JULIO CESAR GARCIA RIBEIRO **RA: 1994484**

QUESTIONÁRIO

1 – A linguagem Java é construída de classes, e, apesar de possuir tipos primitivos (que não são objetos), existem classes para cada um desses tipos, fazendo com que até os tipos primitivos se tornem objetos na linguagem.

2 – A abstração ocorre no processo de levantamento de requisitos e tem como objetivo pegar objetos do mundo real e os colocar dentro no software. Por exemplo, criarmos um sistema de gerenciamento de carros, pegamos o objeto carro, definimos as características necessárias deste objeto carro, como marca e modelo, e definimos as ações que este carro irá realizar dentro do sistema.

3 – São estes: Classe; Atributo; Métodos; e Objetos.

4 – Um tipo primitivo tem sua definição de espaço necessário a ser alocado na memória pré-definido. São exemplos de tipos primitivos: int, float, char.

5 – Um tipo abstrato é um tipo que não tem um valor fixo a ser alocado na memória. Ele é criado utilizando tipos primitivos, como classes, que tem a quantidade de memória que precisa ser alocada para a criação de sua instância calculada com base nos atributos e nos métodos que esta classe possui.

6 – Garbage collector é um processo de gerenciamento automático de memória que localiza objetos que já foram utilizados no programa e não são mais necessários e os exclui, liberando espaço na memória. Este processo faz com que o funcionamento do sistema seja dinâmico pois o gerenciamento de memória é automático, liberando recursos de memória vital.

7 –

a – javac código_fonte.java
b – java código_fonte.class

8 – ByteCode é o arquivo executável gerado pelo compilador Java, que pode ser executado na máquina virtual Java.

9 – Portabilidade garante que o mesmo arquivo executável possa ser rotado em diferentes tipos de hardware sem necessidade de recopilação do código fonte. O Java implementa portabilidade do bytecode (arquivo executável) utilizando a JVM (Java Virtual Machine), que é instalada no sistema operacional da máquina em questão. O Código fonte é compilado para um bytecode, e este mesmo bytecode pode ser executado em diferentes computadores, com diferentes sistemas operacionais, desde que possuam a JVM instalada.

10 – Java implementa segurança em nível de software trazendo funcionalidades como o encapsulamento, onde é possível definir os limites de acesso de cada componente das classes. Não é necessário o código fonte de uma classe para rodar uma aplicação, apenas o seu bytecode, tornando difícil a modificação deste arquivo.

11 – Coesão faz com que cada classe apenas cumpra as suas tarefas e nada mais, não assumindo responsabilidades que não são suas. Ignorado esse principal torna difícil a manutenção e reuso, pois não sabemos qual classe realiza tal operação, já que esta não cumpre apenas as suas tarefas. Acoplagem tem relação as relações entre as classes. Quando temos muito acoplamento estamos garantindo que cada classe realiza a sua função de maneira correta, e, quando é necessário realizar outras operações são chamadas outras classes para realizar as mesmas, gerando acoplamento.

12 –

a – O objeto this faz referência para membros da própria classe. Se possuirmos um atributo chamado nome e queremos atribuir a este um valor de uma variável também chamada de nome, podemos utilizar o objeto this no atributo nome, desta forma garantimos que o atributo nome irá receber o valor da variável nome.

b – O objeto super faz referência a métodos sobescritos. Quando se tem um método na classe mãe que foi sobrescrito por um método na classe filha, se pode acessar esse método sobescrito com a palavra super.

13 –

```
public class Ex12 {
    private int questao12;

    public int getQuestao12() {
        return questao12;
    }

    public void setQuestao12(int questao12) {
        // Aqui a palavra this garante que o atributo questão12 ira receber
        // o conteúdo so parâmetro questão12.
        this.questao12 = questao12;
    }
}
```

14 –

a – Encapsulamento define o tipo de acesso de cada elemento das classes. Existem 3 tipos de acesso: *public* (qualquer classe pode acessar os elementos da classe); *private* (apenas elementos da mesma classe podem acessá-los); *protected* (apenas classes que herda a dona dos elementos com a palavra *protected* podem acessá-las). Encapsulamento ajuda na segurança garantindo que cada classe tem acesso apenas aos elementos necessários, diminuindo brechas no sistema.

b – Generalização é gerada pelas classes mais altas na hierarquia de herança, enquanto especialização é gerada pelas classes mais abaixo da hierarquia. Classes mais acima tem métodos ou atributos mais gerados, que podem ser sobrescritos para especializar uma função. Herança gera re-usabilidade, visto que se temos uma classe mais geral, com métodos que podem pertencer a outras classes, não precisamos ficar declarando os métodos em todas as classes, basta fazer uma herança.

c – Polimorfismo é gerando com a sobrescrita de métodos (muitas formas de uma coisa). Existem 3 tipos de polimorfismo: sobrecarga (métodos na mesma classe com o mesmo nome, mas assinaturas diferentes); sobrescrita (quando acontece a sobrescrita de um método de uma classe mãe pela classe filha é gerado o polimorfismo por sobrescrita, ambos os métodos devem estar em classes diferentes (mãe e filha) e devem possuir a mesma assinatura); coerção (em objetos que se comportam como outros, exemplo, pessoa que se comporta como professor, desta forma, uma classe pessoa pode acessar os métodos da classe

professor, desde que a classe filha (professor) tenha um método sobrescrito da classe pessoa (mesma assinatura)).

15 –

```
public class Mae {
    private String nome;    // Encapsulamento
    private int idade;
    protected int nascimento;

    public String getNome() {   // Encapsulamento
        return nome;
    }
    public int getIdade() {   // Encapsulamento
        return idade;
    }

    public void setNome(String nome) {   // Encapsulamento
        this.nome = nome;
    }
    public void setIdade(int idade) {   // Encapsulamento
        this.idade = idade;
    }

    public void exibeNome() {   // Encapsulamento
        System.out.println("Esta é a classe Mãe");
    }
}

public class Filha extends Mae {    // Herança
    public Filha(String nome) {   // Polimorfismo por sobrecarga
        setNome(nome);
    }
    public Filha(String nome, int idade, int nascimento) {     // Polimorfismo por
sobrecarga
        setNome(nome);
        setIdade(idade);
        this.nascimento = nascimento;   // Protected pode ser acessado sem set()
    }

    public void exibeNome() {   // Polimorfismo por sobrescrita
        System.out.println("Esta é a classe Filha");
    }
}
```

16 – Trocas de mensagens ocorrem em chamadas de métodos de um objeto por outros objetos. É o ato de enviar mensagens de uma classe para a outra, chamando um método com parâmetros e passando elementos de uma classe para outra por esses parâmetros.

17 – O método construtor tem como objetivo dizer ao compilador qual o tamanho de memória que deve ser alocado para que a classe possa ser instanciada. Ele calcula o tamanho necessário para armazenar todos os atributos e métodos na classe na memória.

18 –

a – Classes abstratas são classes que não podem ser instanciadas, possuem apenas atributos e assinaturas de métodos, sendo classes que servem para ser classes mãe (mais generalizadas).

b – Métodos abstratos são métodos que devem ser sobrescritos por classes filhas quando existe uma relação de herança entre uma classe e outra. Eles estão contidos na classe mais geral e apenas possuem assinatura na classe mais generalizada.

c – Uma classe final é uma classe que não pode ser estendida, não podem servir como classes mãe.

d – Um atributo final pode ter o seu valor alterado apenas uma vez, na própria declaração ou no construtor. Podemos fazer uma analogia com constantes.

e – Um método final não pode ser sobrescrito em subclasses.

19 – Interfaces são utilizadas para gerar herança múltipla. Como o Java não suporta herança múltipla, são criadas interfaces que podem ser implementados por classes mais especialistas. Interfaces possuem apenas métodos abstratos, ou sejam, devem ser sobrescritos pelas subclasses. Deste modo, ela obriga as subclasses a possuírem métodos ou propriedades em comum.