SEXTA LISTA DE EXERCÍCIOS DE PROGRAMAÇÃO III

1) Explique a diferença entre o Paradigma Estruturado e o Paradigma Orientado a Objetos.

No Paradigma Estruturado, as informações estão separadas das funções que as manipulam. Assim, as funções precisam receber as informações como parâmetros para poderem realizar as operações sobre elas. No Paradigma Orientado a Objetos, as informações estão acopladas às funções que as manipulam, nos chamados Objetos. Assim, as funções, denominadas de comportamentos dos objetos, podem acessar as informações presentes nos objetos sem recebê-las como parâmetros.

Além disso, o Paradigma Estruturado se preocupa em modelar as informações como variáveis e o programador tem controle mais eficaz do fluxo de execução do programa. Já no Paradigma Orientado a Objetos as informações são modeladas para representar mais fielmente o mundo real, sendo mais próximas do ser humano, mas o programador possui controle menor sobre o fluxo de execução do programa.

2) Explique os quatro conceitos principais da orientação a objetos e dê exemplos.

Os quatro princípios principais são: abstração, encapsulamento, modularização e herança.

Abstração é a capacidade de prover uma visão simplificada do mundo real, considerando apenas os aspectos relevantes para o escopo do sistema sendo desenvolvido.

Encapsulamento diz respeito à separação entre os aspectos externos de um objeto, acessíveis por outros objetos, e seus detalhes internos de implementação, que ficam ocultos dos demais objetos. Assim, abstração e encapsulamento são conceitos complementares: enquanto a abstração enfoca o comportamento observável de um objeto, o encapsulamento oculta a implementação que origina esse comportamento.

Modularização diz respeito ao desenvolvimento de sistemas que sejam decompostos em um conjunto de módulos coesos e fracamente acoplados.

Herança é uma característica que permite descrever propriedades e comportamentos comuns a várias classes, mantendo separadas somente as informações divergentes. A partir da herança pode-se trabalhar com outros conceitos, como polimorfismo e sobrescrita.

3) Qual a diferença entre os dois tipos de declaração de objetos abaixo:

```
int main()
    Pessoa p;
    p.nome = "Gabriel";
    p.imprimeNome();
```

```
int main()
    Pessoa *p;
    p = new Pessoa();
    p->nome = "Gabriel";
    p->imprimeNome();
```

No primeiro caso é feita a declaração de um objeto da classe Pessoa de forma estática, ou seja, é alocada toda a memória necessária para armazenar um objeto do tipo Pessoa na memória principal. Assim, é possível o preenchimento das informações desse objeto desde sua criação.

No segundo caso é feita a declaração de um objeto da classe Pessoa de forma dinâmica, ou seja, é feita a alocação de uma referência ao objeto. Posteriormente, quando for necessário utilizar o objeto, é alocada a memória necessária para armazenar um objeto do tipo Pessoa, utilizando o operador new. Assim, só será possível preencher as informações desse objeto após a alocação da memória referente a ele.

4) Explique como é a destruição de objetos em Java e em C++, comparando as duas formas.

Em C++, a destruição dos objetos alocados dinamicamente fica a cargo do programador, que deve utilizar a palavra reservada delete para limpar a memória previamente ocupada. Já em Java a destruição dos objetos fica a cargo de uma funcionalidade da máquina virtual, chamada Garbage Collector, que limpa toda a memória alocada cuja referência tenha sido perdida pelo programa. A chamada do Garbage Collector não é controlada pelo programador.

5) Dado o código em C++ abaixo explique o problema que pode ocorrer:

```
//File: Airbus.h
#include "Boeing.h"
namespace Airbus
{
	class Carrier
	{
		Carrier();
		~Carrier();
	};
}
```

```
//File: Boeing.h
#include "Airbus.h"
namespace Boeing
{
	class Carrier
	{
		Carrier();
		~Carrier();
	};
}
```

```
// main.cpp : Defines the entry point for the console
application.
#include "Boeing.h"
#include "Airbus.h"

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

O problema que pode ocorrer é a inclusão circular de bibliotecas, uma vez que a biblioteca Boeing.h inclui a biblioteca Airbus.h e a biblioteca Airbus.h inclui a biblioteca Boeing.h . Para resolver esse problema deve-se utilizar header guards, como #pragma once ou #ifndef #define #endif, para que as bibliotecas sejam definidas somente uma vez mesmo que sejam incluídas várias vezes.

6) Explique o funcionamento dos modificadores de acesso;

Os modificadores de acesso permitem que classes acessem atributos ou métodos de outras classes. Existem três tipos principais de modificadores: public, private e protected, presentes em ambas as

linguagens C++ e Java. Java ainda possui um quarto modificador, o modificador default. A primeira tabela sumariza o funcionamento dos modificadores em C++ e a segunda tabela sumariza o funcionamento dos modificadores em Java.

Access	public	protected	private
Same class	yes	yes	yes
Derived classes	yes	yes	no
Outside classes	yes	no	no

	private	default	protected	public
mesma classe	sim	sim	sim	sim
mesmo pacote	não	sim	sim	sim
pacotes diferentes (subclasses)	não	não	sim	sim
pacotes diferentes (sem subclasses)	não	não	não	sim

7) Explique o funcionamento das friend functions em C++.

As funções amigas permitem que funções de fora das classes acessem dados privados ou protegidos dessas classes. A função deve ser declarada em todas as classes das quais ela precisar de acesso às informações, usando a palavra reservada friend.

8) Explique a diferença entre herança privada, protegida e pública em C++.

Na herança pública, membros públicos da classe base se tornam membros públicos na classe derivada, membros protegidos da classe base se tornam membros protegidos na classe derivada e membros privados da classe base nunca serão acessados diretamente pela classe derivada mas podem ser acessados por membros públicos e protegidos da classe base.

Na herança protegida, membros públicos e protegidos da classe base tornam se membros protegidos na classe derivada.

Na herança privada, membros públicos e protegidos da classe base tornam se membros privados na classe derivada.

9) Explique as vantagens do uso de tipos genéricos.

Tipos genéricos nos permitem escrever funções ou tipo de dados genericamente, suportando operações idênticas sem depender do tipo da informação sendo manipulada. São estruturas que permitem a criação de funções e classes genéricas, que serão instanciadas pelo compilador para um tipo específico em tempo de execução.

10) Explique os conceitos de Sobrecarga e Sobrescrita.

Sobrecarga de métodos permite declarar métodos de mesmo nome mas com parâmetros e implementação diferentes. É possível criar múltiplas definições de uma mesma função no escopo da classe. A definição da função precisa ser única nos tipos de parâmetros e/ou no número de parâmetros, não sendo possível sobrecarregar uma função apenas mudando o retorno. Quando uma função sobrecarregada é chamada, o compilador determina a definição mais apropriada a ser utilizada através da comparação dos tipos de parâmetros utilizados para chamar a função.

11) Explique, com exemplos, como Java resolve o problema da herança múltipla.

Como Java não implementa herança múltipla nativamente, uma maneira de declarar uma classe herdando informações de duas classes é através de interfaces. Assim, uma das classes deve ser implementada como uma interface (classe abstrata) e a outra classe como concreta. A partir delas, a terceira classe herdaria as informações da classe concreta e implementaria a interface, recebendo as informações e métodos das duas classes simultaneamente.

12) Explique os conceitos de upcasting, downcasting e amarração tardia de tipos em Java.

Ampliação (upcasting) é a conversão implícita de uma subclasse para uma superclasse. Usa-se ampliação para escrever métodos mais gerais, para poupar tempo e esforço de desenvolvimento. Em linguagens OO com polimorfismo, não tem-se como saber o tipo real do objeto em tempo de compilação; assim, a amarração é feita em tempo de execução, também conhecida como amarração tardia. Ampliação é automática e livre de erros, sendo feita de forma não explícita.

Estreitamento (downcasting) é a conversão explícita de uma superclasse para uma subclasse. Utiliza-se estreitamento para relembrar a classe específica do objeto e chamar métodos que não estão na interface da superclasse. Estreitamento é manual e pode causar erros, tendo de ser feita explicitamente pelo programador.

O mecanismo que verifica o tipo de um objeto em tempo de execução chama-se RTTI. Este mecanismo garante que as conversões são sempre seguras e não permite que um objeto seja convertido para uma classe inválida.