PROVA DE EXAME – resolução
de
Programação Orientada por Objetos

INSTITUTO POLITÉCNICO DE BRAGANÇA
Escola Superior de Tecnologia e Gestão

Engenharia Informática
Informática de Gestão
Época de Recurso – 13 de fevereiro de 2017

Grupo I
(5 valores)

a) Diga o que entende por classe abstrata, se alguma das classes do problema é abstrata e de que forma é que se podem definir classes abstratas em C++. [1 val.]
Classe abstrata é uma classe não insaciável; nenhuma das classes do problema é abstrata; a forma mais habitual de definir em C++ uma classe abstrata é incluir-lhe um ou mais métodos abstratos (virtuais puros).

b) Deveríamos ter definido, em ambas as classes, o método toString() como constate? Justifique. [.6 val.]
Claro que sim, pois em nenhuma das classes o método modifica o valor dos atributos. É claramente um método de consulta.

c) Apresente o resultado que será visualizado na saída standard após a execução do programa. [1.8 val.]

```
OUTPUT 0:
Novo aluno Erasmus de Italia
OUTPUT 1:
Novo aluno Erasmus de Espanha
OUTPUT 2:
Novo aluno
OUTPUT 3:
0: Nome: Pirlo
1: Nome: Paco
2: Nome: Ana
OUTPUT 4:
Pirlo cancelou a sua matricula
OUTPUT 5:
Ana cancelou a sua matricula
O aluno de Espanha Paco cancelou a sua matricula
```

d) Apresente o resultado que seria visualizado na saída standard se os métodos destrutor e toString(), da classe `Base`, fossem virtuais. Refira-se apenas à parte da visualização que resultaria diferente em relação ao output da alínea anterior. [.6 val.]

```
OUTPUT 3:
0: Nome: Pirlo Aluno de Italia
1: Nome: Paco Aluno de Espanha
2: Nome: Ana
OUTPUT 4:
O aluno de Italia Pirlo cancelou a sua matricula
```

e) Se definíssemos o método construtor da classe `AErasmus` como se segue, estaríamos a introduzir, com essa alteração, dois erros. Diga quais. [1 val.]
Erro 1: o atributo nome, sendo privado na classe base, surge oculto na classe derivada AErasmus; logo não se pode aceder ao mesmo de forma direta (é o que se tenta fazer na instrução `nome = nom;`*);*
Erro 2: uma vez que não tem qualquer lista de inicialização, quando executado este construtor, por omissão é invocado o construtor por defeito da classe base, método que, como se sabe, não existe.

Grupo II (15 valores)

a) Defina em C++ todas as classes do problema, respeitando integralmente os métodos e ... [12 val.]

```
class Curso{
    Coleccao<UC> ucs;                          2
    Coleccao<Aluno> alunos;                    2
public:
    bool addAluno(const string &n) {                    2
        Aluno a(n);
        return alunos.insert(a);
    }

    bool addUC(const string &n, int a){                 2
        UC uc(n,a);
        return ucs.insert(uc);
    }
    bool incsreverAlunoEmUC(const string &al, const string &uc, int ano){    4
        Aluno *a=findAluno(al);
        if (a!=NULL) {
            UC *u = findUC(uc, ano);
            if (u!=NULL)
                if (a->addUC(u)) return u->addAluno(a);
        }
        return false;
    }
    bool listarUCsDeAluno(const string &name){          3
        Aluno *a=findAluno(name);
        if (a != NULL) {
            a->listarUCs();
            return true;
        }
        else return false;
    }
private:
    Aluno *findAluno(const string &n){                  1
        Aluno a(n);
        return alunos.find(a);
    }
    UC *findUC(const string &n, int a){                 2
        UC uc(n,a);
        return ucs.find(uc);
    }
};

class Aluno{                                    1
    string nome;
    Coleccao<UC*> ucs;                          2
public:
    Aluno(const string &n): nome(n){}           1
```

```
    bool addUC(UC *u){ return ucs.insert(u); }          2

    void listarUCs(){                                   3
        Coleccao<UC*>::iterator it;
        cout<<"UCs do aluno "<<nome<<":\n";
        for(it=ucs.begin(); it!=ucs.end(); it++)
            cout<<(*it)->getNome()<<" "<<(*it)->getAno()<<endl;
    }

    bool operator<(const Aluno& outro) const {return nome<outro.nome;}      2
};

class UC{                                               1
    string nome;
    int ano;
    Coleccao<Aluno*> alunos;                            2
public:
    UC(const string &n, int a) {nome=n; ano=a;}         1
    string getNome() const {return nome;}               1
    int getAno() const {return ano;}                    1

    bool operator<(const UC& outra) const {                                 3
        if (nome != outra.nome) return nome < outra.nome;
        else return ano<outra.ano;
    }

    bool addAluno(Aluno *a){return alunos.insert(a);}                       2

};
```

b) Acrescente ao problema os métodos que permitam remover do sistema uma dada unidade… [3 val.]

```
void Curso::remUC(const string &n, int a) {             4
    UC *u=findUC(n, a);
    if(u!=NULL){
        u->remAlunos();
        UC uc(n,a);
        ucs.erase(uc);
    }else cout << "UC " << n << " de " <<a<< " nao existe!\n";
}

void UC::remAlunos(){                                   4
    Coleccao<Aluno*>::iterator it;
    for(it=alunos.begin(); it!=alunos.end(); it++)
        (*it)->remUC(this);
    alunos.clear();
}

void Aluno::remUC(UC *u) {ucs.erase(u);}                2
```

Grupo III
(10 valores)

Defina então em C++ a nova classe Nota e acrescente às restantes os métodos … [10 val]

```
class Nota {                                    1
    int valor;
    UC *uc;                                     1
    Aluno *aluno;                               1
public:

    Nota(int v, UC *u, Aluno *a) {              1
        uc = u;
        aluno = a;
        valor = v;
    }

    void print()const{                                      2
        cout << aluno->getNome() << " " << valor << endl;
    }

    Bool operator<(const Nota &outra) const {               2
        return(aluno->getNome() < outra.aluno->getNome());
    }
};



class UC{
    ...
    Coleccao<Nota> notas;                                   2
public:
    ...

    bool addNota(const Nota &n) {                           2
        return notas.insert(n);
    }

    Nota* findNota(const Nota &n) {                         2
        return notas.find(n);
    }

    void listarNotas() {                                3
        Coleccao<Nota>::iterator it;
        cout << "Pauta de " << nome << " "<<ano<<":\n";
        for (it = notas.begin(); it != notas.end(); it++)
                it->print();
    }
};
```

```
class Aluno{
     ...
     Coleccao<Nota*> notas;                                    2
public:
     ...
     string getNome() const { return nome; }                   1

     bool addNota(Nota *p) { return notas.insert(p); }         2

     bool estaInscritoNaUC(UC *u) { return ucs.find(u)!=NULL; }          2
};


bool Curso::atribuirNota(string al, string uc, int ano, int val) {      4
     UC *u = findUC(uc,ano);
     if (u != NULL) {
          Aluno *a = findAluno(al);
          if (a != NULL)
               if (a->inscritoNaUC(u)) {
                    Nota n(val, u, a);
                    if (u->addNota(n))
                         return a->addNota(u->findNota(n));
               }
     }
     return false;
}

void Curso::listarNotasDeUC(string uc, int ano) {          2
      UC *u = findUC(uc, ano);
      if (u != NULL) u->listarNotas();
}
```