République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur
et de la Recherche Scientifique
ECOLE SUPÉRIEURE EN INFORMATIQUE
8 Mai 1945 - Sidi-Bel-Abbès

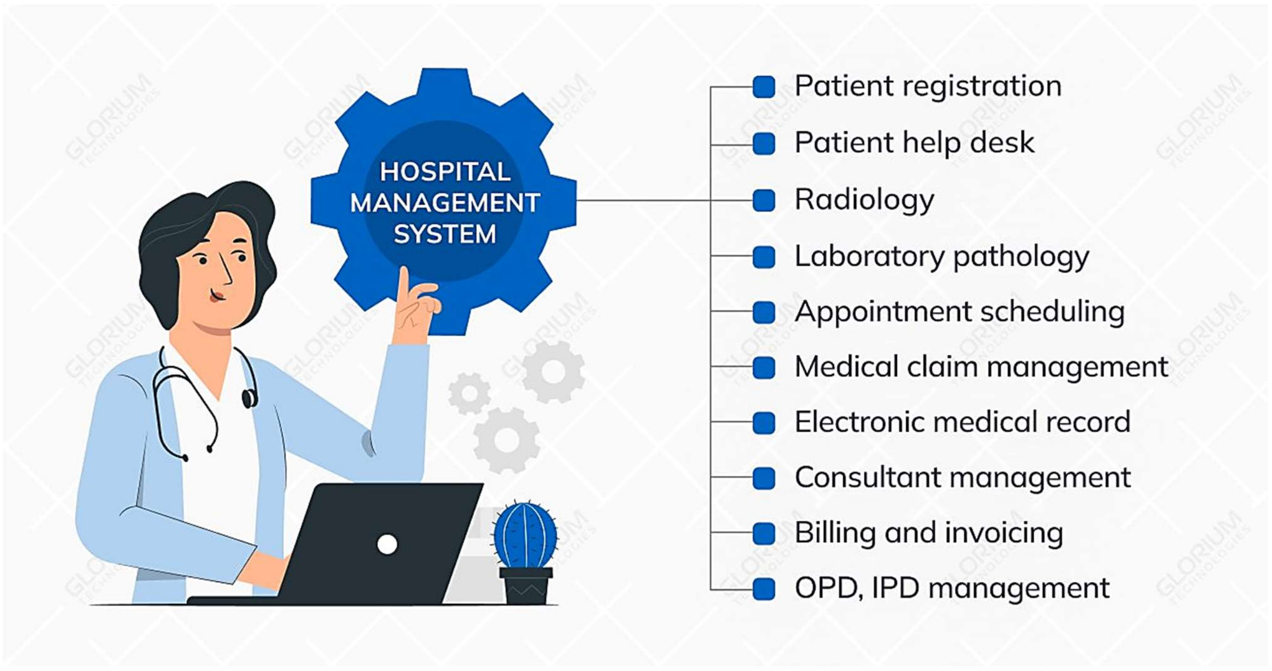
الجمهورية الجزائرية الديمقراطية الشعبية
وزارة التعليم العالي والبحث العلمي
المدرسة العليا للإعلام الآلي
8 ماي 1945 - سيدي بلعباس

Higher School of Computer Science 08 May 1945 - Sidi Bel Abbes

Oriented Object Programming

Mini Project

Hospital Patient Record System



Created By:
Dr, Serhane Oussama

Objective:

This mini-project is designed to build a simplified Hospital Patient Record System using Java programing language, illustrating core **Object-Oriented Programming (OOP)** principles such as **inheritance**, **polymorphism**, and **integrating Gui interfaces**.

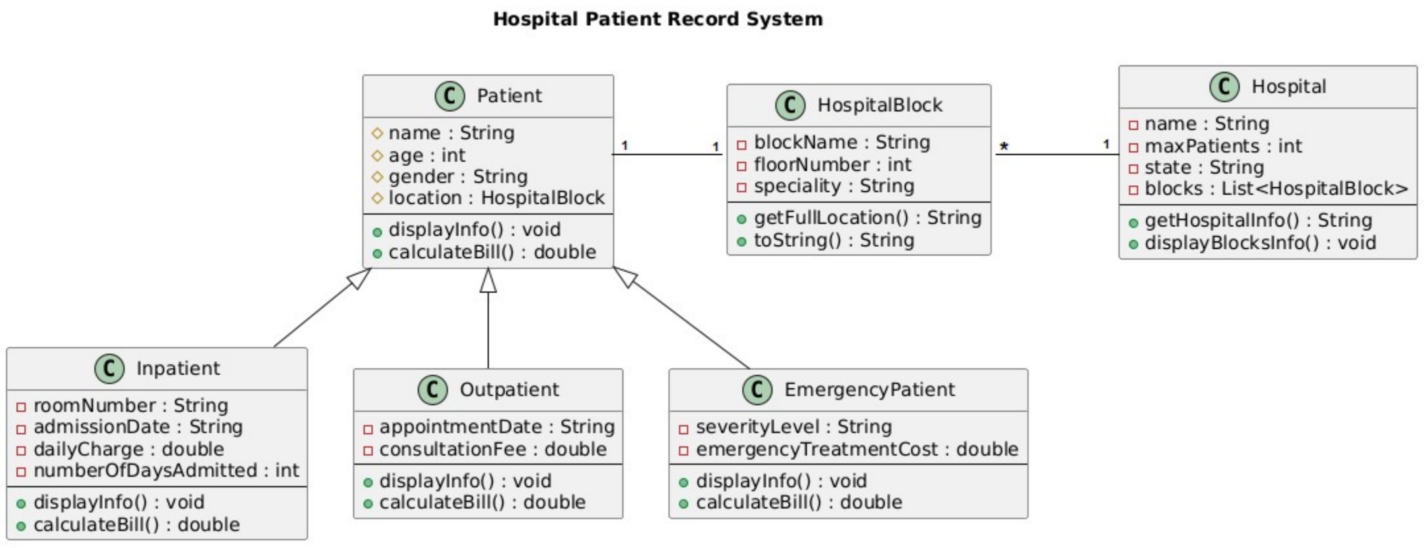
Main Objectives:

- Manage patient records, categorizing patients as **Inpatients**, **Outpatients**, and **Emergency Patients**, each with specialized attributes and behaviors.
- Organize hospital resources by defining a **Hospital** entity, containing multiple specialized **Hospital Blocks** (e.g., Cardiology, Pediatrics, Emergency).
- Implement methods that demonstrate polymorphic (**polymorphism**) behaviors, especially in calculating patient bills and displaying patient information.
- Provide a clear, structured codebase showcasing OOP best practices, serving as a foundation for further GUI integration.

Key Tasks:

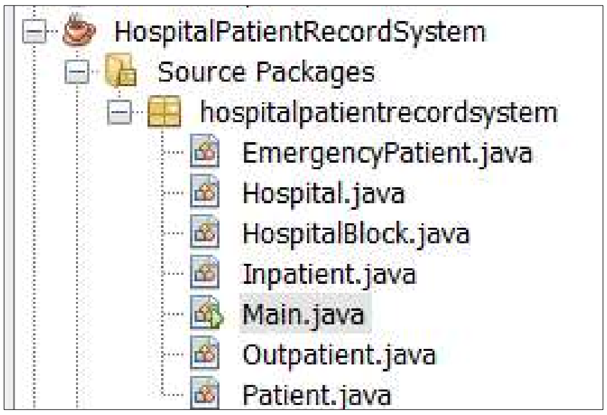
- Define relationships clearly between the hospital, hospital blocks, and patients.
- Apply inheritance by creating specialized patient types.
- Use polymorphism to handle patient-specific functionalities.
- Integrate location-based attributes clearly within the system.

Class Diagram: The class diagram for the **Hospital Patient Record System** contains the primary class **Hospital**, which can manage multiple specialized **HospitalBlocks**. Each **HospitalBlock** accommodates various types of patients represented through the superclass **Patient** and its subclasses: **Inpatient**, **Outpatient**, and **EmergencyPatient**. The diagram highlights **inheritance** (Patient subclasses) and **composition** (Hospital to HospitalBlock). And represent as following:



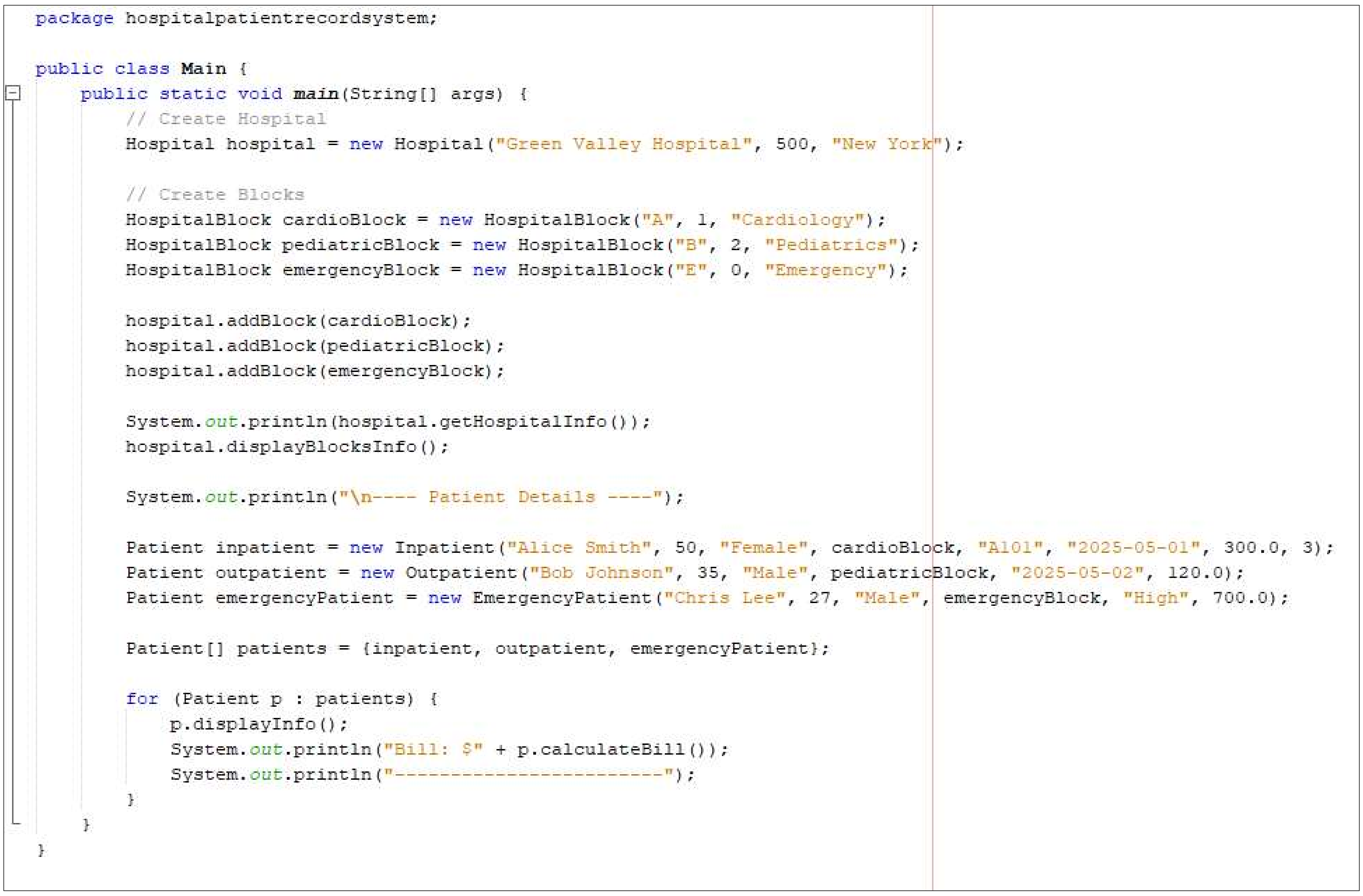
Phase N1: Core implementation and Testing

In this step, you should implement all classes including the main class, the class structure should appear as following:



```
HospitalPatientRecordSystem
  Source Packages
    hospitalpatientrecordsystem
      EmergencyPatient.java
      Hospital.java
      HospitalBlock.java
      Inpatient.java
      Main.java
      Outpatient.java
      Patient.java
```

To test the core software main functionalities. In the following an example of Main class code:

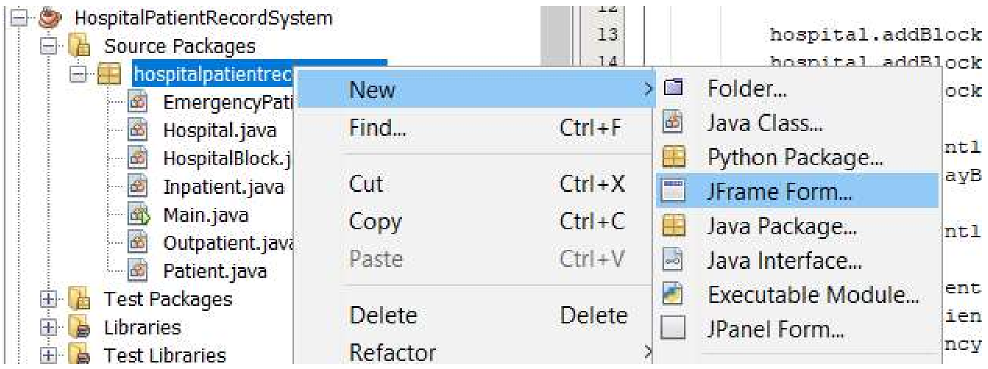


```java
package hospitalpatientrecordsystem;

public class Main {
    public static void main(String[] args) {
        // Create Hospital
        Hospital hospital = new Hospital("Green Valley Hospital", 500, "New York");

        // Create Blocks
        HospitalBlock cardioBlock = new HospitalBlock("A", 1, "Cardiology");
        HospitalBlock pediatricBlock = new HospitalBlock("B", 2, "Pediatrics");
        HospitalBlock emergencyBlock = new HospitalBlock("E", 0, "Emergency");

        hospital.addBlock(cardioBlock);
        hospital.addBlock(pediatricBlock);
        hospital.addBlock(emergencyBlock);

        System.out.println(hospital.getHospitalInfo());
        hospital.displayBlocksInfo();

        System.out.println("\n---- Patient Details ----");

        Patient inpatient = new Inpatient("Alice Smith", 50, "Female", cardioBlock, "A101", "2025-05-01", 300.0, 3);
        Patient outpatient = new Outpatient("Bob Johnson", 35, "Male", pediatricBlock, "2025-05-02", 120.0);
        Patient emergencyPatient = new EmergencyPatient("Chris Lee", 27, "Male", emergencyBlock, "High", 700.0);

        Patient[] patients = {inpatient, outpatient, emergencyPatient};

        for (Patient p : patients) {
            p.displayInfo();
            System.out.println("Bill: $" + p.calculateBill());
            System.out.println("------------------------");
        }
    }
}
```

Phase 2: Software Graphical Interface

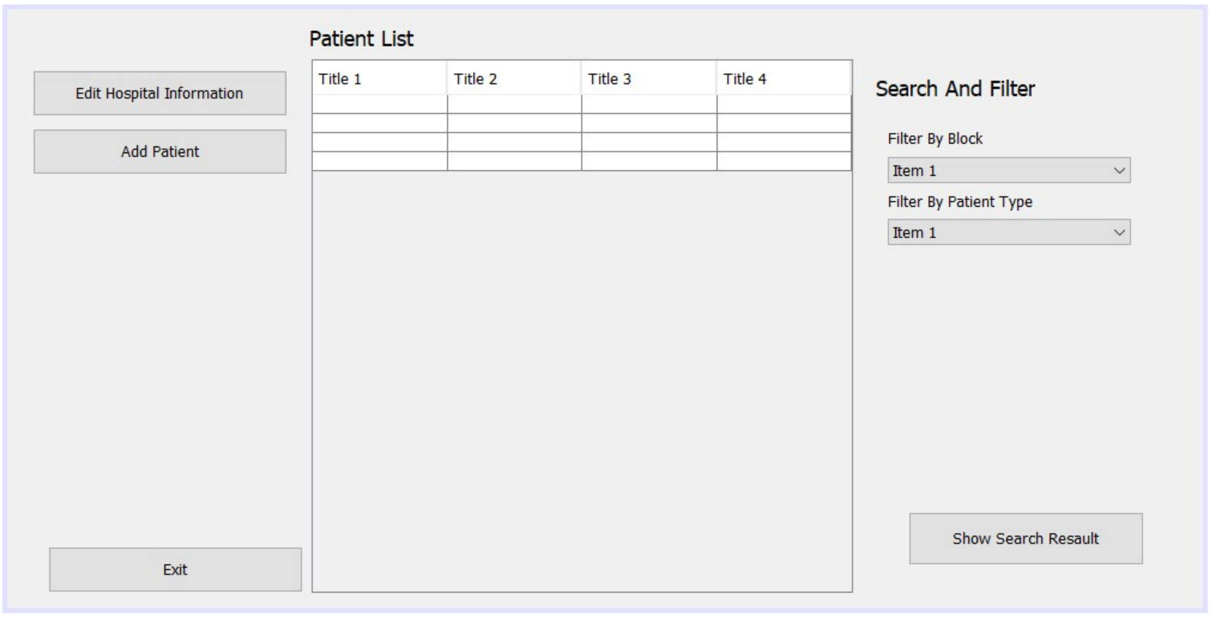
In this phase, we are going to prepare a graphical interface that contain three frames (window).

1/ Create a new JFrame named **HospitalRecordManager** (note that this frame will replace the Main Class) as following

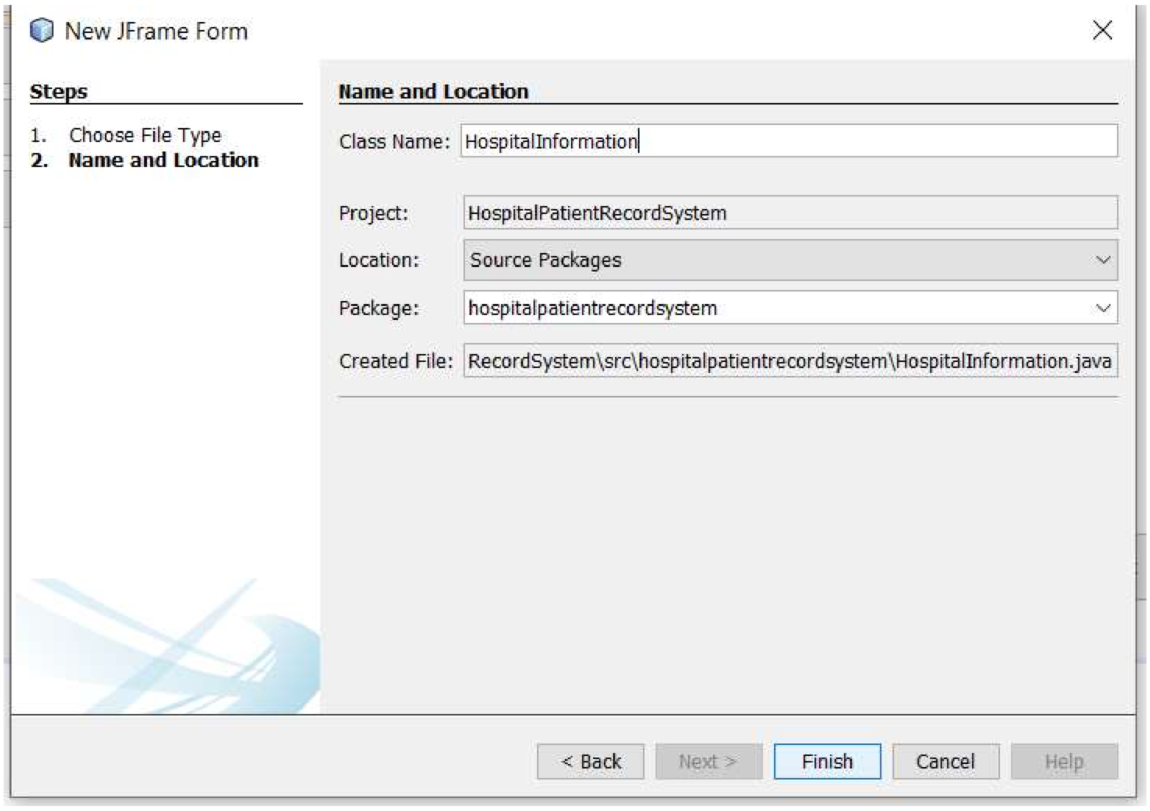


```
hospital.addBlock
hospital.addBlock
```

Then, fill out the following components from the left-side palette of NetBeans. The final result of **HospitalRecordManager** window will appear as following:



2/ Create second Frame that allow us to input the Hospital information, left click on your project and select create new JFrame, then name it **HospitalInformation** as following :


New JFrame Form

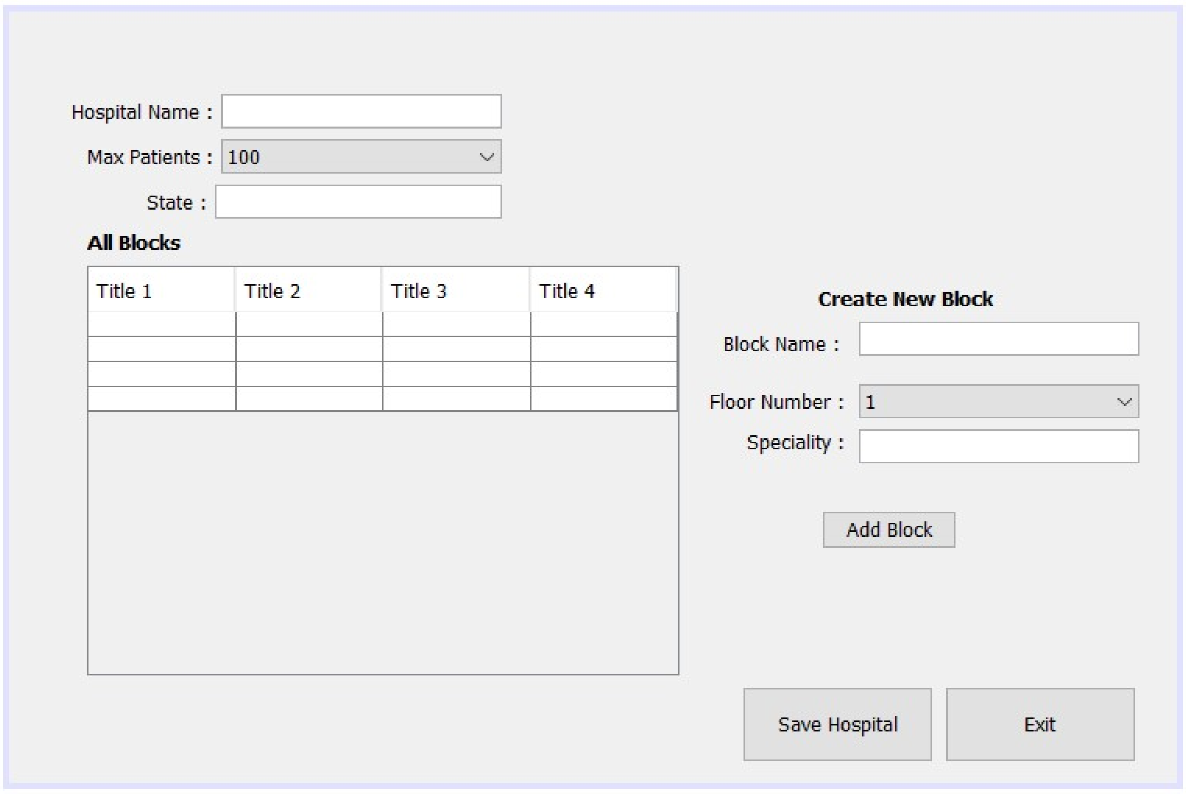
Steps

1. Choose File Type
2. **Name and Location**

Name and Location

Class Name:	HospitalInformation
Project:	HospitalPatientRecordSystem
Location:	Source Packages
Package:	hospitalpatientrecordsystem
Created File:	RecordSystem\src\hospitalpatientrecordsystem\HospitalInformation.java

Then, fill out the required component to create an Hospital instance (from Hospital class) as following :



Hospital Name :	
Max Patients :	100
State :	

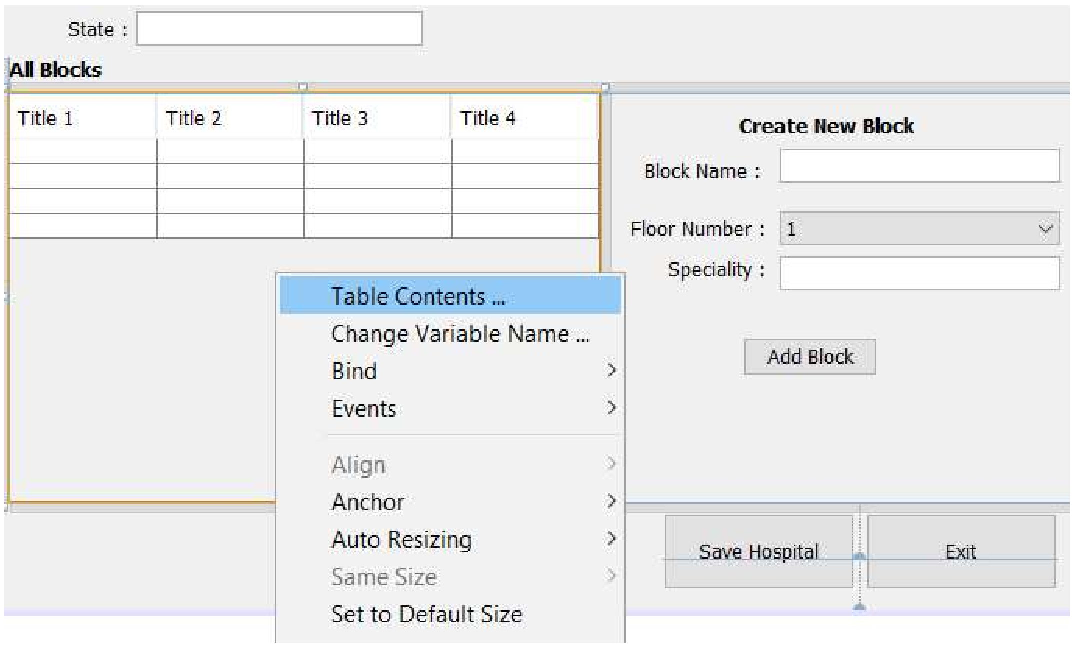
All Blocks

Title 1	Title 2	Title 3	Title 4

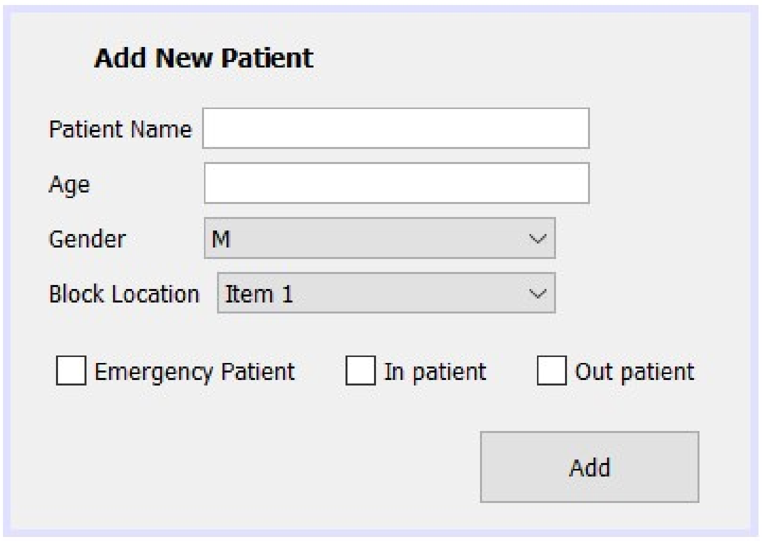
Create New Block

Block Name :	
Floor Number :	1
Speciality :	

You can Edit the Table information (Header) by clicking left and edit Table Contents



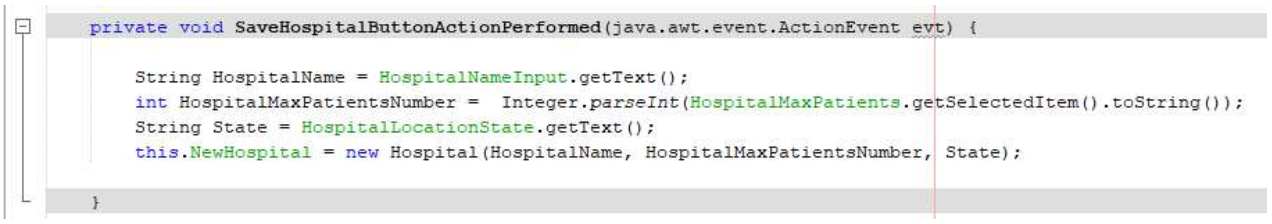
3/ Create Third Frame name it **NewPatient** that allow us to input the patient information as following



Phase 3: Software Functionalities

In this part we are going to link the functionality of our system

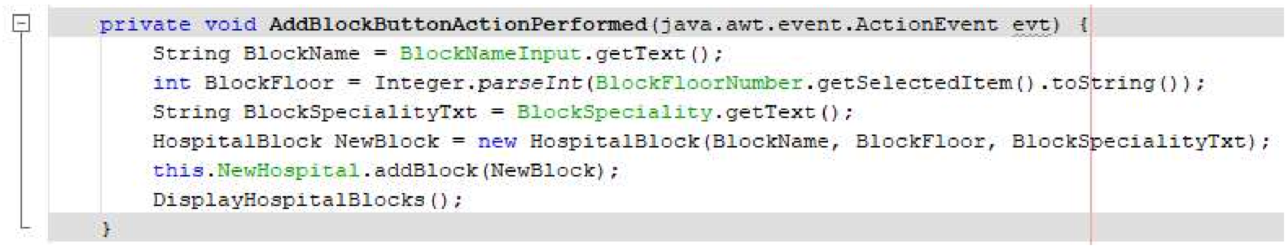
1- We need to be able to create and fill the information of our hospital that include hospital information and block information. Double click on **Save Hospital** Button of **HospitalInformation** Frame then we need to gather user input and call Hospital constructor as showing the following code:



```java
private void SaveHospitalButtonActionPerformed(java.awt.event.ActionEvent evt) {

    String HospitalName = HospitalNameInput.getText();
    int HospitalMaxPatientsNumber =  Integer.parseInt(HospitalMaxPatients.getSelectedItem().toString());
    String State = HospitalLocationState.getText();
    this.NewHospital = new Hospital(HospitalName, HospitalMaxPatientsNumber, State);

}
```

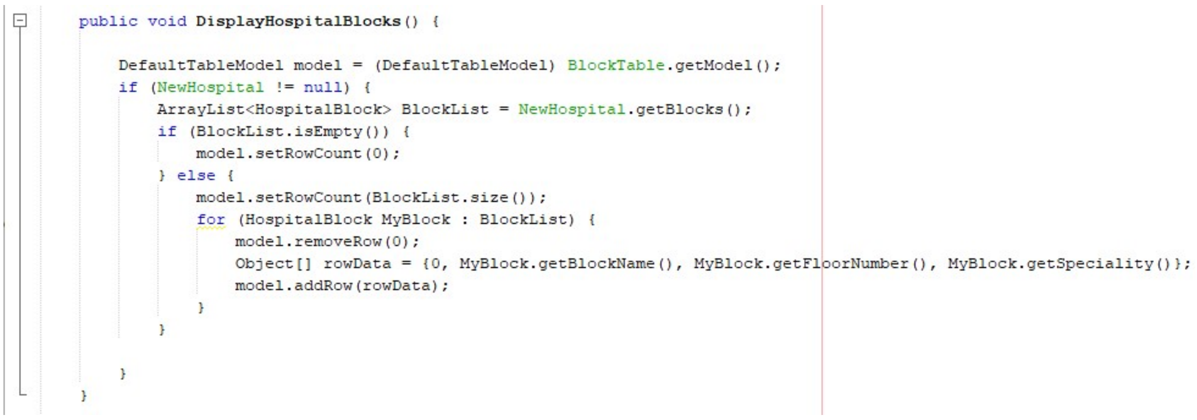
2- The trap in this part is the **HospitalRecordManager (**Frame or class**)** should know about the created object NewHospital**,** the above code missed one line to add.

3- Now, we need to allow the user to create and fill the hospital blocks, then display them in the block table.



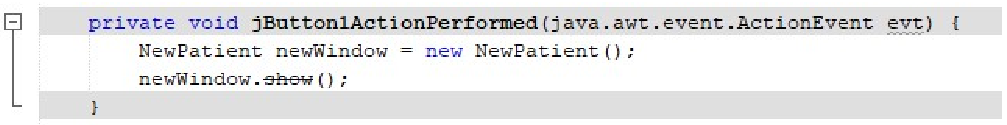
```java
private void AddBlockButtonActionPerformed(java.awt.event.ActionEvent evt) {
    String BlockName = BlockNameInput.getText();
    int BlockFloor = Integer.parseInt(BlockFloorNumber.getSelectedItem().toString());
    String BlockSpecialityTxt = BlockSpeciality.getText();
    HospitalBlock NewBlock = new HospitalBlock(BlockName, BlockFloor, BlockSpecialityTxt);
    this.NewHospital.addBlock(NewBlock);
    DisplayHospitalBlocks();
}
```

4- The last line of the above code call DisplayHospitalBlocks function, that function loops over all available blocks at our hospital then displays then at the hospital Table, DisplayHospitalBlocks is given as following :



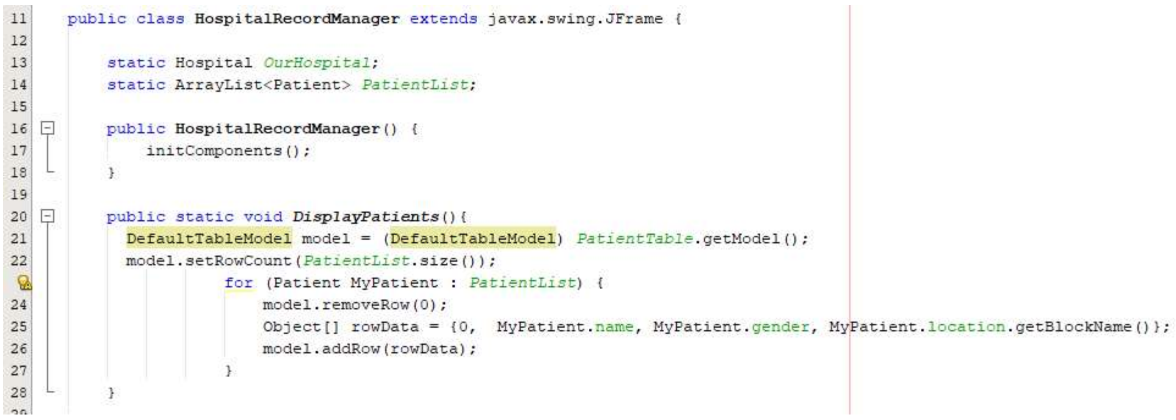
```java
public void DisplayHospitalBlocks() {

    DefaultTableModel model = (DefaultTableModel) BlockTable.getModel();
    if (NewHospital != null) {
        ArrayList<HospitalBlock> BlockList = NewHospital.getBlocks();
        if (BlockList.isEmpty()) {
            model.setRowCount(0);
        } else {
            model.setRowCount(BlockList.size());
            for (HospitalBlock MyBlock : BlockList) {
                model.removeRow(0);
                Object[] rowData = {0, MyBlock.getBlockName(), MyBlock.getFloorNumber(), MyBlock.getSpeciality()};
                model.addRow(rowData);
            }
        }

    }
}
```

5- Now, we are going to implement add patient functionality, when user click on **Add Patient** Button it shows **NewPatient** window the code to shows **NewPatient** given as following:



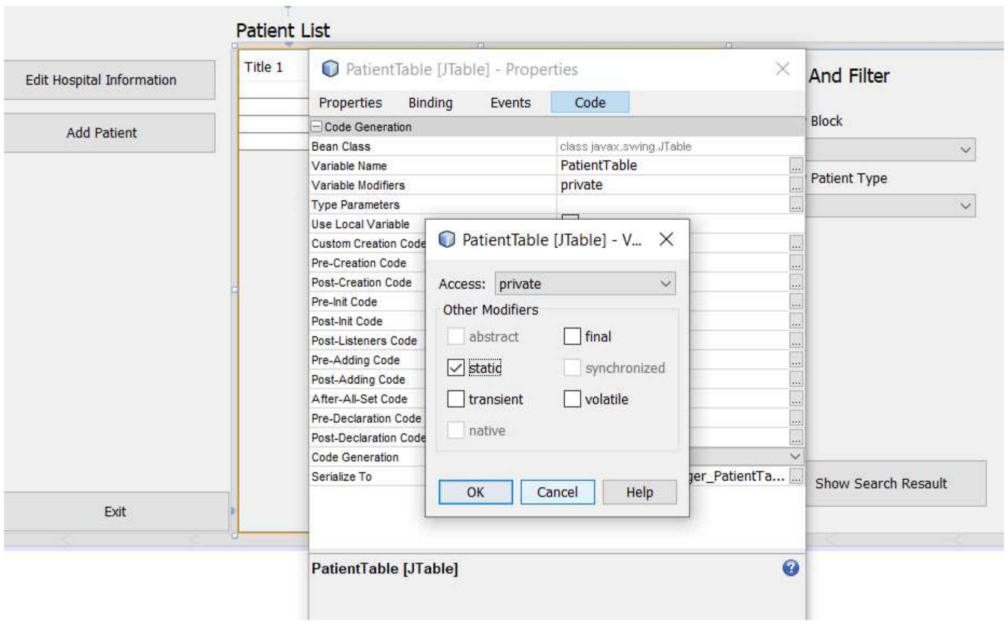
```java
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    NewPatient newWindow = new NewPatient();
    newWindow.~~show~~();
}
```

6- The user now should fill the information of the new patient then save it. After saving the new patient the table of patients should updated by calling **DisplayPatients()** function, the code of saving patient is given as follows:



```java
public class HospitalRecordManager extends javax.swing.JFrame {

    static Hospital OurHospital;
    static ArrayList<Patient> PatientList;

    public HospitalRecordManager() {
        initComponents();
    }

    public static void DisplayPatients(){
      DefaultTableModel model = (DefaultTableModel) PatientTable.getModel();
      model.setRowCount(PatientList.size());
                 for (Patient MyPatient : PatientList) {
                     model.removeRow(0);
                     Object[] rowData = {0,  MyPatient.name, MyPatient.gender, MyPatient.location.getBlockName()};
                     model.addRow(rowData);
                 }
    }
```

Note: You need to change PatientTable modifier to static by access to table propriety/Code as follows:

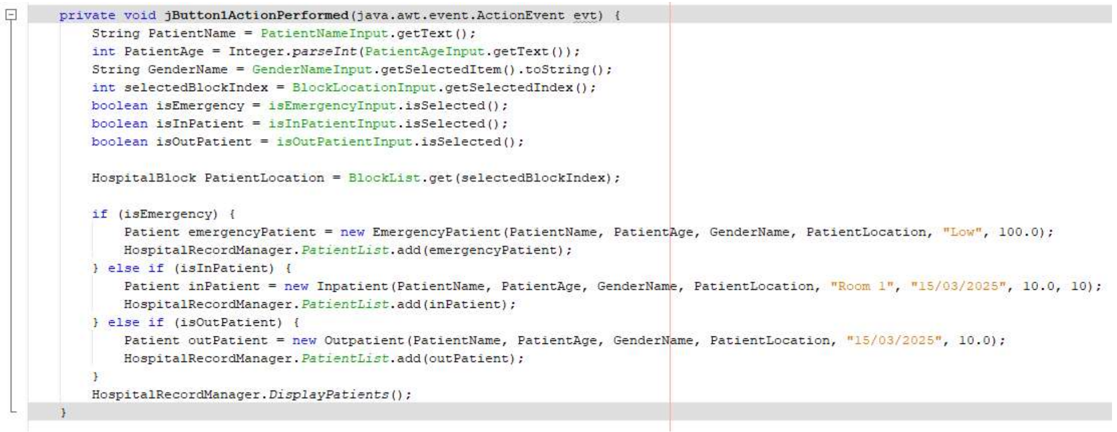


Code Generation
Bean Class: class javax.swing.JTable
Variable Name: PatientTable
Variable Modifiers: private
Type Parameters
Use Local Variable
Custom Creation Code
Pre-Creation Code
Post-Creation Code
Pre-Init Code
Post-Init Code
Post-Listeners Code
Pre-Adding Code
Post-Adding Code
After-All-Set Code
Pre-Declaration Code
Post-Declaration Code
Code Generation
Serialize To: ...ger_PatientTa...

Access: private
Other Modifiers
☐ abstract ☐ final
☑ static ☐ synchronized
☐ transient ☐ volatile
☐ native

By clicking **Add** button on new patient frame, the flow code should call a constructor based on patient type, then create a new patient instance, then save it in global patient list of **HospitalRecordManager** class, the code is given as follows:

7. The following code shows what will happened when a user click on Add patient. It first collect the input information then create appropriate object instance (from different patient classes) call constructor based on selected patient type. And finally call **DisplayPatients()** function to update the table of all patients.



```java
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        String PatientName = PatientNameInput.getText();
        int PatientAge = Integer.parseInt(PatientAgeInput.getText());
        String GenderName = GenderNameInput.getSelectedItem().toString();
        int selectedBlockIndex = BlockLocationInput.getSelectedIndex();
        boolean isEmergency = isEmergencyInput.isSelected();
        boolean isInPatient = isInPatientInput.isSelected();
        boolean isOutPatient = isOutPatientInput.isSelected();

        HospitalBlock PatientLocation = BlockList.get(selectedBlockIndex);

        if (isEmergency) {
            Patient emergencyPatient = new EmergencyPatient(PatientName, PatientAge, GenderName, PatientLocation, "Low", 100.0);
            HospitalRecordManager.PatientList.add(emergencyPatient);
        } else if (isInPatient) {
            Patient inPatient = new Inpatient(PatientName, PatientAge, GenderName, PatientLocation, "Room 1", "15/03/2025", 10.0, 10);
            HospitalRecordManager.PatientList.add(inPatient);
        } else if (isOutPatient) {
            Patient outPatient = new Outpatient(PatientName, PatientAge, GenderName, PatientLocation, "15/03/2025", 10.0);
            HospitalRecordManager.PatientList.add(outPatient);
        }
        HospitalRecordManager.DisplayPatients();
    }
```

8. Your final task now is to implement search and filter functionalities that allow us to:
    a. Perform a search on list of patient by name
    b. Filter the patients list based on their block

<u>Mini Project submission and evaluation :</u>

This mini project is due before the final exam as a maximum deadline. Adding more functionalities, idea, styling will be counted.

The submission should be at Lab session (you are not allowed to submit it on other group sessions), you are free to submit it at any session yet the evaluation will differ based on the following dates:

Submission Date (until)	**Max Mark**
12/04/2025	18/20
19/04/2025	16/20
26/04/2025	14/20
03/05/2025	12/20
10/05/2025	10/20

Note: That is the maximum mark, not the deserved mark. Your mark will reflect your work and will be decided by your lab supervisor.