Lista 6 (EXERCÍCIOS)

1) Identifique os polimorfismos possíveis em C++, utilizando exemplos.

2) Exemplifique em código uma situação de polimorfismo estático e uma de polimorfismo dinâmico em C++.

3) Construa uma classe Base com um método imprimir() que imprima "base" na tela. Construa uma classe Derivada com um método imprimir() que imprima "derivada" na tela. Então, teste o código abaixo para todas as quatro combinações onde imprimir() pode, ou não, ser virtual. Explique o resultado para cada teste.

```
Base A;
Derivada B;
Base &C = B;

A.imprimir();
B.imprimir();
C.imprimir();
```

4) Explique o que são classes abstratas e qual o tipo de comportamento que elas modelam.

5) Pode ser instanciado um objeto de uma classe abstrata? Explique.

6) Explique a sequência de chamadas entre uma classe Base e uma classe Derivada cujos destrutores sejam virtuais.

7) Considere o programa abaixo (**o qual não compila**):

```
#include <iostream>

using namespace std;

class Base {
public:
 Base() { f(); }
 virtual void f() = 0;
 void chama_f() { f(); }
};

class Derivada : public Base {
```

```
public:
 Derivada() { f(); }
 virtual void f() { cout << "D"; }
};

int main()
{
 Derivada d;
 return 0;
}
```

Observe o erro gerado pelo compilador/ligador. Agora, altere o construtor de Base para chamar **chama_f()** em vez de **f()**. O código compila? Apresenta erro em tempo de execução? Explique.

Dica: pesquise por "*purecall error*".

OBS: **nunca** chame funções virtuais no construtor ou destrutor. Leia: http://www.artima.com/cppsource/nevercall.html

8) Para o código anterior, remova qualquer chamada de dentro do construtor Base e adicione a seguinte definição (antes do main):

```
void Base::f()
{
    cout << "B";
}
```

Dentro do main chame **d.Base::f()**. Já que f() é definida como virtual pura, ela pode ter uma definição? Ela pode ser chamada desta maneira? Explique.

Dica: estude o seguinte artigo do Herb Sutter: http://www.gotw.ca/gotw/031.htm

9) Altere o exercício de polígonos da lista anterior para que a classe Polígono seja virtual pura. Chame a função área de um Retangulo ou Triangulo, considerando o objeto como um Polígono.

Dica: para este exercício é possível utilizar cast, ponteiros ou referências.

10) Altere seu código do xadrez para utilizar polimorfismo. Construa a estrutura de dados do Tabuleiro de maneira a conter objetos do tipo Peca. Implemente como virtuais as funções que verificam e realizam movimento.

11) O código abaixo é um erro comum de programação. É intuitivo para o programador que o método correto fosse chamado, entretanto, isto não ocorre. Explique o problema.

```
#include <iostream>

using namespace std;

class Pai
{
public:
    int valor;

    virtual void imprime()
    {
        cout << valor << endl;
    }
};

class Filho : public Pai
{
public:
    int valor2;

    virtual void imprime()
    {
        cout << valor << endl;
        cout << valor2 << endl;
    }
};

int main()
{
    Pai a;
    Filho b;

    a = b;

    // imprime age como Pai e não como Filho
    a.imprime();

    return 0;
}
```

DICA: Pesquise por *slicing*.

12) O código abaixo apresenta um erro comum de programação utilizando classes abstratas. Explique como este código poderia ser consertado.

```
#include <iostream>
#include <vector>

class Pai
{
public:
    virtual void imprime() = 0;
};
```

```
class Filho : public Pai
{
public:
    virtual void imprime() { std::cout << "filho" << std::endl; }
};

int main()
{
    // Armazena um vetor de Pais
    std::vector<Pai> vet;

    // Cria Filho
    Filho obj;

    // Nao é possivel, tenta criar objeto Pai vet[0] que é abstrato
    vet.push_back(obj);

    return 0;

}
```

DICA: Existe um idioma associado à solução. Embora sua aplicação não seja comum. Pesquise por *covariant return types* e leia a entrada 20.8 do C++ FAQ sobre construtores virtuais:

[20.8] *What is a "virtual constructor"?*