Trabalho prático número 2: Gestão Acadêmica

Leandro Lazaro Araújo Vieira (3513), Mateus Pinto da Silva (3489)

Ciência da Computação – Universidade Federal de Viçosa - Campus Florestal (UFV-caf) – Florestal – MG – Brasil

{leandro.lazaro, mateus.p.silva}@ufv.br

Resumo. *Este trabalho consiste em desenvolver um sistema acadêmico para gerenciar alunos, professores, turmas e seus respectivos atributos.*

1. Como simular executar o trabalho

Para a execução correta do trabalho, é necessário:

- Make
- Compilador G++
- Sistema operacional Linux (de preferência Ubuntu)

Para executar o código, basta navegar até a pasta onde está o trabalho e digitar "make" (para compilar) e "make run" (para executar) respectivamente no terminal. Após isso, será exibido um menu cujo mesmo é auto explicativo, o que torna desnecessário seu detalhamento.

2. Solução

O trabalho foi estruturado utilizando as diretrizes MVC (Model View Controller), ou seja, dentro do diretório principal do projeto existem as pastas model, view e controller.

Para criar as classes Teacher e Student no diretório model/entity criou-se a classe Person (no mesmo diretório) que serviu como herança para as duas primeiras classes evitando redundâncias.

Ainda no diretório model/entity, para referenciar estudantes na classe SchoollClass fizemos uso de outra classe auxiliar chamada EnrolledStudent que engloba o cpf do aluno da classe Student e suas respectivas notas bimestrais, ou seja, dentro da classe SchoolClass jamais será salva uma lista de Students, mas de EnrolledStudents, o que indiretamente quer dizer que será salvo apenas o cpf do aluno cadastrado junto com suas notas bimestrais. Isso foi feito buscando eliminar redundâncias, já que, se houver o mesmo dado em dois lugares diferentes, será custoso manter a sua integridade.

Quanto a interface, não focamos em fazer algo frondoso, mas algo simples e fácil de ser utilizado, até porque desenvolver interfaces próprias não é tão ágil de ser feito em C++.

```
namespace model {
   namespace entity{


       class Teacher : public Person
           {
           private:
               std::string area;
               double salaryPerHour;
               bool active;
           public:
                 Teacher(std::string name, std::string cpf, std::string street,
std::string neighborhood, std::string city, std::string CEP, std::string area,
double SalaryPerHour); // Throws BadArgument
               ~Teacher();


               std::string getArea();
               double getSalaryPerHour();
               std::string toString();
               bool isActive();


               void setArea(std::string area); // Throws BadArgument
                       void setSalaryPerHour(double salaryPerHour); // Throws
BadArgument
               void setActive(bool active);
           };


   } // namespace entity


} // namespace model
```

Imagem 2.1: Cabeçalho da classe Teacher

```
namespace model{

   namespace entity{


       class Student : public Person

           {

           private:

               unsigned int registrationNumber;


               std::string fatherName;

               std::string motherName;


               bool active;

           public:

               Student(std::string name, std::string cpf, std::string street,
std::string neighborhood, std::string city, std::string CEP, std::string
fatherName, std::string motherName, unsigned int registrationNumber); //
Throws BadArgument

               ~Student();


               unsigned int getRegistrationNumber();

               std::string getFatherName();

               std::string getMotherName();

               std::string toString();


               void setActive(bool active);

           };


   } // namespace entity


} // namespace model
```

Imagem 2.2: Cabeçalho da classe Student

```
namespace model
{
   namespace entity
   {
       class SchoolClass
       {
       private:
           unsigned int code;
           std::string teacherCPF;
           unsigned int year;
           std::list<model::entity::EnrolledStudent> enrolledStudent;
       public:
           SchoolClass(unsigned int code, unsigned int year, std::string
teacherCPF);  // Throws BadArgument
           ~SchoolClass();

           entity::EnrolledStudent & search(std::string enrolledStudentCPF);

           unsigned int getCode();
           std::string getTeacherCPF();
           unsigned int getYear();
           std::list<model::entity::EnrolledStudent> &
getAllEnrolledStudents();

           std::string toString();

           void insertEnrolledStudent(std::string enrolledStudentCPF);
       };

   } // namespace entity
} // namespace model
```

Imagem 2.3: Cabeçalho da classe SchollClass

```
namespace model{
   namespace entity {


       class Person
       {
       private:
           std::string name;
           std::string cpf;


           std::string street;
           std::string neighborhood;
           std::string city;
           std::string CEP;


           // Birth date


       public:
           Person(std::string name, std::string cpf, std::string street,
std::string neighborhood, std::string city, std::string CEP); // Throws
BadArgument
           ~Person();


           std::string getName();
           std::string getCpf();
           std::string getStreet();
           std::string getNeighborhood();
           std::string getCity();
           std::string getCEP();
           std::string toString();


           void setStreet(std::string street); // Throws BadArgument
```

```
            void setNeighborhood(std::string neighborhood); // Throws
BadArgument
            void setCity(std::string city); // Throws BadArgument
            void setCEP(std::string CEP); // Throws BadArgument


        };


    } // namespace entity


} // namespace model
```

Imagem 2.4: Cabeçalho da classe Person

```
namespace model
{
    namespace entity
    {
        class EnrolledStudent
        {
        private:
            std::string CPF;
            std::vector<double> grades;
        public:
            EnrolledStudent(std::string CPF);  // Throws BadArgument
            ~EnrolledStudent();


            std::string getCPF();
            std::vector<double>& getGrades();


            std::string toString();


            void setGrade(int position, double grade);
        };
```

Imagem 2.5: Cabeçalho da classe EnrolledStudent

3. Classes Exception

Além do que foi citado até agora ainda há a pasta Exception no diretório principal do projeto que é responsável por armazenar todos tratamentos de exceção do projeto. Dentro dessa pasta existem as subpastas BadArgument, BusinessRule e PersistenceError com seus respectivos arquivos cabeçalho e c++. Essas classes são utilizadas nas classes dos diretórios view e controller para tratar argumentos inválido, quebras na regra de negócio e redundâncias.

4. View

Uma diferença do conteúdo da classe view é que embora pareça estar orientado a objeto, na verdade está programado em um paradigma estruturado. Isso é muito comum em C++ quando se precisa criar métodos que não necessitam de parâmetros nativos para se comportar. Além disso, diferente do restante do código, todas as exceções dentro desse diretório são tratadas fazendo uso de try e catch.