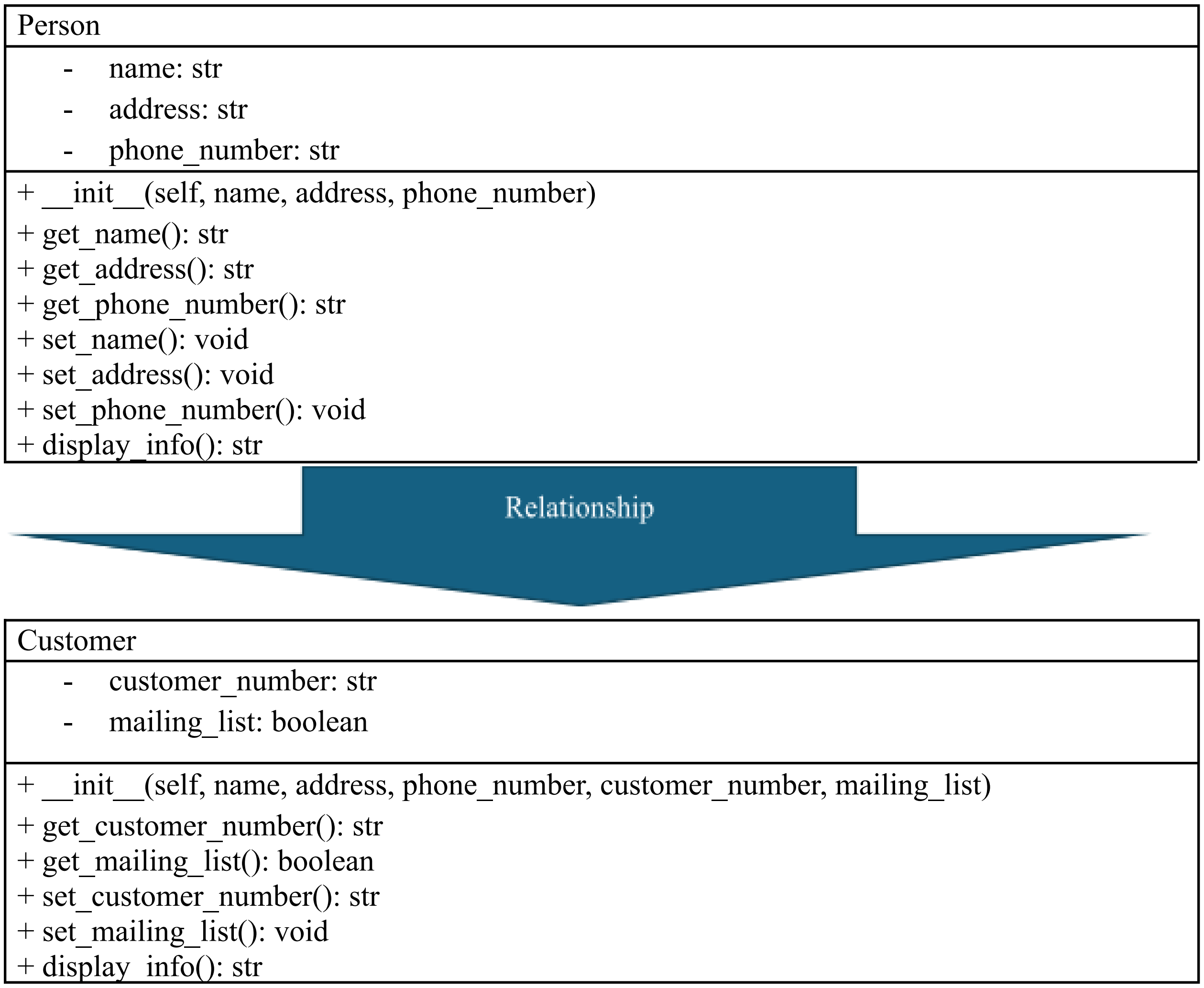
1. **Overview:** This program consists of three main functionalities:

- **ProductionWorker Program**: Creates and displays information for a production worker.
- **ShiftSupervisor Program**: Creates and displays information for a shift supervisor.
- **Customer Program**: Creates and displays information for a customer.

2. **Running the Programs:**
   - **ProductionWorker Program:**
     - **Purpose**: To create and display information for a production worker.
     - **Instructions**:
       - Run the production_worker_program function.
       - When prompted, enter the production worker's name, employee number, shift number (1 for day or 2 for night), and hourly pay rate.
       - The program will output the entered details for the production worker.
   - **ShiftSupervisor Program:**
     - **Purpose**: To create and display information for a shift supervisor.
     - **Instructions**:
       - Run the shift_supervisor_program function.
       - When prompted, enter the shift supervisor's name, employee number, annual salary, and annual bonus.
       - The program will output the entered details for the shift supervisor.
   - **Customer Program:**
     - **Purpose**: To create and display information for a customer.
     - **Instructions**:
       - Run the customer_program function.
       - When prompted, enter the customer's name, address, phone number, customer number, and whether they want to be added to the mailing list (yes or no).
       - The program will output the entered details for the customer, including whether they are on the mailing list.

UML Diagrams





```
Person
-   name: str
-   address: str
-   phone_number: str
+ __init__(self, name, address, phone_number)
+ get_name(): str
+ get_address(): str
+ get_phone_number(): str
+ set_name(): void
+ set_address(): void
+ set_phone_number(): void
+ display_info(): str
```

```
Customer
-   customer_number: str
-   mailing_list: boolean
+ __init__(self, name, address, phone_number, customer_number, mailing_list)
+ get_customer_number(): str
+ get_mailing_list(): boolean
+ set_customer_number(): str
+ set_mailing_list(): void
+ display_info(): str
```

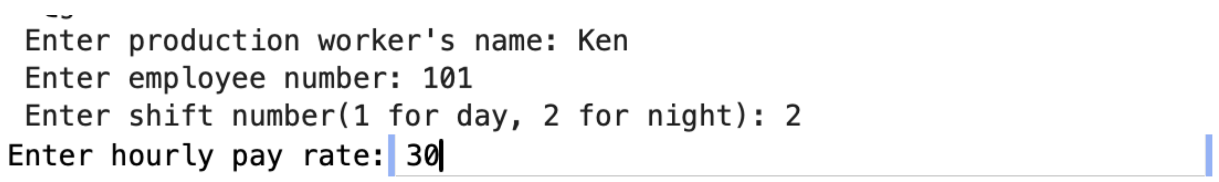
Testing

ProductionWorker Program

Purpose: To verify that a ProductionWorker object can be created and that the object's attributes are correctly set and displayed. This test ensures that the ProductionWorker class handles input correctly and that the display_info method provides accurate information.

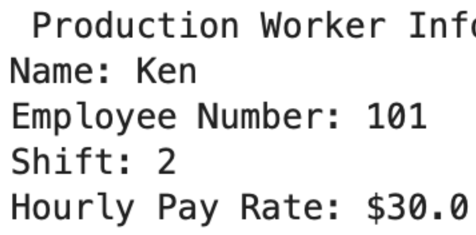
An instance of Production Worker is created. The production_worker_program() function with accesor methods has been called.

Input



```
 Enter production worker's name: Ken
 Enter employee number: 101
 Enter shift number(1 for day, 2 for night): 2
Enter hourly pay rate: 30
```

Output



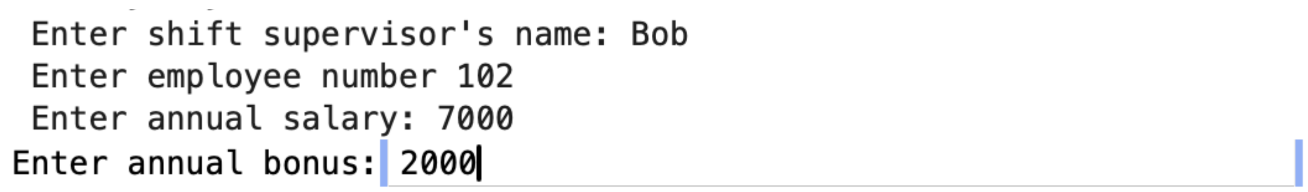
```
 Production Worker Information:
Name: Ken
Employee Number: 101
Shift: 2
Hourly Pay Rate: $30.0
```

ShiftSupervisor Program

Purpose: To check if a ShiftSupervisor object is created properly and if the attributes related to salary and bonus are correctly set and displayed. This test ensures that the ShiftSupervisor class handles inputs for salary and bonus and that the display_info method outputs accurate information.

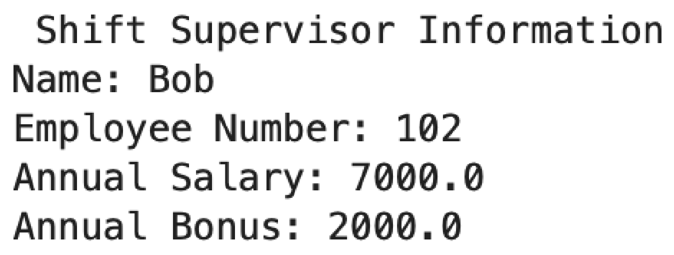
An instance of Shift Supervisor is created. The shift_supervisor_program() function with accesor methods has been called.

Input



```
 Enter shift supervisor's name: Bob
 Enter employee number 102
 Enter annual salary: 7000
Enter annual bonus: 2000
```

Output



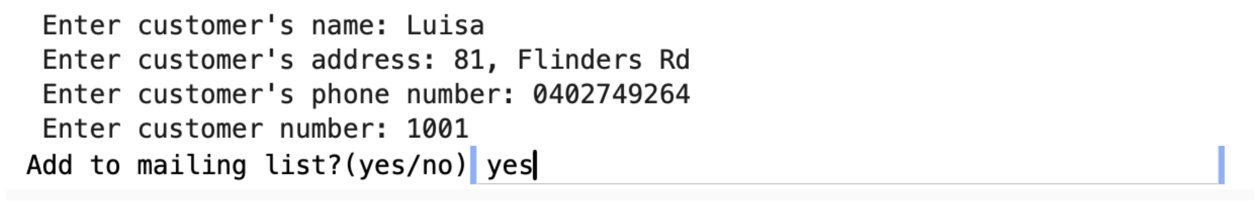
```
  Shift Supervisor Information
 Name: Bob
 Employee Number: 102
 Annual Salary: 7000.0
 Annual Bonus: 2000.0
```

Customer Program

Purpose: To ensure that a Customer object is created correctly and that customer details, including customer number and mailing list, are properly handled and displayed. This test verifies that the Customer class correctly extends the Person class and that the display_info method includes all relevant details.

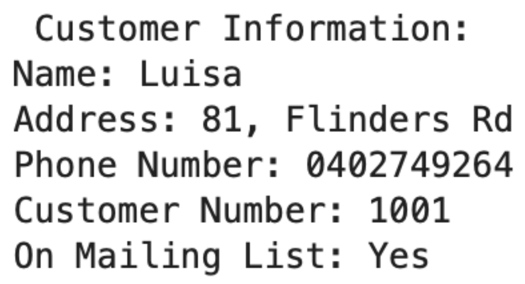
An instance of Customer is created. The customer_program() function with accesor methods has been called.

Input



```
 Enter customer's name: Luisa
 Enter customer's address: 81, Flinders Rd
 Enter customer's phone number: 0402749264
 Enter customer number: 1001
Add to mailing list?(yes/no) yes
```

Output



```
  Customer Information:
 Name: Luisa
 Address: 81, Flinders Rd
 Phone Number: 0402749264
 Customer Number: 1001
 On Mailing List: Yes
```