INSTITUTO POLITÉCNICO DE BRAGANÇA
Escola Superior de Tecnologia e Gestão

Engenharia Informática
Informática de Gestão
Época Normal – 19 de janeiro de 2015

Grupo II
(7.4 valores)

a) Alguma das classes é abstrata? Se sim, diga qual e o que a torna abstrata. [0.5 val.]

Nenhuma das classes é abstrata.

b) Apenas método `print()` é constante. Por que razão se definiu esse método como constante? [0.5 val.]

Porque não afeta o estado do objeto. Acede ao atributo nome apenas para o consultar, não para o alterar.

c) Diga quais são os métodos implícitos ... [1.2 val.]

Os métodos implícitos presentes em ambas as classes são o construtor de cópia e o operador afetação.

d) Apresente o resultado que será visualizado na saída standard após a execução do programa. [3.2 val.]

```
####Criacao###
Criado Diretor Rui Manuel
####Adicao###
####Print1###
Diretor Rui Manuel
####Print2###
Diretor Rui Manuel
####Destruicao###
Funcionario Jose dispensado
Funcionario Pedro dispensado
Diretor dispensado com os seus 2 subordinados
Funcionario Rui Manuel dispensado
```

e) Apresente agora o resultado se nenhum dos métodos da classe `Funcionario` fosse virtual ... [2.0 val.]

```
####Criacao###
Criado Diretor Rui Manuel
####Adicao###
Nao pode ter subordinados!
####Print1###
Diretor Rui Manuel
####Print2###
Funcionario Rui Manuel
####Destruicao###
Funcionario Jose dispensado
Funcionario Pedro dispensado
Diretor dispensado com os seus 1 subordinados
Funcionario Rui Manuel dispensado
```

Grupo I

(12.6 valores)

a) Defina em C++ todas as classes do problema, respeitando integralmente os métodos e ... [9.0 val.]

```
class Aeroporto{                                    [3.3 val.]
      Colecao<Aeronave> aeronaves;                  [1]
      Colecao<Porta> portas;                        [1]
public:
      bool Aeroporto::addPorta(int n){              [1]
            Porta p(n);
            return portas.insert(p);
      }
      bool Aeroporto::addAeronave(int n){           [1]
            Aeronave a(n);
            return aeronaves.insert(a);
      }

      Aeronave *findAeronave(int n){                [1]
            Aeronave  a(n);
            return aeronaves.find(a);
      }
      Porta *findPorta(int n){                      [1]
            Porta p(n);
            return portas.find(p);
      }

      bool conectar(int nporta, int naeronave){     [3]
            Aeronave *a = findAeronave(naeronave);
            if (a != NULL){
                  Porta *p = findPorta(nporta);
                  if (p != NULL) return p->conectar(a);
                  else return false;
            }
            else return false;
      }

      bool desconectar(int nporta){                 [2]
            Porta *p = findPorta(nporta);
            if (p != NULL) return p->desconectar();
            else return false;
      }
};
```

```
class Conexao{                                          [2.1 val.]
      TDataHora inicio, fim;
      Aeronave* aeronave;                               [2]
      Porta* porta;
public:
      Conexao(Aeronave* a, Porta* p){                   [2]
            inicio = TDataHora::hoje_agora();
            fim = TDataHora("1/1/3000 0:0:0");
            aeronave = a;
            porta=p;
      }
      TDataHora getFim()const{ return fim; }            [1]
      void desconectar(){fim=TDataHora::hoje_agora();}  [1]          [1]
      bool operator<(const Conexao &outra)const {return inicio<outra.inicio;}
};

class Aeronave{                                         [1.2 val.]
      int num;
      Colecao<Conexao *> conexoes;                      [1]
public:
      Aeronave(int n){ num = n; }                       [1]
      bool conectar(Conexao *c){ return conexoes.insert(c); }        [1]
      bool operator<(const Aeronave &outra)const { return num<outra.num; }  [1]
};

class Porta{                                            [2.4 val.]
      int num;
      Colecao<Conexao> conexoes;                        [1]
public:
      Porta(int n){num=n;}                              [1]
      bool conectar(Aeronave *a){                       [3]
            if (findConexaoAtual()!=NULL) return false;
            else{Conexao c(a, this);
                  if (conexoes.insert(c)) return a->conectar(conexoes.find(c));
                  else return false;
            }
      }
      bool desconectar(){                               [2]
            Conexao *c = findConexaoAtual();
            if (c == NULL) return false;
            else {c->desconectar(); return true;}
      }
      bool operator<(const Porta &outra)const {return num<outra.num;}      [1]
```

```
      //Conexao *findConexaoAtual()const{
      //    if (conexoes.empty()) return NULL;
      //    else
      //      if (conexoes.rbegin()->getFim()<=TDataHora::hoje_agora()) return NULL;
      //      else return (Conexao*)(conexoes.rbegin().operator->());
      //}
};
```

b) Acrescente ao problema os métodos que permitam mostrar... [2.4 val.]

```
void Aeroporto::printPortasConectadas(){          [1.2 val.]
      cout << "Conexoes atuais:" << endl;
      Colecao<Porta>::iterator it;
      for (it = portas.begin(); it != portas.end(); it++){
            Conexao *c = it->findConexaoAtual();
            if (c != NULL) c->print();
      }
}

void Conexao::print()const{                       [.6 val.]
      cout<< "A Aeronave "<< aeronave->getNum()
          << " encontra-se conectada 'a porta "
          << porta->getNum()<< " desde " << inicio << endl;
};

int Aeronave::getNum()const { return num; }       [.3 val.]

int Porta::getNum()const { return num; }          [.3 val.]
```

c) Implemente um pequeno main que faça uso de todas as funcionalidades da aplicação. [1.2 val.]

```
void main(){
      Aeroporto ae;
      ae.addPorta(5);
      ae.addPorta(1);
      ae.addAeronave(2000);
      ae.addAeronave(7000);
      ae.conectar(5, 7000);
      ae.conectar(1, 2000);
      ae.desconectar(5);
      ae.printPortasConectadas();
}
```