# Trabalho prático de Tecnologias de Programação

Reginaldo Gregório de Souza Neto
Instituto Politécnico de Bragança
Portugal
a61482@ipb.pt

19 de março de 2024

## 1 Introducão

Este trabalho apresenta a SAGUI, uma nova Linguagem Específica de Domínio criada para descrever diagramas de classes de forma textual. Em engenharia de software, os diagramas de classes são essenciais para modelar como diferentes partes de um sistema se relacionam e trabalham juntas. Eles mostram as classes, seus atributos, métodos, e como se conectam através de relações como associação, herança, e composição. A ideia da SAGUI é tornar mais fácil para desenvolvedores e analistas de sistemas especificar essas estruturas textualmente, facilitando a automação e a troca de informações em ferramentas de desenvolvimento. Com a SAGUI, é possível representar todos os elementos importantes de um diagrama de classes, incluindo detalhes sobre multiplicidade, navegabilidade e visibilidade, de forma clara e direta. Este documento introduz a linguagem e demonstra seu uso através de um exemplo prático, mostrando como ela pode ser aplicada para descrever um diagrama de classes específico.

# 2 Exemplo do diagrama de classes

O diagrama de classes proposto como base para realização do trabalho pode ser visto na figura 1.

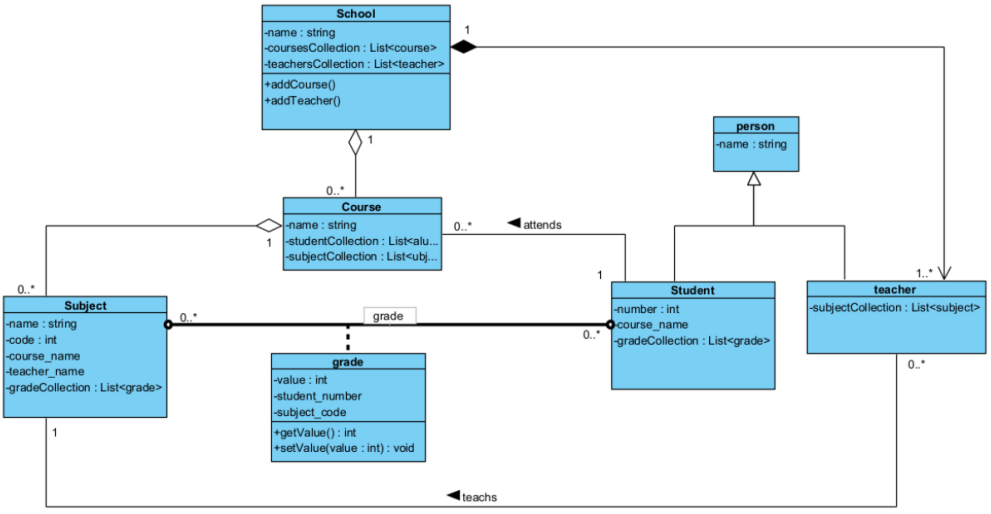



Figura 1: Exemplo do Diagrama de Classes

Através desse diagrama é possível notar todas as necessidades de representação em uma linguagem de Diagrama de Classes. Sendo assim, ele será utilizado como base para criação da SAGUI e consequentemente como sua validação.

# 3 SAGUI - Uma linguagem de Diagrama de Classes

Para a criação do diagrama de classes, o primeiro passo consiste na escolha de um título para o mesmo. Em seguida utilize `DIAGRAMA GERAL:` para iniciar a codificação e «`CLASSES` para abrir o escopo de criação das classes, utilize » para realizar seu fechamento. Nota-se que a organização do código deve ser feita através de tabulações para que seja possível a identificação dos escopos definidos.

Para descrever a visibilidade de um atributo ou método/função que faz parte de uma classe, uma notação opcional de três letras pode ser colocada antes do nome desse membro:

```
PUB = Público
PRI = Privado
PRO = Protegido
PAC = Pacote/Interno
```

A especificação de cada classe se dá através de seus parênteses, onde devem conter os tipos e os nomes de cada atributos e após uma barra reta estipula-se quais serão os métodos.

```
TITULO: NomeDoDiagrama
DIAGRAMA GERAL:
    ≪CLASSES
        NomeDaClasse(TIPO: nomeAtributo | método(parâmetros) );
        NomeDaClasse(TIPO: nomeAtributo | método(parametros) );
    ≫
```

Os atributos por sua vez, se forem do mesmo tipo, devem ser declarados entre vírgulas. Caso contrário, é preciso utilizar um ponto e vírgula entre cada tipagem como consta no exemplo a seguir:

```
TITULO: NomeDoDiagrama
DIAGRAMA GERAL:
≪CLASSES
    NomeDaClasse(
        PRI STRING: a, b, c;
        PRI INT: d, e, f;
        |
        PUB metodo1();
        PUB metodo2();
    );
≫
```

Veja que é possível organizar o código de maneira mais espaçada para uma melhor visualização das informações. Para a criação das `RELAÇÕES` a dinâmica de escopo permanece a mesma, entretanto é preciso se atentar às peculiaridades nas representações.

```
≪RELAÇÕES
    classeA (cardinalidadeA <tipo>"legenda"<tipo> cardinalidadeB) ClasseB
    classeA (cardinalidadeA <tipo>) ClasseC (<tipo> cardinalidadeB) ClasseB
≫
```

As representação das cardinalidades foram inspiradas nos padrões usados no MySQL [2], e na linguagem SAGUI ficou da seguinte maneira:

| TIPO | DESCRIÇÃO |
|---|---|
| 1 | Um, e apenas um |
| * | Vários |
| 01 | Zero ou um |
| 0* | Zero ou vários |
| 1* | Um ou vários |

Tabela 1: Tipos de Cardinalidades

Os tipos a serem inseridos seguem o padrão da tabela 2 e devem ser espelhados para representar os dois sentidos da relação entre as classes, se assemelhando ao padrão utilizado na linguagem Mermaid[1]. Opcionalmente entre os tipos, é possível acrescentar uma legenda para a relação.

| TIPO | DESCRIÇÃO |
|---|---|
| <\|-- | Herança |
| *-- | Composição |
| o-- | Agregação |
| --> | Associação |
| -- | Linha Sólida |
| ..> | Dependência |
| ..\|> | Realização |
| .. | Linha Tracejada |

Tabela 2: Tipos de Relações

# 4 Descrição do Diagrama de Classes usando o SAGUI

Seguindo o modelo do diagrama proposto, a representação do mesmo na linguagem SAGUI ficaria do seguinte modo:

```
TITULO: IPB
DIAGRAMA GERAL:
≪CLASSES
    School(
        PRI STRING: name;
        PRI List<Course>: coursesCollection;
        PRI List<Teacher>: teachersCollection;
        |
        PUB addCourse();
        PUB addTeacher();
    );
    Course(
        PRI STRING: name;
        PRI List<Student>: studentsCollection;
        PRI List<Subject>: subjectsCollection;
        );
    Grade(
        PRI INT: value, subject_code, student_number;
        |
        PUB INT: getValue();
        PUB VOID: setValue();
    );
    Teacher(
        PRI List<Subject>: subjectCollection;
    );
    Student(
        PRI INT: number;
        PRI STRING: course_name;
        PRI List<Grade>: gradesCollection;
        );
    Subject(
        PRI STRING: name, course_name, teacher_name;
```

```
        PRI INT: code;
        PRI List<Grade>: gradesCollection;
    );
    person(
        PRI STRING: name
    );
≫
≪RELAÇÕES
    Course (0* --o 1) School;
    Course (1 o-- 0*) Subject;
    Student (1 -"attends"- 0*) Course;
    Subject (0*-- ) Grade ( -- 0*) Student;
    Teacher (0* --"teaches"--1) Subject;
    person (<|--) Teacher;
    person (<|--) Student;
    School (1 *--> 1*) Teacher;
≫
```

# 5 Criando o diagrama em Mermaid

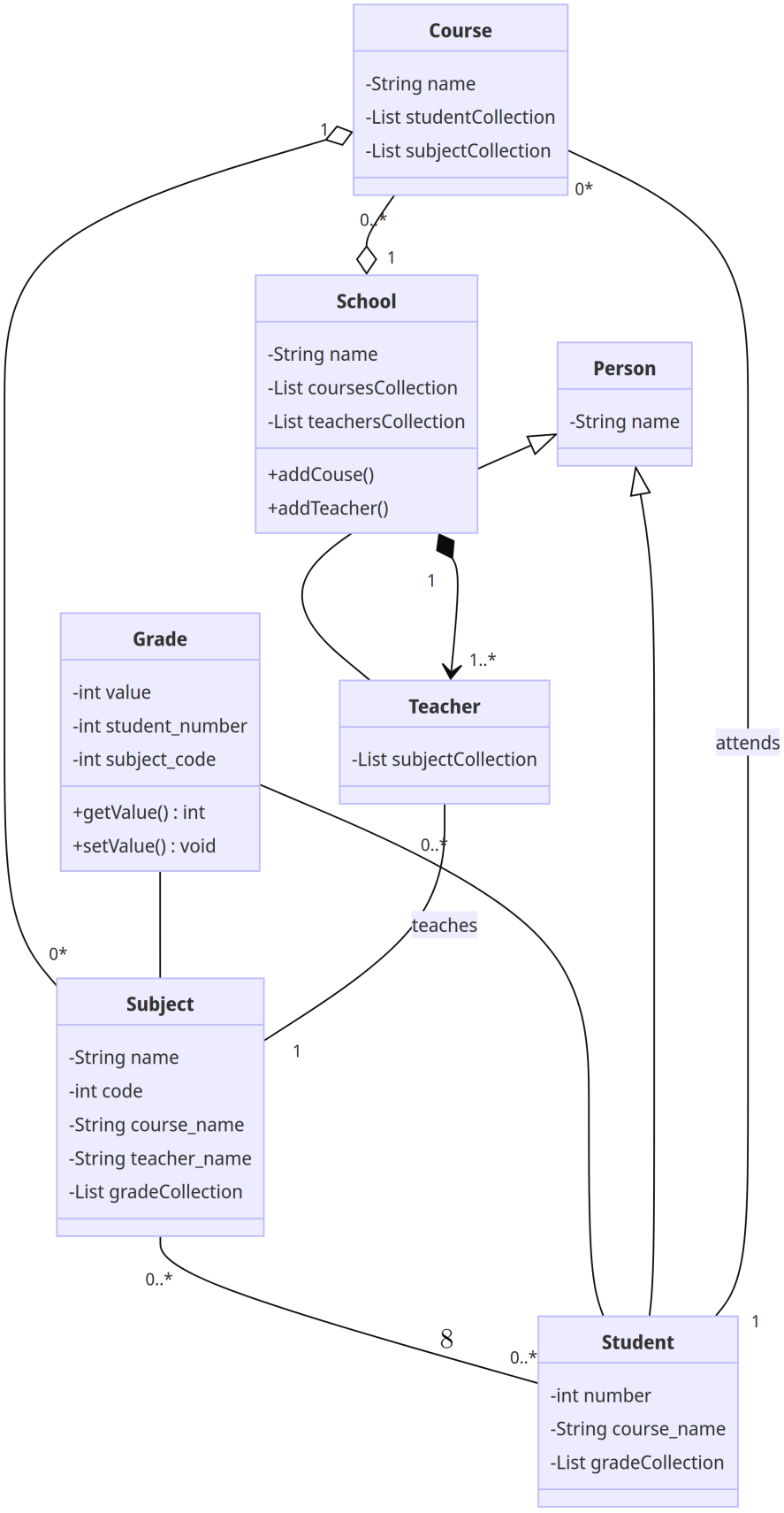
Para recriar o mesmo diagrama utilizando o Mermaid basta realizar a seguinte programação:

```
classDiagram
    Course "0..*" --o "1" School
    Course "1" o-- "0*" Subject
    Student "1" -- "0*" Course : attends
    Subject "0..*" -- "0..*" Student
    Grade -- Subject
    Grade -- Student
    Teacher "0..*" --"1" Subject: teaches
    Person <|-- Student
    Person <|-- Teacher
    School "1" *--> "1..*" Teacher
    class School{
        -String name
        -List<Course> coursesCollection
        -List<Teacher> teachersCollection
```

```
    +addCouse()
    +addTeacher()
}
class Course{
    -String name
    -List<Student> studentCollection
    -List<Subject> subjectCollection

}
class Grade{
    -int value
    -int student_number
    -int subject_code
    +getValue()int
    +setValue()void
}
class Teacher{
    -List<Subject> subjectCollection
}
class Student{
    -int number
    -String course_name
    -List<Grade> gradeCollection
}
class Subject{
    -String name
    -int code
    -String course_name
    -String teacher_name
    -List<Grade> gradeCollection
}
class Person{
    -String name
}
```

Gerando o seguinte diagrama:


Course
-String name
-List studentCollection
-List subjectCollection
1
0*
0..*
1
School
-String name
-List coursesCollection
-List teachersCollection
+addCouse()
+addTeacher()
Person
-String name
1
1..*
Grade
-int value
-int student_number
-int subject_code
+getValue() : int
+setValue() : void
Teacher
-List subjectCollection
attends
0..*
teaches
0*
Subject
-String name
-int code
-String course_name
-String teacher_name
-List gradeCollection
1
0..*
8
0..*
Student
-int number
-String course_name
-List gradeCollection
1

# 6 Conclusão

Após os testes realizados no Mermaid, podemos concluir que é possível a realização da representação dos diagramas através de uma linguagem textual. Embora o Mermaid não fosse capaz de representar o diagrama de uma maneira perfeitamente fidedigna, é possivel comprovar sua eficácia na exibição dos conceitos desejados. Portanto é notavel que a linguagem SAGUI também seja capaz de representar as mesmas características de um diagrama, uma vez que ela foi projetada para isso, sendo idealizado desde o princípio a possibilidade de que seja feita sua conversão/tradução ao Mermaid.

# Referências

[1] Mermaid. https://mermaid.js.org/, 2024.

[2] Mysql. https://dev.mysql.com/doc/, 2024.