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

Devido à possibilidade de implementação dos conceitos de OO, tais como abstração, classes, atributos, objetos, entre outros.

---

2. Explique o que é e como se utiliza o processo de "abstração".

O processo de abstração se dá quando transcrevemos as definições (características e ações) do mundo real para as classes.

---

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?

Os tipos primitivos são blocos de construção básico fornecido por uma determinada linguagem de programação.

---

5. O que é um tipo abstrato de dados?

Os tipos abstratos de dados são um conjunto de procedimentos cuja prioridade é atuar sobre os dados encapsulados.

---

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

O Garbage Collector tem uma função muito importante na linguagem Java. A sua função é fazer um despejo de memória quando os objetos não estão mais sendo usados. Assim que os objetos perdem a referência, são limpos da memória, liberando espaço e impedindo que ocorra qualquer esgotamento de memória.

---

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

- **A. Compilar um código fonte Java;**
  - *No terminal deve-se digitar javac Arquivo.java*
- **B. Fazer com que a J.V.M. (Máquina Virtual Java) execute uma aplicação Java.**
  - *No terminal deve-se digitar java nomedoarquivo que foi compilado anteriormente.*

---

8. O que é o ByteCode?

O ByteCode é gerado na compilação do código Java, gerando assim uma linguagem intermediária de código que é interpretada pela JVM (Java Virtual Machine), e, dessa forma, executar as aplicações em Java.

---

9. Explique o que é a característica Portabilidade. Como isto é possível com aplicação Java? Para esta resposta relacione 4 "personagens" deste cenário:

- **I. O código fonte (arquivo .java);**
- **II. O byteCode (arquivo .class);**
- **III. O Sistema Operacional;**
- **IV. A JVM (Java Virtual Machine).**

A portabilidade é uma característica muito forte em Java, onde, através do ByteCode que é obtido na compilação do código-fonte, pela JVM, é possível executar o mesmo código em diferentes sistemas operacionais.

---

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

Proteção de Hardware: *O Java garante a integridade na gestão da memória principal, devido ao fato da linguagem não implementar ponteiros. O que evita que o programador aloque algum espaço de memória que já esteja a ser usada por alguma outra aplicação.*

Proteção de Software: *O Java se dispõe grandes quantidade de API's, fornecidas nas bibliotecas nativas de Java. Essas bibliotecas, foram testadas inúmeras vezes, o que reduz, significamente, erros durante a construção de uma determinada aplicação.*

---

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

Na programação orientada a objeto (POO), os dados são manipulados por métodos e procedimentos definidos numa unidade única, o objeto, o que permite a ***Modularidade*** *de modo que seja utilizada em vários momentos durante o código, e de forma independente.*

- **A. Como esta característica pode ajudar na questão da Manutenibilidade?**

  Visto que o módulo se torna mais específico e independentes com a modularização, pode-se fazer as alterações necessárias sem afetar o restante do projeto. A manutenibilidade se torna muito mais viável.

---

12. Para servem os objetos:

- **A. this;**

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

- **B. super.**

  O objeto super, representa uma chamada de método ou acesso a um atributo da superclasse.

---

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

```
this.nome = nome;
```

```
super.metodo();
```

---

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

- **A. Encapsulamento:**
  - **I. Seus níveis (explique cada um dos três níveis);**
  - **PÚBLICO (public):** *todos têm acesso. Um atributo pode ter o seu valor alterado a partir de qualquer outro código, mesmo sendo este de uma classe qualquer.*
  - **PROTEGIDO (protected):** *Tem acesso quem está no mesmo pacote ou classes que herdem a classe que contenha atributo ou método protegido.*
  - **PRIVADO (private):** *Restrição total fora da classe. Só têm acesso os membros da própria classe.*
  - **II. Como o Encapsulamento pode nos ajudar na padronização, segurança e "manutenibilidade" no desenvolvimento de sistemas;**
  - *Mantendo os serviços da classe, podemos alterar a estrutura interna da classe sem que outros trechos do código que dependam dela tenham que ser alteradas.*
- **B. Herança:**
  - **I. Explique os conceitos que Generalização e Especialização;**
  - **Generalização:** *Definição de uma entidade que é um superconjunto de outra entidade.*
  - **Especialização:** *Definição de uma entidade que é um subconjunto de outra entidade.*
  - **II. Como o mecanismo de Herança pode nos ajudar na padronização, segurança e "manutenibilidade" no desenvolvimento de sistemas;**
  - *Devido à possibilidade das classes herdarem características de outras classes, a padronização do código se torna muito maior, porque, dessa forma, torna-se mais fácil de ser lido, mantido, e seguro.*
  - **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 classe diversas vezes de forma independente, isso facilita na codificação já que se pode chamar a classe várias vezes.*
- **C. Polimorfismo;**
  - **I. Sobrecarga;**
  - *A sobrecarga de métodos (overload) é um conceito do polimorfismo que consiste basicamente em criar variações de um mesmo método, ou seja, a criação de dois ou mais métodos com nomes totalmente iguais numa classe. A Sobrecarga permite que utilizemos o mesmo nome em mais de um método contanto que as suas listas de argumentos difiram para ser feita a separação dos mesmos.*
  - **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.*
  - **III. Coerção.**
  - *Forçamos um objeto a "vestir" uma roupagem específica. Isto é muito comum em casos em que precisamos definir um modelo genérico que seja, depois, redefinido/ especializado por outras classes de objetos. Para isso ser possível, o template/padrão é construído a utilizar apenas tipos suficientemente abstratos (a partir de interfaces ou ainda classes abstratas), do qual se derivam/especializam diversos objetos conforme a necessidade.*

---

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

```
public abstract class Ave {
  public abstract double obterKmPercorrido();
}

public class Canario extends Ave {
  public double obterKmPercorrido() {
    return 20.0;
  }
}

public class Tucano extends Ave {
  public double obterKmPercorrido() {
    return 0.5;
  }
}

public class Aplicativo {
  public static void main(String args[]) {
    System.out.println("Polimorfismo\n");
    Ave ave1 = new Canario();
    System.out.println("Quilômetros percorrido de canario: " + ave1.obterKmPercorrido());
    Ave ave2 = new Tucano();
    System.out.println("Quilômetros percorrido de rato: " + ave2.obterKmPercorrido());
  }
}
```

---

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

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

---

17. O que é um método construtor? Qual a sua importância? Faça um código que demonstre a sua explicação.

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

```
public class MinhaClasse {
  public MinhaClasse() {
      //esse é o metodo construtor System.out.println("Oi!");
  }
}

public class Teste {
  public static void main(String args[]) {
    MinhaClasse minhaClasse = new MinhaClasse();
  }
}
```

---

18. Explique o que são como e quando utilizamos:

- **A. Classe abstrata;**
  - *As classes abstratas são as que não permitem realizar qualquer instância. São classes feitas especialmente para serem modelos para as suas classes derivadas.*
- **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.*
- **C. Classe final;**
  - *Quando é aplicado na classe, 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.*
- **D. Atributo final;**
  - *Um atributo final de uma classe pode ter o seu valor atribuído uma única vez, seja na própria declaração ou no construtor.*
- **E. Método final.**
  - *O método serve para que quando uma subclasse a chame da mesma maneira que foi criada, sem haver mudanças no seu comportamento. Já quando*

isso acontece 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?

Assim como uma classe, também trata-se de um tipo abstrato de dados: - Todos os métodos que ela contiver, deverão ser construídos nas classes que implementarem esta Interface, logo, na sua forma de uso, assemelha-se aos métodos abstratos;- Caso a Interface tenha algum atributo, este será do tipo constante, isto é, não poderá ter o seu valor alterado. Se comportarão como constantes (atributos "finais");- É utilizada para suprir a necessidade herança múltipla, já que não é possível implementar esta forma de herança em Java.
