Nome: Leonardo Canuto Junior **RA:** 2453533

Lista 1 – Programação Orientada a Objetos

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

R: Podemos dizer que Java é totalmente aderente às técnicas de Orientação a Objetos, pois possuí a implementação e aderência a todos os recursos da Orientação a Objetos, isso incluí um suporte nativo a classes, objetos, encapsulamento, herança, polimorfismos, interface e outros mais, assim podemos dizer que java é uma linguagem que por implementar os demais recursos do paradigma de Programação Orientada a Objetos é uma linguagem aderente.

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

R: A abstração se baseia em criar uma classe baseada em tipo de representação de uma entidade do mundo real a partir de uma listagem de tipos primitivos, por exemplo: criamos uma classe chamada automóveis, podemos criar a partir disso objetos como: moto, carro e outros. Com isso, definimos a abstração se baseando no processo de implementar objetos do mundo real voltando-se para pontos de importância dado ao programa.

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

R: Os artefatos existentes na programação orientada a objetos são 4. Tais são: classes, atributos, métodos e objetos.

4) O que é um "tipo primitivo" de dados?

R: Um tipo primitivo de dado é aquele que consegue ser reconhecido pelo sistema, o sistema já o conhece e sabe qual espaço separar, também cria seu endereçamento quando utilizado, exemplos disso são os inteiros, double, char, boolean e outros.

5) O que é um "tipo abstrato" de dados?

R: Um tipo abstrato de dado é representado por tipos criados pelos programadores ou interpretados artificialmente pelo computador, esses serão reconhecidos pelo sistema a partir de uma explicação de como estes dados funcionam, como por exemplo: String, objetos e outros.

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

R: O Garbage Collector é um recurso que vem no contexto da sua aplicação e apaga por exemplo, objetos que não tem mais referência na aplicação, como objetos criados dentro de métodos e após um tempo determinado serão "limpos" pelo Garbage Collector. O recurso dinamiza o processo pois evita o acúmulo de lixos e permite ao sistema conseguir lidar melhor com o uso de memória evitando gasto desnecessário.

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

7.a) Compilar um código fonte Java;

R: Primeiro é necessário entrar no diretório em que seu código fonte Java foi escrito, isso pode ser feito usando: cd Desktop/nome-do-diretorio

Para compilar digita-se o comando: **javac nomearquivo.java**

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

R: Com o arquivo já compilado, para que a J.V.M execute a aplicação java basta digitar no cmd: **java nomearquivo**, lembrando de estar dentro do diretório onde foi compilado o código.

8) O que é o "ByteCode"?

R: O bytecode é o código em formato já compilado, pronto para leitura por parte da máquina virtual quando executado. É uma forma de facilitar aplicações na sua execução.

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).

R: A característica de portabilidade em uma aplicação pode ser representada pela facilidade em executar de forma fácil e na maior quantidade possível de dispositivos. Nas aplicações em java podemos definir muita portabilidade pois podemos executar em quaisquer máquinas ou sistemas operacionais. Para explicarmos isso, podemos determinar 4 fases da aplicação, o código fonte que será feito pelo programador, em um sistema que tenha instalado o JDK (Java Development Kit), em sequência ele compila o seu código transformando-o em bytecode (um arquivo .class), esse contém instruções de execução que podem ser lidas e qualquer outra máquina por uma JVM (Máquina Virtual Java), nesse caso uma JVM é criada em todos aqueles sistemas operacionais ou máquinas que tem um JRE instalado (Java Runtime Environment), assim podemos definir que o código em formato de bytecode pode ser distribuído e executado por qualquer sistema operacional (tanto Linux como Windows), sem necessidade do código fonte que foi realizado pelo programador.

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

R: Podemos dizer que Java tem sua proteção nesses dois níveis, sejam de hardware e software. A proteção de hardware se dá pela linguagem ser compilada em o que chamamos de bytecode que faz os códigos escritos não serem editáveis e que poderão ser executados pela JVM. E com a JVM pode-se garantir recursos de proteção de hardware como verificação de memória, acessos invalidados ou corrompidos e gerenciamento de recursos, ou seja, a linguagem não permite o acesso direto ao hardware sem autorização. Já no nível de proteção de software podemos justificar pelos recursos oferecidos pela linguagem, como o encapsulamento que se dá por sistema de permissões dentro das classes, também permite a criação de ambientes restritos de execução, assim limita os acessos aos recursos do sistema sem autorização ao ambiente que foi criado.

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

R: Tratando de modularidade em java, podemos analisar dois conceitos que devemos ter em vista quando estamos desenvolvendo orientada a objetos. Coesão: um código mais rápido e leve de se interpretar, aquilo que é buscado pelo programador. Acoplagem: códigos que demandam sistemas maiores, uma divisão de em vários métodos, classes, objetos, com objetivo de facilitar uma organização de grandes projetos.

Com esses dois conceitos, aplicamos a acoplagem quando tratamos de modularidade, pois necessitamos dividir grandes projetos em módulos para facilitar como desenvolvemos e como organizamos nosso código.

11.a) Como esta característica pode ajudar na questão da "Manutenibilidade"?

R: Quando tratamos de manutenibilidades no código, colocamos em vista que a acoplagem nos permite dividir o código em diversos módulos e métodos que organize e facilite quando for necessário adição de novos dados, correção de erros, adição de métodos ou atributos.

12) Para servem os objetos:

12.a) this;

R: O objeto "this" serve para apontar os objetos para os membros da própria classe, por exemplo dentro de métodos ele reverencia o objeto para o atributo da própria classe, num possível caso de variáveis com o mesmo nome dentro de um método.

12.b) super.

R: O objeto "super" tem a função de apontar objetos para uma classe mãe, no caso ele é usado quando há uma classe filha que possuí objetos com nomes similares entre a classe e o objetivo é dar a prioridade a classe que está herda a subclasse.

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

R: Na criação de um método setName(), você utiliza o this para reverenciar o atributo da própria classe.

```
public class Pessoa{
        String name = "";
        public static void main(String arg[]){
                Pessoa c1;
                c1 = new Pessoa();
                c1.setName("Rafael");
        }
        public void setName(String name){
                this.name = name;
        }
}
```

Quando temos que utilizar de objetos com o mesmo nome em duas classes e queremos dar a prioridade nos da classe mãe.

```
public class Aluno extends Pessoa{
        String name = "";
        // Poderia usar essa variável como um apelido;
        public void seApresenta(){
                System.out.println("Meu nome eh " + super.name);
                System.out.println("Meu apelido entre os colegas eh "
        + name);
        }
}
```

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

14.a) Encapsulamento:

R: Encapsulamento tem a função de limitar ou gerenciar as permissões aos acessos de atributos, métodos, classes e objetos pensando na preservação de um modelo de uma hierarquia do algoritmo.

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

R: public: o método, atributo, objeto ou classe pode ser acessado por qualquer outro método ou classe.

private: só pode ser acessado por membros da própria classe e não quaisquer outros fora podem acessar diretamente.

protected: funciona como um intermediário entre public e private, pois deixa que uma classe filha possa acessar, enquanto outra classe que não seja herdeira não. Porém, se torna público para classes que estão na sua mesma pasta, porém para classes de fora que não sejam classes filhas, age como private.

package: usadas para parear com uma hierarquia no código e utilizada para propor a ideia de pacotes/bibliotecas.

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

R: O encapsulamento nos ajuda ditando regras na produção de um sistema ou um determinado código grande que possa ser gerenciado por uma grande equipe, facilitando a manutenibilidade. Isso acontece pois com o encapsulamento podemos determinar quem e onde os membros da classe que foram encapsulados poderão ser usados, assim padronizamos o uso dos métodos (como por exemplo getters e setters) e evitamos o uso irresponsável de atributos desgovernados de classes externas.

14.b)_ Herança:

R: Herança é utilizado quando temos a intenção de criar novas classes que possam herdar dados comuns de outras classes. Ela se baseia em criar um modelo de

subclasse que possa trazer consigo os atributos e métodos de outra classe. Por exemplo: pessoa (classe mãe) e aluno (classe filha), assim a classe filha poderá herdar atributos comuns de pessoa dentro da subclasse aluno.

14.b.i) Explique os conceitos que "Generalização" e "Especialização";

R: Em herança, os conceitos de Generalização e Especialização se assemelham ao uso da herança de classes. Nesse contexto, entendemos como Generalização as classes mães, que apresentam um padrão com dados generalizados, pensando que possam ser reutilizados por outras classes filhas. Já com Especialização, trabalhamos com classes filhas, que vão herdar as informações generalizadas e criarem seus próprios objetos, métodos e atributos específicos.

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

R: Com o mecanismo de Herança, nosso código consegue possuir uma melhor padronização tendo em vista que não será necessário repetir atributos em comum entre outras classes filhas. Isso reforça por consequência a segurança do código, que será possível analisar em um plano seguro e evitar um conflito de atributos e ajudar a manutenibilidade do código, tendo em vista que o programador possa de maneira mais fácil reutilizar os atributos de uma classe mãe em outra classe que seja do mesmo gênero. Como por exemplo: "Um aluno é uma pessoa". Com isso, imaginando grande sistemas com diversas linhas de códigos e arquivos, conseguimos através da Herança ter uma manutenção de código segura e limpa, sabendo a ordem e a organização de classes, reduzindo erros e facilitando a compreensão do código, por exemplo você necessita alterar algo em uma classe mãe e irá automaticamente atualizar nas classes filhas.

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.

R: Visando o conceito de Reusabilidade no contexto de Herança, podemos entender como a facilidade em criar estruturas comuns podem influenciar em códigos bem feitos que consigam ser reaproveitados sem ter a necessidade de um sistema pesado por grandes repetições desnecessárias. Como exemplo, quando visamos a criação de uma Pessoa que possuí características em comum com diversas outras, definimos um padrão que herdará outros tipos de Pessoas com atributos próprios, porém herdando a base, evitando a criação de atributos repetitivos quando formos trabalhar com diversos tipos de "Pessoas". Além disso, visando tudo isso dentro do conceito de Dinamismo, compreendemos que classes reutilizadas com atributos em comuns, facilita a dinamização da nossa codificação, fazendo com que os recursos alocados possam ser distribuídos de melhor forma e otimizando como o código será compilado e executado pelo processador.

14.c) Polimorfismo:

R: A palavra "Polimorfismo" evidencia um significado como "várias formas". Dentro desse contexto, entendemos por Polimorfismo aquilo que utiliza de um mesmo elemento, porém de formas e objetivos diferentes. Como por exemplo o uso de um mesmo objeto para várias funções e manipulações a partir dos parâmetros utilizados.

14.c.i) Sobrecarga;

R: Sobrecarga é um tipo de polimorfismo que apresenta a forma de utilizar de métodos com o mesmo nome porém parâmetros diferentes, significa “sobrecarregar” criando várias vezes a mesma classe com objetivos diferentes.

14.c.ii) Sobrescrita;

R: Sobrescrita em polimorfismo se trata de uma forma de reescrever objetos ou métodos de uma classe mãe dentro de uma classe filha, por exemplo objetos entre uma classe mães e filha que possua o mesmo nome, tipo de retorno e quantidade de parâmetros, no entanto o objetivo gira em torno da possibilidade de alterar esses objetos dentro da classe filha mas mantendo seu padrão dentro da classe mãe.

14.c.iii) Coerção.

R: Coerção em polimorfismo tem a intenção de criar objetos em classes específicas, porém que se comportem como outra classe. Nesse caso, podemos citar como a classe mãe não possuí o acesso aos métodos da classe filha. No entanto, com a coerção e criando objetos com o comportamento da classe filha, ela poderá acessar esses métodos. Se baseia no fundamento de objetos de um tipo com comportamentos diferentes.

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

R:

```
public class Pessoa{
    //Encapsulamento: private
    private String nome;
    private int cpf;
    public Pessoa(){
    //Declaração dos atributos default
        nome = "";
        cpf = 0;
    }
    public Pessoa(String nome, int cpf){
        //Polimorfismo: sobrecarga
        this.nome = nome;
        this.cpf = cpf;
    }
}
public class Aluno extends Pessoa{
    // Herança: Aluno herdando Pessoa
    private int ra;
    private String curso;
}
```

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

R: A “troca de mensagens” em java acontece quando conseguimos transmitir informações entre diferentes classes e realizar códigos a partir dessa transmissão de informações. Como por exemplo podemos citar a Comunicação entre Objetos, que utilizamos para transmitir modelos de dados a partir de objetos entre as classes com métodos getters por exemplo. Podemos citar também a Comunicação entre Atributos, que é uma prática ruim baseando-se na manutenibilidade do código.

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

R: A função do método construtor na aplicação é informar ao sistema operacional o tamanho, espaço e endereço do objeto criado. Sua importância se dá ao fato da necessidade de especificar ao sistema a construção do tipo abstrato criado.

```
public class Contexto{
        public static void main(String arg[]){
                Name c1;
                c1 = new Name();//--> Método construtor
        }
}
```

18) Explique o que são como e quando utilizamos:

18.a) Classe abstrata;

R: Classe abstrata é uma classe que não permite instanciar um objeto dela própria, usada no contexto de classes mães, onde são classes generalizadas e modelos, por isso tem a intenção de não serem instanciadas em objetos.

18.b) Método abstrato;

R: O método abstrato faz com que a implementação do método da classe mãe seja obrigatório em todas suas classes filhas. Isso é usado quando definimos um método que seja em comum entre todas as classes e ela pode sobrescrever em cada uma das classes filhas dependendo do objetivo.

18.c) Classe final;

R: A classe final é uma forma de privar uma classe para que ela não possa ser herdade por outra, ou seja não pode ser uma classe mãe. É utilizada quando temos a intenção de que uma classe não seja manipulável e não possa ser estendida.

18.d) Atributo final;

R: O atributo final basicamente força uma variável a tornar-se uma constante, assim, ela nunca pode ser alterada após ser inicializada com um valor. Apresenta uma função como uso de constantes.

18.e) Método final.

R: O método final é um método que não pode ser sobrescrito, por exemplo quando definimos um método final em uma classe mãe, ele não poderá ser sobrescrito

em outras classes filhas. É utilizado por questões de segurança, quando necessitamos privar o uso de um método exclusivo que não pode ser sobrescrito dentro de outra classe.

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

R: A estrutura de dados “Interface” deve se basear em métodos comuns que deverão ser sobrescritos por cada classe que utilizá-lo. O uso da criação de classe em modo de “Interface” se dá quando precisamos criar uma estrutura como uma Herança multiplica, visando que em Java não há essa possibilidade. Assim, com o uso da interface, os métodos terão objetivo de serem usados por vários outras classes de forma que consigam implementar múltiplas heranças e assim utilizando do Polimorfismo de sobrescrita para adaptar o método da Interface com a sua necessidade. No uso de estruturas de Interfaces, faz necessário que todos os métodos escritos na Interface sejam obrigatoriamente implementados na classe o qual vai implementá-lo.