PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
ESCOLA POLITÉCNICA – Curso: Bacharelado em Engenharia de Computação
Disciplina: Programação Orientada a Objetos – EC
Professor: Roland Teodorowitsch

EXERCÍCIOS DE REVISÃO

1. Sobre o conceito de Paradigma de Programação, qual das sentenças a seguir está ERRADA?
   (A) São exemplos de paradigmas: procedural, funcional, orientado a objetos.
   (B) Um programa pode NÃO aproveitar as funcionalidades de um paradigma.
   (C) Trata-se de uma forma de classificar as linguagens de programação a partir de suas funcionalidades.
   (D) Uma linguagem de programação suporta um único paradigma.
   (E) Fornece e determina a visão que o programador possui sobre a estruturação e execução de seus programas.

2. Qual das afirmações a seguir pode ser associada sem restrições à Programação Orientação a Objetos?
   (A) Permite criar programas que podem manipular suas próprias fórmulas e componentes como se fossem dados puros.
   (B) Busca gerar programas que sejam obtidos de forma produtiva.
   (C) Trata a computação como uma avaliação de funções matemáticas, evitando estados ou dados mutáveis.
   (D) Tem como foco o desenvolvimento de programas com ênfase em sequência, decisão e iteração, permitindo estruturas de bloco e sub-rotinas.
   (E) Tem como principal característica o conceito de chamadas a procedimentos (também conhecidos como rotinas, subrotinas, métodos ou funções), que contêm um conjunto de passos computacionais a serem executados.

3. Sobre os conceitos básicos da Programação Orientada a Objetos, qual das afirmações a seguir está ERRADA?
   (A) Uma classe define as propriedades e o comportamento dos objetos gerados por ela.
   (B) Instanciar objetos significa gerar novos exemplares a partir de uma descrição abstrata de um objeto genérico (ou seja, de uma classe).
   (C) Uma classe determina um conjunto de objetos com propriedades e comportamentos semelhantes.
   (D) Um objeto é formado fundamentalmente por atributos e métodos.
   (E) Os métodos definidos em uma classe correspondem ao estado dos objetos dessa classe.

4. Quando se disponibiliza um conjunto padronizado de métodos, escondendo a implementação interna de uma classe, qual conceito da Orientação a Objetos está sendo utilizado?
   (A) Polimorfismo
   (B) Abstração
   (C) Coesão
   (D) Associação
   (E) Encapsulamento

5. Qual das afirmações a seguir sobre o comando `make` e o arquivo `Makefile`, frequentemente utilizados no Unix para compilação de aplicações, está ERRADA?
   (A) Caso `make` seja executado sem a definição de um alvo, NÃO existindo `Makefile` no diretório corrente, será mostrada uma mensagem de erro.
   (B) Basicamente um `Makefile` contém uma série de regras que definem: um alvo, as dependências para construir esse alvo e os comandos para se construir esse alvo a partir das dependências.
   (C) Para decidir se determinado alvo, especificado em um `Makefile`, deve ser reconstruído, o comando `make` leva em consideração as estampas de tempo nos arquivos alvo e suas dependências.
   (D) O comando `make` pode ser executado mesmo se NÃO houver um arquivo `Makefile` no diretório corrente.
   (E) Caso `make` seja executado sem a definição de um alvo, existindo Makefile no diretório corrente, todos os alvos do `Makefile` serão considerados.

6. Considerando-se o projeto orientado a objetos, quando as classes de uma aplicação possuem uma finalidade única e bem orientada, sem assumir responsabilidades que não são suas, tem-se
   (A) alto encapsulamento.
   (B) baixo encapsulamento.
   (C) baixa coesão.
   (D) alta coesão.
   (E) baixa sobrecarga.

7. No contexto da Programação Orientada a Objetos, qual das afirmações a seguir sobre composição está ERRADA?
   (A) NÃO é um relacionamento entre pares (ou seja entre objetos de mesma importância).
   (B) Os objetos envolvidos nesse relacionamento NÃO podem ser usados independentemente dos outros.
   (C) É um tipo especial de agregação.
   (D) Em termos de programação, que quando o objeto todo é destruído, todos os objetos parte são automaticamente destruídos também.
   (E) Os objetos NÃO são independentes uns dos outros.

8. Analise o código em C++ a seguir.

```
#include <iostream>
using namespace std;
class MinhaClasse {
    public:
        MinhaClasse() { }
        ~MinhaClasse() { }
};
int main() {
    const int TAM = 20;
    MinhaClasse v0, *v1;
    MinhaClasse *v2 = new MinhaClasse();
    MinhaClasse *v3 = new MinhaClasse[TAM];
    delete v2;
    return 0;
}
```

E responda:

a) Quantas vezes o construtor de MinhaClasse será chamado para v0? _____

b) Quantas vezes o construtor de MinhaClasse será chamado para v1? _____

c) Quantas vezes o construtor de MinhaClasse será chamado para v2? _____

d) Quantas vezes o construtor de MinhaClasse será chamado para v3? _____

e) Quantas vezes o destrutor de MinhaClasse será chamado para v0? _____

f) Quantas vezes o destrutor de MinhaClasse será chamado para v1? _____

g) Quantas vezes o destrutor de MinhaClasse será chamado para v2? _____

h) Quantas vezes o destrutor de MinhaClasse será chamado para v3? _____

9. Considere as seguintes declarações de classes em C++:

```
class Base {
public: string whoami() { return "Base"; }
};
class Derivada : public Base {
public: string whoami() { return "Derivada"; }
};
```

E também as seguintes declarações de variáveis:

```
Base b, *pb;
Derivada d, *pd;
```

Após estas declarações, qual das operações a seguir NÃO é válida?

(A) `pb = &d;`
(B) `pd = &b;`
(C) `b = *pb;`
(D) `pd = &d;`
(E) `b = d;`

10. Considere as seguintes declarações de classes em C++:

```
class Veiculo {
public: virtual string tipo() { return "Veiculo"; }
};
class Carro : public Veiculo {
public: string tipo() { return "Carro"; }
};
```

O que será impresso após a execução do seguinte trecho de código?

```
Veiculo v, *pv;
Carro   c;
pv = &v;
cout << pv->tipo() << " ";
pv = &c;
cout << pv->tipo() << endl;
```

(A) Veiculo Veiculo
(B) Carro Carro
(C) Veiculo Carro
(D) Carro Veiculo

11. O programa em C++ abaixo utiliza sobrecarga de operadores, porém está INCOMPLETO. Complete-o para que ele funcione corretamente, SEM MODIFICAR o que já está escrito, apenas inserindo linhas novas.

```
#include <iostream>
#include <sstream>
using namespace std;
class Empresa {
    private:
        string nome;
        int numFuncionarios;
    public:
        Empresa(string nome="", int numFuncionarios=0);
        void define(string nome,int numFuncionarios);
        string obtemNome();
        int obtemNumFuncionarios();
        string toString();
        Empresa operator+ (int funcionarios);
        bool operator== (Empresa &e);
        bool operator!= (Empresa &e);
};
Empresa::Empresa(string nome,int numFuncionarios) {
    define(nome,numFuncionarios);
}
void Empresa::define(string nome,int numFuncionarios) {
    this->nome = nome;
    this->numFuncionarios = numFuncionarios;
}
string Empresa::obtemNome() {
    return nome;
}
int Empresa::obtemNumFuncionarios() {
    return numFuncionarios;
}
string Empresa::toString() {
  stringstream ss;
  ss << nome << " (" << numFuncionarios << " funcionarios)";
  return ss.str();
}
int main(){
  Empresa supermercado("Supermercado 123",5), s("Supermercado 123",18);
  cout << supermercado.toString() << endl;
  if (s!= supermercado) cout << "DIFERENTE" << endl;
  supermercado = supermercado + 2;
  cout << supermercado.toString() << endl;
  if (s!= supermercado) cout << "DIFERENTE" << endl;
  supermercado = supermercado + 11;
  cout << supermercado.toString() << endl;
  if (s== supermercado) cout << "IGUAL" << endl;
  return 0;
}
```

12. Considere uma aplicação em C++ que precisa armazenar informações sobre produtos. Para cada produto é preciso armazenar código de barras (*string*), descrição (*string*), preço (valor real) e informações para desconto (2 números que indicam respectivamente para quantos produto forem comprados, quantos serão pagos - permitindo, por exemplo, descontos no estilo "leve x pague y"). Crie uma classe chamada `Produto` para armazenar e gerenciar as informações dos produtos. Para esta classe defina:
    a) construtor sem parâmetros;
    b) construtor que recebe como parâmetros: código de barras, descrição, preço, número de produtos que deve ser comprado ("x" no estilo de desconto "leve x, pague y") e quantos produtos serão pagos ("y" no estilo de desconto "leve x pague y");
    c) método chamado `defineCodigoBarras` para definir o código de barras;
    d) método chamado `obtemCodigoBarras` para obter o código de barras;
    e) método chamado `defineDescricao` para definir a descrição do produto;
    f) método chamado `obtemDescricao` para obter a descrição do produto;
    g) método chamado `definePreco` para definir o preço do produto;
    h) método chamado `obtemPreco` para obter o preço do produto;
    i) método chamado `defineDesconto` que recebe dois números inteiros definido número de produtos que deve ser comprado ("x" no estilo de desconto "leve x, pague y") e quantos produtos serão pagos ("y" no estilo de desconto "leve x pague y");
    j) método chamado `obtemTotal` que recebe a quantidade (valor inteiro) de produtos comprados e retorna o valor a ser pago, já com o desconto calculado;
    k) método chamado `toCSV` para obter uma linha no formato CSV (*Comma Separated-Values*) com todas as variáveis de estância separadas por ponto-e-vírgula.

    O seguinte trecho de código deve ser um exemplo de uso desta classe.

```
#include <iostream>
#include <sstream>
using namespace std;
int main() {
    Produto p1;
    cout << p1.toCSV() << endl;
    p1.defineCodigoBarras("7896324274608");
    p1.defineDescricao("Biscoito Mosmann Maria Tradicional 400g");
    p1.definePreco(4.8);
    p1.defineDesconto(4,3);
    cout << p1.toCSV() << endl;
    for (int i=0;i<=9;++i)
        cout << p1.obtemTotal(i) << endl;
    Produto p2("7891095100552","Lentilha Yoki 500g",4.5,5,3);
    cout << p2.obtemCodigoBarras() << endl;
    cout << p2.obtemDescricao() << endl;
    cout << p2.obtemPreco() << endl;
    for (int i=0;i<=10;++i)
        cout << p2.obtemTotal(i) << endl;
    return 0;
}
```

E a execução deste código deve gerar como resultado exatamente a seguinte saída.

```
;;0;1;1
7896324274608;Biscoito Mosmann Maria Tradicional 400g;4.8;4;3
0
4.8
9.6
14.4
14.4
19.2
24
28.8
28.8
33.6
7891095100552
Lentilha Yoki 500g
4.5
0
4.5
9
13.5
18
13.5
18
22.5
27
31.5
27
```

13. Utilizando a STL (*Standard Template Library*), implemente um trecho de código em C++ para manipular uma lista (`list` da STL) de objetos de uma classe chamada `Contato` (que armazena nome e telefone), cuja declaração deve ser exatamente a seguinte:

```
class Contato {
  private:
    string nome;
    string telefone;
  public:
    Contato(string n="",string t="");
    ~Contato();
    string obtemNome();
    string obtemTelefone();
    void defineNome(string n);
    void defineTelefone(string t);
    string toString();
};
```

Seu código deve:

1. declarar uma lista de ponteiros para objetos do tipo `Contato` (`Contato*`);
2. alocar 5 objetos e inseri-los nessa lista;
3. percorrer a lista mostrando cada um de seus itens (use o método `toString` para mostrar cada item);
4. ler um nome (`string`) e a seguir percorrer esta lista para verificar se este nome está entre os objetos da lista, e, se estiver, mostrar o respectivo telefone do contato com este nome;
5. desalocar todos os objetos alocados, limpando a lista.

O número de vezes que o construtor da classe citada é executado e o número de vezes que o seu destrutor é executado devem ser iguais.

14. Sobre exceções em C++, qual das afirmações a seguir está ERRADA?
    (A) Exceções são uma forma de, inicialmente, identificar situações excepcionais da execução de um código, para, em seguida, associar uma ação à ocorrência destas circunstâncias.
    (B) Usa-se asterisco em um comando ou bloco de tratamento para indicar que este comando ou bloco será um tratador padrão, capturando todas as exceções NÃO capturadas por outros tratadores.
    (C) É possível sobrescrever métodos da classe `exception`, para personalizar o tratamento de exceções.
    (D) É possível criar exceções personalizadas através de herança.
    (E) O parâmetro fornecido para o comando `throw` é passado como argumento para os blocos de tratamento.