Access Modifiers in this diagram may be wrong

You choose the the right Access Modifiers

+ = public
# = protected
~ = default
- = private

RegistrationSystem

- studentsList: StudentsList
- coursesList: CoursesList

- final maxStudentsForCourse: int
- final maxCoursesForStudent: int
- final minCoursesForStudent: int

+ RegistrationSystem(maxStudentsForCourse: int, minCoursesForStudent: int, maxCoursesForStudent: int): void // Constructor

+ addStudent(studentId: int)
+ addCourse(courseId: int)
+ removeStudent(studentId: int)
+ removeCourse(courseId: int)

+ getLastStudentAdded(): int // returns the id
+ getLastCourseAdded(): int // returns the id

+ getAllStudentsList(): int[] // Sorted by Id
+ getAllCoursesList(): int[] // Sorted by Id
+ displayAllStudentsList(): void // Sorted by Id
+ displayAllCoursesList(): void // Sorted by Id

+ addStudentToCourse(int StudentId, int courseId): void
+ removeStudentFromCourse(int StudentId, int courseId): void

+ getCoursesOfStudent(studentId: int): int[]
+ getStudentsOfCourse(courseId: int): int[]
+ displayCoursesOfStudent(studentId: int): void
+ displayStudentsOfCourse(courseId: int): void

+ isFullCourse(courseId: int): boolean
+ isNormalStudent(courseId: int): boolean

Those public methods
are the interface

and must those methods never return a object of
the internal system like: StudentNode, Cell or
History

internal system must be inaccessible

Node

next: Node // pointer to the next node
id : int (final variable , prevent wrong edit)
- fistCell: Cell

No thing

StudentNode extends Node

- coursesCount: int // with Getter

~ getCourses(): int[] // returns a list of IDs
~ removeAllCoursesOfMe(): void // need it when remove a student

CourseNode extends Node

- studentsCount: int // with Getter

~ getStudents(): int[] // returns a list of IDs
~ removeAllStudentsOfMe(): void // need it when remove a course

In addStudent() and addCourse()
you as a programmer must check that the given id has not used
before, so you will loop over all list items
and this is O(n)

why not make the list is sorted and when add you make loop over the
items to the nearst id and id after it
this makes the list is sorted always
but still o(n)

LinkedList

// This is special Singly linked list with some features i need

head: Node

- lastAdded: int // the id of the last added node
// with Getter
// default value is 0 because there is non-positive id number

+ add(): void
+ remove(): void

+ getAllTheList(): int[] // returns array of IDs

StudentsList extends LinkedList

head: StudentNode

...

CoursesList extends LinkedList

head: CourseNode

...

History Management part

when you remove a student or course
may some actions stored in the history
can't be back

Need a solution or system desgin for this
part

History
// for enrollment operations only

undoStack: Stack
redoStack: Stack

+ Undo(): void
+ Redo(): void

// Those methods takes the appropriate operations according to the event happend
+ studentAddedToCourse(studentId): void // or courseAddedToStudent
+ studentRemovedFromCourse(studentId): void // courseRemovedFromStudent
+ studentWasRemoved(studentId: int, coursesList: int[]): void
+ courseWasRemoved(courseId: int, studentsList: int[]): void

the idea of Action class you can replace it
with your idea that give the correct final
output

Action

+ final actionType: int OR byte // stores the type of action "add" or "remove"
+ final studentId: int
+ final courseId: int

+ Action(actionType: int, studentId: int, courseId: int) // Constructor

+ doTheAction(): void
+ doTheReverseAction(): void

Stack

// This is Stack using Singly linked list

+ push(actions: Action[])
+ pop(): Action[]
+ top(): Action[]
+ clear() // Empty all the stack

When remove a student, you will remove all
his courses in one action

This is the cause, why array of Actions are in
the Stack

Cell is also a Node as you know in data
structures but
i renamed it to cell to be
easier name only
but Cell is a Node

Cell // You may say EnrollmentNode

// There no info part, only need know to connect this student with this course

studentId: StudentNode (final variable)
courseId: CourseNode (final variable)

- nextCourseForThisStudent: Cell // if equals null, there is no next cell or enrollment
- nextStudentForThisCourse: Cell // if equals null, there is no next cell or enrollment
- previousCourseForThisStudent: Cell // if equals null, the previous is the StudentNode it self
- previousStudentForThisCourse: Cell // if equals null, the previous is the CourseNode it self

Nothing

Sparse Table Example