Hospital Patient Management System

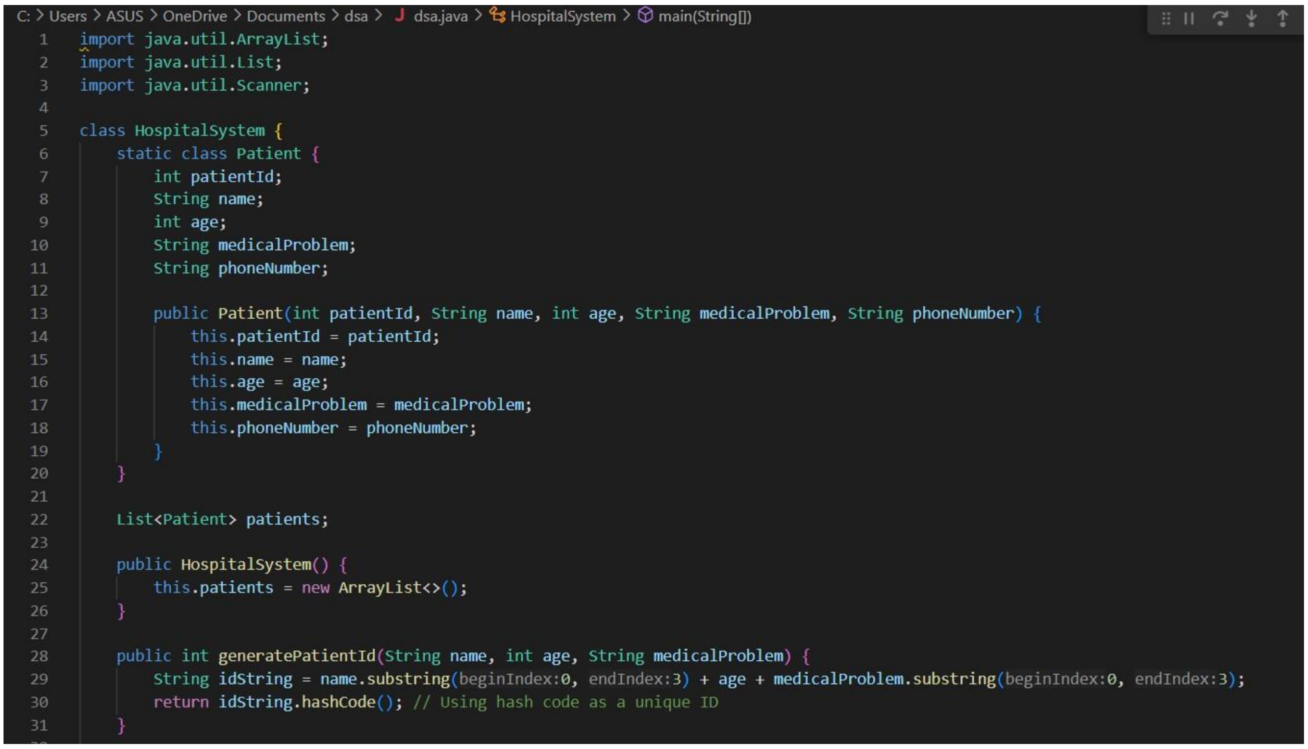
Saw Win Nwe
Sam

Contents

INTRODUCTION

The current project involves creating a Hospital Patient Management System, a crucial software solution that aims to enhance patient care and streamline hospital operations. As experienced software developers at Galaxy Software Company leading this initiative, our focus is on developing efficient data structure and algorithms to ensure the system’s effectiveness. Interion sort is a key algorithm that plays vital roles in sorting patient records based on different criteria. By conducting asymptotic analysis, we assess the performance of these algorithms, considering factors like time complexity and space complexity in various scenarios. We have developed a thorough test plan to verify the accuracy and resilience of the implemented algorithms making sure they can handle a wide range of inputs and edge cased seamlessly. Furthermore, our report delves into the intricacies. Trade-offs, and advantages of using abstract data types (ADTs) in the system’s design and development, highlighting how they contribute to improving modularity, encapsulation, and maintainability. This project offers us valuable hands-on experience in algorithm design and data structure implementation while tracking real world software development challenges.

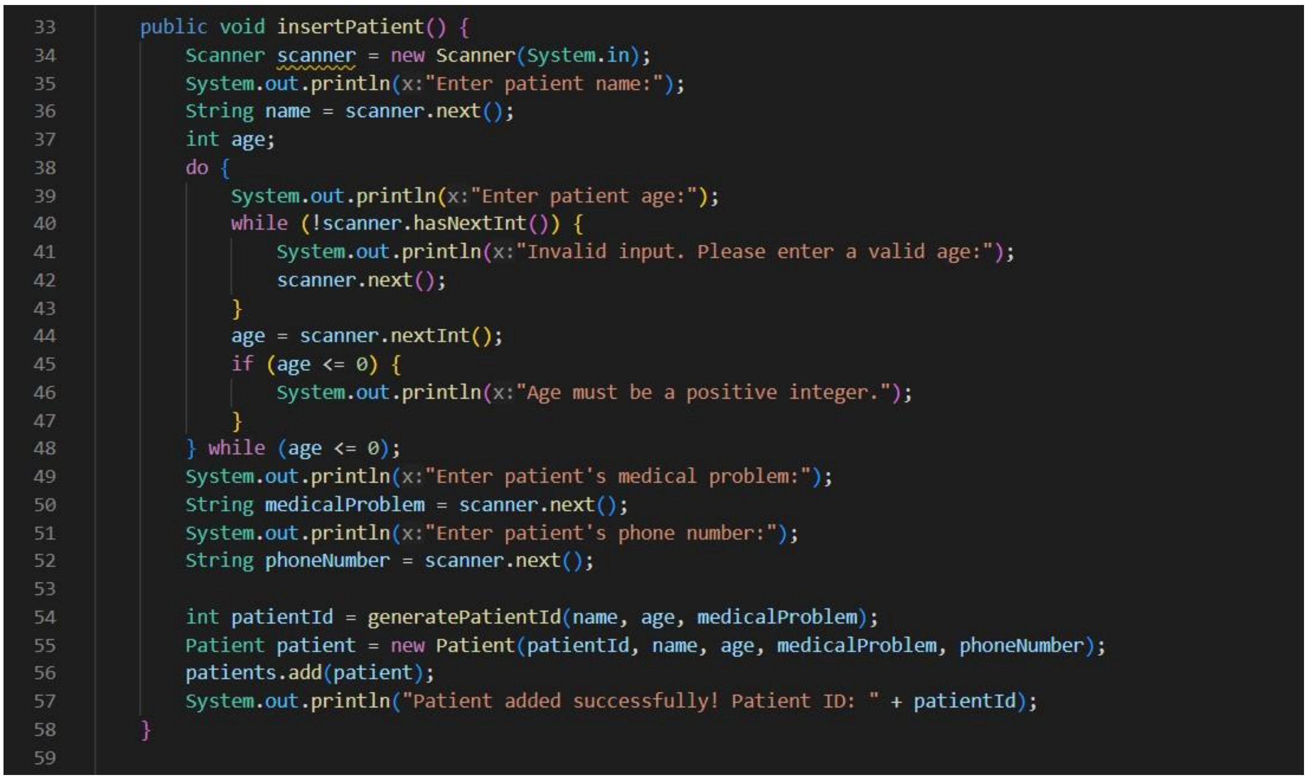
Task1



C: > Users > ASUS > OneDrive > Documents > dsa > J dsa.java > HospitalSystem > main(String[])

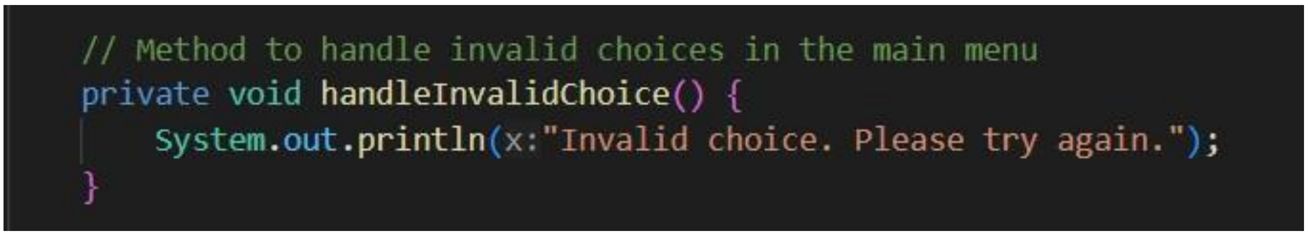
```java
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

class HospitalSystem {
    static class Patient {
        int patientId;
        String name;
        int age;
        String medicalProblem;
        String phoneNumber;

        public Patient(int patientId, String name, int age, String medicalProblem, String phoneNumber) {
            this.patientId = patientId;
            this.name = name;
            this.age = age;
            this.medicalProblem = medicalProblem;
            this.phoneNumber = phoneNumber;
        }
    }

    List<Patient> patients;

    public HospitalSystem() {
        this.patients = new ArrayList<>();
    }

    public int generatePatientId(String name, int age, String medicalProblem) {
        String idString = name.substring(beginIndex:0, endIndex:3) + age + medicalProblem.substring(beginIndex:0, endIndex:3);
        return idString.hashCode(); // Using hash code as a unique ID
    }
```

Figure 1: Import.



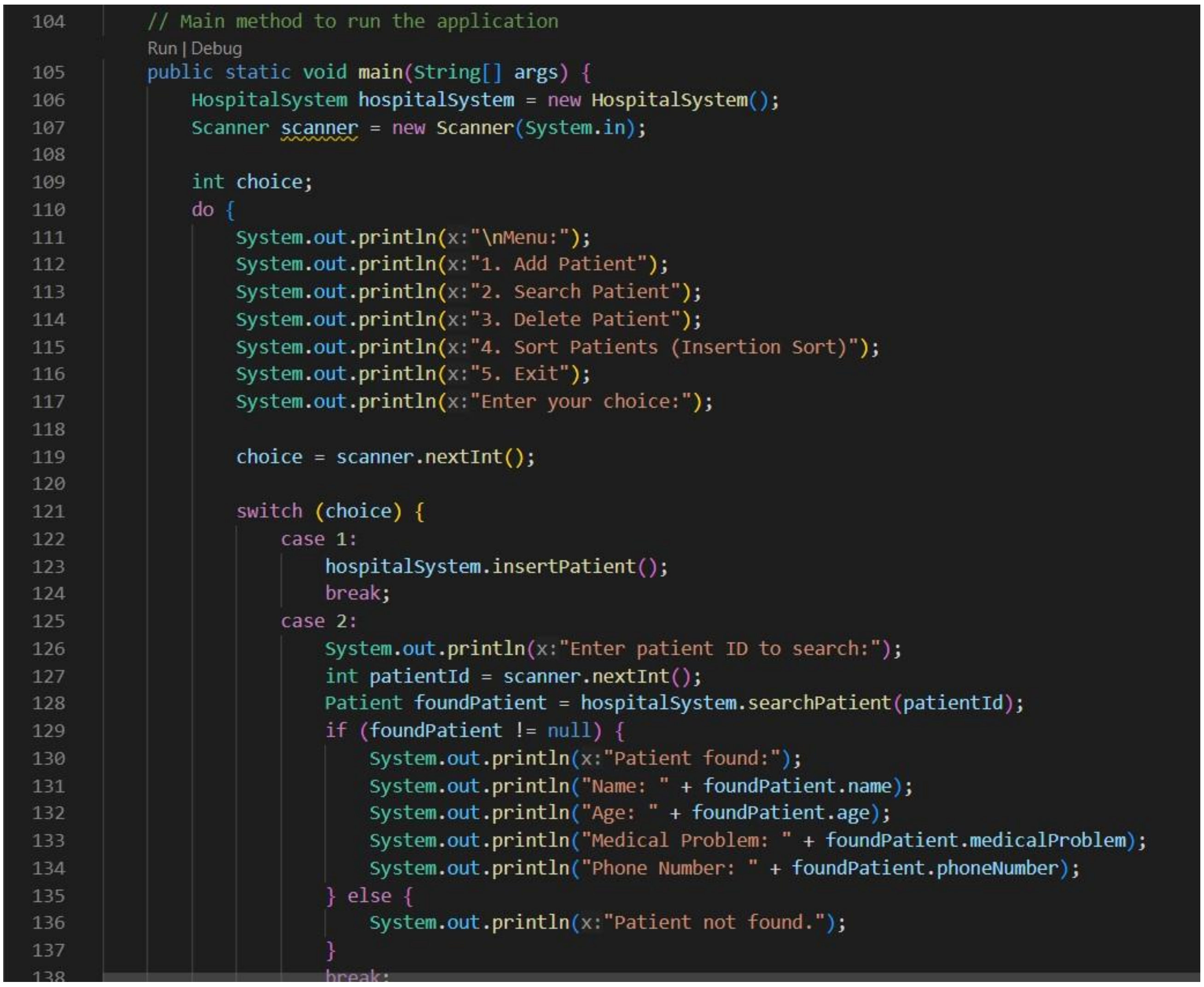
```java
    public void insertPatient() {
        Scanner scanner = new Scanner(System.in);
        System.out.println(x:"Enter patient name:");
        String name = scanner.next();
        int age;
        do {
            System.out.println(x:"Enter patient age:");
            while (!scanner.hasNextInt()) {
                System.out.println(x:"Invalid input. Please enter a valid age:");
                scanner.next();
            }
            age = scanner.nextInt();
            if (age <= 0) {
                System.out.println(x:"Age must be a positive integer.");
            }
        } while (age <= 0);
        System.out.println(x:"Enter patient's medical problem:");
        String medicalProblem = scanner.next();
        System.out.println(x:"Enter patient's phone number:");
        String phoneNumber = scanner.next();

        int patientId = generatePatientId(name, age, medicalProblem);
        Patient patient = new Patient(patientId, name, age, medicalProblem, phoneNumber);
        patients.add(patient);
        System.out.println("Patient added successfully! Patient ID: " + patientId);
    }

```

Figure 2:insert patient.



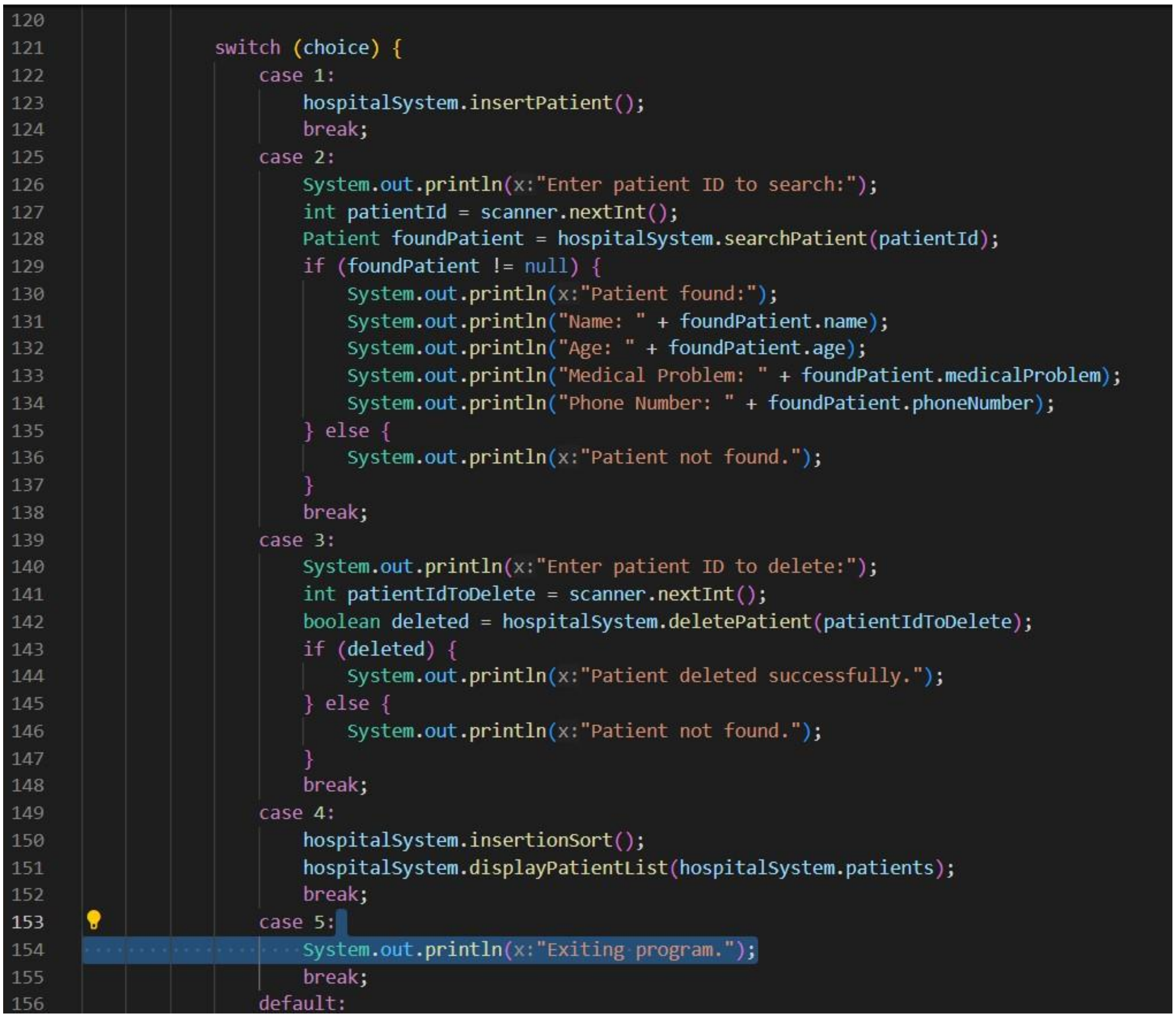
```java
// Method to handle invalid choices in the main menu
private void handleInvalidChoice() {
    System.out.println(x:"Invalid choice. Please try again.");
}
```

Figure 3: Error Handling



```java
    // Main method to run the application
    Run | Debug
    public static void main(String[] args) {
        HospitalSystem hospitalSystem = new HospitalSystem();
        Scanner scanner = new Scanner(System.in);

        int choice;
        do {
            System.out.println(x:"\nMenu:");
            System.out.println(x:"1. Add Patient");
            System.out.println(x:"2. Search Patient");
            System.out.println(x:"3. Delete Patient");
            System.out.println(x:"4. Sort Patients (Insertion Sort)");
            System.out.println(x:"5. Exit");
            System.out.println(x:"Enter your choice:");

            choice = scanner.nextInt();

            switch (choice) {
                case 1:
                    hospitalSystem.insertPatient();
                    break;
                case 2:
                    System.out.println(x:"Enter patient ID to search:");
                    int patientId = scanner.nextInt();
                    Patient foundPatient = hospitalSystem.searchPatient(patientId);
                    if (foundPatient != null) {
                        System.out.println(x:"Patient found:");
                        System.out.println("Name: " + foundPatient.name);
                        System.out.println("Age: " + foundPatient.age);
                        System.out.println("Medical Problem: " + foundPatient.medicalProblem);
                        System.out.println("Phone Number: " + foundPatient.phoneNumber);
                    } else {
                        System.out.println(x:"Patient not found.");
                    }
                    break;
```

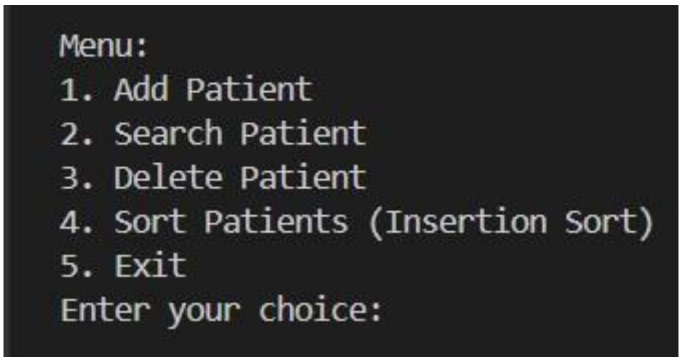
Figure 4: Menu



```java

            switch (choice) {
                case 1:
                    hospitalSystem.insertPatient();
                    break;
                case 2:
                    System.out.println(x:"Enter patient ID to search:");
                    int patientId = scanner.nextInt();
                    Patient foundPatient = hospitalSystem.searchPatient(patientId);
                    if (foundPatient != null) {
                        System.out.println(x:"Patient found:");
                        System.out.println("Name: " + foundPatient.name);
                        System.out.println("Age: " + foundPatient.age);
                        System.out.println("Medical Problem: " + foundPatient.medicalProblem);
                        System.out.println("Phone Number: " + foundPatient.phoneNumber);
                    } else {
                        System.out.println(x:"Patient not found.");
                    }
                    break;
                case 3:
                    System.out.println(x:"Enter patient ID to delete:");
                    int patientIdToDelete = scanner.nextInt();
                    boolean deleted = hospitalSystem.deletePatient(patientIdToDelete);
                    if (deleted) {
                        System.out.println(x:"Patient deleted successfully.");
                    } else {
                        System.out.println(x:"Patient not found.");
                    }
                    break;
                case 4:
                    hospitalSystem.insertionSort();
                    hospitalSystem.displayPatientList(hospitalSystem.patients);
                    break;
                case 5:
                    System.out.println(x:"Exiting program.");
                    break;
                default:
```

Figure 5: main code

Output

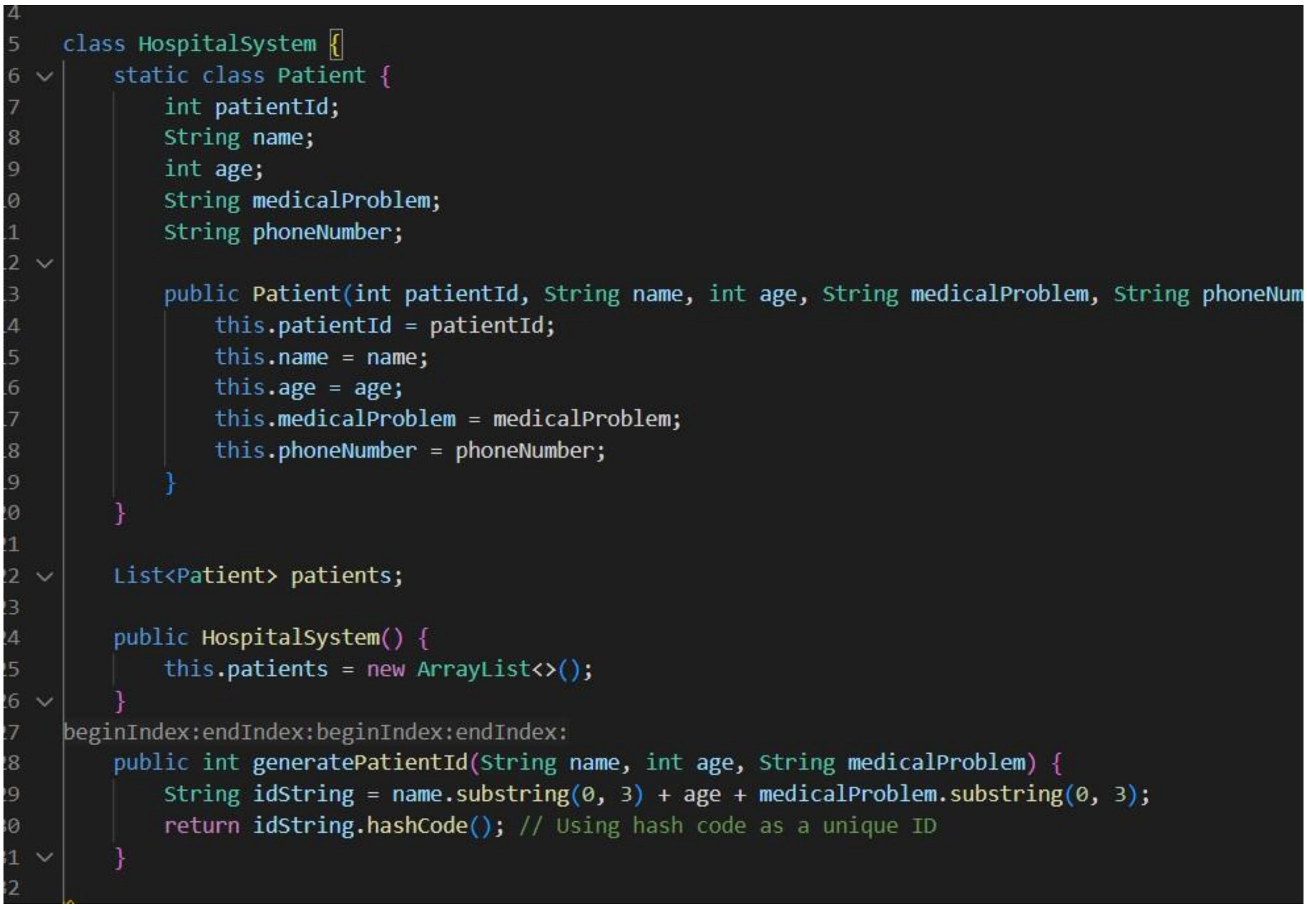


```
Menu:
1. Add Patient
2. Search Patient
3. Delete Patient
4. Sort Patients (Insertion Sort)
5. Exit
Enter your choice:
```

Figure 6: Menu Output

The output given is a menu for a Hospital Patient Management System, showing different choices for using the system. Here is a breakdown of each option on the menu.

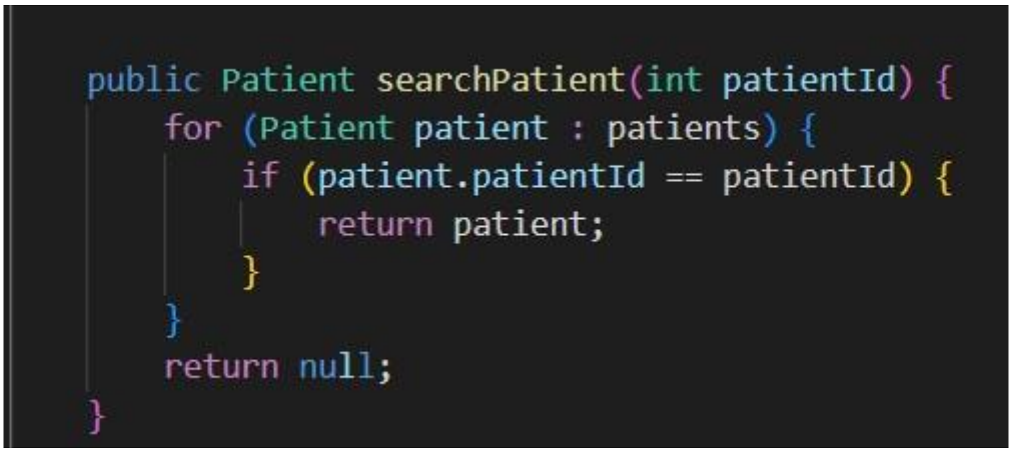
- Add Patient: Users can add a new patient by entering details like name, age, and department.



```java
class HospitalSystem {
    static class Patient {
        int patientId;
        String name;
        int age;
        String medicalProblem;
        String phoneNumber;

        public Patient(int patientId, String name, int age, String medicalProblem, String phoneNum
            this.patientId = patientId;
            this.name = name;
            this.age = age;
            this.medicalProblem = medicalProblem;
            this.phoneNumber = phoneNumber;
        }
    }

    List<Patient> patients;

    public HospitalSystem() {
        this.patients = new ArrayList<>();
    }
beginIndex:endIndex:beginIndex:endIndex:
    public int generatePatientId(String name, int age, String medicalProblem) {
        String idString = name.substring(0, 3) + age + medicalProblem.substring(0, 3);
        return idString.hashCode(); // Using hash code as a unique ID
    }
```

Figure 1.1: Add patients.

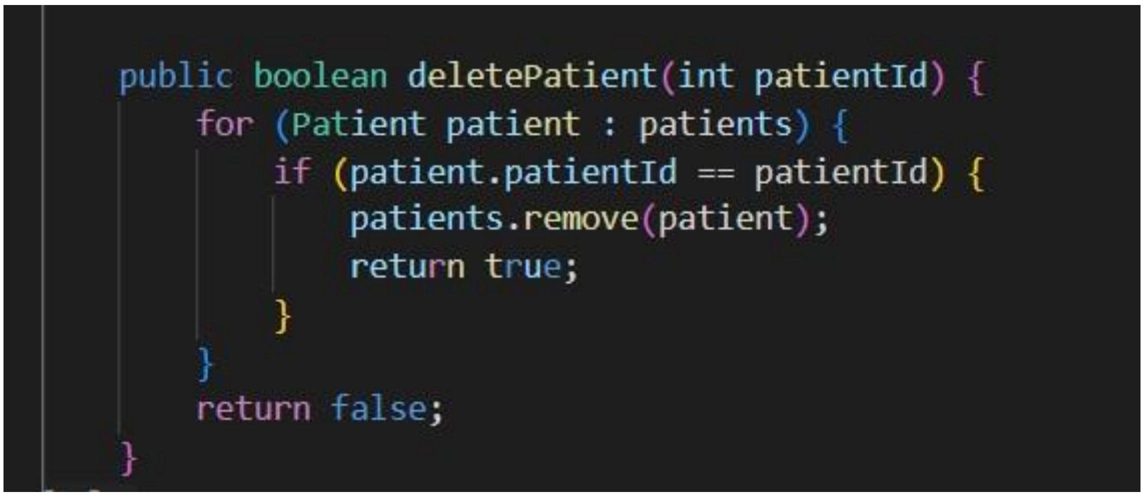
- Search Patient: Users can search for a patient by entering their ID to view their information.



```java
public Patient searchPatient(int patientId) {
    for (Patient patient : patients) {
        if (patient.patientId == patientId) {
            return patient;
        }
    }
    return null;
}
```

Figure 1.2: Search patients.

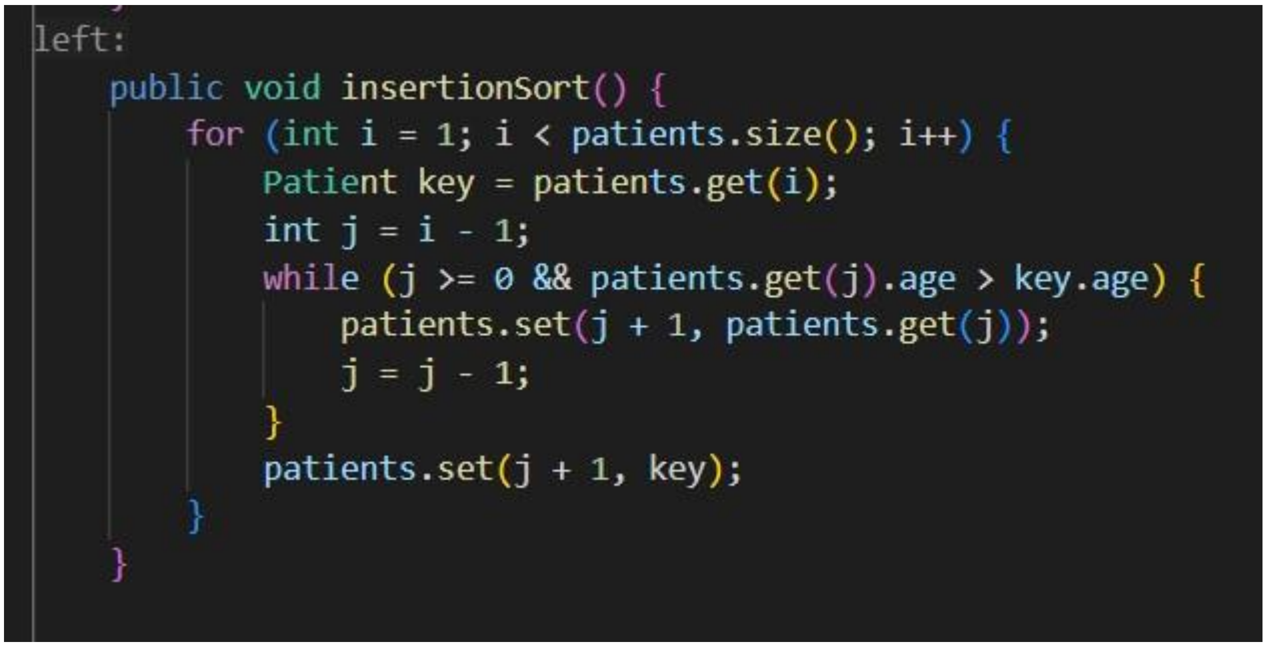
- Delete Patient: Users can delete a patient by providing their ID to remove their record from the system.



```java
public boolean deletePatient(int patientId) {
    for (Patient patient : patients) {
        if (patient.patientId == patientId) {
            patients.remove(patient);
            return true;
        }
    }
    return false;
}
```

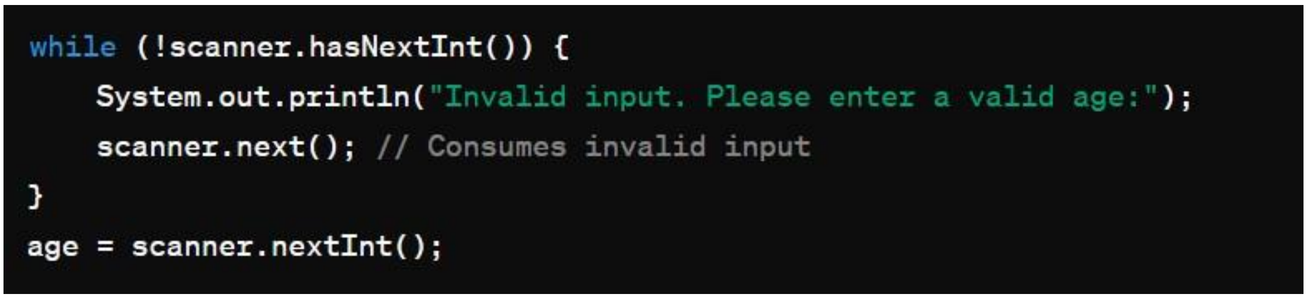
Figure 1.3: Delete patients.

- Sort Patients (Insertion Sort): This feature starts sorting patients' records using the Insertion Sort algorithm. When chosen, the system will arrange the patient records by their IDs using the Insertion Sort method and show the sorted list.



```
left:
    public void insertionSort() {
        for (int i = 1; i < patients.size(); i++) {
            Patient key = patients.get(i);
            int j = i - 1;
            while (j >= 0 && patients.get(j).age > key.age) {
                patients.set(j + 1, patients.get(j));
                j = j - 1;
            }
            patients.set(j + 1, key);
        }
    }
```

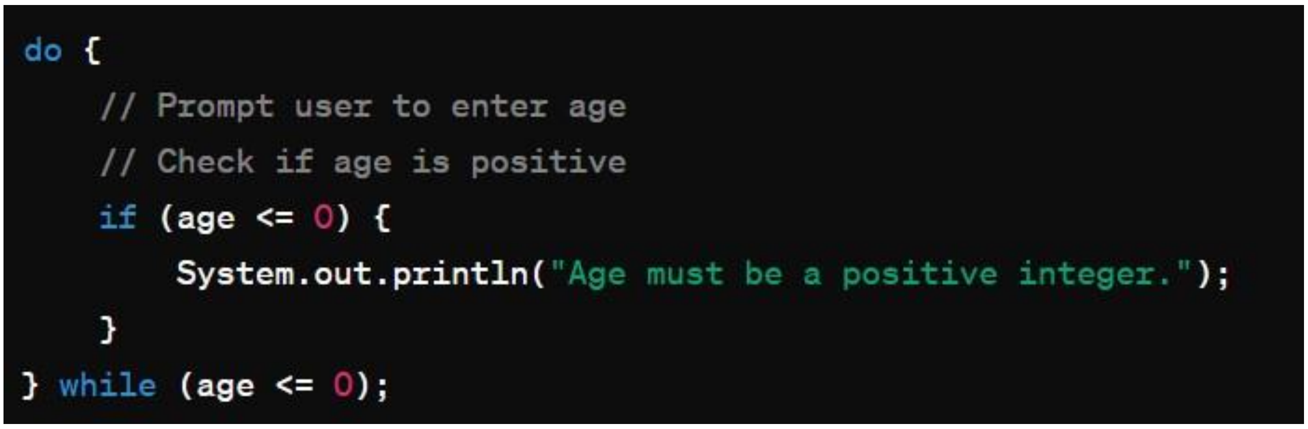
Figure 1.4: Insertion Sort.



```
while (!scanner.hasNextInt()) {
    System.out.println("Invalid input. Please enter a valid age:");
    scanner.next(); // Consumes invalid input
}
age = scanner.nextInt();
```

Figure 1.5: Error Handling.

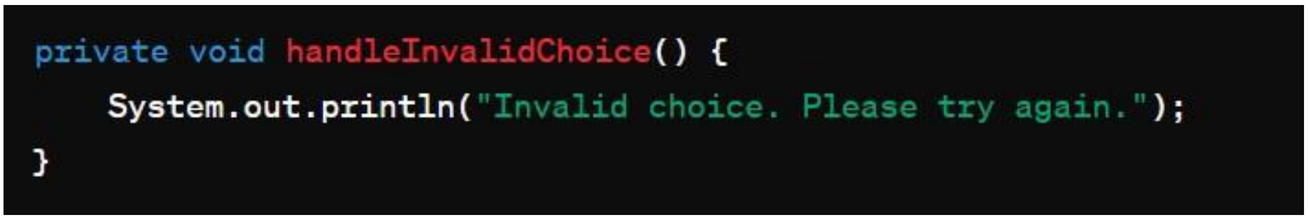
- Age Validation: Upon requesting the user to input the age of the patient, the program verifies whether the entered value is an integer. If the input is not an integer, the program will continuously prompt the user to input a valid age until a valid integer is provided.
- Furthermore, a loop is implemented to guarantee that the age input is a positive integer. If the entered age is not positive, an error message is exhibited by the program, urging the user to re-enter a positive integer.



```java
do {
    // Prompt user to enter age
    // Check if age is positive
    if (age <= 0) {
        System.out.println("Age must be a positive integer.");
    }
} while (age <= 0);
```

Figure 1.6: Error Handling.

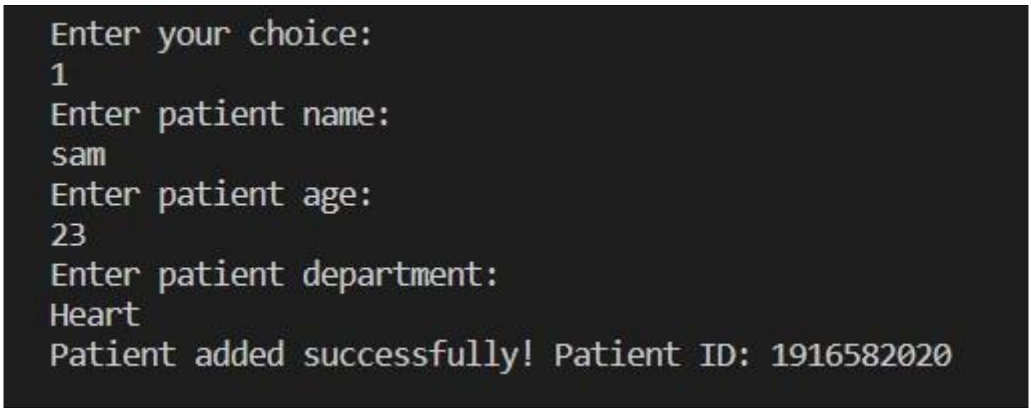
- The handleInvalidChoice() function merely displays an error message to indicate that the selected choice is invalid.



```java
private void handleInvalidChoice() {
    System.out.println("Invalid choice. Please try again.");
}
```

Figure 1.7: Error Handling.

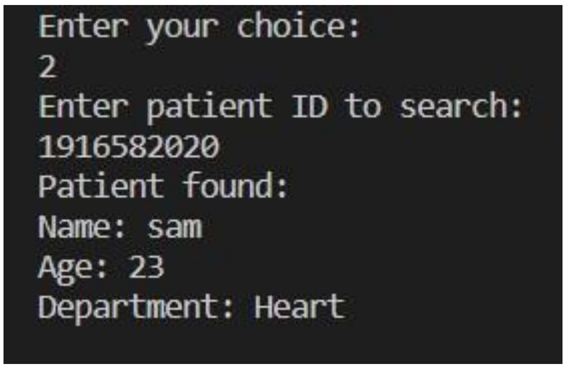
- Exit: By selecting this option, the program will close, ending the Hospital Patient Management System.



```
Enter your choice:
1
Enter patient name:
sam
Enter patient age:
23
Enter patient department:
Heart
Patient added successfully! Patient ID: 1916582020
```

Figure 7: Add Patient

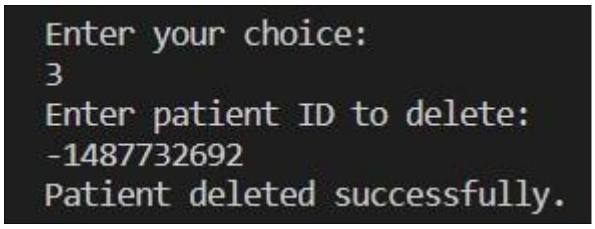
When users choose the "Add Patient" option in the Hospital Patient Management System, they can enter new patient information such as name, age, and department. The system will then create a special patient ID using this information. Once the patient is successfully added, the system will confirm the addition and provide the new patient ID for future reference. This feature helps to grow the patient database and ensures that important patient information is stored accurately for easy management and access.



```
Enter your choice:
2
Enter patient ID to search:
1916582020
Patient found:
Name: sam
Age: 23
Department: Heart
```

Figure 8: Search Patient

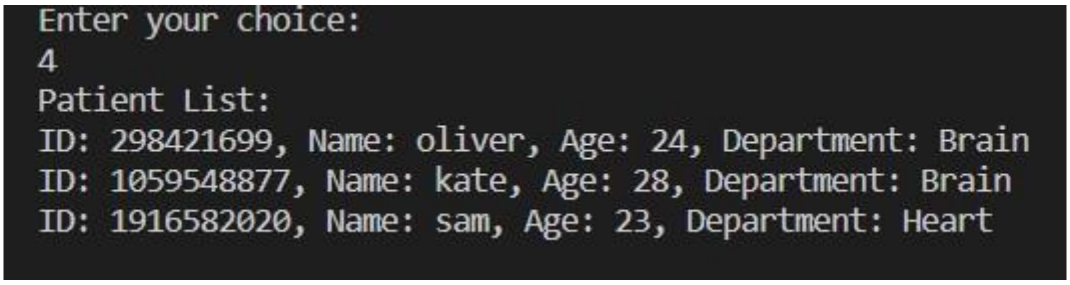
In the Hospital Patient Management System, users can search for specific patient information by entering their patient ID using Option 2, "Search Patient." If the patient is found, the system displays their details including name, age, and department. If the patient is not found, the user is notified by the system.



```
Enter your choice:
3
Enter patient ID to delete:
-1487732692
Patient deleted successfully.
```

Figure 9: Deleting Patient

Users can remove a patient from the Hospital Patient Management System by entering the patient's ID when choosing option 3. After the deletion is successful, the system verifies it and maintains the database's correctness. In case the patient ID is not found, the system informs the user.



```
Enter your choice:
4
Patient List:
ID: 298421699, Name: oliver, Age: 24, Department: Brain
ID: 1059548877, Name: kate, Age: 28, Department: Brain
ID: 1916582020, Name: sam, Age: 23, Department: Heart
```

Figure 10:Sorting Patient using Age

Insertion Sort (as choosing 4) allows users to organize the patient list using the insertion sort algorithm. It arranges patient records in ascending order by their unique IDs. Once sorted, the system shows the updated list for easy reference and navigation, improving the organization and accessibility of patient data in the Hospital Patient Management System.

- Start: The algorithm begins with the second item in the list and assumes that the first item is already in order.
- Compare and Insert: It then goes through the unsorted section of the list, comparing each item with the items in the sorted section. If an item is smaller than the items before it, it is moved to the correct position in the sorted section by shifting other items to the right.
- Repeat: This process continues until all items are sorted, with each iteration adding one more item to the sorted section.
- Finish: Once the iterations are finished, the entire list is sorted.

Insertion Sort code uses the insertionSort() method to sort the patient list. It goes through each patient, starting from the second one, and checks their ID against previous patients' IDs. If needed, it moves patients to the right to place the current patient in the correct sorted spot. The sorted patient list is then shown for quick access.

Task 2

Sample Test Plan

Test No.	Test Description	Test Data	Expected Result	Actual Result	Pass/ Fail
1	Add patient	Entire add patient	Data stored	Successful	Pass

2	Search Patient	Search By ID	Generated ID that matches with user input	Successful	Pass
3	Delete Patient	Delete Patient data with ID	Delete Patient	Easy deletion without confirmation	Pass
4	Insertion Sort	Show patient as insertion sort algorithm	Show results	Show results but not with step by step	Pass
5	Exits	Exits program	Exits program	Exits program	Pass
	Age Input Validation - Non-Numeric	Input: "abc"	System prompts user enter valid age to	As Expected,	Pass
	Age Input Validation - Negative Value	Input: -10	System prompts user to enter a valid age	As Expected,	Pass

	Menu Choice Validation - Non-Integer	Input: "abc"	System displays error message for invalid menu choice	As Expected,	Pass
	Menu Choice Validation - Out of Range	Input: 10	System displays error message for invalid menu choice	As Expected,	Pass