Gabarito – Simulado P1

1. **(1,0)** Todas as classes no Java herdam direta ou indiretamente da classe *Object*; portanto, seus 11 métodos são herdados por todas as outras classes. A classe *Object em Java* provê o método *clone*, cujo objetivo é criar uma cópia de um objeto. Nesse contexto, explique por que a expressão *x.clone().equals(x)* é tipicamente verdadeira, enquanto a expressão *x.clone() == x é tipicamente falsa*.

R: A implementação padrão do método `clone()` cria uma cópia superficial do objeto, ou seja, copia todos os campos do objeto, mas não copia os objetos referenciados por esses campos. Portanto, a cópia resultante e o objeto original têm o mesmo conteúdo, e o método `equals()` compara o conteúdo dos objetos, não suas identidades de referência.

Por outro lado, a expressão x.clone() == x é tipicamente falsa porque o operador == verifica se dois objetos têm a mesma identidade de referência na memória, ou seja, se eles apontam para a mesma localização de memória

2. **(0,5) Complete com V(verdadeiro) ou F(falso). Corrija as afirmações incorretas.**

Dentre as características da linguagem Java destacam-se:

a) (F) Orientação a objetos: suporte ao paradigma de programação orientada a objetos. Portabilidade: é possível rodar um *software* feito em Java em qualquer máquina que disponha de máquina virtual implementada para ela, e sintaxe baseada na sintaxe da linguagem C. -> sintaxe baseada em linguagem Java.

b) (F) A JVM é um *software* que funciona sobre o sistema operacional, sendo responsável pelo processo de tradução de um programa Java para uma plataforma específica. Assim, um programa feito em Java pode rodar em qualquer SO de qualquer arquitetura, desde que exista uma JVM implementada para ele. Um programa Java precisa passar por um processo de Interpretação para ser analisada a existência de erros de sintaxe. Esse processo de Interpretação traduz o código-fonte escrito pelo programador para uma linguagem intermediária chamada Java bytecodes. Esse processo de tradução dos códigos fontes para Java bytecodes é feito por um programa chamado Interpretador. Então, é necessário que outra ferramenta chamada Tradutor se responsabilize por Traduzir esses bytecodes para o sistema operacional. Essa ferramenta que interpreta bytecodes é a

máquina virtual Java (JVM). -> JVM é responsável pelo processo de compilação de um programa Java. O programa JAVA não passa por um processo de interpretação para analise de existência de erros de sintaxe. O processo que traduz o código fonte é chamado Processo de Compilação e traduz o código fonte para bytecodes. Esse processo é feito pelo Java Compiler (JavaC). Não é necessário uma ferramenta Tradutor, já que o compilador já realiza a tradução.

c) (F) O conjunto de ferramentas necessárias para desenvolver, compilar e rodar aplicativos Java é disponibilizado em um kit conhecido como Java Development Kit (JDK). Assim, para começar a programar em Java você deve realizar o download do JRE e instalá-lo. Existem disponíveis vários Ambientes de Desenvolvimento – *Integrated Development Environment* (IDE), que dão suporte à linguagem Java. Um IDE é um programa de computador que reúne ferramentas de apoio ao desenvolvimento de *software* com o objetivo principal de agilizar o processo de codificação. Há vários IDEs para programação Java. Os dois mais amplamente utilizados são o NetBeans e o Eclipse. -> Para começar a programar em Java é necessário realizar o download do JDK. (NetBeans e Eclipse pode haver discordância).

d) (F) A orientação a objetos tenta gerenciar a complexidade inerente aos problemas do mundo real abstraindo o conhecimento relevante e encapsulando-o dentro de objetos. Um objeto define as características e o comportamento de um conjunto de classes. Assim, a criação de uma classe implica definir um tipo de objeto em termos de seus métodos (variáveis que conterão os dados) e seus atributos (funções que manipulam tais dados). Já uma classe é uma instância de uma classe. Ele é capaz de armazenar estados por meio de seus atributos e reagir a mensagens enviadas a ele, assim como se relacionar e enviar mensagens a outros objetos. -> Encapsuland-o dentro de classes. Uma classe define as características e o comportamento de um conjunto de objetos. A criação de um objeto implica definir o tipo de classe em termos de seus atributos (variáveis que conterão dados) e seus métodos (funções que manipulam tais dados). Já um objeto é uma instância de uma classe.

e) (V) A palavra reservada ***this*** é utilizada para acessar atributos do objeto corrente. A palavra *this* é obrigatória apenas quando temos um argumento ou variável local de mesmo nome que um atributo. Entretanto, é recomendável sua utilização, mesmo quando não obrigatório, por uma questão de padronização.

3. (0,5) Complete com V(verdadeiro) ou F(falso). Corrija as afirmações incorretas.

Dentre as características da linguagem Java destacam-se:

a) (V) A estrutura de dados lista é utilizada para armazenar conjuntos de elementos. A vantagem em utilizar listas em lugar de vetores é o fato de as listas serem alocadas dinamicamente de forma que não precisamos prever seu tamanho máximo. Java fornece classes que implementam o conceito de lista. O Java, por padrão, possui uma série de recursos prontos (APIs) para que possamos tratar de estrutura de dados, também chamados de coleções (collections). ArrayList é uma classe genérica para coleções. Coleções podem armazenar qualquer tipo ou dado, não somente de tipos primitivos.

b) (F) Sempre que queremos criar um novo objeto de uma determinada classe utilizamos a palavra *Var* acompanhada por um **construtor**.. -> Para instanciar um objeto da classe, utiliza-se a definição de variável a partir da tipagem e cria-se um novo objeto: Classe objeto = new Classe();

c) (F) Um dos conceitos de orientação a objetos que possibilita a reutilização de código é o conceito de **Encapsulamento**. Pelo conceito de encapsulamento é possível criar uma nova classe a partir de outra classe já existente. -> Herança.

d) (V) Construtores de subclasses podem utilizar os construtores das superclasses pelo uso da palavra reservada *super*.

e) (F) Quando estudamos **encapsulamento** aprendemos que devemos preferencialmente manter os atributos com nível de acesso publico *(public)* de forma que para acessá-los outras classes precisem utilizar métodos. Mas, vimos também que há um nível de acesso protegido (*protected*) que faz com que o atributo se comporte como público para classes da mesma hierarquia ou do mesmo pacote e como privado para as demais classes. Apesar de potencialmente facilitar a implementação de métodos nas subclasses, a utilização de atributos protegidos é perigosa, pois o atributo ficará acessível a todas as classes que estejam no mesmo pacote e não somente às subclasses. -> Devemos preferencialmente manter os atributos com nível de acesso privado (private), de forma quer para acessá-los outras classes precisem utilizar métodos.

4. (0,5) Complete com V(verdadeiro) ou F(falso). Corrija as afirmações incorretas.

Dentre as características da linguagem Java destacam-se:

a) (F) A palavra polimorfismo vem do grego *poli morfos* e significa muitas formas. Na orientação a objetos, isso representa uma característica que permite que classes diferentes sejam tratadas de uma mesma forma. Podemos ver o polimorfismo como a possibilidade de um mesmo método ser executado de forma diferente de acordo com a classe do objeto que aciona o método e com os parâmetros passados para o método. O polimorfismo pode não ser obtido pela utilização dos conceitos de herança, sobrecarga de métodos e sobrescrita de método. -> Polimorfismo permite

que diferentes classes reajam de maneira diferente a estímulos semelhantes. Pode ser obtido pela utilização dos conceitos de herança, sobrecarga de métodos e sobrescrita de métodos.

b) (F) A técnica de sobrecarga ou override permite reescrever um método em uma subclasse de forma que tenha comportamento diferente do método de mesma assinatura existente na sua superclasse. -> Sobreescrita.

c) (F) Métodos de mesmo nome podem ser declarados na mesma classe, contanto que tenham diferentes conjuntos de parâmetros (determinado pelo número, tipos e ordem dos parâmetros). Isso é chamado sobreescrita de método -> Sobrecarga

d) (V) Quando comparamos dois objetos com o operador ==, na realidade estamos comparando se eles são o mesmo objeto e não se seus valores são iguais. Isso ocorre porque os objetos em Java são ponteiros para espaços de memória. Assim, dois objetos podem ter os mesmos valores em seus atributos e não serem iguais, pois podem apontar para locais diferentes. Dessa forma, para comparar os valores de dois objetos, devemos utilizar o método *equals*. Por isso que quando queremos comparar *Strings*, por exemplo, utilizamos o método *equals.*

5.(1,0) Considere o código baixo para responder as questões:

```
public class ContaEspecial extends Conta {
    private double limite;

    public double getLimite() {
        return limite;
    }

    public void setLimite(double limite) {
        this.limite = limite;
    }

}
```

a) Qual o nome da superclasse: Conta

b) Qual o nome da classe filha ou subclasse: ContaEspecial

c) Explique o conceito de generalização e especialização baseado no exemplo deste código. Generalização e Especialização dizem respeito a hierarquia das associações entre classes envolvendo herança, onde a classe mais genérica é a classe de generalização, correspondente a classe pai ou superclasse, enquanto a especialização diz respeito a classe

mais especializada, sendo as classes filhas ou subclasses. No exemplo do enunciado a classe mais genérica é a classe Conta e a classe especializada é a classe ContaEspecial.

6. (1,0) Considerando o código abaixo explique o que significa cada linha.

```
public ContaEspecial(int numero, String nome_titular, double limite) {
    super(numero, nome_titular);
    this.limite = limite;
}
```

Linha1: Define o método construtor de ContaEspecial, onde há a visibilidade do método, seu nome e os parâmetros passados para instanciar um objeto da classe.

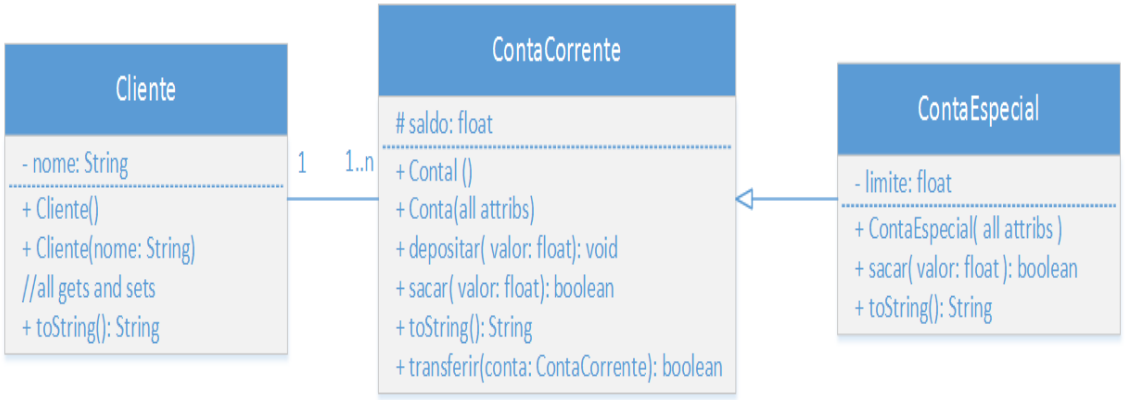
Linha2: Chama o método construtor da superclasse, passando os parâmetros relativos a ele para instanciar um objeto ContaEspecial.

Linha3: Atribui ao campo "limite" da classe ContaEspecial o valor passado por parâmetro com o nome "limite".

7. (0,5) Considere uma classe denominada Funcionario. Qual o comando para instanciar uma lista de Funcionários, usando ArrayList, denominada lista?

ArrayList<Funcionario> listaDeFuncionarios = new ArrayList<Funcionario>();

Para as próximas questões considere as classes em Java de forma a representar o diagrama UML a seguir:



- ✓ **<u>Podem substituir o tipo float por double.</u>**
- ✓ A classe ContaEspecial herda da classe ContaCorrente.
- ✓ Clientes que possuem conta especial possuem um limite de crédito. Dessa forma, podem fazer saques até esse valor limite, mesmo que não possuam saldo suficiente na conta.

8. (1,0)Complete o código da classe ContaCorrente.

```
public class ContaCorrente {

   private Cliente titular;

   protected double saldo;

   public ContaCorrente( Cliente titular, double saldo) {

      this.titular = titular;

      this.saldo = saldo;

   }

   public ContaCorrente(){}

    public Cliente getTitular() {
      return titular;
   }

   public void setTitular(Cliente titular) {
      this.titular = titular;
   }

   public double getSaldo() {
      return saldo;
   }

   public void setSaldo(double saldo) {
      this.saldo = saldo;
   }

   public boolean saca(double valor){
      if(this.saldo >=valor) {
         this.saldo - =valor;
         return true;
      }else{
         return false;
      }
   }
    public void deposita (double valor){
      this.saldo += valor;
   }

   @Override
   public String toString() {
      return ("\nTitular: "+ this.titular +"Saldo R$ "+ saldo);
   }
```

```
}
```

9. (1,0) Implemente o método transfere da classe ContaCorrente.

```
public boolean  transfere(ContaCorrente contaDestino, double valor){
        if (saca(valor)) {
                contaDestino.deposita(valor);
                return true;
        } else {
                return false;
        }
}
```

10. **(0,5)** O construtor da classe ContaEspecial deve receber como parâmetro, além dos parâmetros da superclasse, o limite que o banco disponibiliza para o cliente- utilizar super. Complete o código abaixo de forma a implementar estes requisitos.

```
public class ContaEspecial extends ContaCorrente {
 private double limite;
   public ContaEspecial(double limite, Cliente titular, double saldo) {
            super(titular, saldo) ;
            this.limite = limite;
   }
   public double getLimite() {
```

```
    return limite;
  }
  public void setLimite(double limite) {
    this.limite = limite;
  }
// restante do código
```

11.(1,0) Sobrescreva o método sacar na classe ContaEspecial, de modo que o cliente possa ficar com saldo negativo até o valor de seu limite. Note que o atributo saldo da classe ContaCorrente deve ser do tipo protected para que possa ser modificado na subclasse.

```
public boolean saca(double valor){

 if (this.saldo > (valor – this.limite)) {

        this.saldo -= valor;
        return true;
    }else{
        return false;
    }
  }
```

12. (0,5) O que acontece quando tentamos ordenar uma lista de Clientes usando o comando sort? Como resolvemos este problema?

Ao acessar o método sort pela lista de clientes é solicitado que passe como parâmetro um comparador, não executando o código. É possível corrigir implementando a interface Comparable dentro da classe Clientes e sobreescrever o método compareTo, acessando o sort() por Collections.sort(listaClientes). Outra opção é criar um comparador dentro do método sort():

```
listaClientes.sort(new Comparator<Cliente>() {
        @Override
        public int compare(Cliente c1, Cliente c2) {
          return c1.getNome().compareTo(c2.getNome());
        }
```

13. (1,0) Na classe principal Main:

a) Instancie um objeto do tipo ContaCorrente com os dados: nome do Cliente Solange Pereira, saldo R$ 4500,00

```
public static void main(String[] args) {
```

```
        Cliente cliente = new Cliente("Solange Pereira");
        ContaCorrente conta = new ContaCorrente(cliente, 4500.00);



    }
```

b) Realize um saque da conta criada no exercício anterior, informe se o saldo foi realizado com sucesso ou se o saldo era insuficiente e mostre os dados da conta com o saldo atualizado.

```
        If (conta.saca(1000)) {
            System.out.println("Saque realizado com sucesso!");
        } else {
            System.out.println("Saldo insuficiente!");
        }

        System.out.println(conta.toString());
```