UNIP UNIVERSIDADE PAULISTA Instituto de Ciências Exatas *Campus Brasília*	Curso: Ciência da Computação / Sistema de Informação Disciplina: Linguagem de Programação Orientação a Objetos Prof(a): Msc Wanderley Gonçalves Freitas Lista de Exercícios 05.1 – herança, polimorfismo, classe abstrata e interface

1) A respeito dos conceitos fundamentais de programação orientada a objeto, julgue os itens seguintes:

I. Uma é classe concreta quanto tem todos os seus métodos implementados. Por outro lado, uma classe é abstrata obrigatoriamente quando tem declara um método abstrato na classe.
II. Uma interface declara métodos que deverão ser implementados pelas classes concretas.
III. Uma classe pode implementar diversas interfaces

Assinale a opção correta.

(a) Apenas a afirmação I é verdadeira.
(b) As afirmações I e III são verdadeiras.
(c) As afirmações I e II são verdadeiras.
(d) As afirmações II e III são verdadeiras.
(e) As afirmações I, II e III são verdadeiras.

2) A respeito dos conceitos fundamentais de programação orientada a objeto, julgue os itens seguintes:

I. Uma classe que tem uma ou mais métodos abstratos precisa ser declarada como abstrata
II. Uma classe concreta que herda de uma classe abstrata precisa implementar todos os métodos abstratos
III. É necessário aplicar o modificador *abstract* junto à declaração da *classe* para defini-la como *classe concreta.*

Assinale a opção correta.

(a) Apenas a afirmação I é verdadeira.
(b) As afirmações I e III são verdadeiras.
(c) As afirmações I e II são verdadeiras.
(d) As afirmações II e III são verdadeiras.
(e) As afirmações I, II e III são verdadeiras.

3) A respeito dos conceitos fundamentais de programação orientada a objeto, julgue os itens seguintes:

I. Se uma classe tem um método que é abstrato, ela deve ser declarada como abstrata.

II. Uma classe pode implementar somente uma única interface.

III. No contexto da orientação a objetos, uma classe que NÃO possui objetos instanciados a partir dela é conhecida como classe abstrata.

IV. No desenvolvimento de um novo sistema, a empresa utilizou o seguinte princípio da orientação a objetos: em uma mesma classe pode adicionar vários métodos com comportamento diferentes com o mesmo nome, porém com assinaturas diferentes. Nesse contexto, o conceito mencionado da orientação objeto no texto é sobrecarga.

Assinale a opção correta?

(a) As afirmações I e II são verdadeiras.

(b) As afirmações I e III são verdadeiras.

(c) As afirmações II e IV são verdadeiras.

(d) As afirmações I, III e IV são verdadeiras.

(e) As afirmações II, III e IV são verdadeiras.

4) A respeito dos conceitos fundamentais de programação orientada a objeto, julgue os itens seguintes:

I. Classe abstrata é uma classe que define o comportamento e atributos para uma subclasse concreta.

II. Uma classe abstrata pode ser instanciada.

III. Uma classe abstrata pode possuir métodos concretos e métodos abstratos..

IV. Se uma classe possuir pelo menos um método abstrato, ela deve ser uma classe concreta.

Assinale a opção correta?

(a) As afirmações I e II são verdadeiras.

(b) As afirmações I e III são verdadeiras.

(c) As afirmações II e IV são verdadeiras.

(d) As afirmações I, III e IV são verdadeiras.

(e) As afirmações II, III e IV são verdadeiras.

5) A respeito dos conceitos fundamentais de programação orientada a objeto, julgue os itens seguintes:

I. No desenvolvimento do novo sistema, a empresa aproveitou partes do sistema antigo e estendeu os componentes de maneira a usar código já validado, acrescentando as novas funções solicitadas. Nesse contexto, o conceito mencionado da orientação objeto é herança.

II. Métodos abstratos são métodos que não possuem implementação, ou seja, sem corpo.

III. Em java, poder ocorrer o mecanismo de herança entre duas classes explicitamente

IV. Quando duas ou mais classes derivadas de uma mesma superclasse podem invocar métodos que têm a mesma assinatura, mas com comportamentos distintos. Nesse contexto, o conceito mencionado da orientação objeto é sobrecarga.

Estão certos apenas os itens?

(a) I, II, III e IV.
(b) I, II e III, apenas.
(c) II e III, apenas.
(d) II, III e IV, apenas.
(e) I e II, apenas

6) A respeito dos conceitos fundamentais de programação orientada a objeto, julgue os itens seguintes designando (C) para os certos e (E) para os errados:

(a) (C) (E) - A polimorfismos é um mecanismo para o compartilhamento de métodos e atributos entre classes e subclasses, permitindo a criação de novas classes através da programação das diferenças entre a nova classe e a classe-pai.

(b) (C) (E) Os métodos protected e os atributos protected são visíveis apenas para a classe a que eles pertencem.

(c) (C) (E) Quando uma subclasse é criada, essa herda todas as características da superclasse, podendo possuir propriedades e métodos próprios.

(d) (C) (E) Uma classe concreta é uma classe que pode ser instanciada.

(e) (C) (E) Uma interface pode possuir métodos concretos e métodos abstratos.

(f) (C) (E) É necessário aplicar o modificador abstract junto à declaração da classe para defini-la como abstrata.

(g) (C) (E) Métodos abstratos em classes abstratas são aqueles que possuem o modificador abstract e possuem corpo.

(h) (C) (E) Não é possível instanciar objetos de classes abstratas, ou seja, classes abstratas não podem ser instanciadas.

(i) (C) (E) A subclasse herda as propriedades comuns da superclasse e pode ainda adicionar novos métodos ou reescrever métodos herdados.

7) A programação orientada a objeto não é apenas uma forma de programar, é também um jeito de pensar em um problema utilizando conceitos do mundo real e, não somente conceitos computacionais. Assim, O paradigma de programação orientado a objetos tem sido largamente utilizado no desenvolvimento de sistemas. Considerando o conceito de orientação programação orientado objeto, julgue os seguintes itens.

I. Uma classe abstrata pode possuir métodos concretos e métodos abstratos.

II. *Herança* é uma propriedade que dificuldade a implementação de reuso. Utiliza-se a extensão(extends) é uma das formas de se implementar *herança*.

III. Polimorfismo sobrescrito é a característica de se ter dois métodos com o mesmo nome e comportamento distintos utilizando herança.

Assinale a opção correta.

(a) Apenas a afirmação I é verdadeira.

(b) As afirmações I e III são verdadeiras.

(c) As afirmações I e II são verdadeiras.

(d) As afirmações II e III são verdadeiras.

(e) As afirmações I, II e III são verdadeiras.

8) A respeito dos conceitos acerca dos construtores em java, julgue os itens seguintes designando (C) para os certos e (E) para os errados:

(a) (C) (E) Uma classe pode implementar somente uma única interface.

(b) (C) (E) Uma interface só pode ter assinatura dos métodos e os atributos são todos constantes.

(c) (C) (E) Uma instância de uma classe abstrata herda atributos e métodos de sua superclasse direta.

(d) (C) (E) Métodos abstratos em classes abstratas são aqueles que possuem o modificador abstract e não possuem corpo.

(e) (C) (E) O modificador protected, quando aplicado a um atributo de classe, não permite que classes filhas desta classe, tenham acesso a este atributo.

(f) (C) (E) Não é possível instanciar objetos de classes abstratas, ou seja, classes abstratas não podem ser instanciadas.

(g) (C)(E) Métodos abstratos em classes abstratas são aqueles que possuem o modificador abstract e não possuem corpo.

9) A figura 1 mostra a hierarquia de classe de exceção. As exceções são eventos inesperados que ocorrem durante a execução de um programa. Uma exceção pode ser o resultado de uma condição de erro ou simplesmente uma entrada inesperada. De qualquer forma, em linguagens orientadas a objetos como Java, as exceções são vistas como objetos.

Considerando os conceitos relacionados acerca de tratamento de exceção, julgue os seguintes itens.

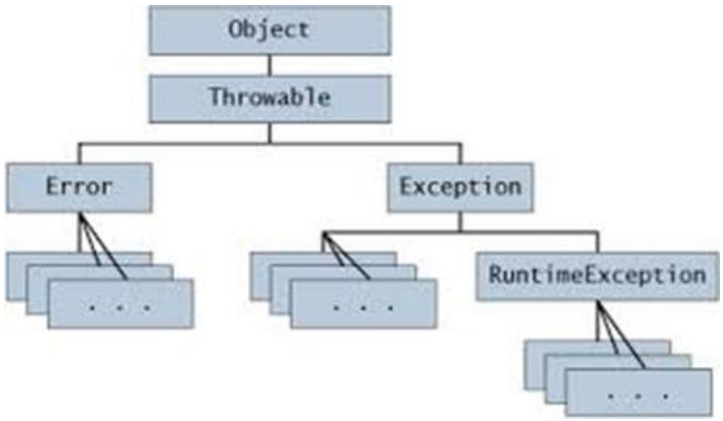

Figura 1 : hierarquia de classe de exceção

I. No tratamento de exceção, não é possível utilizar vários blocos *catch* para capturar exceções vindas de um único bloco *try*.

II. O código que pode gerar uma exceção deve ser incluído em um bloco *try* em uma instrução *try – catch – finally*..

III. Todos os objetos que podemos dar *throw* e *catch* são de classes derivadas direta ou indiretamente de *Throwable*.

IV. O bloco *finally* em uma instrução *try – catch – finally* não será executado quando uma exceção ocorrer dentro do bloco *try*.

Assinale a opção corretas

(A) Apenas um item está certo

(B) Apenas os itens I e II estão certos

(C) Apenas os itens II e III estão certos

(D) Apenas os itens III e IV estão certos

(E) Apenas os itens II, III e IV estão certos

10) A figura 2 mostra a hierarquia de classe de exceção. As exceções são eventos inesperados que ocorrem durante a execução de um programa. Uma exceção pode ser o resultado de uma condição de erro ou simplesmente uma entrada inesperada. De qualquer forma, em linguagens orientadas a objetos como Java, as exceções são vistas como objetos.

Considerando os conceitos relacionados acerca de tratamento de exceção, julgue os seguintes itens.

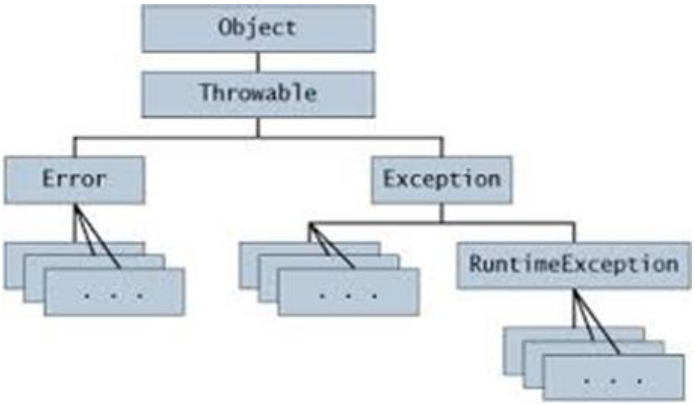

Figura 2 : hierarquia de classe de exceção

I. O código que pode gerar uma exceção deve ser incluído em um bloco *catch* em uma instrução try – catch – finally..

II. As exceções que são *subclasse* de *RuntimeException* não precisam ser tratadas..

III. Não é natural ocorrerem exceções do tipo *Error* em *Java*. Exceções deste tipo representam erros na *JVM* e não devem ser capturadas.

IV. O bloco *finally* em uma instrução *try – catch – finally* sempre será executado quer ocorra ou não uma exceção no bloco *try* .

Assinale a opção corretas

(a) Apenas um item está certo
(b) Apenas os itens I e II estão certos
(c) Apenas os itens II e III estão certos
(d) Apenas os itens III e IV estão certos
(e) Apenas os itens II, III e IV estão certos

11) A figura 2 mostra a hierarquia de classe de exceção. As exceções são eventos inesperados que ocorrem durante a execução de um programa. Uma exceção pode ser o resultado de uma condição de erro ou simplesmente uma entrada inesperada. De qualquer forma, em linguagens orientadas a objetos como Java, as exceções são vistas como objetos.

Considerando os conceitos relacionados acerca de tratamento de exceção, julgue os seguintes itens.

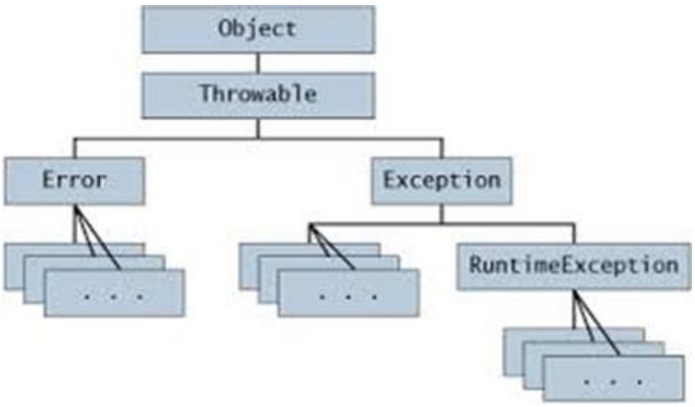

Figura 2 : hierarquia de classe de exceção

I. O código que pode gerar uma exceção deve ser incluído em um bloco *try* em uma instrução try – catch – finally..

II. No tratamento de exceção, é possível utilizar vários blocos *catch* para capturar exceções vindas de um único bloco *try*.

III. Não é natural ocorrerem exceções do tipo *Error* em *Java*. Exceções deste tipo representam erros na *JVM* e devem ser capturadas.

IV. O bloco *finally* em uma instrução *try – catch – finally* não será executado quando uma exceção ocorrer dentro do bloco *try*.

Assinale a opção corretas

(a) Apenas um item está certo

(b) Apenas os itens I e II estão certos

(c) Apenas os itens II e III estão certos

(d) Apenas os itens III e IV estão certos

(e) Apenas os itens II, III e IV estão certos

12) O código a seguir mostra um programa escrito na linguagem de programação java e logo abaixo a estrutura do relatorio técnico com as perguntas elaboradas para dar apoio ao diagnostico do software.

Para esta questão, considere os seguintes conceitos da orientação a objetos:

- *Superclasse* é uma classe que é um supertipo(classe base) de uma ou mais classes. Assim, é a classe base que é estendida por outra com objetivo de reutilizar seus membros(atributos e métodos).
- *Subclasse* é uma classe que é um subtipo de classe. A classe herda todas as características da superclasse.
- *Assinatura de um método* é uma combinação do nome do método, tipo; ordem e número de seus parâmetros.
- *Sobrescrita* são métodos de mesma assinatura existindo em classes relacionadas por herança.
- *Sobrecarga* são métodos na mesma classe, com o mesmo nome, mas com assinaturas diferentes.

```
public class Funcionario {

    protected  String nome;
    protected double salario;

    public void imprimir() {
        System.out.println("nome:   " + nome );
    }

    public double calcularBonificacao(){
        double calculo = salario * 0.10;
        return calculo;
    }
```

```
public class Coordenador extends Funcionario{
    protected int nivel;

    public void verificarDesempenho(){
        double resultado = nivel + 2;
        System.out.println("nivel:   " + resultado);
    }
    public double calcular(double valor1, int valor2){
        double calculo = valor1 * valor2;
        return calculo;
    }
    public double calcular(int valor1, double valor2){
        double calculo = valor1 / valor2;
        return calculo;
```

```
public abstract class  Gerente extends Funcionario {
    protected String senha;

    public double calcularBonificacao() {
        double calculo = salario * 0.30;
        return calculo;
    }
    public void imprimir() {
        System.out.println("Categoria Especial");
        System.out.println("senha: " + senha);
    }
    public abstract double obterSalario();
```

```
public class SubGerente extends Gerente {

    public double obterSalario() {
        double vResultado = salario + 100;
        return vResultado;
    }
}
```

A) Escreva as assinaturas de todos os métodos sobrecargas?

B) Escreva as assinaturas de todos os métodos sobrescritos?

C) Escreva nome superclasse com seus atributos e nome da subclasse com seus atributos?

D) Escreva nome classe abstrata e assinatura do método abstrato?

E) O que será impresso na execução do programa Teste

```
public class Teste {

    public static void main(String[] args) {
        Funcionario marcos = new Funcionario();
        marcos.setSalario(1000);
        System.out.println(marcos.calcularBonificacao());
    }

}
```

?

F) O que será impresso na execução do programa Teste

```
public class Teste {

    public static void main(String[] args) {
        Coordenador marcos = new Coordenador();
        marcos.setSalario(2000);
        System.out.println(marcos.calcularBonificacao());
    }

}
```

?

G) O que será impresso na execução do programa Teste

```
public class Teste {

    public static void main(String[] args) {
        Coordenador marcos = new Coordenador();
        System.out.println(marcos.calcular(10.0, 2));
    }

}
```

H) O que será impresso na execução do programa Teste

```
public class Teste {

    public static void main(String[] args) {
        Coordenador marcos = new Coordenador();
        System.out.println(marcos.calcular(50,2.0));
    }

}
```

13) A empresa de informatica WG contratou uma analista de sistema com perfil de desenvolvedor de aplicação com conhecimento em várias linguagens de programação para realiza-se um trabalho de analise de código para entendimento do software que encontra-se em produção. A Análise do programa deverá utiliza-se da seguinte metodologia:

I. Compilação bem sucedida e justifique sua resposta.

- ❖ Exemplo : compilação bem sucedida porque todos os métodos estão implementados

II. Erro de compilação – informar a linha e explicar o erro

- ❖ Exemplo : Erro de compilação na linha 6 porque falta ponto e virgula

(A) O código a seguir mostra uma programa escrito na linguagem de programação java

```
1  public class  Empregado {
2      protected double salario;
3
4      public double calcularBonificacao() {
5          double calculo = salario * 0.30;
6          return calculo;
7      }
8      public void imprimir() {
9          System.out.println("salario: " + salario);
10     }
11     public abstract double obterSalario();
12 }
```

.

(B) O código a seguir mostra uma programa escrito na linguagem de programação java

```
1  public abstract class  Empregado {
2      protected double salario;
3
4      public double calcularBonificacao() {
5          double calculo = salario * 0.30;
6          return calculo;
7      }
8      public void imprimir() {
9          System.out.println("salario: " + salario);
10     }
11     public abstract double obterSalario();
12 }
13
```

(C) O código a seguir mostra uma programa escrito na linguagem de programação java

```
1  public interface Operador {
2
3      void calcular();
4      double calcularImposto(double pValor);
5  }
6
7  public class Pedreiro implements Operador {
8      double salario;
9      public void calcular() {
10         salario = salario + 1500;
11
12     }
13     public double calcularImposto(double pValor) {
14         double vResultado = salario * pValor;
15         return vResultado;
16     }
17 }
```

(D) O código a seguir mostra uma programa escrito na linguagem de programação java

```
1  public interface Operador {
2
3      void calcular();
4      double calcularImposto(double pValor);
5  }
6
7  public class Pedreiro extends Operador {
8      double salario;
9      public void calcular() {
10         salario = salario + 1500;
11
12     }
13     public double calcularImposto(double pValor) {
14         double vResultado = salario * pValor;
15         return vResultado;
16     }
17 }
```

(E) O código a seguir mostra uma programa escrito na linguagem de programação java

```
1  public abstract class AlunoMestrado {
2      protected double nota;
3      protected double salario;
4
5      public void calcularNota() {
6          nota = nota + 3;
7      }
8
9      public abstract double obterSalario(){
10         return (salario *2)
11     }
12
13     public void imprimir() {
14         System.out.println("nota: " + nota);
15     }
16 }
```

(F) O código a seguir mostra uma programa escrito na linguagem de programação java

```
1  public interface Operador {
2
3      void calcular();
4      double calcularImposto(double pValor){
5          double vCalculo = pValor * 2;
6      }
7  }
```

(G) O código a seguir mostra uma programa escrito na linguagem de programação java

```
1  public abstract class AlunoMestrado {
2      protected double nota;
3      protected double salario;
4
5      public void calcularNota() {
6          nota = nota + 3;
7      }
8
9      public abstract double obterSalario();
10
11
12     public void imprimir() {
13         System.out.println("nota: " + nota);
14     }
15 }
16
17 public class Mestre extends AlunoMestrado {
18
19     public void calcularNota() {
20         nota = nota + 4.5;
21     }
22
23     public void imprimir() {
24         System.out.println("nota: " + nota);
25     }
26 }
```

(H) O código a seguir mostra uma programa escrito na linguagem de programação java

```
1  public class Aluno {
2      protected double nota;
3
4      public void calcularNota() {
5          nota = nota + 3;
6      }
7  }
```

(I) O código a seguir mostra uma programa escrito na linguagem de programação java

```
1  public abstract class AlunoMestrado {
2      protected double nota;
3      protected double salario;
4
5      public void calcularNota() {
6          nota = nota + 3;
7      }
8
9      public abstract double obterSalario();
10
11     public void imprimir() {
12         System.out.println("nota: " + nota);
13     }
14 }
15
16 public class Mestre extends AlunoMestrado {
17
18     public void calcularNota() {
19         nota = nota + 4.5;
20     }
21
22     public void imprimir() {
23         System.out.println("nota: " + nota);
24     }
25
26     public double obterSalario() {
27         double vCalculo = salario * 2.5;
28         return vCalculo;
29     }
30 }
```

(J) O código a seguir mostra uma programa escrito na linguagem de programação java

```
1  public interface Operador {
2
3      void calcular();
4      double calcularImposto(double pValor);
5  }
```

14) O Sindicato dos professores do Estado do Nordeste deseja o desenvolvido um sistema on-line de cadastramento de associados. O sistema on-line de cadastramento deverá descrever as caraterísticas essências para todos os tipos de associados. Na primeira etapa de desenvolvimento do sistema o arquiteto de software projetou uma hierarquia de classes no diagrama de classe da aplicação e solicitou que fossem feitas as seguintes tarefas:

(a) Crie a pacote : . Pacote : br.unip.lpoo.np2.<turma aluno>.

(b) Crie a interface Sindicato.

- Cria os seguintes métodos conforme a descrição abaixo:

Tipo Retorno	Nome método	Parâmetro	Objetivo do método - corpo
double	calcularContribuicao	double pValor	Métodos é abstrato.

(c) Crie a classe abstrata Professor que implementa a interface sindicato.

- Atributos (protected) : nome(String), salario (double).
- Cria os seguintes métodos concretos conforme a descrição abaixo:

Tipo Retorno	Nome método	Parâmetro	Objetivo do método - corpo
double	obterPrimeiraParcela	double pTaxa	Dever retorna o resultado da formula: ❖ Regra de Negócio – Todo professor recebe seu salário em duas parcelas, sendo alguns professores tem taxa diferencial definido pelo justiça. formula : vResultado = (salario * pTaxa) + salario
double	obterPrimeiraParcela	ausente	Dever retorna o resultado da formula: ❖ Regra de Negócio – Todo professor recebe seu salário em duas parcelas, sendo 60% na primeira parcela. formula : vResultado = (salario * 0.60)
double	obtersegundaParcela	ausente	Dever retorna o resultado da formula: Regra de Negócio – Todo professor recebe seu salário em duas parcelas, sendo 40% na segunda parcela. ❖ formula : vResultado = (salario * 0.40)

- Cria o método abstrato, conforme a descrição abaixo:

Tipo Retorno	Nome método	Parâmetro	Objetivo do método - corpo
void	Imprimir	Ausente	

(d) Crie classe ProfessorGraduacao subclasse da classe professor:

- Atributos (protected) : gratificacao (double).
- O métodos getPrimeiraParcela e getSegundaParcela serão sobrescritos (sobrescritos, redefine, override), conforme tabela

Nome método	Objetivo do método - corpo
obterPrimeiraParcela	Dever retorna o resultado da formula: ❖ Regra de Negócio – Todo professor graduação recebe seu salário integral em única parcela. formula : retorna o atributo salario
obtersegundaParcela	Dever retorna o resultado da formula: ❖ Regra de Negócio – Todo professor graduação não recebe a segunda parcela . formula : retorna zero

- O métodos imprimir e calcularContribuição serão implementados na classe concreta. Essa implementação é sobrescrever o método, conforme tabela

Nome método	Objetivo do método - corpo
Imprimir	Exibe no console todos os atributos da classe
calcularContribuicao	Regra de Negócio – Todo professor graduação dever contribuir para o sindicado com 2.5% sobre salario . ❖ formula : vResultado = (salario * 0.025) + pValor

(e) Crie classe ProfessorHorista subclasse da classe professor:

- Atributos (protected) : salarioHora (double), totalHoras (int).
- O métodos getPrimeiraParcela e getSegundaParcela serão sobrescritos

Nome método	Objetivo do método - corpo
obterPrimeiraParcela	Dever retorna o resultado da formula: ❖ Regra de Negócio – Todo professor horista recebe seu salário em duas parcelas, sendo 30% na primeira parcela sobre o calculo do salario final [(salario por hora * total de horas) + salario] .formula : vSalarioBase=salarioHora * totalHora vResultado= (salario + vSalarioBase) * 0.30
obtersegundaParcela	Dever retorna o resultado da formula: ❖ Regra de Negócio – Todo professor horista recebe seu salário em duas parcelas, sendo 70% na primeira parcela sobre o calculo do salario final [(salario por hora * total de horas) + salario] .formula : vSalarioBase=salarioHora * totalHora vResultado = (salario + vSalarioBase) * 0.70

- O métodos imprimir e calcularContribuição serão implementados na classe concreta. Essa implementação é sobrescrever o método, conforme tabela

Nome método	Objetivo do método - corpo
Imprimir	Exibe no console todos os atributos da classe
calcularContribuicao	❖ Regra de Negócio – Todo professor horista dever contribuir para o sindicado com 3% sobre salario final [(salario por hora * total de horas) + salario] . formula : vSalarioBase=salarioHora * totalHora vResultado= (salario + vSalarioBase) * 0.03