

**CESAR SCHOOL**  
**CIÊNCIAS DA COMPUTAÇÃO 3P - POO 2ª PROVA**

**NOME:** \_\_\_\_\_

**INSTRUÇÕES SOBRE RESPOSTAS:**

- **QUESTÕES 1 e 2 – Usar caneta.** Indicar a linha por referência cruzada. Colocar um sequencial crescente em cada linha com erro no código e, na resposta, colocar o sequencial, em seguida o problema encontrado e finalmente a descrição da causa do problema, conforme trabalhado em sala de aula.
- **QUESTÃO 3 – Usar caneta.** Indicar o que é impresso no console EM ORDEM dos Sysouts executados.
- **QUESTÃO 4 – Usar caneta.** Novo código: escrever o código todo. Código alterado: marcar a linha a ser alterada no QUADRO com uma referência cruzada e REESCREVER TODA A LINHA alterada. Código excluído e código alterado em uma mesma classe: reescrever a classe toda.
- **QUESTÃO 5 – Usar lápis ou caneta.**

**ATIVIDADE CONTINUADA (4,0 PONTOS):** Entrega das fases 4 e 5 (opcional) até 28/11/2022.

**QUESTÃO 1 (1,5 PONTOS):** Dado o código abaixo, quais linhas possuem erros de compilação ou de execução? Descreva as causas dos erros apontados.

```
public class Alimento {}
public class Lactinio extends Alimento {}
public class Fruta extends Alimento {}
public class Leite extends Lactinio {}
public class Banana extends Fruta {}
public class Uva extends Fruta {}
public class CasosConversao {
    public void processarConversoes1() {
        Fruta m1 = new Uva();
        Lactinio c = new Leite();
        Alimento ma = new Alimento();
        c = ma; (1)
        Leite n1 = m1; (2)
        Banana mc = new Banana();
        m1 = mc;
    }
    public void processarConversoes2() {
        Alimento ma = new Banana();
        Lactinio c = new Leite();
        Fruta m1 = new Uva();
        Fruta f = (Fruta)c; (3)
        Fruta mt = (Banana)ma;
        Banana mc = (Fruta)ma; (4)
        c = (Leite)ma; (5)
    }
}
```

**QUESTÃO 2 (0,5 PONTO):** Dado o código abaixo, quais linhas possuem erros de compilação? Descreva as causas dos erros apontados.

```
public abstract class Alimento {
    public abstract void origem();
    public abstract void calorias();
}
public abstract class Lactinio extends Alimento {}
```

```

public abstract class Fruta extends Alimento {
    public void calorias() {
        System.out.println("NAO DISPONIVEL");
    }
}
public class Banana extends Fruta {
    public final void origem() {
        System.out.println("AREA EQUATORIAL");
    }
}
public final class Uva extends Fruta {
    public void origem() {
        System.out.println("AREA NAO EQUATORIAL");
    }
    public void calorias() { }
}
public class Leite extends Lactinio { (1)
    public void origem() {
        System.out.println("MAMIFEROS");
    }
}
public abstract final class Verdura extends Alimento { (2)
    public void temSemente(); (3)
}
public class BananaNanica extends Banana {
    public void origem() { (4)
        System.out.println("BRASIL");
    }
}
public class UvaPassa extends Uva { (5)

```

**QUESTÃO 3 (1,5 PONTOS):** Dado o código abaixo, indique as saídas no console quando executamos o programa em questão.

```

public class Alimento {
    public void origem() {
        System.out.println("UNIVERSO");
    }
    public void calorias() {
        System.out.println("NAO DISPONIVEL");
    }
}
public class Fruta extends Alimento {
    public void origem() {
        super.origem();
        System.out.println("TERRA");
    }
}
public class Lactinio extends Alimento {}
public class Banana extends Fruta {
    public void origem() {
        super.origem();
        System.out.println("AREA EQUATORIAL");
    }
}
public class Uva extends Fruta {}
public class Leite extends Lactinio {
    public void calorias() {
        super.calorias();
        System.out.println("200");
    }
}

```

```

public class ProgramaAlimento {
    public static void main(String[] args) {
        Alimento m1 = new Fruta();
        m1.origem();
        m1.calorias();
        Alimento m2 = new Leite();
        m2.origem();
        m2.calorias();
        Alimento m3 = new Lactinio();
        m3.origem();
        m3.calorias();
        Banana m4 = new Banana();
        m4.origem();
        m4.calorias();
        Fruta m5 = new Uva();
        m5.origem();
        m5.calorias();
    }
}

```

**QUESTÃO 4 (1,0 PONTO):** O código abaixo apresenta duas hierarquias já implementadas e um código duplicado. Usando os recursos de polimorfismo avançado, evolua a solução em questão, SEM QUEBRAR AS HIERARQUIAS atuais, de forma que o código duplicado seja único, servindo às duas classes que originalmente lá aparecem.

```

public class Ativo {
    double valor;
}
public class Moeda extends Ativo {}
public class MoedaTradicional extends Moeda {}
public class Acao extends Ativo {}
public class AcaoPreferencial extends Acao {}
public class Criptomoeda extends Moeda { (1)
    boolean semVencimento;
    double adicional;
    public boolean isSemVencimento() {
        return semVencimento;
    }
    public double retornarImposto() {
        if (!isSemVencimento()) {
            return valor * 0.05;
        } else {
            return (adicional + valor) * 0.035;
        }
    }
}
public class AcaoOrdinaria extends Acao { (2)
    double aliquotaBase;
    public double retornarImposto() {
        return (1 + aliquotaBase/100) * valor;
    }
}
public class AvaliadorExpectativa {
    public boolean excedeExpectativa(AcaoOrdinaria acao, int prazo, double taxa) {
        double imposto = acao.retornarImposto();
        if (prazo < 180) {
            return imposto < taxa;
        } else {
            return imposto < (taxa * 0.9);
        }
    }
}

```

```

    public boolean excedeExpectativa(Criptomoeda criptoMoeda, int prazo, double taxa) {
        double imposto = criptoMoeda.retoronarImposto();
        if (prazo < 180) {
            return imposto < taxa;
        } else {
            return imposto < (taxa * 0.9);
        }
    }
}

```

**QUESTÃO 5 (1,5 PONTOS):** Escreva uma classe em JAVA **CalculadoraDistanciaCoordenadas** cujo objetivo é calcular a distância entre duas coordenadas cartesianas, representados por tuplas (x, y – no plano) ou (x, y, z – no espaço). Premissas:

- A classe **CalculadoraDistanciaCoordenadas** deve ter dois métodos sobrecarregados para calcular as e retorna as distâncias entre duas coordenadas, sendo um método que calcule a distância entre duas coordenadas NO PLANO, e outro método que calcule a distância entre duas coordenadas NO ESPAÇO.
- Os parâmetros de entrada dos métodos que representam coordenadas no plano e no espaço devem ser dos tipos dados no quadro abaixo.
- Deve haver reuso de um método por outro.

```

public class CoordenadaPlano {
    private double x;
    private double y;
    public CoordenadaPlano(double x, double y) {
        this.x = x;
        this.y = y;
    }
    public double getX() {
        return x;
    }
    public double getY() {
        return y;
    }
}

public class CoordenadaEspaco extends CoordenadaPlano {
    private double z;
    public CoordenadaEspaco(double x, double y, double z) {
        super(x, y);
        this.z = z;
    }
    public double getZ() {
        return z;
    }
}

```

No PLANO – Dadas duas coordenadas A(x<sub>A</sub>, y<sub>A</sub>) e B(x<sub>B</sub>, y<sub>B</sub>), a distância entre elas é:

$$d_{AB} = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$$

No ESPAÇO – Dadas duas coordenadas A(x<sub>A</sub>, y<sub>A</sub>, z<sub>A</sub>) e B(x<sub>B</sub>, y<sub>B</sub>, z<sub>B</sub>), a distância entre elas é:

$$d_{AB} = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2 + (z_B - z_A)^2}$$

## QUESTÃO 1 (1,5 PONTOS)

### REFERÊNCIA NO MATERIAL

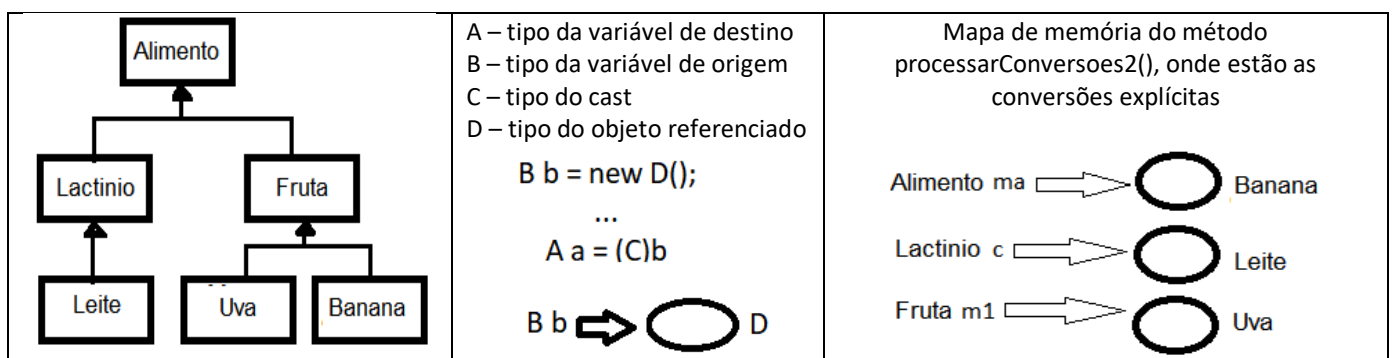
POO – Heranca.pdf; POO – Polimorfismo\_PrincipioSubstituicao.pdf.

### EXPLICAÇÃO SOBRE A SOLUÇÃO

É imprescindível visualizar a hierarquia em questão (dada abaixo) e desenhar o mapa de memória para conversões explícitas, pois é necessário avaliar os tipos dos objetos referenciados. A solução passa por aplicar, a cada linha de código que contém conversão implícita e explícita de tipos não primitivos as regras aplicáveis de compilação e de execução, a saber:

- Em uma conversão implícita, o código compila ou não compila. Compila se o tipo da expressão (variável) de origem for igual ou subtipo do tipo da variável de destino.
- Em uma conversão explícita (CAST), o código pode compilar ou não, e se compilar, pode rodar ou não. O código compila se: o tipo do CAST e o tipo da expressão (variável) de origem estiverem na mesma hierarquia

E se o tipo do CAST for igual ou subtipo do tipo da variável de origem. O código roda se o tipo do objeto referenciado pela expressão (variável) de origem for igual ou subtipo do tipo do CAST.



### RESPOSTA (linhas com erro indicadas no código)

(1) – Não compila porque o tipo da variável de origem **ma (Alimento)** não é igual ou subtipo do tipo da variável de destino **c (Lactinio)**.

(2) – Não compila porque o tipo da variável de origem **m1 (Fruta)** não é igual ou subtipo do tipo da variável de destino **n1 (Leite)**.

(3) – Não compila porque o tipo do CAST (**Fruta**) e o tipo da variável de origem **c (Lactinio)** não estão na mesma hierarquia.

(4) – Não compila porque o tipo do CAST (**Fruta**) não é igual ou subtipo do tipo da variável de destino **mc (Banana)**.

(5) – Compila mas não roda porque o tipo do objeto referenciado pela variável de origem **ma (Banana)** não é igual ou subtipo do tipo do CAST (**Leite**).

### CRITÉRIO DE PONTUAÇÃO

Na questão há 05 linhas com erros de compilação. Cada erro identificado e corretamente justificado vale 0,3. Há desconto de 0,3 por linha incorreta apontada. Se não fosse assim, poderia se apontar todas as linhas como incorretas e sempre pontuar potencialmente pelo máximo possível.

## QUESTÃO 2 (0,5 PONTO)

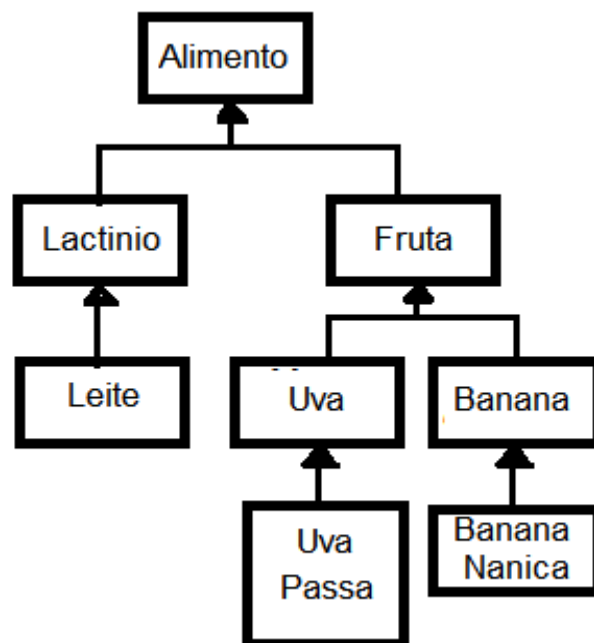
### REFERÊNCIA NO MATERIAL

POO – Heranca.pdf; POO – Polimorfismo\_ClasesAbstratas.pdf.

### EXPLICAÇÃO SOBRE A SOLUÇÃO

É imprescindível visualizar a hierarquia em questão (dada abaixo). O código deve ser analisado sob a luz das amarrações de compilação envolvendo os conceitos de classes abstratas e do modificador de acesso final. O código não compila se...

- Uma classe abstrata for instanciada (new...);
- Um método abstrato definido em uma superclasse abstrata não for sobrescrito em uma subclasse concreta (diretamente ou por herança da hierarquia superior);
- Houver chamada, em uma subclasse, via super, de um método definido como abstrato em uma superclasse, mas que não tenha implementação encontrada na super hierarquia;
- Um método abstrato for definido em uma classe concreta;
- Um método abstrato tiver corpo;
- Um método não abstrato não tiver corpo;
- Um método abstrato for final;
- Uma classe abstrata for final;
- Uma classe tentar herdar de outra classe final;
- Uma classe tentar sobrescrever um método final de uma de suas superclasses.



### RESPOSTA (linhas com erro indicadas no código)

- (1) – Não compila porque a classe **Leite** tem que implementar o método **calorias()**.
- (2) – Não compila porque a classe **Verdura** não pode ser abstrata e final.
- (3) – Não compila porque o método **temSemente()** é concreto (ou não abstrato) e não tem implementação (corpo).
- (4) – Não compila porque o método **origem()** da classe **BananaNanica** não pode sobrescrever o seu homônimo final da superclasse **Banana**.
- (5) – Não compila porque a classe **UvaPassa** não pode herdar de uma classe final (**Uva**).

### CRITÉRIO DE PONTUAÇÃO

Na questão há 05 linhas com erros de compilação. Cada erro identificado e corretamente justificado vale 0,1. Há desconto de 0,1 por linha incorreta apontada. Se não fosse assim, poderia se apontar todas as linhas como incorretas e sempre pontuar potencialmente pelo máximo possível.

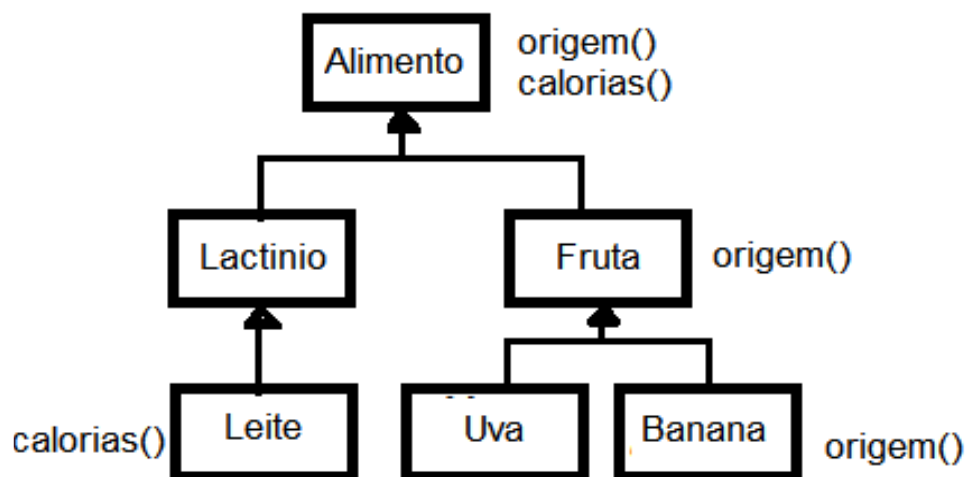
### QUESTÃO 3 (1,5 PONTOS)

#### REFERÊNCIA NO MATERIAL

POO – Heranca.pdf; POO – Polimorfismo\_Sobrescrita.pdf.

#### EXPLICAÇÃO SOBRE A SOLUÇÃO

É importante montar a hierarquia em questão e identificar, em cada classe, que métodos estão implementados e/ou reimplementados por sobrescrita. A regra de sobrescrita diz que um método definido e implementado em uma superclasse pode ser definido (com a mesma assinatura) na subclasse e nela reimplementado. Pelo princípio da substituição, é possível referenciar objetos de subtipos com variáveis de supertipos. Através da variável de um supertipo, é possível “enxergar” o método, pois ele foi definido na superclasse. Mas, quando o método é chamado, a “versão” executada é a primeira encontrada na hierarquia, quando esta é percorrida a partir do tipo do objeto referenciado para cima. Um método sobrescrito pode chamar seu homônimo na hierarquia superior através da palavra reservada **super**, seguida de ponto e do nome do método em questão. O fluxo de execução segue a sequência da implementação do método na hierarquia superior, e depois volta para o método sobrescrito. A solução consiste em aplicar às chamadas dos métodos estas premissas.



#### RESPOSTA

UNIVERSO

TERRA

NAO\_DISPONIVEL

UNIVERSO

NAO\_DISPONIVEL

200

UNIVERSO

NAO\_DISPONIVEL

UNIVERSO

TERRA

AREA\_EQUATORIAL

NAO\_DISPONIVEL

UNIVERSO

TERRA

NAO\_DISPONIVEL

#### CRITÉRIO DE PONTUAÇÃO

No código apresentado existem 15 Sysouts. Cada Sysout escrito correto, casado com, e NA ORDEM CORRESPONDENTE à ORDEM em que aparecem os Sysouts quando executamos o código, vale 0,1. Se mais de 15 saídas forem indicadas, as saídas de ordem superior a 15 serão ignoradas. Se menos de 15 saídas forem indicadas, serão avaliadas até a quantidade apresentada, com pontuação eventualmente proporcional.

## QUESTÃO 4 (1,0 PONTO)

### REFERÊNCIA NO MATERIAL

Assunto da 1ª unidade; POO – Heranca.pdf; POO; POO – Polimorfismo\_interface.pdf.

### EXPLICAÇÃO SOBRE A SOLUÇÃO

Interfaces servem para “emular”, em JAVA, herança múltipla, atribuindo a classes que já estão encaixadas em hierarquias pré-definidas outras características comuns inerentes a outros conceitos, que não necessariamente estão associados às superclasses das hierarquias originais. No caso, há duas hierarquias pré-definidas, e lá constam duas subclasses, de hierarquias diferentes, com características comuns, que são usadas por uma classe funcional através de uma clara replicação de código. A ideia é usar interface para eliminar tal duplicação.

A solução passa por definir uma interface que espelhe o comportamento comum das duas subclasses das hierarquias já definidas que possuem o mesmo método utilizado pela classe funcional. O passo a passo de alteração do código é:

- Criar a dita interface com o método comum observado nas duas subclasses das hierarquias pré-definidas.
- Fazer as duas subclasses das hierarquias pré-definidas implementarem a interface.
- Eliminar um dos métodos com lógica duplicada na classe funcional.
- Alterar o método remanescente da classe funcional para receber como parâmetro o tipo da interface criada, em vez do subtipo específico original.

Como a interface possui o mesmo método utilizado pelo método da classe funcional, não há necessidade de alterar a lógica do código deste, apenas, por uma questão estética, se altera o nome do parâmetro.

### RESPOSTA (as linhas alteradas estão marcadas no código)

```
public interface Imposto {
    double retornarImposto();
}
(1) - public class Criptomoeda extends Moeda implements Imposto {
(2) - public class AcaoOrdinaria extends Acao implements Imposto {
public class AvaliadorExpectativa {
    public boolean excedeExpectativa(Imposto imp, int prazo, double taxa) {
        double imposto = imp.retornarImposto();
        if (prazo < 180) {
            return imposto < taxa;
        } else {
            return imposto < (taxa * 0.9);
        }
    }
public boolean excedeExpectativa(Criptomoeda criptoMoeda, int prazo, double taxa) {
double imposto = criptoMoeda.retornarImposto();
if (prazo < 180) {
return imposto < taxa;
} else {
return imposto < (taxa * 0.9);
}
}
}
```

### CRITÉRIO DE PONTUAÇÃO

Até 0.3 para definição da interface, até 0.4 para eliminação do código duplicado (um dos métodos) e 0.3 para implementação da interface pelas duas subclasses.



## QUESTÃO 5 (1,5 PONTOS)

### REFERÊNCIA NO MATERIAL

Assunto da 1ª unidade; POO – Heranca.pdf; POO; POO – Polimorfismo\_sobrecarga.pdf.

### EXPLICAÇÃO SOBRE A SOLUÇÃO

A questão pede a implementação de uma classe com dois métodos sobrescritos que, por definição, possuem mesmo nome, mesmo tipo de retorno (na grande maioria dos casos, e este é um deles) e listas de parâmetros diferentes, sejam pela quantidade de parâmetros, sejam pela sequência dos tipos especificados. O reuso entre métodos sobrescritos é frequente, já que eles tratam de situações ou de regras de negócio muito similares, normalmente envolvendo o mesmo conceito. Em muitas situações, e esta é um caso, a lógica de implementação de um método é um “caso especial” de uma outra versão sobrecarregada, e a simples chamada de um pelo outro passando convenientemente uma combinação de parâmetros pode promover reuso de uma forma bastante eficiente. Na questão, deve-se observar que o cálculo da distância entre dois pontos/coordenadas no plano é um caso especial do cálculo da distância entre dois pontos/coordenadas no espaço. É só olhar para as fórmulas com atenção e perceber que a fórmula da distância no plano é equivalente à fórmula da distância no espaço se as duas coordenadas Z dos dois pontos forem iguais, ou forem iguais a zero, critério este que será adotado na solução apresentada.

“Mindset” para resolução da questão	
Criar classe pedida	<code>public class CalculadoraDistanciaPontos { ... }</code>
Na classe pedida, criar dois métodos sobrecarregados	<code>public double calcularDistancia(lista1 parâmetros)</code> <code>public double calcularDistancia(lista2 parâmetros)</code>
Usar os tipos dados como parâmetros	<code>public double calcularDistancia(CoordenadaPlano p1, CoordenadaPlano p2)</code> <code>public double calcularDistancia(CoordenadaEspaco p1, CoordenadaEspaco p2)</code>
Promover reuso entre os métodos	Aproveitar o fato de que a distância no plano é igual à distância no espaço com as coordenadas Z dos dois pontos/coordenada iguais a zero. Ou seja, o método <code>calcularDistancia</code> que recebe os <code>CoordenadaPlano</code> vai instanciar dois <code>CoordenadaEspaco</code> com coordenadas X e Y correspondentes às coordenadas X e Y dos <code>CoordenadaPlano</code> recebidos, e <u>coordenadas Z zero</u> . Depois, vai retornar a chamada do outro método <code>calcularDistancia</code> que recebe os <code>CoordenadaEspaco</code> , passando para este os <code>CoordenadaEspaco</code> instanciados.

### RESPOSTA

```
public class CalculadoraDistanciaPonto {
    public double calcularDistancia(CoordenadaPlano c1, CoordenadaPlano c2) {
        CoordenadaEspaco ce1 = new CoordenadaEspaco(c1.getX(), c1.getY(), 0.0);
        CoordenadaEspaco ce2 = new CoordenadaEspaco(c2.getX(), c2.getY(), 0.0);
        return calcularDistancia(ce1, ce2);
    }
    public double calcularDistancia(CoordenadaEspaco c1, CoordenadaEspaco c2) {
        double dx = c2.getX() - c1.getX();
        double dy = c2.getY() - c1.getY();
        double dz = c2.getZ() - c1.getZ();
        return Math.sqrt(dx*dx + dy*dy + dz*dz); // quem usou Math.pow no lugar dos produtos OK!!!
    }
}
```

### CRITÉRIO DE PONTUAÇÃO

A implementação dos dois métodos “repetindo” a fórmula vale ATÉ 0,9. O reuso vale ATÉ 0,6.