Disciplina...: Programação Orientada a Objetos
Professor...: José Antonio Gonçalves
Aluno...: Victor Ribeiro Calado RA:2313553

1º Questionário

Obs.: As questões desta lista de exercícios deverão ser respondidas ao longo da disciplina e entregues (via Moodle) ao final da disciplina. As respostas serão encontradas nos slides (inclusive nos já postados).

- As questões foram elaboradas com o intuito de ser um check-list de estudos.
- Poderão ser usadas na elaboração das provas;
- Caso tenha dúvidas a respeito de qualquer uma delas, por favor, me envie uma mensagem **via Moodle.**

---

1)_ Por que falamos que Java é totalmente aderente às técnicas de Orientação a Objetos?

Java permite que implemente de conceitos de abstração, atributos, classes, objetos, etc... por isso dizemos que a linguagem é totalmente aderente às técnicas de Orientação a Objetos.

2)_ Explique o que é e como se utiliza o processo de “abstração”.

É utilizada para a definição de entidades do mundo real. Onde são criadas as classes. Essas entidades são consideradas tudo que é real, tendo como consideração as suas características e ações.

3)_ Quais são os artefatos produzidos na programação orientada a objetos?

Classes, atributos, métodos e objetos.

4)_ O que é um “tipo primitivo” de dados?

Um tipo primitivo (nativo ou básico) é fornecido por uma linguagem de programação como um bloco de construção básico.

5)_ O que é um “tipo abstrato” de dados?

Um tipo abstrato de dados pode ser visto como um modelo matemático que encapsula um modelo de dados e um conjunto de procedimentos que atuam com exclusividade sobre os dados encapsulados.

6)_ Explique o que é o “Garbage Collector”. Como este recurso pode dinamizar o funcionamento do sistema?

Garbage Collector é um Coletor de Lixo que “limpa” da memória principal os objetos que não estão sendo mais usados. Isso acontece assim que eles perdem a referência, liberando espaço na memoria e impedindo que ocorra o esgotamento da memória.

7)_ Considerando o **modo Shell** (linhas de comando) do sistema operacional Windows, como se faz para:

7.a)_ Compilar um código fonte Java;

No terminal deve-se digitar javac *arquivo*.java

7.b)_ Fazer com que a J.V.M. (Máquina Virtual Java) execute uma aplicação Java.

Terminal deve-se digitar java *arquivo* que foi compilado anteriormente.

8)_ O que é o “ByteCode”?

O código escrito em linguagem java é compilado em uma linguagem intermediaria de código, o Bytecode, que é interpretada pelas máquinas virtuais java.

9)_ Explique o que é a característica “Portabilidade”. Como isto é possível com aplicações Java? Para esta resposta relacione 4 “personagens” deste cenário: o código fonte (arquivo .java), o byteCode (arquivo .class), o Sistema Operacional e a JVM (Java Virtual Machine).

É a característica que a linguagem java possui que é possível executar o mesmo código em diferentes sistemas, através do bytecode que obtido através do código fonte, que quando compilado vira o bytecode, que ira ser interpretado pela máquina virtual java do sistema.

10)_ Justifique a afirmação que diz que “a segurança em Java se dá em dois níveis: proteção de hardware e proteção de software”.

Proteção do Hardware (proteção da RAM): Pelo fato de Java não implementar “ponteiros”, garante a integridade no gerenciamento da memória principal. O que evita que inadvertidamente o “programador” aloque um espaço que já está sendo utilizado por outra aplicação.

Proteção ao software: Grande quantidade de API´s (Interfaces para Programação de Aplicações). Estas API´s, fornecidas na “bibliotecas” nativas de Java, durante sua instalação foram testadas inúmeras vezes, reduzindo assim a margem de erros durante a construção de uma aplicação.

11)_ Explique como aplicamos o conceito de “Modularidade” em Java. Na resposta desta questão deve-se trata dos conceitos sobre “Acoplagem” e “Coesão”.

Na programação orientada a objeto os dados e procedimentos que manipulam estes dados são definidos numa unidade única, o objeto, o que permite a modularidade, podendo assim ser usado em vários momentos durante o código de forma independente.

11. a)_ Como esta característica pode ajudar na questão da “Manutenibilidade”?

A modularidade, fica mais fácil de fazer manutenção em um modulo especifico, uma vez que são independentes pode-se fazer as alterações necessárias sem afetar o restante do projeto.

12)_ Para servem os objetos:

12. a)_ this;

O objeto this faz uma referência a um membro (atributo ou método) da classe.

12.b)_ super.

A palavra super representa uma chamada de método ou acesso a um atributo da superclasse, por isso tem esse nome.

13)_ Usando Java, dê um exemplo que contemple as respostas das questões 12.a e 12.b.

```
this .cpf =cpf;

super.metodo();
```

14)_ Dentre os conceitos de sustenta a Orientação a Objetos, explique:

a)_ Encapsulamento:

14.a.i)_Seus níveis (explique cada um dos três níveis);

(public): todos têm acesso. Um atributo pode ter seu valor alterado a partir de qualquer outro código, mesmo sendo este de uma classe qualquer.

(protected): em Java tem acesso quem está no mesmo pacote ou classes que herdem a classe que contenha atributo ou método protegido.

(private): Restrição total fora da classe. Só têm acesso membros da própria classe.

14.a.ii)_ Como o Encapsulamento pode nos ajudar na padronização, segurança e “manutenibilidade” no desenvolvimento de sistemas;

14. b)_ Herança:

14.b.i)_ Explique os conceitos que “Generalização” e “Especialização”;

Generalização é definição de uma entidade que é um superconjunto de uma outra entidade.

Especialização é definição de uma entidade que é um subconjunto de uma outra entidade

14.b.ii)_ Como o mecanismo de Herança pode nos ajudar na padronização, segurança e “manutenibilidade” no desenvolvimento de sistemas;

Como as classes podem herdar características de outras classes isso facilita a padronizar um programa de modo que ele fica mais fácil de ser lido e mantido e seguro, já que vão ter classes que só podem ser acessadas através de outras por conta da herança.

14.b.iii)_ Explique o conceito de “Reusabilidade”. Como este é aplicado no mecanismo de Herança e, ainda, como esta possibilidade nos ajuda no dinamismo da codificação.

O conceito de reusabilidade reside na capacidade de se utilizar a mesma classes diversas vezes de forma independente, isso facilita na codificação já que se pode chamar a classe varias vezes.

14. c)_ Polimorfismo:

14.c.i)_ Sobrecarga;

A sobrecarga de métodos (overload) é um conceito do polimorfismo que consiste basicamente em criar variações de um mesmo método, “a criação de dois ou mais métodos com nomes totalmente iguais em uma classe”. A Sobrecarga permite que utilizemos o mesmo nome em mais de um método contanto que suas listas de argumentos sejam diferentes para que seja feita a separação dos mesmos.

14.c.ii)_ Sobrescrita;

A sobrescrita (ou override) está diretamente relacionada à orientação a objetos, mais especificamente com a herança. Com a sobrescrita, conseguimos especializar os métodos herdados das superclasses, alterando o seu comportamento nas subclasses por um mais específico.

14.c.iii)_ Coerção.

Força um objeto a “vestir” uma roupagem específica. É comum em casos em que precisamos definir um modelo genérico que seja, depois, redefinido/ especializado por outras classes de objetos. Para que isso seja possível, o template/padrão é construído utilizando apenas tipos suficientemente abstratos (a partir de interfaces ou ainda classes abstratas), do qual se derivam/especializam diversos tipos de objetos de acordo com a necessidade.

15)_ Construa um programa para exemplificar as respostas das questões 14.a, 14.b e 14.c.

```
abstract class Mamifero {   public abstract double
obterCotaDiariaDeLeite();} class Vaca extends Mamífero {
 public double obterCotaDiariaDeLeite(){
 return 20.0;}}
class Cabra extends Mamifero {
 public double obterCotaDiariaDeLeite() {
  return 5.0;}}
class Aplicativo {   public static void main(String args[]){
   System.out.println("Polimorfismo\n");
   Mamifero mamifero1 = new Vaca();
   System.out.println("Cota diaria de leite da Vaca: " + mamifero1.obterCotaDiariaDeLeite());
   Mamifero mamifero2 = new Cabra();
   System.out.println("Cota diaria de leite da Cabra: " + mamifero2.obterCotaDiariaDeLeite());  }}
```

16)_ Explique o que são trocas de mensagens? Como isso acontece?

Esta troca de mensagens se dá pela declaração de objetos dos tipos de classes e pela invocação dos métodos através dos objetos declarados.

17)_ O que é um “método construtor”? Qual sua importância? Faça um código que demonstre sua explicação.

Construtores são métodos especiais chamados pelo sistema no momento da criação de um objeto. Eles não possuem valor de retorno, porque você não pode chamar um construtor para um objeto, você só usa o construtor no momento da inicialização do objeto.

```
public class TesteClasse {    public TesteClasse() {
// o metodo construtor
    System.out.println("Hellou World!!"); }}
public class Teste {
 public static   void main(String args[]) {
     TesteClasse Obj1 = new TesteClasse();}}
```

18)_ Explique o que são como e quando utilizamos:

18.a) Classe *abstrata*;

As classes abstratas são as que não permitem realizar qualquer tipo de instância. São classes feitas especialmente para serem modelos para suas classes derivadas

18.b) Método *abstrato*;

Em orientação a objetos, método abstrato é o método de uma classe abstrata que não possui implementação. Na classe abstrata, é definido o método abstrato com palavra reservada abstract.

18.c)_ Classe *final*;

A classe final quando aplicada, não permite estende-la, nos métodos impede que o mesmo seja sobrescrito (overriding) na subclasse, e nos valores de variáveis não pode ser alterado depois que já tenha sido atribuído um valor.

18.d)_ Atributo *final*;

Um atributo final de uma classe pode ter seu valor atribuído uma única vez, na própria declaração ou no construtor.

18.e)_ Método *final*.

O método final serve para que quando uma subclasse a chame da mesma maneira que foi criada, sem que haja mudanças em seu comportamento. Isso acontecendo com uma classe ela não pode ser herdada.

19)_ Dentro da tecnologia Java, explique o que é a estrutura de dados "*Interface*". Quando a utilizamos?

Uma classe trata-se de um tipo abstrato de dados:

Todos os métodos que ela contiver, deverão ser construidos nas classes que implementam esta Interface, assemelhando-se aos métodos abstratos.

A Interface contendo um atributo, então será constante, ou seja, não podendo ser alterado seu valor. Se comportarão como (atributos finais), que é utilizada para suprir a necessidade de herança múltipla, já que não é possível implementar está forma de herança em Java.