INSTITUTO POLITÉCNICO DE BRAGANÇA
Escola Superior de Tecnologia e Gestão

Engenharia Informática
Informática de Gestão
Época Normal – 18 de janeiro de 2016

Grupo I (13.2 valores)

a) Defina em C++ todas as classes do problema, respeitando integralmente os métodos e ... [9.3 val.]

```
class Produto {                                                          [1]
      int id;
      double preco;
      int quantidade;
public:
      Produto(int i, double p) { quantidade = 0; id = i; preco = p; }    [1]
      double getPreco() const { return preco; }                          [1]
      bool venderUnidade(){                                              [1]
            if (quantidade > 0) { quantidade--; return true; }
            else return false;
      }
      void reporStock(int quant) {quantidade += quant;}                  [1]
      bool operator<(const Produto &outro)const { return id<outro.id; }  [1]
};

class Carrinho {
      Colecao<Produto *> produtos;                                       [1]
public:
      bool addProduto(Produto *p) {                                      [2]
            if (p->venderUnidade()) return produtos.insert(p);
            else return false;
      }
      double valorTotal() const {                                        [3]
            double soma = 0;
            Colecao<Produto *>::iterator it;
            for (it = produtos.begin(); it != produtos.end(); it++)
                  soma += (*it)->getPreco();
            return soma;
      }
      void encomendar() { produtos.clear(); }                            [1]
};

class Cliente{                                                           [1]
      int num;
      Carrinho carrinho;
public:
      Cliente(int n){ num = n; }                                         [1]
      bool addProdutoAoCarrinho(Produto *p){return carrinho.addProduto(p);} [1]
      void encomendar() { carrinho.encomendar(); }                       [1]
      bool operator<(const Cliente &outro)const { return num<outro.num; } [1]
};
```

```
class LojaOnline{
      Colecao<Cliente> clientes;                                    [1]
      Colecao<Produto> produtos;                                    [1]

      Cliente *findCliente(int n) {                                 [1]
            Cliente cl(n);
            return clientes.find(cl);
      }
      Produto *findProduto(int i) {                                 [1]
            Produto p(i,0.0);
            return produtos.find(p);
      }
public:
      bool registarCliente(int n) {                                 [1]
            Cliente cl(n);
            return clientes.insert(cl);
      }
      bool registarProduto(int i, double p) {                       [1]
            Produto pr(i, p);
            return produtos.insert(pr);
      }
      bool reporStockProduto(int idprod, int quant) {               [2]
            Produto *p = findProduto(idprod);
            if (p != NULL) { p->reporStock(quant); return true; }
            else return false;
      }
      bool addProdutoAoCarrinho(int ncliente, int idprod) {         [3]
            Produto *p = findProduto(idprod);
            if (p == NULL) return false;
            else {
                  Cliente *c = findCliente(ncliente);
                  if (c == NULL) return false;
                  else return c->addProdutoAoCarrinho(p);
            }
      }
      bool encomendar(int ncliente) {                               [2]
            Cliente *c = findCliente(ncliente);
            if (c != NULL) { c->encomendar(); return true;}
            else return false;
      }
};
```

b) Acrescente ao problema os métodos que permitam mostrar o recheio do carrinho... [2.1 val.]

```
void LojaOnline::mostrarCarrinho(int ncliente) {                    [2]
    Cliente *c = findCliente(ncliente);
    if (c != NULL) c->mostrarCarrinho();
}

void Cliente::mostrarCarrinho() { carrinho.mostrar(); }             [1]

void Carrinho::mostrar() const {                                    [3]
    cout<<"<Carrinho de compras>"<<endl;
    Colecao<Produto *>::iterator it;
    for (it = produtos.begin(); it != produtos.end(); it++)
         (*it)->print();
    cout << "Total: " << valorTotal() << " EUROS." << endl;
}

void Produto::print()const {                                        [1]
    cout<< '\t' << "Produto " << id << ' ' << preco << " euros." << endl;
}
```

c) Acrescente ao problema os métodos que permitam excluir um dos produtos do carrinho... [1.8 val.]

```
bool LojaOnline::remProdutodoCarrinho(int ncliente, int idprod) {   [3]
    Produto *p = findProduto(idprod);
    if (p == NULL) return false;
    else {
         Cliente *c = findCliente(ncliente);
         if (c == NULL) return false;
         else return c->remProdutoDoCarrinho(p);
    }
}

bool Cliente::remProdutoDoCarrinho(Produto *p){                     [1]
    return carrinho.remProduto(p);
}

bool Carrinho::remProduto(Produto *p) {                             [2]
    if (produtos.find(p) != NULL) {
         produtos.erase(p);
         p->reporStock(1);
         return true;
    }else return false;
}
```

Grupo II
(6.8 valores)

a) Quantos métodos construtores estarão presentes em cada uma das classes do problema. [0.3 val.]
3.

b) Alguma das classes do problema é abstrata? Diga o que entende por classe abstrata. [1.0 val.]
Nenhuma das classes do problema é abstrata. Uma classe abstrata é uma classe não instanciável, normalmente caraterizada em C++ por conter um ou mais métodos virtuais puros.

c) Considerando as seguintes instanciações: `A a1("um"), a2("dois");` diga, justificando ... [0.7 val.]
A instrução não é válida dado não estarem definidos, nem como métodos das respetivas classes, nem como funções globais, os operadores < (menor) e << (insersor) para os operandos usados. (Também não será válida devido a um erro não intencional: identificador c1 não declarado)

d) Apresente o resultado que será visualizado na saída standard após a execução do programa. [3.8 val.]

```
-----1                [3]
A(str)
B(str,str)
C(str,str,str)
-----2                [3]
A()
B()
C()
-----3                [1]
A()
B()
C()
-----4
Area com sub-regioes com subsolo impermeavel      [3]
objeto da classe C com subobjeto da classe B com subobjeto da classe A    [3]
objeto da classe C com subobjeto da classe B com subobjeto da classe A    [1]
-----5                [3]
~C()
~B()
~A()
-----6                [1]
~C()
~B()
~A()
-----7                [1]
~C()
~B()
~A()
```

e) Apresente o resultado que seria visualizado na saída standard se nenhum dos métodos... [1.0 val.]

```
-----4
...
objeto da classe C com subobjeto da classe B com subobjeto da classe A      [.6]
-----5                                                                       [.4]
~C()
~B()
~A()
-----6
```