The Group:

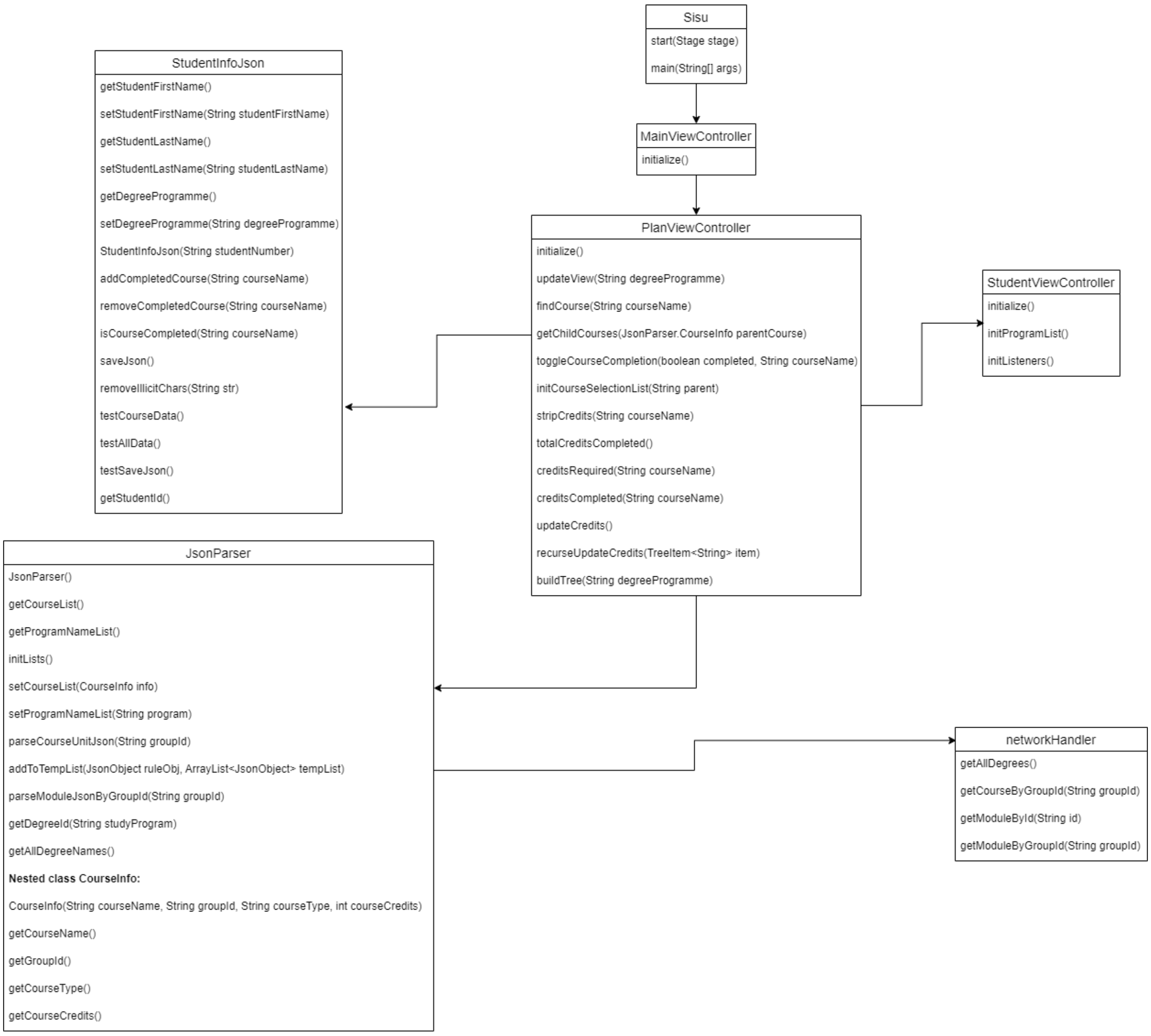
Leevi Hokkanen 050253975,

Samuli Ruuskanen H291595,

Miika Valli 292271

Documentation of the *Programming 3: Interfaces and Techniques* Project *Sisu Unraveled*

Structure of the Program


Sisu
start(Stage stage)
main(String[] args)
MainViewController
initialize()
StudentInfoJson
getStudentFirstName()
setStudentFirstName(String studentFirstName)
getStudentLastName()
setStudentLastName(String studentLastName)
getDegreeProgramme()
setDegreeProgramme(String degreeProgramme)
StudentInfoJson(String studentNumber)
addCompletedCourse(String courseName)
removeCompletedCourse(String courseName)
isCourseCompleted(String courseName)
saveJson()
removeIllicitChars(String str)
testCourseData()
testAllData()
testSaveJson()
getStudentId()
PlanViewController
initialize()
updateView(String degreeProgramme)
findCourse(String courseName)
getChildCourses(JsonParser.CourseInfo parentCourse)
toggleCourseCompletion(boolean completed, String courseName)
initCourseSelectionList(String parent)
stripCredits(String courseName)
totalCreditsCompleted()
creditsRequired(String courseName)
creditsCompleted(String courseName)
updateCredits()
recurseUpdateCredits(TreeItem<String> item)
buildTree(String degreeProgramme)
StudentViewController
initialize()
initProgramList()
initListeners()
JsonParser
JsonParser()
getCourseList()
getProgramNameList()
initLists()
setCourseList(CourseInfo info)
setProgramNameList(String program)
parseCourseUnitJson(String groupId)
addToTempList(JsonObject ruleObj, ArrayList<JsonObject> tempList)
parseModuleJsonByGroupId(String groupId)
getDegreeId(String studyProgram)
getAllDegreeNames()
Nested class CourseInfo:
CourseInfo(String courseName, String groupId, String courseType, int courseCredits)
getCourseName()
getGroupId()
getCourseType()
getCourseCredits()
networkHandler
getAllDegrees()
getCourseByGroupId(String groupId)
getModuleById(String id)
getModuleByGroupId(String groupId)

Project Features

Main features:

Our program aimed to fulfill the intermediate requirements of the program.

- The project has a UI build entirely from scratch.
- Degree programs, modules and courses are received from Sisu's API.
- Degree, its situation, and progress is shown.
- Program handles errors in file handling.
- Unit tests have been implemented.

Additional feature:

- Student settings have been implemented. The study program can be set by the student at the start of the program. The situation of the degree can be read from a JSON file.

Key Classes and Their Functionality

JsonParser

Class JsonParser's main function is to find all the courses and modules of one study program by parsing a string of JSON-format which has been got from an URL made in class networkHandler. Then JsonParser stores the useful info of modules and courses to an ArrayList named courseList which can then be fetched by a call. JsonParser also gets all the names of study programs and stores them in to an ArrayList named programNameList which can also then be fetched by a call.

networkHandler

This class is used for making HTTP requests to the Kori API, so that we can receive the relevant module and course info. The module, degree, and course info is searched via ids and group-ids of said information. The class works with JsonParser by returning data in JSONs of data in string format.

StudentInfoJson

StudentInfoJson is used for keeping track of the completed courses of students and for creating JSON files of those said courses. Class works by initializing the class object for an individual student with a student identification. The methods of the class can be used for adding and removing courses from student. The courses can then be stored JSON file: *StudentInfoJson.json.* This same file is created to the root of the program directory.

MainViewController

MainViewController is the controller for the main view that contains the *TabPane* for *PlanView* and *StudentView*. The only real functionality provided be *MainViewController* is to pass along a reference to the instance of *PlanViewContoller*.

StudentViewController

StudentViewController is used to initialize student data in the view and for updating stored data when updated using the UI. This also calls *PlanViewController*'s *updateView* method via a reference stored in *Sisu* when the degree programme is changed. When the program starts the i*nitialize()* method get called and the class sets the student info field values to the ones stored for the student. The *initialize()* method also adds listeners to the *TextFields* for updating the stored data when the values are changed.

PlanViewController

PlanViewController is responsible for initializing and handing events in the *PlanView*. When the *initialize* method gets called it will add a listener to the *TreeView* that is responsible for detecting which *TreeItem* the user clicks on. At this point it will also initialize the *TreeView* depending on whether there is a degree programme saved for the user. The *updateView* method is responsible for clearing the tree and calling the *buildTree* method to build a new tree when the degree programme is changed. The controller is also responsible for generating the course selection list and updating the course credits and completions accordingly.

Sisu

The *Sisu* class contains the *main* function of the program. The class is responsible for making the start dialog, loading the main view and holding references to *PlanViewController* and *StudentInfoJson* objects. It is also responsible for calling the *saveJson* method of *StudentInfoJson* when the program exits.

Division of Work

Class JsonParser and the unit tests of that class are done by Miika. Classes networkHandler, StudentInfoJson and the unit tests of those classes are done by Leevi. The rest of the classes (classes related to GUI) are done by Samuli. Also, the documentation was done by all of us group members together. The division of work was in the end exactly as we had planned in the beginning.

User Manual

When the program is run, a window pops up and asks the student's student number. After writing your number click "OK" or press ENTER. If you haven't previously planned your studies, you may now add your first and last name to the text fields in the new window and choose your study program from the list. The student number you wrote before is automatically added in the text field of student number. If you had previously planned your studies, all your info is automatically added. Once all your info is added, you can press the tab named *Study plan* to plan your studies. You can always return to the tab *Student info* to re-enter your info, for example change your study program.

In the tab named *Study plan*, you can now plan your studies. All fetched modules and courses are now available as an interactive tree-type view in left side of the tab. The modules can be expanded by double-clicking the names of the modules or by clicking the arrow next to the modules' names. Now you will see all the sub-modules and sub-courses of a module. By clicking the line of a module, all the courses belonging to the module are listed in the right side of the window. By clicking the checkbox next to the course's name, you mark the course completed. Once you want to close the program, you may press the X on the top right corner of a window. All your info and the courses you marked as completed will be saved once the X has been pressed. Now, all your info and completed courses are loaded to the program automatically when you use the program again.

If you haven’t previously planned your studies, the opening of the tab *Study plan* can take quite a long time depending on the amount of all the courses in your chosen study program. If you had previously planned your studies, the fetching of the courses is done after entering your student number instead, so the time before you see the next window can be quite long. Also, changing the study program from the list makes the window freeze for a little while, but it will be operable again shortly.

Known Bugs or Missing Features

The test testGetAllDegrees() of test class networkHandlerTest can cause a build error if the test fails, which it has done on numerous occasions. But this doesn’t affect the functionality of the program. The reason for the failure is probably that the order of objects in an array in the JSON fetched from the Kori API is randomized. This bug also causes that the pipeline in GitLab can fail.

Although not a bug or missing feature, as mentioned in the *User Manual* section of the documentation, the program can be quite slow after entering the student number in the first window, after changing the study program or after changing the tab to *Study plan*. It might seem like the program has crashed, but it will start working and be interactive again after a short while.