An Nguyen documentation on: API calling and degree program, study module, course unit structure of the program.

This explanation is for files: APIReader.java, StudyModule.java, CourseUnit.java, DegreeProgram.java, AnyRule.java, AnException.java, SubCompositeRule.java, AbstractModule.java .

Goal: This document explain the "Why" for mentioned files. I want to create the structure as close as what the API has. Because I believe in that way, the program can have as much information as SISU has. And I have to make sure the structure is true for everything since there're many variations of the nested structure in Rule that each API has.

Files explanation:

AbstractModule.java : provide a basic template (has name, id, groupId, id, API) for CourseUnit, StudyModule, DegreeProgram

StudyModule.java, DegreeProgram.java: This 2 module has somewhat same structure as they have the Rule in their API. In which can have type "CreditsRule" or "CompositeRule". And I looked through all the Degree and most of StudyModule API, **I can see that every CreditsRule has a "rule" of type CompositeRule** which makes sense because CreditsRule's "rule" property can only contain an Object, not an array so it can't have many courses or study modules within it. Therefore, the DegreeProgam and StudyModule have a subCompositeRule which contain the "rule" it has.

AnyRule.java: This is the special type and don't have the API of its own because any thing is acceptable. So I made this class especially for it and it can be either "AnyCoureUnitRule" or "AnyStudyModuleRule". The reason I left it as String but not Boolean because I don't know if this is true for every AnyRule, maybe there's more (but not in current state).

SubCompositeRule.java: This can be a CreditsRule or CompositeRule (the different is the existence of max and min credits, which only CreditsRule have). Because the variations the "rule" can have. This class must have 4 array: CourseUnit, StudyModule, AnyRule, SubCompositeRule. For example, this API: https://sis-tuni.funidata.fi/kori/api/modules/by-group-id?groupId=otm-e72e3162-292c-4702-be0e-ae2bbc42280b&universityId=tuni-university-root-id ; it has both CourseUnit and CompositeRule (which has 2 more CourseUnit). Therefore, having courseUnit Array and SubCompositeRule is necessary.

APIReader.java: This class handle all the heavy work of calling and converting the Json into the structure given above. The JsonToDegreeProgram create the DegreeProgram base on the JsonObject that we have from the API, it construct the full DegreeProgram. The JsonToStudyModule construct only the AbstractModule part of the StudyModule because I want to simulate the clicking of the user and lazy load the API (when user click on the StudyModule, it will construct the SubCompositeRule and call API of those in the Rule to create them in a basic sense (AbstractModule)). That leads to function onClickStudyModule which take on the "next level" of the given StudyModule (get API and create Course, Module,… in the SubCompositeRule arrays), in other words, it's a recursive function in a sense to go 1 level deeper on call. Another notice I have is that: **When a CompositeRule is a subrule of CreditsRule in the API, it sometimes can be useless when it has only 1 element in its "rules" and that element is another CompositeRule (which truly contain all needed information), so my code eliminate**

that level of complexity, only take the second inner CompositeRule as the SubCompositeRule. Example is the API above also .This can be seen in function JsonToCompositeRule as followed:

```
if (ruleType.equals(anObject: "CreditsRule")){
    if (checkNotJsonNull(rule, entity: "credits")){
        JsonObject credits = rule.get("credits").getAsJsonObject();
        minCredit = credits.get("min").getAsInt();
        if (checkNotJsonNull(credits, entity: "max")){
            maxCredit = credits.get("max").getAsInt();
        } else {
            maxCredit = minCredit;
        }
    }
    isCredits = true;
    JsonArray subRules = rule.get("rule").getAsJsonObject().get("rules").getAsJsonArray();
    if (subRules.size() == 1 && subRules.get(0).getAsJsonObject().get("type").getAsString().equals
    (anObject: "CompositeRule")){
        rule = subRules.get(0).getAsJsonObject();
    } else {
        rule = rule.get("rule").getAsJsonObject();
    }
}
```

AnException.java: It's actually named after the creator 😊 but you can understand is as just an exception. It's a basic exception class that handles mostly type error in the APIReader.java implementation (because the type in the API is limited, maybe extend more in the future so I just put it there in case that happens)

Conclusion: I believe the structure I made fully capable of getting the full SISU like information and construct SISU in some sense if used properly. And it scale rather well with modern application when lazy loading the structure.