

LINGUAGEM DE PROGRAMAÇÃO I

↳ UNIDADE 2

~~REVISÃO~~ SORTEAMENTO ~~IMPLEMENTAÇÃO DE CLASSES~~

COMPOSIÇÃO

HERANÇA

POLIMORFISMO

REFERENCIA THIS

CLASSES ABSTRACTAS

INTERFAZES

LISTA 6

TRATAMENTO DE ERROS

INTERFAZES GRÁFICAS

1 Sobrecarga

① MÉTODOS COM O MESMO NOME EM UMA CLASSE.

↳ 2 FORMAS DE ACHAR UMA TABELA.

↳ MÉTODOS SOBRECARREGADOS.

[Ex]

② ASSINATURA = NOME MÉTODO + TIPOS DE ARGUMENTOS →

↳ DIFERENCIAMOS PELOS TIPOS DE PARÂMETROS

Construtor() { } 3

Construtor(int x) { } 3

③ TWS = Parâmetro Reservado Para Se Referir De Um Construtor Sobrescrito Para Outro.

```
class criaEmpregado{
    public static void main(String args[]){
        Empregado emp1 = new Empregado("Maria","123123123","Rua das flores 245", 25, 500);
        System.out.println("Idade Maria = " + emp1.Devolveldade());
        System.out.println("Idade Maria 10 anos = " + emp1.Devolveldade(10));

        Empregado emp2 = new Empregado("Tadeu",700);
        System.out.println("Salario Tadeu = " + emp2.salario);
    }
}
```

INSTA const *Programa Iniciando*

EMP2

```
public class Empregado{
    String nome, endereço, cpf;
    float salario;
    int idade;

    public Empregado(String n, String end, String c, int ida, float sal){
        this(n,sal);
        endereço = end;
        cpf = c;
        idade = ida;
    }

    public Empregado(String n, float sal){
        nome = n;
        salario = sal;
    }

    public Devolveldade(){
        return idade;
    }

    public Devolveldade(int numAnos){
        return idade+numAnos;
    }
}
```

Construtor Sobrecarregado

this(n,sal)

nome = n;
salario = sal

Sobrecarregado
Assinatura Distinta

Uso do TWS P/ chamar construtores
do outro construtor sobrecarregado.

(2) A Referência THIS

• NA CHAMADA DOS MÉTODOS DE UMA CLASSE A JVM PASSA IMPLÍCITAMENTE como PARÂMETRO uma REFERÊNCIA ao OBJETO AO QUAL A MENSAGEM FOI ENVIADA.

• QUANDO SE DESEJA RETORNAR ESSE REFERÊNCIA(OBJETO) DE DENTRO DO CORPO DA CLASSE, USA-SE A PALAVRA RESERVADA THIS.

THIS(OBJETO)

• PODE-SE UTILIZAR A PALAVRA CHAVE THIS PARA SE REFERIR A UM CONSTRUTOR SOBREEMULGADO.

NESSA CASO, THIS NÃO DIRIGE REFERÊNCIA AO OBJETO, POIS, O MESMO(OBJETO) JÁ ESTÁ SENDO USADO.

THIS(Constructor)

FAZ DE COM BOMBO PÉ UM CONSTRUTOR QUE MUDA A REFERÊNCIA QUE PÔ DE OBJETO FOI USADO?

• RESUMO THIS

• REFERÊNCIA AO OBJETO;

• PERMITE MÉTODOS E PROPRIEDADES NA INSTÂNCIA DE UM OBJETO.

• PASSA O OBJETO COMO PARÂMETRO.

```
public class ExemploThis{  
    public static void main(String args[]){  
  
        Funcionario func1 = new Funcionario("Maria",500);  
        Funcionario func2 = new Funcionario("Joao",700);  
  
        func1.salario = func1.ajustaSalario(20);  
        func2.salario = func2.ajustaSalario(30);  
  
        System.out.println("Novo Salario de " + func1.nome + " é = " + func1.salario);  
        System.out.println("Novo Salario de " + func2.nome + " é = " + func2.salario);  
    }  
}
```

ATRIBUTO
DO
OBJETO.

```
public class Funcionario{  
  
    String nome;  
    float salario;  
  
    public Funcionario(String n, float s){  
        nome = n;  
        salario = s;  
    }  
  
    public float ajustaSalario(float percent){  
        float novoSalario;  
        novoSalario = this.salario + calculo(percent,this);  
        return novoSalario;  
    }  
  
    private float calculo(float percentual, Funcionario objfunc){  
        float valorAcrescido;  
        valorAcrescido = objfunc.salario * percentual / 100;  
        return valorAcrescido;  
    }  
}
```

OBJETO

↓

OBJETO

③ REUTILIZAÇÃO DE CLASSES

① BENEFÍCIO P.D.O = REUTILIZAÇÃO DE CÓDIGO

② REUTILIZAÇÃO DE CÓDIGO

↳ AMPLIAMENTO DE CLASSES E SEUS MÉTODOS

↳ DIVIDI ACESSIVAMENTE PARA CRIAR NOVOS MÉTODOS E CLASSES GERANDO ECONOMIA DE TIPO E SEGURANÇA.

③ TIPOS DE REUTILIZAÇÃO DE CÓDIGO:

④ COMPOSIÇÃO

⑤ HERANÇA

3.1) COMPOSIÇÃO

⑥ COMPOSIÇÃO = UMA CLASSE X IRA INSTANCIAR OBJETOS DE UMA CLASSE Y (OU MAIS OUTRAS CLASSES).

↳ UTILIZANDO DIFERENTES CLASSES PARA OBTER OS OBJETOS PARA COMPOZER FUNCIONALIDADES EM UMA NOVA CLASSE.



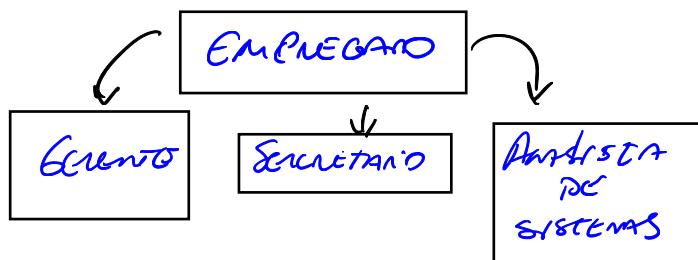
3.2) HERANÇA

*) Herança = CAPACIDADE DE UMA CLASSE DEFINIR O SEU COMPORTAMENTO E SUA ESTRUTURA APROVITANDO DEFINIÇÕES DE OUTRA CLASSE, CONhecida como CLASSE BASE.

↳ A HERANÇA NÃO PRECISA SER INTERATORIA NA DEFINIÇÃO DE UMA COLEÇÃO DE CLASSES.

↳ A COLEÇÃO DE TANTAS AS CLASSES QUE SE ESTENDEM DE UMA CLASSE BASE CHAMA-SE HERANÇA DE HERANÇA.

*) EXTENSÃO = Processo Reservado Visto P/ DEFINIR QUE UMA CLASSE X ESTÁ HERANÇANDO DE OUTRA CLASSE Y

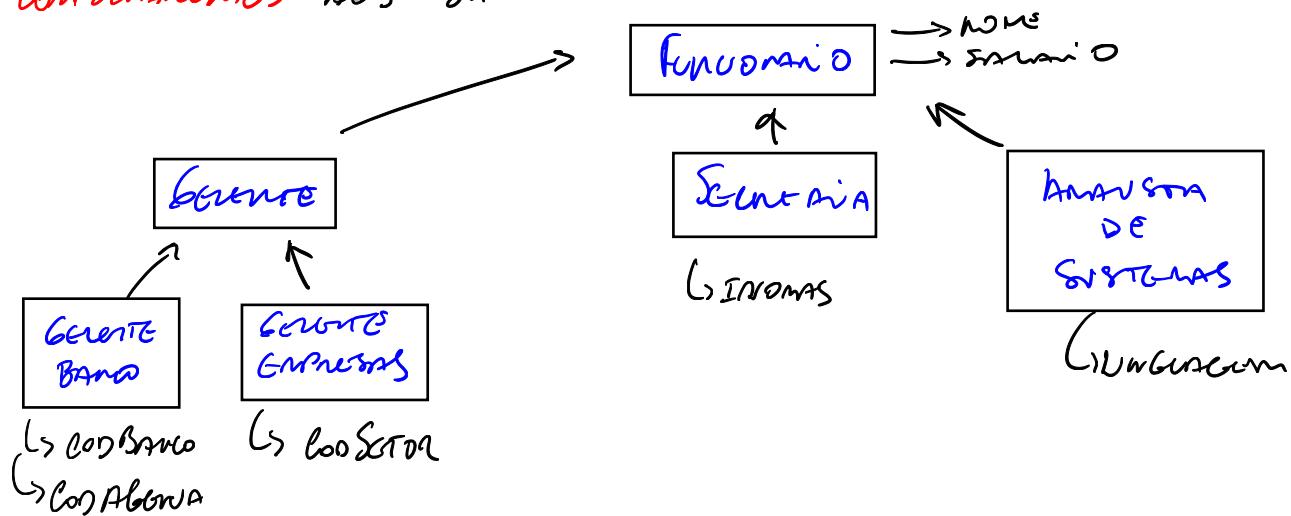


↳ GENERALIZAÇÃO DE UMA CLASSE

*) ATRAVÉS DO MECANISMO DA HERANÇA É POSSUIR DIFERENTES CLASSES GERAIS QUE AGLOREM UM CONJUNTO DE DEFINIÇÕES COMUNS A MUITOS MUNDOS DOS OBJETOS.

↳ ESPECIAZIAR DE UMA CLASSE

*) A PARTIR DESTAS ESPECIAZAÇÕES GERAIS PODEMOS CONSTRUIR NOVAS CLASSES, MAIS ESPECÍFICAS, QUE ACRESCENTEM NOVAS CARACTERÍSTICAS E COMPORTAMENTOS NAS SUAS EXISTENCIAS.



3.3) Herança e a Reclusão Super

④ SUPER

↳ REFERENCIA A CLASSE BASE DO OBJETO.

↳ ACESSO A VARIAVEIS ESCONDIDAS (SHADOW), OU SERA, VARIAVEIS DE UMA CLASSE QUE TEM O MESMO NOME DE UMA VARIAVEL DEFINIDA EM UMA SUBCLASSE.

↳ ACESSO A METODOS QUE FORAM REDEFINIDOS, OU SERA, METODOS QUE POSSUEM A MESMA ASSINATURA NA CLASSE BASE E NA CLASSE DERIVADA.

↳ UTILIZACAO PARA CLASSE DERIVADA DD CONSTANTE DA CLASSE BASE.

```
// CLASSE BASE
public class Automovel{
    // DECLARAÇÃO DE CONSTANTES
    public static final byte movidoAGasolina = 1;
    public static final byte movidoAAlcool = 2;
    public static final byte movidoADisel = 3;
    public static final byte numeroMaximoDePrestacoes = 24;

    // ATRIBUTOS
    private String modelo, cor;
    private byte combustivel;

    // CONSTRUTOR
    public Automovel(String m, String c, byte comb){
        modelo = m;
        cor = c;
        combustivel = comb;
    }
    public byte quantasPrestacoes(){
        return numeroMaximoDePrestacoes;
    }
    public float quantasCusta(){
        float preco = 0;
        switch(combustivel){
            case movidoAGasolina:
                preco = 12000.0f;
                break;
            case movidoAAlcool:
                preco = 10500.0f;
                break;
            case movidoADisel:
                preco = 11000.0f;
                break;
        }
        return preco;
    }
    public String toString(){
        String resultado;
        resultado = modelo + " " + cor + "\n";
        switch(combustivel){
            case movidoAGasolina:
                resultado += "Gasoline \n";
                break;
            case movidoAAlcool:
                resultado += "Alcool \n";
                break;
            case movidoADisel:
                resultado += "Diesel \n";
                break;
        }
        return resultado;
    }
}
```

```
// USO DO EXTENDS PARA AUTOMOVEL BASICO HERDAR DE AUTOMOVEL
public class AutomovelBasico extends Automovel{
    // atributos
    private boolean retrovisorDoLadoDoPassageiro;
    private boolean limpadorDoVidroTraseiro;
    private boolean radioAMFM;

    public AutomovelBasico(String m, String c, byte comb, boolean r, boolean l, boolean af){
        // SUPER -> INDICA QUE DEVE SER USADO O CONSTRUTOR DA CLASSE PAI.
        super(m,c,comb);
        retrovisorDoLadoDoPassageiro = r;
        limpadorDoVidroTraseiro = l;
        radioAMFM = af;
    }
    public AutomovelBasico(String m, String c, byte comb){
        super(m,c,comb);
        retrovisorDoLadoDoPassageiro = true;
        limpadorDoVidroTraseiro = true;
        radioAMFM = true;
    }
    // OS DOIS METODOS A SEGUIR POSSUEM A MESMA ASSINATURA DA CLASSE AUTOMOVEL, O QUE CARACTERIZA UMA REDEFINIÇÃO DE METODOS DA CLASSE BASE.
    public float quantoCusta(){
        float preco = super.quantoCusta();
        if(retrovisorDoLadoDoPassageiro == true){
            preco = preco + 280;
        }
        if(limpadorDoVidroTraseiro == true){
            preco = preco + 650;
        }
        if(radioAMFM == true){
            preco = preco + 190;
        }
        return preco;
    }
    // SUPER -> INDICA QUE DEVE SER CHAMADO O METODO QUANTOCUSTA() E TOSTRING() DA CLASSE BASE.
    public String toString(){
        String resultado = super.toString();
        if(retrovisorDoLadoDoPassageiro == true){
            resultado += "Com limpador traseiro \n";
        }
        if(limpadorDoVidroTraseiro == true){
            resultado += "Com limpador traseiro \n";
        }
        if(radioAMFM == true){
            resultado += "Com radio \n";
        }
        return resultado;
    }
}
```

```

public class AppAutomovel {
    public static void main(String args[]){
        // instancia de um objeto da classe automovel;
        Automovel objA = new Automovel("Fusca","verde",Automovel.movidoAAlecool);

        System.out.println(objA.toString());
        System.out.println(objA.quantoCusta());
        System.out.println(objA.quantasPrestacoes());

        // instancia de um objeto da classe AUTOMOVELBASICO
        AutomovelBasico objAB = new AutomovelBasico("Corolla","cinza",Automovel.movidoAGasolina,true,true,false);

        System.out.println(objAB.toString());
        System.out.println(objAB.quantoCusta());
        System.out.println(objAB.quantasPrestacoes());
    }
    // Observe que o metodo quantasPrestacoes() esta sendo acessado atraves de um objeto da classe AutomovelBasico. Isso so é
    // possivel porque a classe AUTOMOVELBASICO herda da classe AUTOMOVEL, assim todos os ATRIBUTOS e METODOS da CLASSE BASE
    // podem ser usados pela CLASSE DERIVADA.
}

```

3.4) Polimorfismo

④ O MECANISMO DE TIPO PERmite A ENCARTE DE CLASSES A PARTIR DE
CLASSES JA EXISTENTES COM RELACAO E-UM-TIPO-DE, DES TURNA QUE, A PARTIR DE
UMA CLASSE GENERICA, CLASSES MAIS ESPECIALIZADAS POSSAM SER ENCARTE.

④ A RELACAO E-UM-TIPO-DE ENTRE CLASSES PERmite IA EXISTENCIA DE OUTRA
CARACTERISTICA CHAMADA DE VANTAGENS O.O, O Polimorfismo.

④ Polimorfismo = PERmite A MANIPULACAO DE INSTANCIAS DE CLASSE C,
E OS MESMOS METODOS SEDAS CAPAZES DE PROCESSAR
INSTANCES DE QUALQUER CLASSE QUE HERDE DE C, JA'
QUE QUALQUER CLASSE QUE HERDE DE E E-UM-TIPO-DE C.

④ Polimorfismo VANTAGENS:

↳ PERmite O ENVIOL DE MENSAGENS A UM OBRETO SEM
A DETERMINACAO EXATA DO SEU TIPO.

↳ PERmite A EXTENSIBILIDADE DO SISTEMA COM POCO OU
NENHUM IMPACTO NAS CLASSES EXISTENTES.

↳ PERmite A CONSTRUCAO DE CLASSES GENERICAS PARA EFETUAR TAREHAS GERAIS COMUNS AOS SISTEMAS, COMO
AS ESTRUTURAS DE DADOS.

Ex | → Polimorfismo

```
// CLASSE BASE
public class Automovel{
    // DECLARAÇÃO DE CONSTANTES
    public static final byte movidoAGasolina = 1;
    public static final byte movidoAAlcool = 2;
    public static final byte movidoADisel = 3;
    public static final byte numeroMaximoDePrestacoes = 24;

    // ATRIBUTOS
    private String modelo, cor;
    private byte combustivel;

    // CONSTRUTOR
    public Automovel(String m, String c, byte comb){
        modelo = m;
        cor = c;
        combustivel = comb;
    }

    public byte quantasPrestacoes(){
        return numeroMaximoDePrestacoes;
    }

    public float quantoCusta(){
        float preco = 0;
        switch(combustivel){
            case movidoAGasolina:
                preco = 12000.0f;
                break;
            case movidoAAlcool:
                preco = 10500.0f;
                break;
            case movidoADisel:
                preco = 11000.0f;
                break;
        }
        return preco;
    }

    public String toString(){
        String resultado;
        resultado = modelo + " " + cor + "\n";
        switch(combustivel){
            case movidoAGasolina:
                resultado += "Gasoline \n";
                break;
            case movidoAAlcool:
                resultado += "Alcool \n";
                break;
            case movidoADisel:
                resultado += "Diesel \n";
                break;
        }
        return resultado;
    }
}
```

```
// USO DO EXTENDS PARA AUTOMOVEL BASICO HERDAR DE AUTOMOVEL
public class AutomovelBasico extends Automovel{
    // atributos
    private boolean retrovisorDoLadoDoPassageiro;
    private boolean limpadorDoVidroTraseiro;
    private boolean radioAMFM;
    public AutomovelBasico(String m, String c, byte comb, boolean r, boolean l, boolean af){
        // SUPER -> INDICA QUE DEVE SER USADO O CONSTRUTOR DA CLASSE PAI.
        super(m,c,comb);
        retrovisorDoLadoDoPassageiro = r;
        limpadorDoVidroTraseiro = l;
        radioAMFM = af;
    }

    public AutomovelBasico(String m, String c, byte comb){
        super(m,c,comb);
        retrovisorDoLadoDoPassageiro = true;
        limpadorDoVidroTraseiro = true;
        radioAMFM = true;
    }

    // OS DOIS METODOS A SEGUIR POSSUEM A MESMA ASSINATURA DA CLASSE AUTOMOVEL, O
    // QUE CARACTERIZA UMA REDEFINIÇÃO DE METODOS DA CLASSE BASE.
    public float quantoCusta(){
        float preco = super.quantoCusta();
        if(retrovistorDoLadoDoPassageiro == true){
            preco = preco + 280;
        }
        if(limpadorDoVidroTraseiro == true){
            preco = preco + 650;
        }
        if(radioAMFM == true){
            preco = preco + 190;
        }
        return preco;
    }

    // SUPER -> INDICA QUE DEVE SER CHAMADO O METODO QUANTOCUSTA() E TOSTRING() DA
    // CLASSE BASE.
    public String toString(){
        String resultado = super.toString();
        if(retrovistorDoLadoDoPassageiro == true){
            resultado += "Com limpador traseiro \n";
        }
        if(limpadorDoVidroTraseiro == true){
            resultado += "Com limpador traseiro \n";
        }
        if(radioAMFM == true){
            resultado += "Com radio \n";
        }
        return resultado;
    }
}
```

METODO que RECEBE um PARÂMETRO do TIPO AUTOMOVEL

```
public class ConcessionariaDeAutomoveis{
    public static void imprime(Automovel a){
        System.out.println("Dados do automovel escolhido: ");
        System.out.println(a.toString());
        System.out.println("Valor: R$" + a.quantoCusta());
        System.out.println(a.quantasPrestacoes() + "prestações de " + (a.quantoCusta()/a.quantasPrestacoes()));

    }

    public static void main(String args[]){
        Automovel objA1 = new Automovel("Fiat", "bege", Automovel.movidoAAlcool);
        AutomovelBasico objA2 = new AutomovelBasico("Corsa", "cinza", Automovel.movidoAGasolina);

        // O metodo IMPRIME() recebe um objeto da classe Automovel como parametro. Observe que, nesse exemplo
        // chamamos o metodo passando (OBJA1) e (OBJA2), ou seja, objetos de classes diferentes mas da mesma
        // HIERARQUIA DE CLASSES, caracterizando dessa forma a utilização de polimorfismo.

        imprime(objA1);
        imprime(objA2);
    }
}
```

3 USO DO METODO que RECEBE UM PARÂMETRO DO TIPO UMA CLASSE BASE, USANDO REGRAS (CLASSES HERDADAS).

④ Classes Abstratas

• São classes compostas por implementações genéricas e especificações de procedimentos.

↳ Não produzem instâncias, ou seja, não podemos criar objetos dessa classe.

↳ Agregam características e comportamentos que serão herdados por outras classes.

↳ Forneceem padrões de comportamento que serão implementados nas subclasses.

• A classe abstrata passa a ser o molde genérico para as classes que serão definidas.

• Para se definir uma classe abstrata usa-se a palavra-chave **Abstract**.

• Método abstrato = molde de uma implementação a ser provada a classes filhas concretas.

↳ Pode haver várias implementações de um mesmo procedimento.

• Métodos abstratos → Classes Abstratas.

↓ CONCRETA

• Métodos na abstrata ⇒ podem ser vistos em classes que derivam das classes abstratas.

• As classes que derivam de classes abstratas devem obrigatoriamente implementar todos os métodos abstratos definidos na classe base.

Ex) -> Classe Abstrata

```
// definição da CLASSE ABSTRATA através da palavra ABSTRACT
abstract class Figura{
    int x;
    int y;

    // a classe pode ter um CONSTRUTOR, mesmo que não possa instanciar objetos.
    public Figura(int x1, int y1){
        x = x1;
        y = y1;
    }

    // Declaração dos métodos abstratos. Esses métodos devem OBRIGATORIAMENTE ser implementados nas classes derivadas.
    public abstract void desenha();
    public abstract void apaga();

    // Declaração de um MÉTODO NORMAL que poderá ser utilizado pelos objetos de classes derivadas;
    public void move(int x1, int y1){
        apaga();
        x = x1;
        y = y1;
        desenha();
    }
}
```

```
class Quadrado extends Figura{
    public Quadrado(int x1, int y1){
        // utilização do CONSTRUTOR DA CLASSE BASE.
        super(x1,y1);
    }
    // implementação OBRIGATÓRIA dos MÉTODOS definidos na CLASSE ABSTRATA.
    public void desenha(){
        System.out.println("Desenhando quadrado(" + x + "," + y + ")");
    }
    public void apaga(){
        System.out.println("Apagando quadrado(" + x + "," + y + ")");
    }
}
```

```
// CLASSE/PROGRAMA PARA TESTAR O ABSTRACT
public class TesteAbstract{
    public static void main(String args[]){
        Quadrado q = new Quadrado(10,10);
        q.desenha();
        q.move(50,50);
        q.apaga();
    }
}
```

3 Interfaces

- ① Se uma **Classe** é definida por apenas **MÉTODOS ABSTRATOS** essa é melhor usar a **Parâma Classe ABSTRACT**.
- ② Para estes casos o Java fornece uma técnica chamada **Interfaces** que podem ser usadas para definir um **MERCADO DE COMPATIBILIDADE**.
- ③ As **Interfaces** diferem dos **MÉTODOS ABSTRATOS** no fato de que NENHUM MÉTODO DA INTERFACE PODE TER UM CORPO.
- ④ Interfaces:
- ↳ ESTRITAMENTE MÉTODOS.
 - ↳ NÃO PODE SER INSTANCIADA.
 - ↳ UMA CLASSE PODE IMPLEMENTAR VÁRIAS **Interfaces**, MAS APENAS **HERDA** (EXTENDE) DE UMA UNICA CLASSE.
 - ↳ Interfaces COMPLICADAS = classes.
 - ↳ Interface = Coleção de Definição de métodos.
 - ↳ Interfaces PODE DECLARAR CONSTANTES.
- ⑤ Declaração de uma Interface = P.R Interface
- ⑥ SINTATICAMENTE Interface equivale a uma classe comumente ABSTRAIA
- ↳ TODOS SÓS MÉTODOS SÃO ABSTRATOS.
 - ↳ TÓMOS AS SEUS VARIÁVEIS SÃO PUBLIC STATIC final;
- ⑦ SEMANTICAMENTE as duas diferem, pois uma Interface GUARDA UMA **ÍDEA DE PROTOCOLO** (comunicação) entre classes.
Porque?

Ex → Interfaces

```
interface FiguraGeometrica{  
    public void desenha();  
    public void apaga();  
    public void move(int x1, int y1);  
}
```

④ Nesse exemplo a classe [Linha] implementa a interface [FiguraGeometrica] e precisa desenhar todos os métodos definidos pela interface.

além?

```
public class Linha implements FiguraGeometrica{
```

```
    int x, y;  
    public Linha(int x1, int y1){  
        x = x1;  
        y = y1;  
    }  
    public void desenha(){  
        System.out.println("Desenhando Linha (" + x + "," + y + ")");  
    }  
    public void apaga(){  
        System.out.println("Apagando Linha (" + x + "," + y + ")");  
    }  
    public void move(int x1, int y1){  
  
        this.apaga();  
        x = x1;  
        y = y1;  
        this.desenha();  
    }  
}
```

Classe

```
class Quadrado extends Figura{  
    public Quadrado(int x1, int y1){  
        // utilização do CONSTRUTOR DA CLASSE BASE.  
        super(x1,y1);  
    }  
    // implementação OBRIGATÓRIA dos MÉTODOS definidos na CLASSE ABSTRATA.  
    public void desenha(){  
        System.out.println("Desenhando quadrado(" + x + "," + y + ")");  
    }  
    public void apaga(){  
        System.out.println("Apagando quadrado(" + x + "," + y + ")");  
    }  
}
```

⑥ Packages

④ Packages = Aglomerado de classes e interfaces ou conjunto lógico unis relacionados.

↳ Em Java é um conjunto de classes e interfaces que estão dentro de um mesmo diretório de arquivos

④ PACKAGE é usado para dar p/:

↳ garantir a unicidade do nome da classe.

④ Sem nomes iguais no mesmo pacote.

↳ Realizar controle de acesso;

④ Membros de uma classe tem a capacidade de controle de acesso das variáveis membros que podem ser acessadas pelas classes do mesmo pacote.

④ Para colocar uma **Class** em um determinado pacote, deve-se colocar a palavra reservada **PACKAGE** na **primeira linha de código do arquivo fonte (*.java)**.

Package java.lang;

⑤ Utilização **Classes** de outros **Pacotes**:

Import java.lang.String;

ou assim

import java.lang.*;

→ Importa **classes** n
classes **String**.

→ Importa **todas** as **classes**
do **Pacote Lang**.

⑥ Principais Pacotes da Linguagem Java:

- **java.applet** □ Classes para criação de Applets.
- **java.awt** □ Classes para criação de interfaces gráficas.
- **java.io** □ Classes para gerenciamento de Entradas e Saídas.
- **java.lang** □ Classes Básicas da linguagem (incluídas automaticamente, não necessita o uso do import).
- **java.net** □ Classes para trabalho em rede.
- **java.util** □ Classes de utilitários (Data, Dicionário, Pilha, Vetor, Random, etc.)
- **java.sql** □ Classes para manipulação de Banco de Dados

Exercícios Ueca 6 - Revisão de Classes

Questão 1 → Classes LivroLiquido e LivroBiblioteca que herdam da classe Livro.

OBS → Campos em comum representam sua classe Arcosnac.

