```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package boundary;

import adt.ArrList;
import adt.DoublyLinkedList;
import adt.LinkedListInterface;
import adt.ListInterface;
import entity.Course;
import entity.CourseProgram;
import entity.Program;
import java.util.Iterator;
import utility.InputValue;
import utility.MessageUI;
import dao.DAO;
import utility.Command;
import utility.ConsoleColor;

/**
 *
 * @author Chew Lip Sin
 */
public class CourseProgramMaintenanceUI {

    private final InputValue iv = new InputValue();
    private final DAO<CourseProgram> cpDAO = new DAO();

    public CourseProgramMaintenanceUI() {
    }


    public int getMenuChoice() {
        Command.cls();
        int choice = 0;
        System.out.println("================================================");
        System.out.println("||Course and Program Management Subsystem Menu||");
        System.out.println("================================================");
        System.out.println("1. Add new program to the course");
        System.out.println("2. Remove program from the course");
        System.out.println("0. Exit");
        do {
            System.out.print("Enter choice: ");
            choice = iv.readInteger();
            if (choice > 2 || choice < 0) {
                MessageUI.displayInvalidChoiceMessage();
            }
        } while (choice > 2 || choice < 0);

        return choice;
    }

    public int getCourseChoices(ListInterface<Course> courseList) {
        int choice = 0;
        do {
            System.out.print("Enter choice(Enter '0' to exit): ");
            choice = iv.readInteger();
            if (choice > courseList.size() || choice < 0) {
                MessageUI.displayInvalidChoiceMessage();
            }
        } while (choice > courseList.size() || choice < 0);
        return choice;
    }

    public void addCourseProgramUI(LinkedListInterface<CourseProgram> courseProgramList, 
ListInterface<Course> courseList, ListInterface<Program> programList, int index) {
        ListInterface<Program> tempProgram = new ArrList<>();
        ListInterface<Program> tempProgram2 = new ArrList<>();
        ListInterface<Boolean> isElective = new ArrList<>();
        ListInterface<Program> tempProgram3 = new ArrList<>();
        ListInterface<Program> tempProgram4 = new ArrList<>();

        boolean loop = true;
        for (int i = 1; i <= programList.size(); i++) {
            tempProgram3.add(programList.getEntry(i));
        }
        int choice = 0, choice2 = 0;
        validateProgram(courseProgramList, courseList, programList, tempProgram2, index);
        for (int i = 1; i <= tempProgram2.size(); i++) {
            tempProgram3.remove(tempProgram2.getEntry(i));
        }
        tempProgram4 = tempProgram2;
        do {
            int no = 1;
            if (tempProgram3.size() > 0) {
                no = displayProgramList(tempProgram3, no);
                choice = getProgramChoice(choice, no);

                if (choice != 0) {
                    choice2 = displayElective(choice2, isElective);
                    if (choice2 == 1 || choice2 == 2) {
                        tempProgram.add(tempProgram3.getEntry(choice));
                        tempProgram2.add(tempProgram3.getEntry(choice));
                        tempProgram3.remove(choice);
                    }

                }

                if (choice == 0 || choice2 == 0) {
                    loop = false;
                }

            }
            if (tempProgram3.size() == 0) {
                loop = false;
            }
        } while (loop);
        if (tempProgram.size() > 0) {
            askAddConfirmation(tempProgram, courseList, isElective, courseProgramList, index);
            for (int i = 0; i < courseProgramList.sizeOf(); i++) {
                System.out.println(String.format("%s  %s  %s", courseProgramList.get(i).getCourseCode(), courseProgramList.get(i).getProgramCode(), courseProgramList.get(i).isIsElective()));
            }

        }
        if (tempProgram.size() == 0 && tempProgram4.size() == programList.size()) {
            MessageUI.printFormattedText("There is no available program to choose\n", ConsoleColor.YELLOW);
            Command.pressEnterToContinue();
        }
        tempProgram.clear();
        tempProgram2.clear();
        tempProgram3.clear();
        isElective.clear();

    }

    public void validateProgram(LinkedListInterface<CourseProgram> courseProgramList, ListInterface<Course> courseList,
            ListInterface<Program> programList, ListInterface<Program> tempProgram2,
            int index) {
        for (int i = 0; i < courseProgramList.sizeOf(); i++) {
            if
```

```
(courseProgramList.get(i).getCourseCode().equals(courseList.getEntry(index).getCourseCode()))
{
                for (int j = 1; j <= programList.size(); j++) {
                    if
(courseProgramList.get(i).getProgramCode().equals(programList.getEntry(j).getCode())) {
                        tempProgram2.add(programList.getEntry(j));
                    }
                }
            }
        }

    }

    private void askAddConfirmation(ListInterface<Program> tempProgram,
            ListInterface<Course> courseList, ListInterface<Boolean> isElective,
            LinkedListInterface<CourseProgram> courseProgramList, int index) {
        int choice;
        do {
            MessageUI.printFormattedText("Are you sure to add it?(1 is yes, 0 is no): ",
ConsoleColor.BRIGHTBLUE);
            choice = iv.readInteger();
            if (choice == 1) {
                for (int i = 1; i <= tempProgram.size(); i++) {
                    CourseProgram cp = new
CourseProgram(courseList.getEntry(index).getCourseCode(),
                            tempProgram.getEntry(i).getCode(),
                            isElective.getEntry(i).booleanValue());
                    courseProgramList.add(cp);
                }
                MessageUI.displaySuccessConfirmationMessage("Added");
                Command.pressEnterToContinue();
                cpDAO.saveToFile(courseProgramList, "courseProgram.dat");

            } else if (choice > 1 || choice < 0) {
                MessageUI.displayInvalidChoiceMessage();
            }
        } while (choice > 1 || choice < 0);
    }

    private int displayProgramList(ListInterface<Program> tempProgram3, int no) {
        Iterator<Program> it = tempProgram3.getIterator();
        System.out.println("Program List:");
        System.out.println(String.format("%-5s|%-15s|%-80s", "No.", "Program Code",
"Program Name"));
        String line = "";

```

```java
        for (int i = 0; i <= 100; i++) {
            line += "-";
        }
        System.out.println(line);
        while (it.hasNext()) {
            Program program = it.next();
            System.out.println(String.format("%2d   |%-15s|%-80s", no, program.getCode(), program.getName()));
            no++;
        }

        return no;

    }

    private int getProgramChoice(int choice, int no) {
        do {
            System.out.print("Enter choice(Enter '0' to exit/continue): ");
            choice = iv.readInteger();
            if (choice >= no || choice < 0) {
                MessageUI.displayInvalidChoiceMessage();
            }
        } while (choice >= no || choice < 0);
        return choice;
    }

    private int displayElective(int choice2, ListInterface<Boolean> isElective) {
        do {
            System.out.println("Elective");
            System.out.println("========");
            System.out.println("1. Yes");
            System.out.println("2. No");
            System.out.println("0. Exit");
            System.out.print("Is it an Elective?: ");
            choice2 = iv.readInteger();
            if (choice2 > 2 || choice2 < 0) {
                MessageUI.displayInvalidChoiceMessage();
            }
        } while (choice2 > 2 || choice2 < 0);
        if (choice2 == 1) {
            isElective.add(Boolean.TRUE);
        } else if (choice2 == 2) {
            isElective.add(Boolean.FALSE);
        }
        return choice2;
    }

    public void removeCourseProgramUI(LinkedListInterface<CourseProgram> 
courseProgramList, ListInterface<Course> courseList, ListInterface<Program> programList, int 
index) {
        ListInterface<Program> tempProgram = new ArrList<>();
        ListInterface<Program> tempProgram2 = new ArrList<>();
        ListInterface<Boolean> isElective = new ArrList<>();
        ListInterface<Program> tempProgram3 = new ArrList<>();
        ListInterface<Program> tempProgram4 = new ArrList<>();
        LinkedListInterface<CourseProgram> cp1 = new DoublyLinkedList<>();
        boolean loop = true;
        int choice = 0;
        validateProgram(courseProgramList, courseList, programList, tempProgram2, index);
        for (int i = 1; i <= tempProgram2.size(); i++) {
            tempProgram3.add(tempProgram2.getEntry(i));
        }
        for (int i = 1; i <= tempProgram2.size(); i++) {
            tempProgram4.add(tempProgram2.getEntry(i));
        }
        do {
            int no = 1;
            if (tempProgram2.size() > 0) {
                no = displayDelProgramList(tempProgram2, no);
                choice = getProgramChoice(choice, no);

                if (choice != 0) {
                    tempProgram.add(tempProgram2.getEntry(choice));
                    tempProgram2.remove(choice);

                }

                if (choice == 0) {
                    loop = false;
                }

            }
            if (tempProgram2.size() == 0) {
                loop = false;
            }
        } while (loop);
        if (tempProgram.size() > 0) {
            validateCourseProgram(tempProgram, courseList, courseProgramList, index, 
cp1);
            askDelConfirmation(tempProgram, courseList, courseProgramList, index, cp1);
            for (int i = 0; i < courseProgramList.sizeOf(); i++) {
                System.out.println(String.format("%s  %s  %s",
courseProgramList.get(i).getCourseCode(), courseProgramList.get(i).getProgramCode(), courseProgramList.get(i).isIsElective()));
            }

        }
        if (tempProgram4.size() == 0) {
            MessageUI.printFormattedText("There is no available program to delete\n", ConsoleColor.YELLOW);
            Command.pressEnterToContinue();
        }
        tempProgram.clear();
        tempProgram2.clear();
        tempProgram3.clear();
        isElective.clear();
    }

    private int displayDelProgramList(ListInterface<Program> tempProgram2, int no) {
        Iterator<Program> it = tempProgram2.getIterator();
        System.out.println("Program List:");
        System.out.println(String.format("%-5s|%-15s|%-80s", "No.", "Program Code", "Program Name"));
        String line = "";

        for (int i = 0; i <= 100; i++) {
            line += "-";
        }
        System.out.println(line);
        while (it.hasNext()) {
            Program program = it.next();
            System.out.println(String.format("%2d   |%-15s|%-80s", no, program.getCode(), program.getName()));
            no++;
        }

        return no;

    }

    private void askDelConfirmation(ListInterface<Program> tempProgram, ListInterface<Course> courseList,
            LinkedListInterface<CourseProgram> courseProgramList,
            int index,
            LinkedListInterface<CourseProgram> cp1) {
        int choice;
        do {
            MessageUI.printFormattedText("Are you sure to delete it?(1 is yes, 0 is no):
```

```java
", ConsoleColor.BRIGHTBLUE);
            choice = iv.readInteger();
            if (choice == 1) {
                for (int i = 0; i < cp1.sizeOf(); i++) {
                    courseProgramList.remove(cp1.get(i));
                }
                cpDAO.saveToFile(courseProgramList, "courseProgram.dat");
                MessageUI.displaySuccessConfirmationMessage("Deleted");
                Command.pressEnterToContinue();
            } else if (choice > 1 || choice < 0) {
                MessageUI.displayInvalidChoiceMessage();
            }
        } while (choice > 1 || choice < 0);
    }

    private void validateCourseProgram(ListInterface<Program> tempProgram,
            ListInterface<Course> courseList,
            LinkedListInterface<CourseProgram> courseProgramList,
            int index, LinkedListInterface<CourseProgram> cp1) {
        for (int i = 0; i < courseProgramList.sizeOf(); i++) {
            if
(courseProgramList.get(i).getCourseCode().equals(courseList.getEntry(index).getCourseCode()))
{
                for (int j = 1; j <= tempProgram.size(); j++) {
                    if
(courseProgramList.get(i).getProgramCode().equals(tempProgram.getEntry(j).getCode())) {
                        cp1.add(courseProgramList.get(i));
                    }
                }
            }
        }
    }
}
```