National University of Computer & Emerging Sciences, Karachi
Computer Science Department
Fall 2024, Lab Manual - 05

Course Code: SL3001	**Course: Software Development and construction**
Instructor:	**Yasir Arfat**

Lab # 05

1. Collections in Java

The **Collection in Java** is a framework that provides an architecture to store and manipulate the group of objects.

Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.

Java Collection means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList

, Vector, LinkedList

, PriorityQueue

, HashSet, LinkedHashSet, TreeSet).

What is Collection in Java

A Collection represents a single unit of objects, i.e., a group.

What is a framework in Java

- It provides readymade architecture.
- It represents a set of classes and interfaces.
- It is optional.

What is Collection framework

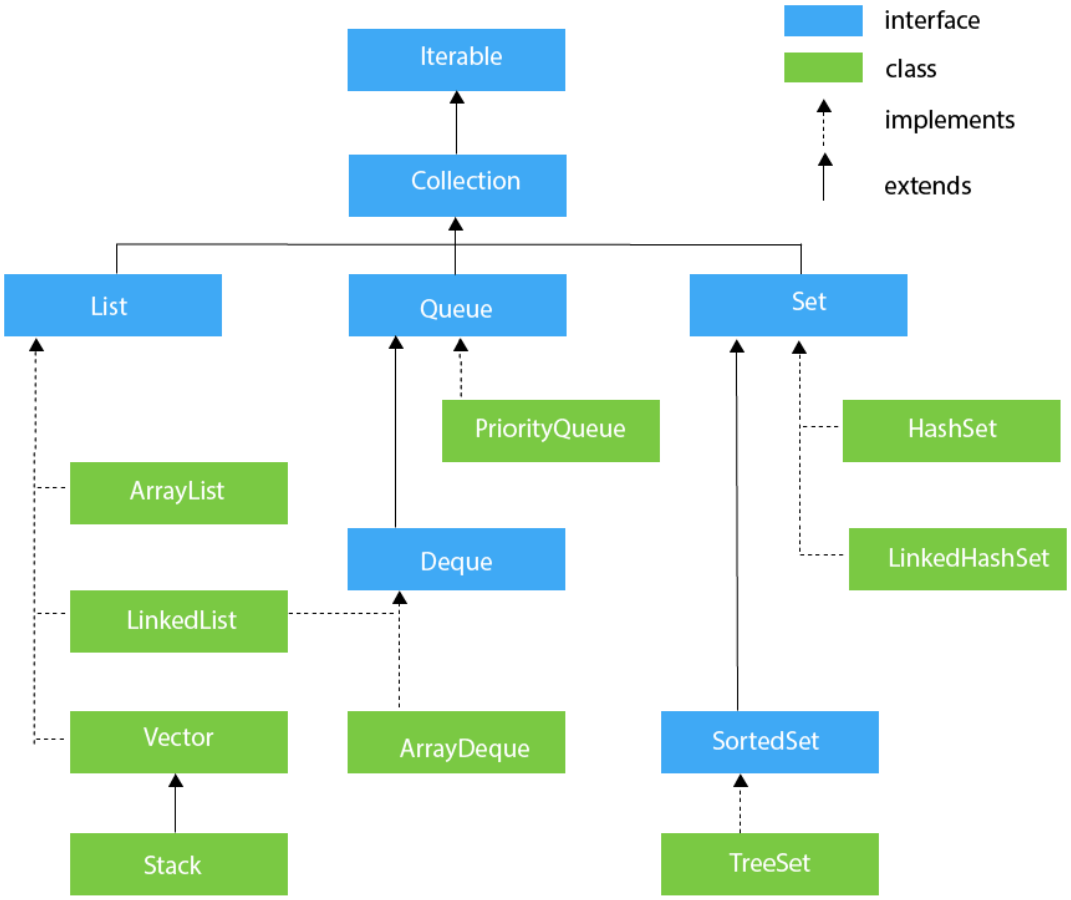
The Collection framework represents a unified architecture for storing and manipulating a group of objects. It has:

1. Interfaces and its implementations, i.e., classes
2. Algorithm

Hierarchy of Collection Framework

The java.util package contains all the classes

and interfaces



List Interface

List interface is the child interface of Collection interface. It inhibits a list type data structure in which we can store the ordered collection of objects. It can have duplicate values.

List interface is implemented by the classes ArrayList, LinkedList, Vector, and Stack.

To instantiate the List interface, we must use :

1. List <data-type> list1= **new** ArrayList();
2. List <data-type> list2 = **new** LinkedList();
3. List <data-type> list3 = **new** Vector();
4. List <data-type> list4 = **new** Stack();

ArrayList

The ArrayList class implements the List interface. It uses a dynamic array to store the duplicate element of different data types. The ArrayList class maintains the insertion order and is non-synchronized. The elements stored in the ArrayList class can be randomly accessed. Consider the following example.

Ex 1:

```
package collection;
import java.util.ArrayList;
import java.util.List;
public class Ex_1 {
    public static void main(String[] args) {
        List<Integer> l = new ArrayList();
        l.add(1);
        l.add(2);
        l.add(3);
        System.out.println(l);
    }}
```

```
C:\Users\Yasir-PC\.jdks\openjdk-22.0.2\bin\java.exe
[1, 2, 3]
```

Ex 2:

```java
package collection;
import java.util.ArrayList;
import java.util.List;
public class Ex_1 {
    public static void main(String[] args) {
        List<Integer> l = new ArrayList();
        l.add(1);
        l.add(2);
        l.add(3);
        System.out.println(l);
        l.remove(0);
        System.out.println(l);
        l.clear();
        System.out.println(l);
    }}
```

```
C:\Users\Yasir-PC\.jdks\openjdk-22.0.2\bin\java.exe "-
[1, 2, 3]
[2, 3]
[]
```

Accessing a Collection via an Iterator

To cycle through the elements in a collection we have two ways

1. Using an Iterator

For example, you might want to display each element. One way to do this is to employ an *iterator*, which is an object that implements either the **Iterator** or the **ListIterator** interface. **Iterator** enables you to cycle through a collection,obtaining or removing elements. **ListIterator** extends **Iterator** to allow bidirectional traversal of a list, and the modification of elements. **Iterator** and **ListIterator** are generic interfaces which are declared as shown here:

interface Iterator<E>
interface ListIterator<E>

By using this iterator object, you can access each element in the collection, one element at a time. In general, to use an iterator to cycle through the contents of a collection, follow these steps:

1. **Obtain an iterator to the start of the collection by calling the collection's iterator( ) method.**
2. **Set up a loop that makes a call to hasNext( ). Have the loop iterate as long as hasNext( ) returns true.**
2. **Within the loop, obtain each element by calling next( ).**

For collections that implement **List**, you can also obtain an iterator by calling **listIterator( )**. As explained, a list iterator gives you the ability to access the collection in either the forward or backward direction and lets you modify an element. Otherwise, **ListIterator** is used just like **Iterator**.

Ex 3:

```
package collection;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
public class Ex_1 {
    public static void main(String[] args) {
        List<Integer> l = new ArrayList();
        l.add(1);
        l.add(2);
        l.add(3);
        Iterator i = l.iterator();
        //without loop
        System.out.println(i.next());
        System.out.println(i.next());
        System.out.println(i.next());
    }}
```

```
C:\Users\Yasir-PC\.jdks\openjdk-22.0.2\bin\java.exe "-
1
2
3
```

Ex 4:

```java
package collection;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
public class Ex_1 {
    public static void main(String[] args) {
        List<Integer> l = new ArrayList();
        l.add(1);
        l.add(2);
        l.add(3);
        Iterator<Integer> i = l.iterator();//you can also specify type
        //with loop
        while(i.hasNext())
            System.out.println(i.next());
    }}
```

```
C:\Users\Yasir-PC\.jdks\openjdk-22.0.2\bin\java.exe "-
1
2
3
```

ListIterator for modification

```java
package collection;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.ListIterator;
public class Ex_1 {
    public static void main(String[] args) {
        List<String> l = new ArrayList();
        l.add("A");
```

```
        l.add("B");
        //modify objects using listiterator
        ListIterator<String> i = l.listIterator();
        while(i.hasNext()) {
            String e = i.next();
            i.set(e + "+");
        }
        //printing
        i = l.listIterator();
        while(i.hasNext()) {
            System.out.println(i.next());
        }
    }}
```

```
C:\Users\Yasir-PC\.jdks\openjdk-22.0.2\bin\java.exe "-
A+
B+
```

ListIterator for Backward display

```
package collection;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.ListIterator;
public class Ex_1 {
    public static void main(String[] args) {
        List<String> l = new ArrayList();
        l.add("A");
        l.add("B");
        //modify objects using listiterator
        ListIterator<String> i = l.listIterator();
        while(i.hasNext()) {
            String e = i.next();
            i.set(e + "+");
        }
        //printing
        i = l.listIterator();
        while(i.hasNext()) {
            System.out.println(i.next());
        }
        System.out.println("Backward");
        while(i.hasPrevious()) {
            System.out.println(i.previous());
        }
    }}
```

```
C:\Users\Yasir-PC\.jdks\openjdk-22.0.2\bin\java.exe "-
A+
B+
Backward
B+
A+
```

2.The For-Each Alternative to Iterators

If you won't be modifying the contents of a collection or obtaining elements in reverse order, then the for-each version of the **for** loop is often a more convenient alternative to cycling through a collection than is using an iterator. Recall that the **for** can cycle through any collection of objects that implement the **Iterable** interface. Because all of the collection classes implement this interface, they can all be operated upon by the **for**.

```java
package collection;
import java.util.ArrayList;
class ForEachDemo {

    public static void main(String[] args) {
        ArrayList<Integer> vals = new ArrayList<Integer>();

        vals.add(1);
        vals.add(2);
        vals.add(3);
        vals.add(4);
        vals.add(5);

        System.out.print("Contents of vals: ");
        for (int v : vals)
            System.out.print(v + " ");

        System.out.println();

        int sum = 0;
        for (int v : vals)
            sum += v;

        System.out.println("Sum of values: " + sum);
    }
}
```

```
C:\Users\Yasir-PC\.jdks\openjdk-22.0.2\bin\java.exe "-
Contents of vals: 1 2 3 4 5
Sum of values: 15
```

As you can see, the **for** loop is substantially shorter and simpler to use than the iterator-based approach. However, it can only be used to cycle through a collection in the forward direction, and you can't modify the contents of the collection.

2. Java Vector

Vector is like the *dynamic array* which can grow or shrink its size. Unlike array, we can store n-number of elements in it as there is no size limit. It is a part of Java Collection framework since Java 1.2. It is found in the java.util package and implements the *List* interface, so we can use all the methods of List interface here.

It is similar to the ArrayList, but it consumes more memory than ArrayList

Empty vector has initial size of 10

```
package collection;

import java.util.Vector;
class vector {
    public static void main(String[] args) {
        Vector<Integer> s = new Vector<>();
        System.out.println(s.capacity());
    }}
```

```
C:\Users\Yasir-PC\.jdks\openjdk-22.0.2\bin\java.exe "-
10
```

```
import java.util.Vector;
class vector {
    public static void main(String[] args) {
        Vector<Integer> s = new Vector<>();
```

```
        System.out.println(s.capacity());
        s.add(66);
        s.add(66);
        s.add(66);
        s.add(66);
        s.add(66);
        s.add(66);
        s.add(66);
        s.add(66);
        s.add(66);
        s.add(66);
        s.add(66);//size will be double when you insert 11th element
        System.out.println(s.capacity());
    }}
```

```
C:\Users\Yasir-PC\.jdks\openjdk-22.0.2\bin\java.exe "-
10
20
```

Empty arraylist has initial size of 0

```
package collection;

import java.util.ArrayList;
class vector {
    public static void main(String[] args) {
        ArrayList<Integer> s = new ArrayList<>();
        System.out.println(s.size());
        s.add(66);
        System.out.println(s.size());
        s.add(66);
        System.out.println(s.size());
        s.add(66);
        System.out.println(s.size());
        s.add(66);
        System.out.println(s.size());
        s.add(66);
        System.out.println(s.size());

    }}
```

```
C:\Users\Yasir-PC\.jdks\openjdk-22.0.2\bin\java.exe "-j
0
1
2
3
4
5
```

3. Java LinkedList class

Java LinkedList class uses a doubly linked list to store the elements.ArrayList and Vector uses Dynamic Array to store elements It provides a linked-list data structure. It inherits the AbstractList class and implements List and Deque interfaces.

The important points about Java LinkedList are:

- Java LinkedList class can contain duplicate elements.
- Java LinkedList class maintains insertion order.
- In Java LinkedList class, manipulation is fast because no shifting needs to occur.(when we insert/remove elements in between shifting happens upwards/downwards)
- Java LinkedList class can be used as a list, stack or queue.

Doubly Linked List

In the case of a doubly linked list, we can add or remove elements from both sides.

```
package collection;

import java.util.LinkedList;
class vector {
    public static void main(String[] args) {
        LinkedList<Integer> s = new LinkedList<>();
        System.out.println(s.size());
        s.add(66);
        System.out.println(s.size());
        s.add(66);
        System.out.println(s.size());
        s.add(66);
        System.out.println(s.size());
        s.add(66);
```

```
        System.out.println(s.size());
        s.add(66);
        System.out.println(s.size());
        for (int i : s) {
            System.out.println(i);
        }}}
```

```
C:\Users\Yasir-PC\.jdks\openjdk-22.0.2\bin\java.exe "-
0
1
2
3
4
5
66
66
66
66
66
```

4. Set in Java

The **set** is an interface available in the **java.util** package. The **set** interface extends the Collection interface. An unordered collection or list in which duplicates are not allowed is referred to as a **collection interface**. The set interface is used to create the mathematical set. The set interface use collection interface's methods to avoid the insertion of the same elements. **SortedSet** and **NavigableSet** are two interfaces that extend the set implementation.

Duplicates are not allowed in Set

```
import java.util.HashSet;
import java.util.Set;
public class hashset {
    public static void main(String[] args) {
        Set<Integer> s = new HashSet<>();
        s.add(66);
        s.add(66);
```

```
        s.add(77);
        s.add(66);
        for (int i : s) {
            System.out.println(i);
        }    }}
```

```
C:\Users\Yasir-PC\.jdks\openjdk-22.0.2\bin\java.exe "-
66
77
```

Set doesn't maintain sequence

HashSet follows hashing in which your values are storing inside the heap and hashing used some algorithm to fetch nearest value in heap

```
package collection;

import java.util.HashSet;
import java.util.Set;
public class hashset {
    public static void main(String[] args) {
        Set<Integer> s = new HashSet<>();
        s.add(66);
        s.add(55);
        s.add(88);
        s.add(99);
        for (int i : s) {
            System.out.println(i);
        }    }}
```

```
C:\Users\Yasir-PC\.jdks\openjdk-22.0.2\bin\java.exe "-
66
99
55
88
```

If you want to maintain sequence in ascending order than use TreeSet

```
package collection;

import java.util.Set;
import java.util.TreeSet;
public class treeset {
```

```
    public static void main(String[] args) {
        Set<Integer> s = new TreeSet<>();
        s.add(66);
        s.add(55);
        s.add(88);
        s.add(99);
        for (int i : s) {
            System.out.println(i);
        }    }}
```

```
C:\Users\Yasir-PC\.jdks\openjdk-22.0.2\bin\java.exe "-
55
66
88
99
```

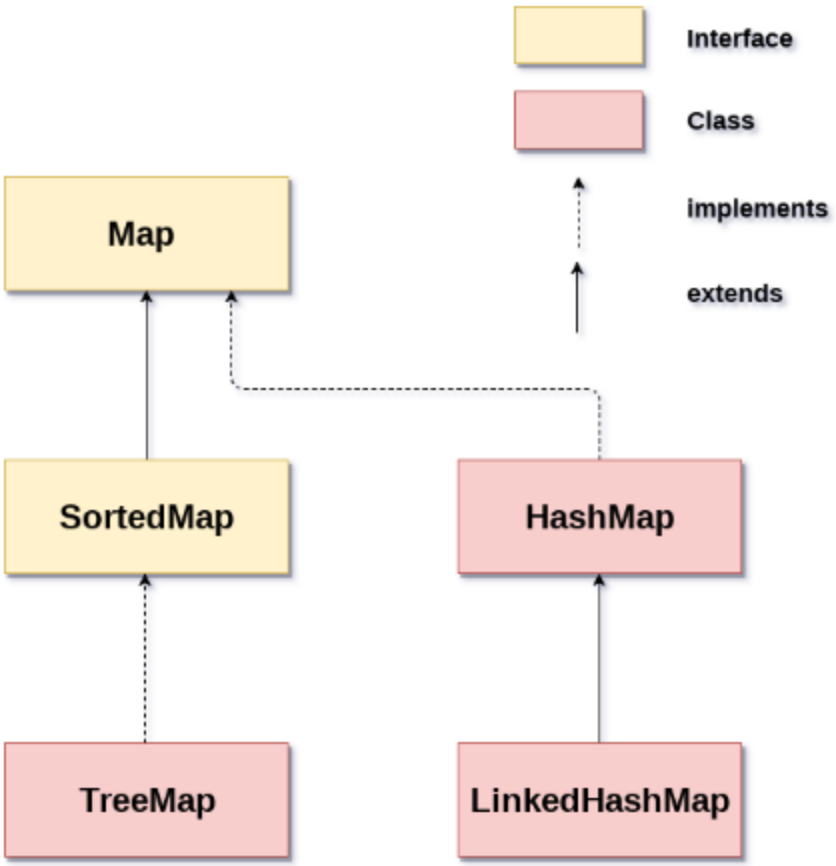
5. Java Map Interface

A map contains values on the basis of key, i.e. key and value pair. Each key and value pair is known as an entry. A Map contains unique keys.

A Map is useful if you have to search, update or delete elements on the basis of a key.

Java Map Hierarchy

There are two interfaces for implementing Map in java: Map and SortedMap, and three classes: HashMap, LinkedHashMap, and TreeMap. The hierarchy of Java Map is given below:



A Map doesn't allow duplicate keys, but you can have duplicate values. HashMap and LinkedHashMap allow null keys and values, but TreeMap doesn't allow any null key or value.

HashMap:doesn’t maintain sequence like HashSet

```
package collection;

import java.util.HashMap;
import java.util.Map;
class map {
    public static void main(String[] args) {
        Map<String, String> s = new HashMap<>();
        s.put("MyName" , "Yasir");
        System.out.println(s);
    }}
```

```
C:\Users\Yasir-PC\.jdks\openjdk-22.0.2\bin\java.exe "-
{MyName=Yasir}
```

HashMap:doesn’t maintain sequence like HashSet

```java
package collection;

import java.util.HashMap;
import java.util.Map;
class map {
    public static void main(String[] args) {
        Map<String, String> s = new HashMap<>();
        s.put("MyName" , "Yasir");
        s.put("LName" , "Arfat");
        s.put("Designation" , "Instructor");
        System.out.println(s);
    }}
```

```
C:\Users\Yasir-PC\.jdks\openjdk-22.0.2\bin\java.exe "-
{Designation=Instructor, MyName=Yasir, LName=Arfat}
```

Search with key

V get(Object key)	This method returns the object that contains the value associated with the key.

```java
package collection;

import java.util.HashMap;
import java.util.Map;
class map {
    public static void main(String[] args) {
        Map<String, String> s = new HashMap<>();
        s.put("MyName" , "Yasir");
        System.out.println(s.get("MyName"));
    }}
```

```
C:\Users\Yasir-PC\.jdks\openjdk-22.0.2\bin\java.exe "-
Yasir
```

For any key which is not available it will give you null

```
import java.util.HashMap;
import java.util.Map;
class map {
    public static void main(String[] args) {
        Map<String, String> s = new HashMap<>();
        s.put("MyName" , "Yasir");
        s.put("LName" , "Arfat");
        s.put("Designation" , "Instructor");
        System.out.println(s.get("SCD"));
    }}
```

```
C:\Users\Yasir-PC\.jdks\openjdk-22.0.2\bin\java.exe "-
null

Process finished with exit code 0
```

We have 2 ways for Printing all keys and values

1.using For each loop
We are using methods of Set to store keys

Set keySet()	It returns the Set view containing all the keys.

keyset() will return all the keys present in the map

```
package collection;

import java.util.HashMap;
import java.util.Map;
import java.util.Set;
class map {
    public static void main(String[] args) {
        Map<String, String> s = new HashMap<>();
        s.put("MyName" , "Yasir");
        s.put("LName" , "Arfat");
        Set<String> s1 = s.keySet();
        for(String k : s1)
```

```
            System.out.println(k + "   " + s.get(k));
    }}
```

2.Map.Entry Interface

Entry is the subinterface of Map. So we will be accessed it by Map.Entry name. It returns

```
C:\Users\Yasir-PC\.jdks\openjdk-22.0.2\bin\java.exe
MyName   Yasir
LName   Arfat
```

a collection-view of the map, whose elements are of this class. It provides methods to get key and value.

The **entrySet( )** method declared by the Map interface returns a Set containing the map entries. Each of these set elements is a Map.Entry object.

What is entry?

Key and Value pair makes one entry

```
import java.util.HashMap;
import java.util.Map;
class map {
    public static void main(String[] args) {
        Map<Integer,String> map=new HashMap();
        map.put(1,"Yasir");
        map.put(2,"Shafiq");
        map.put(3,"Obaid");
        map.put(4,"Ali");
        //Traversing Map
        for(Map.Entry m:map.entrySet()){
            System.out.println(m.getKey()+" "+m.getValue());
        }
    }}
```

Or you can use set to store keys and values

```
package collection;

import java.util.HashMap;
import java.util.Map;
import java.util.Set;
class map {
    public static void main(String[] args) {
        Map<Integer,String> map=new HashMap();
        map.put(1,"Yasir");
        map.put(2,"Shafiq");
        map.put(3,"Obaid");
        map.put(4,"Ali");
        Set<Map.Entry<Integer,String>> pair = map.entrySet();
        //Traversing Map
        for(Map.Entry m:pair){
            System.out.println(m.getKey()+" "+m.getValue());
        }
    }}
```

```
C:\Users\Yasir-PC\.jdks\openjdk-22.0.2\bin\java.exe "-j
1 Yasir
2 Shafiq
3 Obaid
4 Ali
```

We cannot repeat/duplicate keys

```
package collection;

import java.util.HashMap;
import java.util.Map;
import java.util.Set;
class map {
    public static void main(String[] args) {
        Map<String, String> s = new HashMap<>();
        s.put("MyName" , "Yasir");
```

```
        s.put("LName" , "Arfat");
        s.put("MyName" , "Ali");
        s.put("MyName" , "Sahfiq");
        Set<String> s1 = s.keySet();
        for(String k : s1)
            System.out.println(k + "  " + s.get(k));
    }}
```

```
C:\Users\Yasir-PC\.jdks\openjdk-22.0.2\bin\java.exe "-
MyName  Sahfiq
LName  Arfat
```

We can repeat/duplicate values

```
package collection;

import java.util.HashMap;
import java.util.Map;
import java.util.Set;
class map {
    public static void main(String[] args) {
        Map<String, String> s = new HashMap<>();
        s.put("MyName" , "Yasir");
        s.put("LName" , "Arfat");
        s.put("MyName" , "Yasir");
        s.put("Name" , "Arfat");
        Set<String> s1 = s.keySet();
        for(String k : s1)
            System.out.println(k + "  " + s.get(k));
    }}
```

```
C:\Users\Yasir-PC\.jdks\openjdk-22.0.2\bin\java.exe "-
MyName  Yasir
LName  Arfat
Name  Arfat
```

6. Collections in swing

Student Management System in Java Swing

Here in This Example we are Creating Student Management System with the help of collection (Arraylist) in Swing. Follow The Steps Given below.

Create a New Project and Add the following Code to your Project

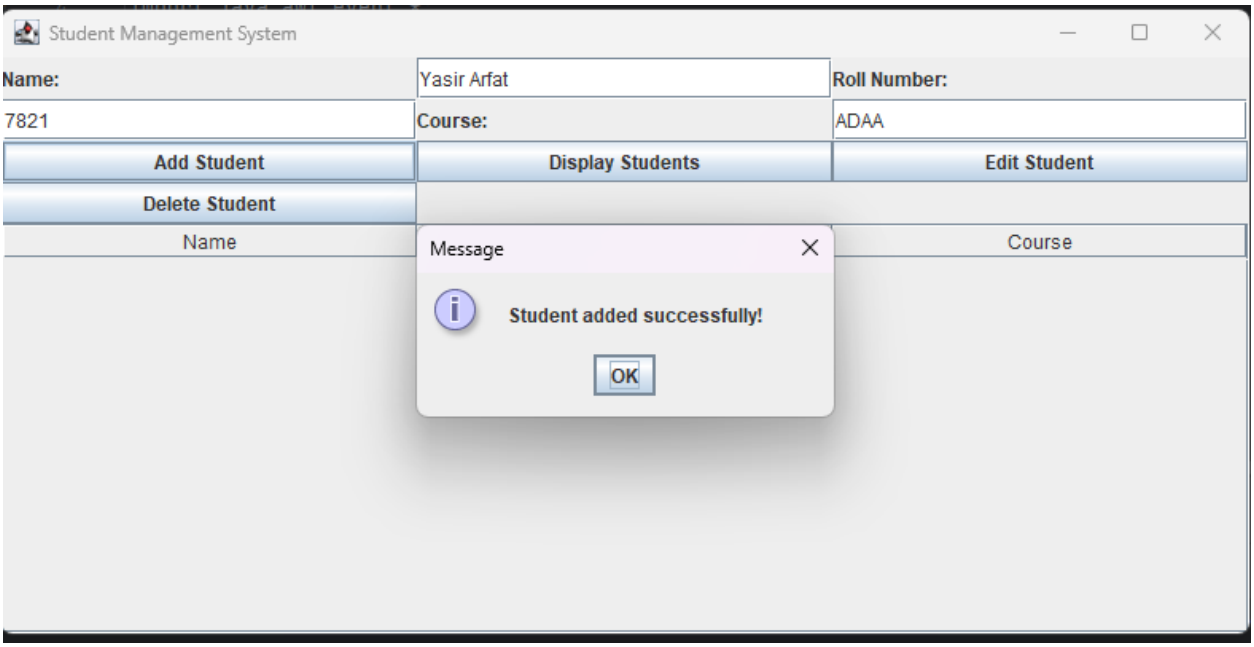
```
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.awt.event.*;
import java.util.ArrayList;

class StudentManagementApp extends JFrame {

    // Collection to hold students
    private ArrayList<Student> studentList = new ArrayList();

    // UI components
    private JTextField nameField, rollNumberField, courseField;
    private JButton addButton, displayButton, editButton, deleteButton;
    private JTable studentTable;
    private DefaultTableModel tableModel;

    // Currently selected student index (for editing)
    private int selectedRow = -1;

    public StudentManagementApp() {
        // Setting up the frame
        setTitle("Student Management System");
        setSize(800, 400);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);

        // Panel for input fields
        JPanel inputPanel = new JPanel(new GridLayout(4, 2));

        inputPanel.add(new JLabel("Name:"));
        nameField = new JTextField();
        inputPanel.add(nameField);

        inputPanel.add(new JLabel("Roll Number:"));
        rollNumberField = new JTextField();
        inputPanel.add(rollNumberField);

        inputPanel.add(new JLabel("Course:"));
        courseField = new JTextField();
        inputPanel.add(courseField);

        addButton = new JButton("Add Student");
        inputPanel.add(addButton);
```

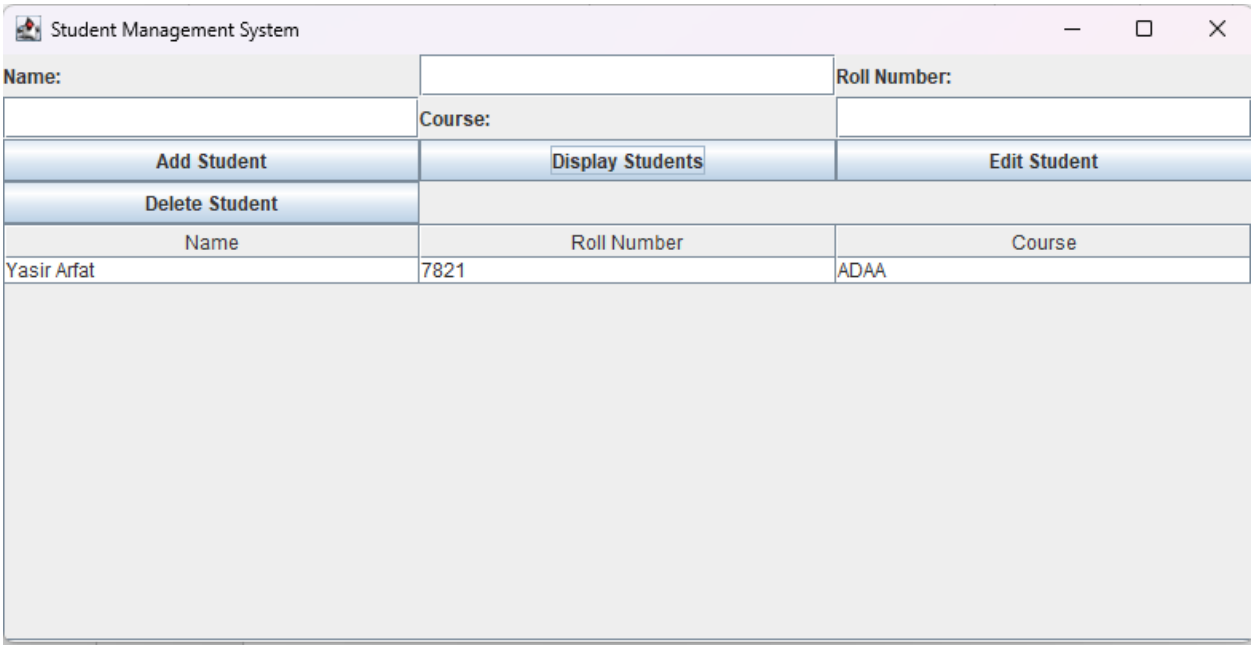
```
        displayButton = new JButton("Display Students");
        inputPanel.add(displayButton);

        editButton = new JButton("Edit Student");
        inputPanel.add(editButton);

        deleteButton = new JButton("Delete Student");
        inputPanel.add(deleteButton);

        add(inputPanel, BorderLayout.NORTH);

        // Table to display students
        tableModel = new DefaultTableModel(new Object[]{"Name", "Roll
Number", "Course"}, 0);
        studentTable = new JTable(tableModel);
        add(new JScrollPane(studentTable), BorderLayout.CENTER);

        // Button actions
        addButton.addActionListener(e -> addStudent());
        displayButton.addActionListener(e -> displayStudents());
        editButton.addActionListener(e -> editStudent());
        deleteButton.addActionListener(e -> deleteStudent());

        // Table row selection listener
        studentTable.getSelectionModel().addListSelectionListener(e -> {
            if (!e.getValueIsAdjusting()) {
                selectedRow = studentTable.getSelectedRow();
                if (selectedRow >= 0) {
                    // Load selected student data into fields
                    nameField.setText(studentTable.getValueAt(selectedRow,
0).toString());

rollNumberField.setText(studentTable.getValueAt(selectedRow, 1).toString());
                    courseField.setText(studentTable.getValueAt(selectedRow,
2).toString());
                }
            }
        });
    }

    // Method to add a student to the ArrayList
    private void addStudent() {
        String name = nameField.getText();
        String rollNumber = rollNumberField.getText();
        String course = courseField.getText();

        if (name.isEmpty() || rollNumber.isEmpty() || course.isEmpty()) {
            JOptionPane.showMessageDialog(this, "Please fill all fields!",
"Error", JOptionPane.ERROR_MESSAGE);
            return;
        }

        Student student = new Student(name, rollNumber, course);
        studentList.add(student);
        JOptionPane.showMessageDialog(this, "Student added successfully!");
```

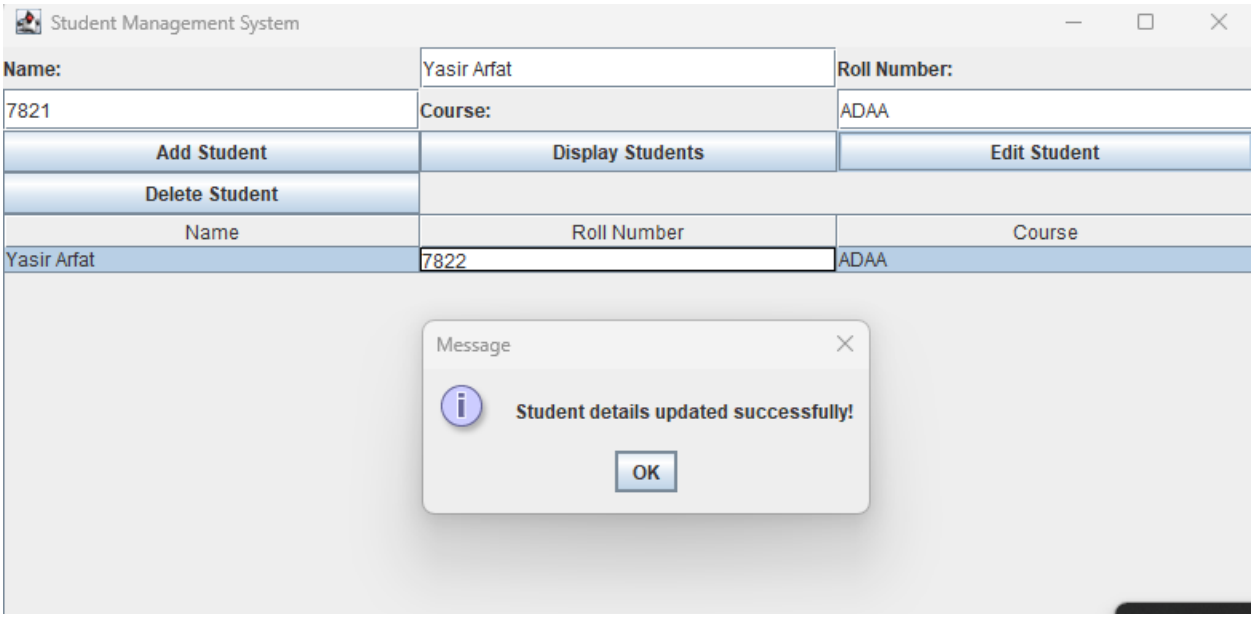
```
        // Clear fields after adding
        clearFields();
    }

    // Method to display students in the JTable
    private void displayStudents() {
        tableModel.setRowCount(0); // Clear existing data

        for (Student student : studentList) {
            tableModel.addRow(new Object[]{student.getName(),
student.getRollNumber(), student.getCourse()});
        }
    }

    // Method to edit a selected student
    private void editStudent() {
        if (selectedRow >= 0) {
            String name = nameField.getText();
            String rollNumber = rollNumberField.getText();
            String course = courseField.getText();

            if (name.isEmpty() || rollNumber.isEmpty() || course.isEmpty()) {
                JOptionPane.showMessageDialog(this, "Please fill all
fields!", "Error", JOptionPane.ERROR_MESSAGE);
                return;
            }

            // Update the student in the ArrayList
            Student student = studentList.get(selectedRow);
            student.setName(name);
            student.setRollNumber(rollNumber);
            student.setCourse(course);

            // Update the JTable row
            tableModel.setValueAt(name, selectedRow, 0);
            tableModel.setValueAt(rollNumber, selectedRow, 1);
            tableModel.setValueAt(course, selectedRow, 2);

            JOptionPane.showMessageDialog(this, "Student details updated
successfully!");
            clearFields();
        } else {
            JOptionPane.showMessageDialog(this, "Please select a student to
edit!", "Error", JOptionPane.ERROR_MESSAGE);
        }
    }

    // Method to delete a selected student
    private void deleteStudent() {
        if (selectedRow >= 0) {
            // Remove the student from the ArrayList
            studentList.remove(selectedRow);
            // Remove the row from the JTable
            tableModel.removeRow(selectedRow);
            JOptionPane.showMessageDialog(this, "Student deleted
successfully!");
            clearFields();
```

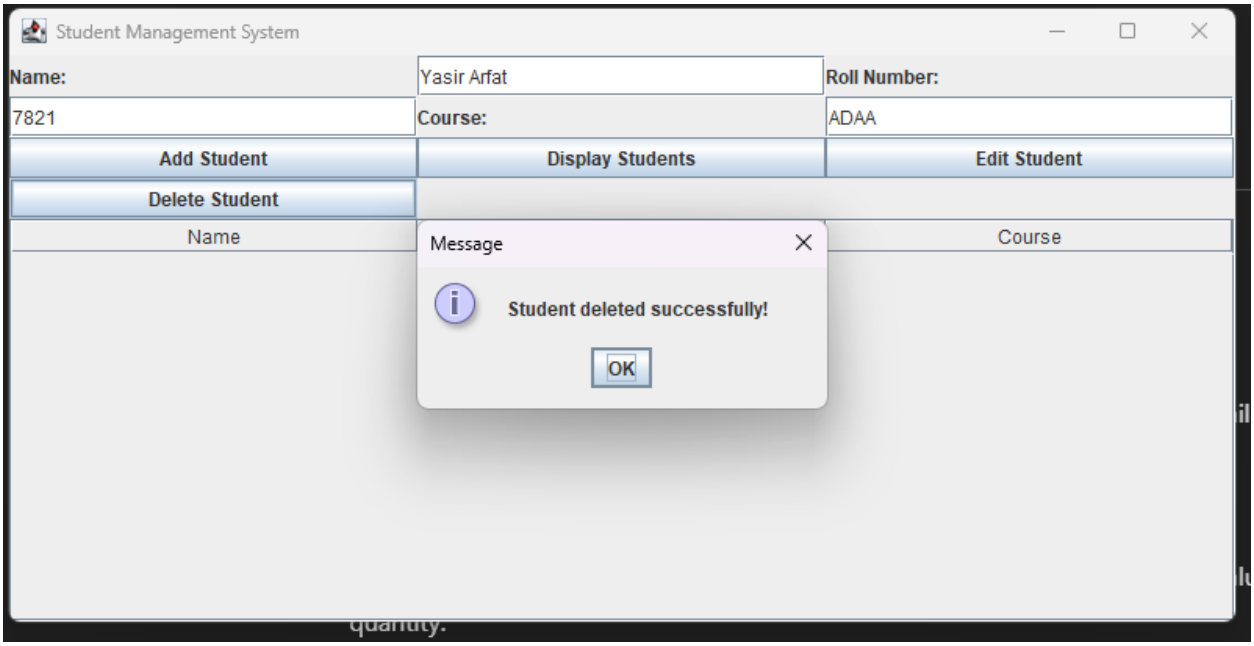
```
        } else {
            JOptionPane.showMessageDialog(this, "Please select a student to
delete!", "Error", JOptionPane.ERROR_MESSAGE);
        }
    }

    // Helper method to clear input fields
    private void clearFields() {
        nameField.setText("");
        rollNumberField.setText("");
        courseField.setText("");
        selectedRow = -1;  // Reset the selected row index
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> new
StudentManagementApp().setVisible(true));
    }
}

// Student class
class Student {
    private String name;
    private String rollNumber;
    private String course;

    public Student(String name, String rollNumber, String course) {
        this.name = name;
        this.rollNumber = rollNumber;
        this.course = course;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getRollNumber() {
        return rollNumber;
    }

    public void setRollNumber(String rollNumber) {
        this.rollNumber = rollNumber;
    }

    public String getCourse() {
        return course;
    }

    public void setCourse(String course) {
        this.course = course;
    }
}
```

Output


Student Management System
Name:
Yasir Arfat
Roll Number:
7821
Course:
ADAA
Add Student
Display Students
Edit Student
Delete Student
Name
Course
Message
Student added successfully!
OK


Student Management System
Name:
Roll Number:
Course:
Add Student
Display Students
Edit Student
Delete Student
Name
Roll Number
Course
Yasir Arfat
7821
ADAA


Student Management System
Name:
Yasir Arfat
Roll Number:
7821
Course:
ADAA
Add Student
Display Students
Edit Student
Delete Student
Name
Roll Number
Course
Yasir Arfat
7822
ADAA
Message
Student details updated successfully!
OK


Student Management System
Name:
Yasir Arfat
Roll Number:
7821
Course:
ADAA
Add Student
Display Students
Edit Student
Delete Student
Name
Course
Message
Student deleted successfully!
OK