Team project :

الاسم : ادم محمد شاكر عبد الفتاح

ID: 2401243069

الاسم : احمد السيد احمد عبد اللطيف

ID : 2401243656

الاسم : اسلام عبد العال محمد

ID : 2401249211

الاسم :احمد طارق احمد علي

ID : 2401244361

الاسم : حازم ايهاب محمد عبدالحليم

ID : 2401245291

الاسم : احمد رمضان محمد محمود

ID: 2401245405

الاسم : ادم محمد شاكر

(Stack)

```
public class EnrollmentAction {


int studentId;

   int courseId;

   boolean isEnroll; //if enroll then true
if delete then false
```

```
   public EnrollmentAction(int
studentId, int courseId, boolean
isEnroll) {
       this.studentId = studentId;
       this.courseId = courseId;
       this.isEnroll = isEnroll;
   }
}
```

Methods :

_Undo

```
public void undo() {
   if (undoStack.isEmpty()) {
        System.out.println("Nothing to
undo.");
        return;
    }

    EnrollmentAction lastAction =
undoStack.pop();
    if (lastAction.isEnroll) {
        // لو كانت تسجيل هنلغيه

remove_enrollment(lastAction.student
Id, lastAction.courseId);
    } else {
        // لو كانت إزالة هنرجع التسجيل
```

```
enrollStudent(lastAction.studentId,
lastAction.courseId);
    }
    redoStack.push(lastAction);
}
```

_Redo

```
public void redo() {
    if (redoStack.isEmpty()) {
        System.out.println("Nothing to
redo.");
        return;
    }
    EnrollmentAction action =
redoStack.pop();
    if (action.isEnroll) {
        enrollStudent(action.studentId,
action.courseId);
    } else {

remove_enrollment(action.studentId,
action.courseId);
    }
    undoStack.push(action);
}
```

الاسم : احمد السيد احمد عبد اللطيف

Methods:

_ Search student

```
  public Nodestudent
searchStudent(int studentId) {
        Nodestudent current = head;
        while (current != null) {
           if (current.studentId ==
studentId)   return current; // Found
           current = current.next; // Move
to next node
```

```
        }
        return null; // Not found after
traversing entire list
    }
_Search course
    public Nodecourse
searchCourse(int courseId) {
        Nodecourse current = head;
        while (current != null) {
            if (current.courseId ==
courseId)   return current; // Found
            current = current.next; // Move
to next node
        }
        return null;      // Not found after
traversing entire list
    }
```

_Last student

```
   public void laststudent(){
    System.out.println("The last
student added:
"+students.laststudent());
  }
```

_Last course

```
  public void lastcourse(){
        System.out.println("The last
course added: "+courses.lastcourse());
    }

_Enrollment
   public void enrollStudent(int
studentId,int courseId){
        Nodestudent student
=students.searchStudent(studentId);
        Nodecourse course
=courses.searchCourse(courseId);
        //validate existence
        if (student ==null||course ==null) {
            System.out.println("Student or
course not found");
            return;
        }

        //check student limit
        if (course.studentCount >=30) {
            System.out.println("Course
"+courseId+" is full");
            return;
        }
        //add course to student's list
        Nodecourse newCourseNode
=new Nodecourse(courseId);
```

```
      newCourseNode.next
=student.enrolledCourse;
      student.enrolledCourse
=newCourseNode;
//student.enrolledcourse represent the
head pointer.


      //add student to course's list.
      Nodestudent newStudentNode
=new Nodestudent(studentId);
      newStudentNode.next
=course.enrolledStudent;
      course.enrolledStudent
=newStudentNode;  //note
```

course.enrolledstudent represent the head pointer.

```
    //update counters
    student.enrolledCouresCount++;
    course.studentCount++;

  undoStack.push(new EnrollmentAction(studentId, courseId, true));
redoStack.clear(); // لو عملت حاجة جديدة، نمسح الـ redo القديم
```

```
  }
_List student in course


  public void ListStudentinCourse(int
courseId){
      Nodecourse course
=courses.searchCourse(courseId);
      if (course == null) {
         System.out.println("Course not
found");
         return;
      }
      Nodestudent current =
course.enrolledStudent;
```

```
        System.out.println("Students
enrolled in course "+course.courseId+"
:");
        while (current !=null) {
            System.out.println("Student ID:
"+current.studentId);
            current =current.next;
        }


}
```

اسلام عبد العال محمد

Methods:

List_of_Courses_For_Student

```
public void
list_of_Courses_For_Student(int
studentId) {
   Nodestudent student =
students.searchStudent(studentId);

   if (student == null) {
       System.out.println("Student not found");
       return;
   }

   Nodecourse currentCourse = student.enrolledCourse;

   if (currentCourse == null) {
       System.out.println("Student " + studentId + " is not enrolled in any courses.");
       return;
   }
```

```
   System.out.println("Courses for
student " + studentId + ":");
   while (currentCourse != null) {
      System.out.println("Course ID: " +
currentCourse.courseId);
      currentCourse =
currentCourse.next;
  }
   }
```

_Remove enrollment

```
public void remove_enrollment(int
studentId,int courseId){
    Nodestudent student
=students.searchStudent(studentId);
    Nodecourse course
=courses.searchCourse(courseId);
    //validate existence
    if (student ==null||course ==null) {
      System.out.println("Student or
course not found");
      return;
    }
    Nodecourse
currentcourse=student.enrolledCours
e;
```

```
        //remove course from student's list
        //if student in the first node
        if (student.enrolledCourse != null
&& student.enrolledCourse.courseId
== courseId) {
    student.enrolledCourse =
student.enrolledCourse.next;
    student.enrolledCouresCount--;
}
        else{
        while (currentcourse != null &&
currentcourse.next != null) {
            if (currentcourse.next.courseId
== courseId) {
```

```
            currentcourse.next =
currentcourse.next.next;
            student.enrolledCouresCount-
-;
            break; }
          currentcourse =
currentcourse.next;
      }
      }
   // remove student from course's list
   //if student in the first node
   if (course.enrolledStudent != null &&
course.enrolledStudent.studentId ==
studentId) {
        course.enrolledStudent =
course.enrolledStudent.next;
        course.studentCount--;
    } else {
         Nodestudent current =
course.enrolledStudent;
        while (current != null &&
current.next != null) {
            if (current.next.studentId ==
studentId) {
                current.next =
current.next.next;
                course.studentCount--;
                break;
        }
        current = current.next;
      }
   }

 System.out.println("Student " +
studentId + " deleted from course " +
courseId);

   undoStack.push(new
EnrollmentAction(studentId, courseId,
false));
redoStack.clear();
```

```
   }
```

احمد طارق احمد علي

Methods :

_Add student

```
   public void addstudent(int id){
      students.addstudent(id);
   }
```

_Remove student:

```
public void removestudent(int id){
      students.removestudent(id);
   }
```

حازم ايهاب محمد عبدالحليم

Methods :

_Add course

```
public void addcourse(int id){
      courses.addcourse(id);
   }
```

_Remove course :

```
public void removecourse(int id){
      courses.removecourse(id);
   }
```

احمد رمضان محمد محمود

Methods :

_Sort students in course

```
Nodestudent currentstudent =
students.searchStudent(studentId);
    if (currentstudent == null ||
currentstudent.enrolledCourse ==
null) {
        return; // No student or no courses
to sort
    }
    Nodecourse mainhead =
currentstudent.enrolledCourse;
    currentstudent.enrolledCourse =
null; // Detach the original list to build
a new sorted one

    while (mainhead != null) {
        // Find the node with the minimum
courseid in the remaining list
        Nodecourse minPrev = null; //
Previous node of the minimum node
        Nodecourse min = mainhead; //
Minimum node
        Nodecourse prev = mainhead; //
For traversal
```

```
    Nodecourse current =
mainhead.next;


    while (current != null) {
      if (current.courseId <
min.courseId) {
        min = current;
        minPrev = prev;
      }
      prev = current;
      current = current.next;
    }
    // Remove the min node from the
original list
    if (min == mainhead) {
      mainhead = mainhead.next;
    } else {
      minPrev.next = min.next;
    }

    // Append the min node to the new
sorted list
    if (currentstudent.enrolledCourse
== null) {
      currentstudent.enrolledCourse =
min;
```

```
        min.next = null;
      } else {
        // Find the last node in the new
list
        Nodecourse last =
currentstudent.enrolledCourse;
        while (last.next != null) {
          last = last.next;
        }
        last.next = min;
        min.next = null;
      }
    }
}
```

_Sort courses in student

```
public void
sort_students_in_course(int courseId)
{
   Nodecourse currentcourse =
courses.searchCourse(courseId);
   if (currentcourse == null ||
currentcourse.enrolledStudent == null)
{
      return; // No course or no students
to sort
   }

   Nodestudent mainhead =
currentcourse.enrolledStudent;
```

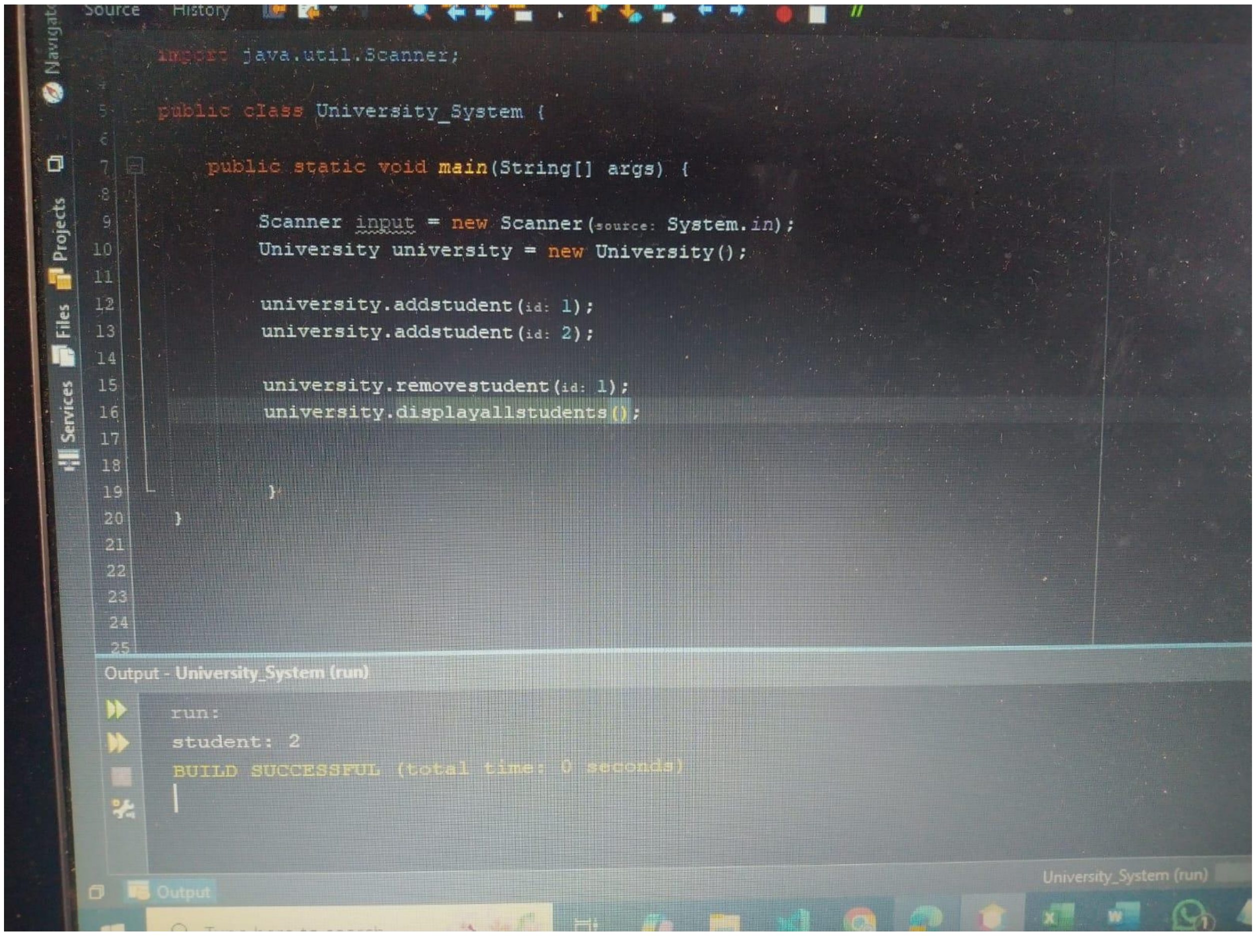
```
    currentcourse.enrolledStudent =
null; // Detach the original list to build
a new sorted one


    while (mainhead != null) {
        // Find the node with the minimum
studentID in the remaining list
        Nodestudent minPrev = null; //
Previous node of the minimum node
        Nodestudent min = mainhead; //
Minimum node
        Nodestudent prev = mainhead; //
For traversal
        Nodestudent current =
mainhead.next;
```

```
      while (current != null) {
         if (current.studentId <
min.studentId) {
            min = current;
            minPrev = prev;
         }
         prev = current;
         current = current.next;
      }

      // Remove the min node from the
original list
      if (min == mainhead) {
            mainhead = mainhead.next;
        } else {
            minPrev.next = min.next;
        }

        // Append the min node to the new
sorted list
        if (currentcourse.enrolledStudent
== null) {
            currentcourse.enrolledStudent =
min;
            min.next = null;
        } else {
```

```
        // Find the last node in the new
list
        Nodestudent last =
currentcourse.enrolledStudent;
        while (last.next != null) {
          last = last.next;
        }
        last.next = min;
        min.next = null;
      }
      }
    }
```
_Is full course

```
public boolean is_full_course(int
courseId){
   Nodecourse
currentcourse=courses.searchCourse(
courseId);
      if
(currentcourse.studentCount>30){
         System.out.println("This course
is full");
         return true;
      } else{
         System.out.println("This course
has "+(30-
currentcourse.studentCount)+" free
sets");
```

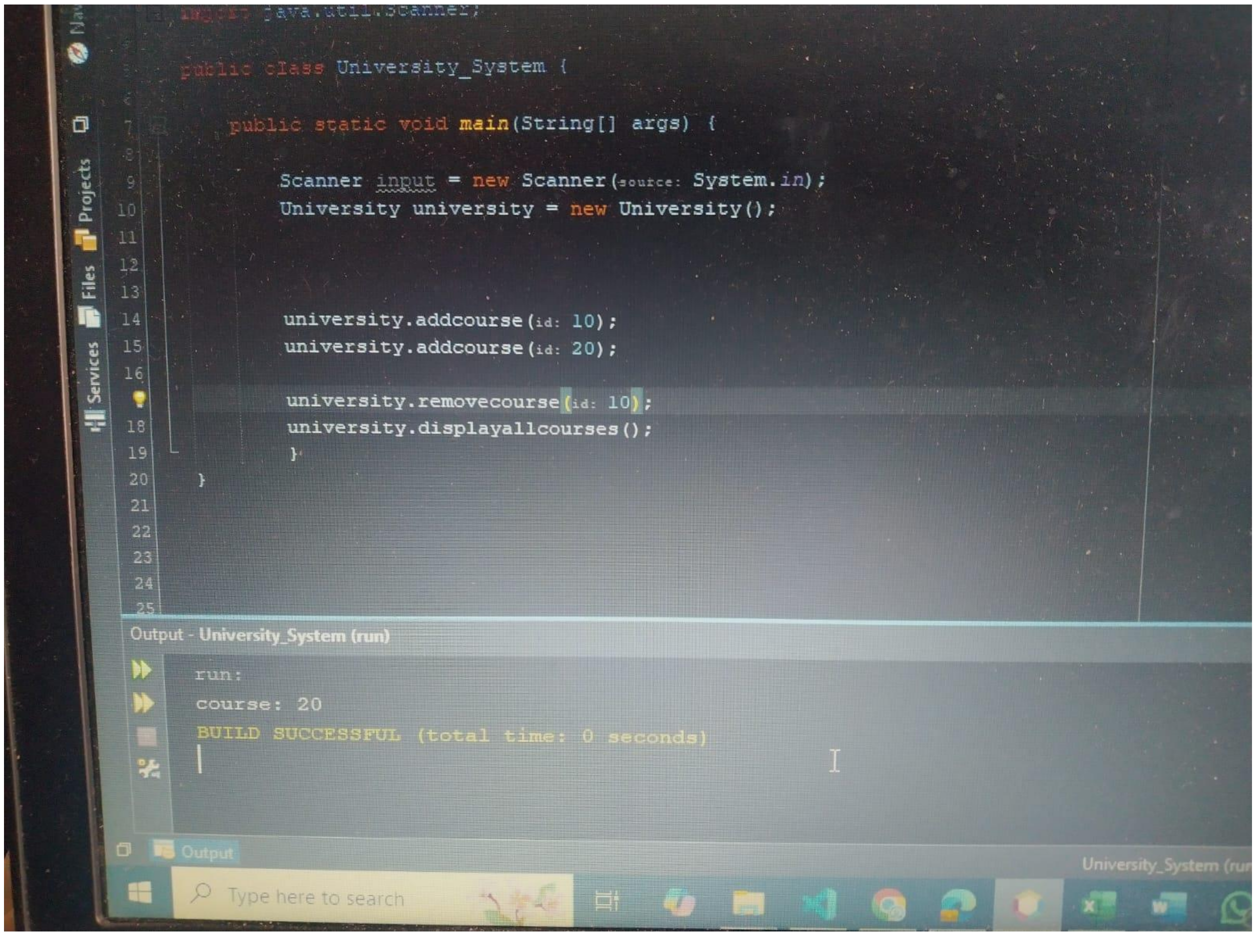
```
      return false;
      }


   }
_Is_normal student
public boolean is_normal_student(int
studentId){
   Nodestudent
currentstudent=students.searchStude
nt(studentId);
      if
(currentstudent.enrolledCouresCount
<2||currentstudent.enrolledCouresCo
unt>7) {
        return false;
    }else{
    return true;
    }


}
```



```
import java.util.Scanner;

public class University_System {

    public static void main(String[] args) {

        Scanner input = new Scanner(source: System.in);
        University university = new University();

        university.addstudent(id: 1);
        university.addstudent(id: 2);

        university.removestudent(id: 1);
        university.displayallstudents();


    }
}
```

Output - University_System (run)

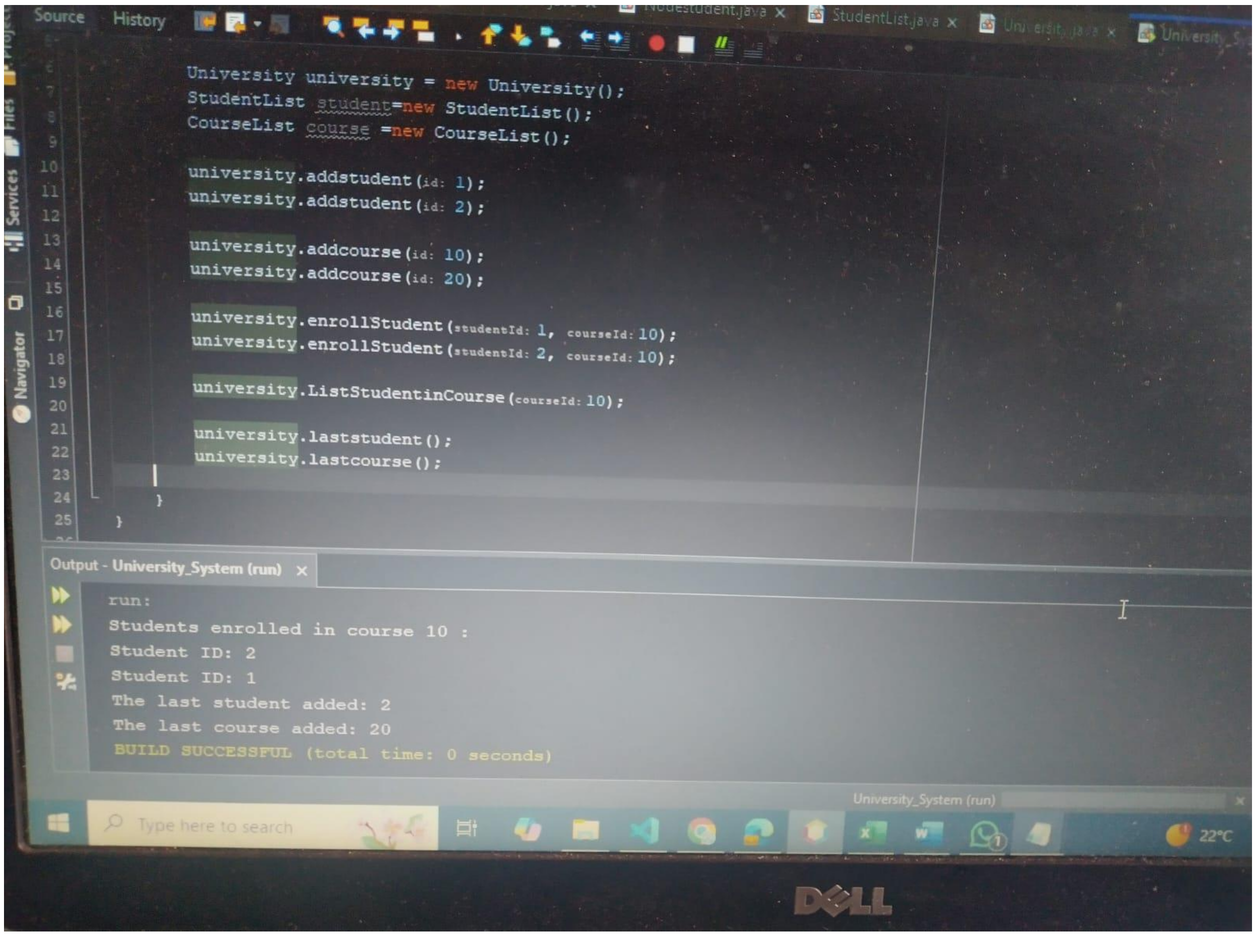
```
run:
student: 2
BUILD SUCCESSFUL (total time: 0 seconds)
```



```
import java.util.Scanner;

public class University_System {

    public static void main(String[] args) {

        Scanner input = new Scanner(source: System.in);
        University university = new University();



        university.addcourse(id: 10);
        university.addcourse(id: 20);

        university.removecourse(id: 10);
        university.displayallcourses();
        }
}
```

Output - University_System (run)

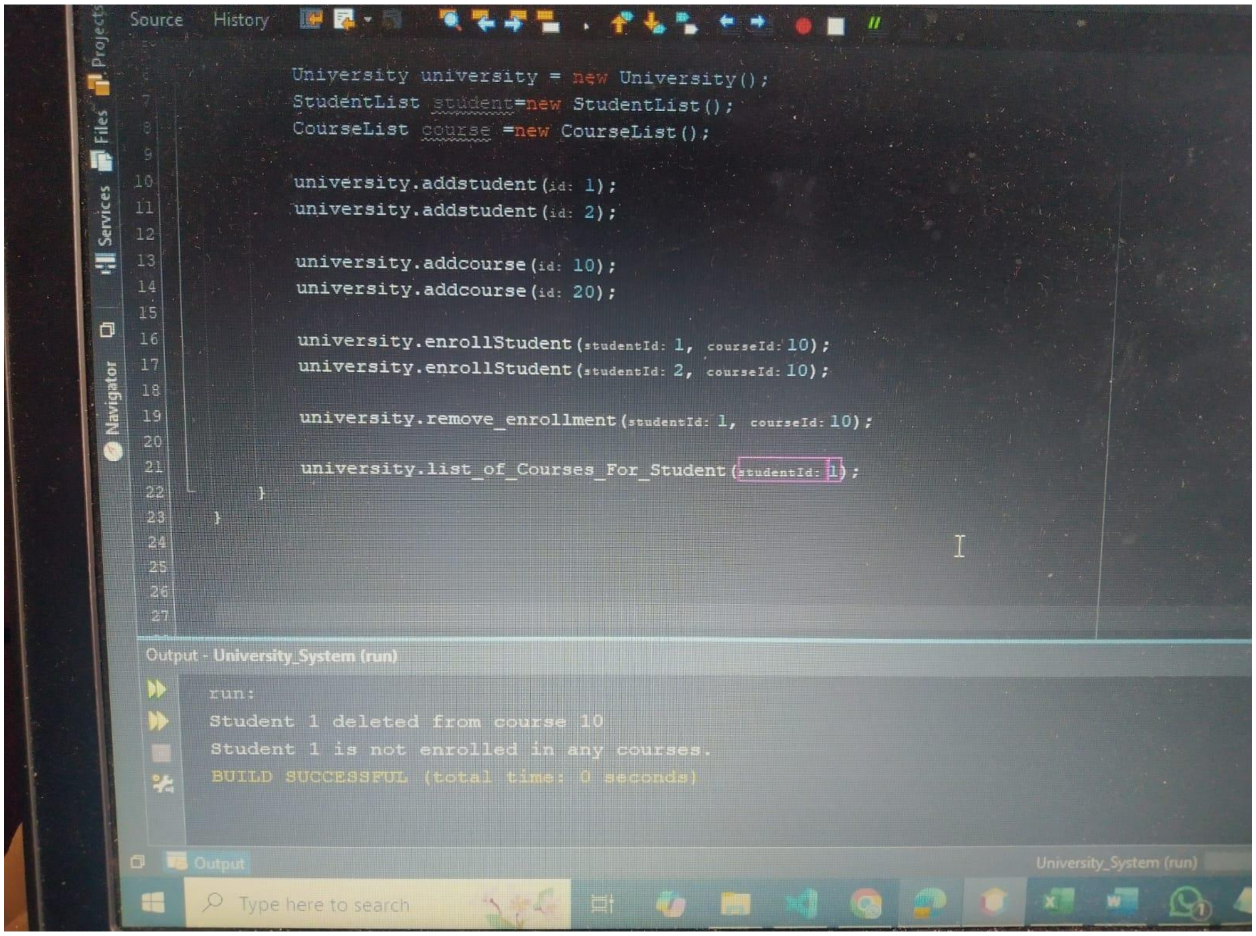
```
run:
course: 20
BUILD SUCCESSFUL (total time: 0 seconds)
```



```
        University university = new University();
        StudentList student=new StudentList();
        CourseList course =new CourseList();

        university.addstudent(id: 1);
        university.addstudent(id: 2);

        university.addcourse(id: 10);
        university.addcourse(id: 20);

        university.enrollStudent(studentId: 1, courseId: 10);
        university.enrollStudent(studentId: 2, courseId: 10);

        university.ListStudentinCourse(courseId: 10);

        university.laststudent();
        university.lastcourse();

    }
}
```

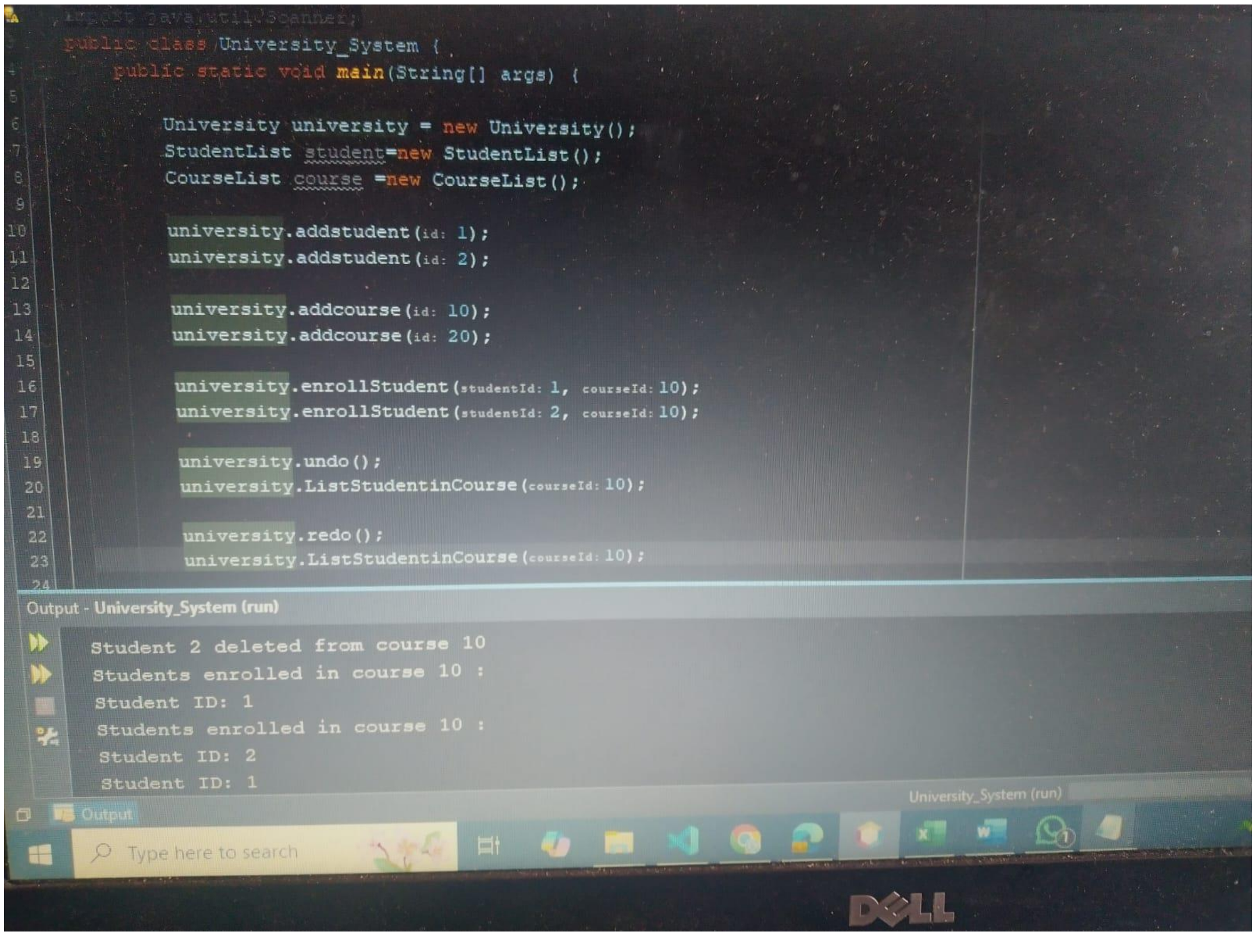
Output - University_System (run)

```
run:
Students enrolled in course 10 :
Student ID: 2
Student ID: 1
The last student added: 2
The last course added: 20
BUILD SUCCESSFUL (total time: 0 seconds)
```


```java
        University university = new University();
        StudentList student=new StudentList();
        CourseList course =new CourseList();

        university.addstudent(id: 1);
        university.addstudent(id: 2);

        university.addcourse(id: 10);
        university.addcourse(id: 20);

        university.enrollStudent(studentId: 1, courseId: 10);
        university.enrollStudent(studentId: 2, courseId: 10);

        university.remove_enrollment(studentId: 1, courseId: 10);

        university.list_of_Courses_For_Student(studentId: 1);
    }
}
```

Output - University_System (run)

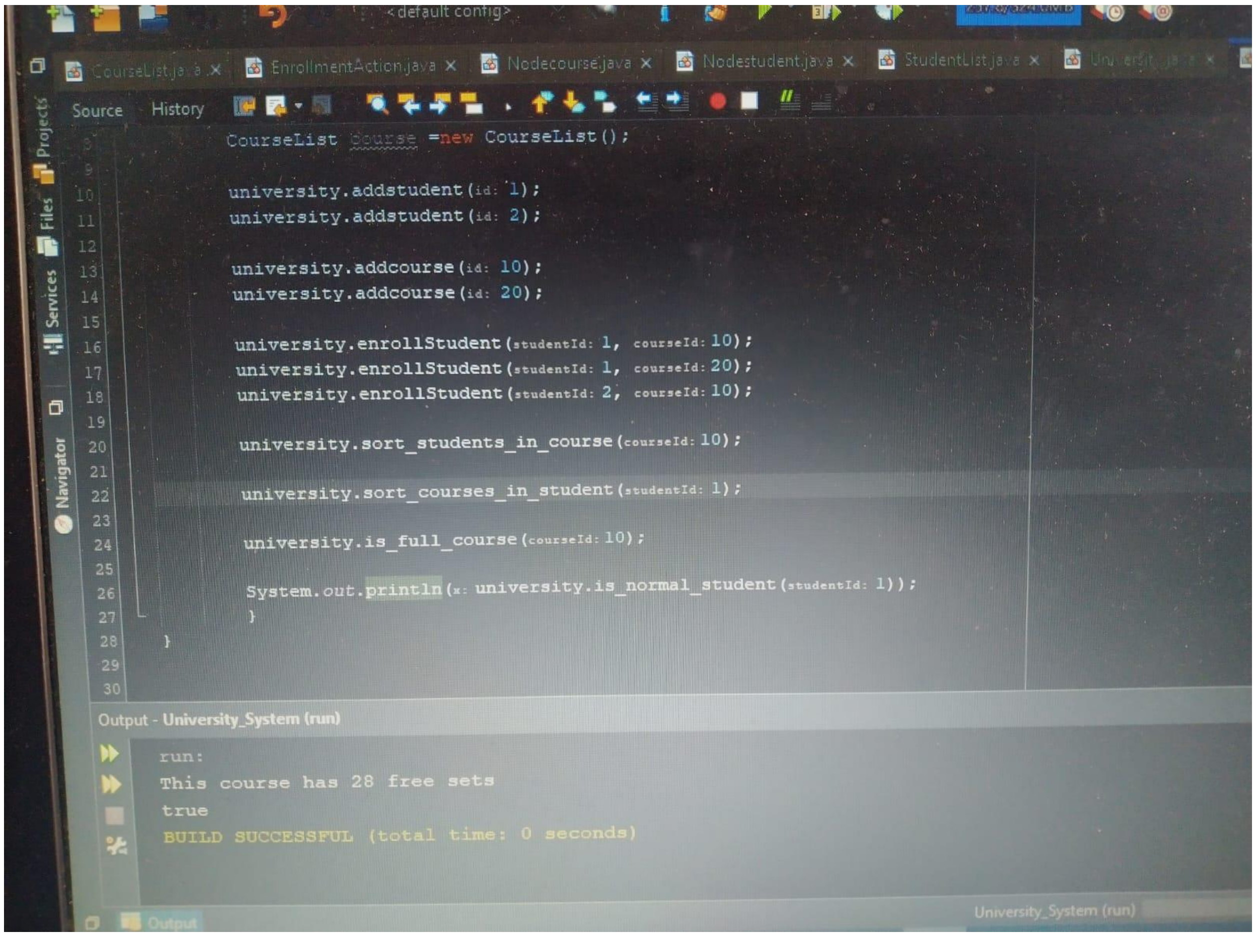
```
run:
Student 1 deleted from course 10
Student 1 is not enrolled in any courses.
BUILD SUCCESSFUL (total time: 0 seconds)
```



```java
import java.util.Scanner;
public class University_System {
    public static void main(String[] args) {

        University university = new University();
        StudentList student=new StudentList();
        CourseList course =new CourseList();

        university.addstudent(id: 1);
        university.addstudent(id: 2);

        university.addcourse(id: 10);
        university.addcourse(id: 20);

        university.enrollStudent(studentId: 1, courseId: 10);
        university.enrollStudent(studentId: 2, courseId: 10);

        university.undo();
        university.ListStudentinCourse(courseId: 10);

        university.redo();
        university.ListStudentinCourse(courseId: 10);
```

Output - University_System (run)

```
Student 2 deleted from course 10
Students enrolled in course 10 :
Student ID: 1
Students enrolled in course 10 :
Student ID: 2
Student ID: 1
```



```
        CourseList course =new CourseList();

        university.addstudent(id: 1);
        university.addstudent(id: 2);

        university.addcourse(id: 10);
        university.addcourse(id: 20);

        university.enrollStudent(studentId: 1, courseId: 10);
        university.enrollStudent(studentId: 1, courseId: 20);
        university.enrollStudent(studentId: 2, courseId: 10);

        university.sort_students_in_course(courseId: 10);

        university.sort_courses_in_student(studentId: 1);

        university.is_full_course(courseId: 10);

        System.out.println(x: university.is_normal_student(studentId: 1));
        }
}
```

Output - University_System (run)

```
run:
This course has 28 free sets
true
BUILD SUCCESSFUL (total time: 0 seconds)
```

University_System (run)