

# ELE2 – LPG3

Eletiva 2 – Linguagem de Programação 3 (JAVA)  
Orientação a Objetos – Revisão Básica

## Conceitos

- Classe
  - Objeto
  - Instância
- Construtor e Destrutor
- Encapsulamento (Membros)
  - Atributos
  - Métodos
- Herança
- Polimorfismo

2

## Classe

- Abstrai um conjunto de objetos com características e comportamentos similares
- Define o comportamento de seus objetos através de:
  - Métodos
  - Atributos (estados possíveis destes objetos)
- “Descreve” a funcionalidade de seus objetos

3

## Estrutura de um Classe

- Atributos
  - Métodos
  - Construtores
  - Destrutor
- Sintaxe:

```
class NomeDaClasse(  
    modificador tipo atributo  
    modificador tipo método  
(tipo parâmetro1,  
    tipo parâmetro2...,  
    tipo parâmetroN){  
    //Código...  
}  
}
```

4

### Atributos

- São declarações das classes que representam características de instância que possam assumir estados
- No exemplo, dia, mês e ano são atributos do tipo inteiro da classe Data, enquanto horas, minutos e segundos são atributos do tipo inteiro da classe Horário. Na classe Main, nenhum atributo foi declarado.

```
1 package java11anoes;  
2  
3 class Data {  
4     int dia;  
5     int mes;  
6     int ano;  
7 }  
8  
9 class Horario {  
10     int horas;  
11     int minutos;  
12     int segundos;  
13 }  
14  
15 public class Main {  
16     public static void main(String[] args) {  
17         Data d = new Data();  
18         d.ano = 2008; d.mes = 0; d.dia = 19;  
19         Horario h = new Horario();  
20         h.horas = 12; h.minutos = 0; h.segundos = 0;  
21         System.out.println("Agora são " + h.horas  
22             + " horas de " + d.dia + "/" + d.mes);  
23     }  
24 }  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100
```

### Métodos

- São declarações das classes que representam ações que a mesma pode executar em função dos valores de seus atributos
- No exemplo, um método chamado retornarTexto() em cada classe retorna uma String representando a data e a hora em função dos valores dos atributos de suas respectivas instâncias

```
1 package java11anoes;  
2  
3 class Data {  
4     int dia; int mes; int ano;  
5     String retornarTexto() {  
6         return dia + "/" + mes + "/" + ano;  
7     }  
8 }  
9  
10 class Horario {  
11     int horas; int minutos; int segundos;  
12     String retornarTexto() {  
13         return horas + ":" + minutos + ":" + segundos;  
14     }  
15 }  
16  
17 public class Main {  
18     public static void main(String[] args) {  
19         Data d = new Data();  
20         d.ano = 2008; d.mes = 0; d.dia = 19;  
21         Horario h = new Horario();  
22         h.horas = 12; h.minutos = 0; h.segundos = 0;  
23         System.out.println(d.retornarTexto() + ", "  
24             + h.retornarTexto());  
25     }  
26 }  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100
```

### Objetos e Instâncias

- Objeto é uma entidade que pode ser física, conceitual ou de software. Ou seja, uma representação genérica.
- Instância é usada com o sentido de exemplo. É a concretização da classe. Ou seja, são os objetos de fato criados e ocupando espaço na memória.



### Construtor

- Método especial. É disparado automaticamente na instanciação da classe
- Utilizamos ele basicamente para inicializar os estados dos atributos
- No exemplo a seguir, foram criados dois construtores. Cada um deles é chamado de acordo com a criação da instância, pela lista de argumentos

### Construtor - Exemplo

```
1 package jappclasses;
2
3 class Data{
4     int dia; int mês; int ano;
5     Data(){
6         dia = 0; mês = 0; ano = 0;
7     }
8     Data(int dia, int mês, int ano){
9         this.dia = dia; this.mês = mês; this.ano = ano;
10    }
11    public String retornarTexto(){
12        return dia + "/" + mês + "/" + ano;
13    }
14 }
15 public class Main {
16     public static void main(String[] args) {
17         Data d1 = new Data(19, 8, 2008);
18         Data d2 = new Data();
19         d2.ano = 2008; d2.mês = 8; d2.dia = 20;
20         System.out.println("Data 1: " + d1.retornarTexto());
21         System.out.println("Data 2: " + d2.retornarTexto());
22     }
23 }
```

Salda - jappClasses (run)

deps-jar:  
Compiling 1 source  
compile:  
run:  
Data 1: 19/8/2008  
Data 2: 20/8/2008

### Destrutor

- O método Destrutor não pode ser sobrecarregado porque não aceita parâmetros e é executado quando o objeto é excluído.

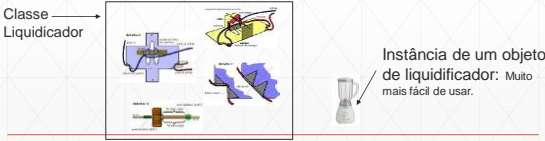
```
obj.finalize();
```

### Encapsulamento

- Significa limitação de acesso aos membros da classe
- Utiliza-se de modificadores de acesso, tais como:
  - Public: Acessível a partir de qualquer outra classe, independente do package
  - Protected: Acessível a partir de qualquer classe derivada, no mesmo package
  - Private: Acessível apenas dentro da própria classe
  - Static: Membro compartilhado com todas as instâncias da classe
  - Final: Constante da classe
  - Friendly: Acessível a partir de qualquer outra classe do mesmo package

### Encapsulamento

- Mecanismo que permite separar detalhes de funcionamento – características (atributos) e funções (métodos) – de sua interface.
- Exemplo: Para utilizarmos um liquidificador, não precisamos saber detalhes de seu funcionamento. A única interface que conhecemos são seus botões.



### Encapsulamento - Atributo

- São características (variáveis) da classe e podem ser divididos em dois tipos básicos:
  - Atributos de Instância: "determinam o estado de cada objeto."
  - Atributos de Classe: "possui um estado que é compartilhados por todos os objetos de uma classe."

```
class CartãoDeBaralho{
    public char simbolo;
    public image Naipes;

    public static ListaDeSimbolos[] = {0,1,2,3,4,5,6,7,8,9,J,Q,K,A};
    public static ListaDeNaipes[] = { ♥ , ♠ , ♦ , ♣ };
}
```

### Encapsulamento - Método

- É uma rotina que é executada por um objeto ao receber uma mensagem: "É a descrição da forma como o objeto realiza uma tarefa."
- Os métodos determinam o comportamento dos objetos de uma classe e são análogos à funções ou procedimentos da programação estruturada.

```
class Chachorro{
    //Atributos de instância:
    public String Nome;
    public Color CorDoPelo;
    public String Raça;
    ;
    //Atributos de classe:
    public static ListaDeRaças[]
        = {Doberman, Pit Bull, ...};
    ;
    //Métodos:
    public void Correr(int Direção){
        //Código-fonte
    }
    public void Sentar(){
        //Código-fonte
    }
    public void Parar(){
        //Código-fonte
    }
    ;
}
```

### Exemplo de encapsulamento

```
package jappclasses;

class Data{
    private int dia;
    public int getDia() { return dia; }
    public void setDia(int dia) {
        if(dia > 0 && dia < 32) this.dia = dia;
    }
    private int mês;
    public int getMês() { return mês; }
    public void setMês(int mês) {
        if(mês > 0 && mês < 13) this.mês = mês;
    }
    private int ano;
    public int getAno() { return ano; }
    public void setAno(int ano) { this.ano = ano; }
    public String retornarTexto() { return dia + "/" + mês + "/" + ano; }
}

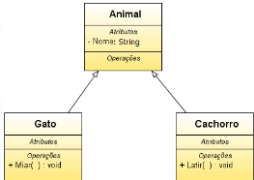
public class Main {
    public static void main(String[] args) {
        Data d = new Data();
        d.setAno(2008); d.setMês(8); d.setDia(19);
        System.out.println("Data: " + d.retornarTexto());
    }
}
```

### Herança - Definição

- Conhecida também como Generalização;
- Definição de uma classe a partir de outra, herdando os seus membros;
- A classe herdeira é chamada subclasse;
- A classe herdada é chamada superclasse;
- Pode ser utilizado em cascata, criando hierarquias com várias gerações de classes;
- Permite que as classes compartilhem atributos e operações baseados em um relacionamento.

## Subclasse e superclasse

- Superclasse é a classe herdada. Ou seja, a classe Pai;
- Subclasse é a classe herdeira. Ou seja, a classe Filha;
- No exemplo, animal é a superclasse;
- No exemplo, cachorro e gato são subclasses.



## Exemplo

```
public class Pessoa {
    /** Creates a new instance of Pessoa */
    public Pessoa() {
    }
    public String Nome;
    public Date nascimento = new Date();
}

public class Aluno extends Pessoa {
    /** Creates a new instance of Aluno */
    public Aluno() {
        disciplinas = new Vector();
    }
    public int matricula;
    public Vector disciplinas;
}

public class Professor extends Pessoa {
    /** Creates a new instance of Professor */
    public Professor() {
        disciplinas = new Vector();
    }
    public float salário;
    public Vector disciplinas;
    public Date dataDeAdmissão = new Date();
}
```

## Exemplo II

```
package br.com.fatecpg.heranca;

public class MeuForm extends javax.swing.JFrame {
    javax.swing.JLabel MeuFormLabel;

    public MeuForm() {
        this.setTitle("Meu Formulário JFrame");
        MeuFormLabel = new javax.swing.JLabel();
        MeuFormLabel.setText("Meu Formulário");
        this.add(MeuFormLabel);
        this.setSize(300, 50);
    }
}

package br.com.fatecpg.heranca;

public class Main {
    public static void main(String[] args) {
        MeuForm f = new MeuForm();
        f.setVisible(true);
    }
}
```

## Polimorfismo

- Grego: “muitas formas”
- *poli* = muitas, *morphos* = formas
- Permite que referências de tipos de classes mais abstratas representem o comportamento das classes concretas que referenciam;
- Um mesmo método pode apresentar várias formas, de acordo com seu contexto;

### Polimorfismo

- Permite que a semântica de uma interface seja efetivamente separada da implementação que a representa;
- Basicamente, override e overload;
- Benefícios:
  - Clareza e manutenção do código;
  - Divisão da complexidade;
  - Aplicações flexíveis.

### Sobreposição (Override)

- Utilizamos quando queremos reescrever um método na classe Filha que já existe e está implementado na classe Pai;
- Podemos chamar o método pai (se for desejado) através da palavra reservada super;
- Também chamado de sobrescrita ou polimorfismo vertical.

### Exemplo

```
1 package br.com.fatecpg.heranca;
2
3 import java.awt.Graphics;
4
5 abstract class Forma {
6     abstract void Desenha (Graphics g, int x, int y);
7     abstract double getArea();
8     void ImprimeArea(){
9         System.out.println("Área da Forma: " + this.getArea());
10    }
11 }
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

### Exemplo (utilizando o “super.”)

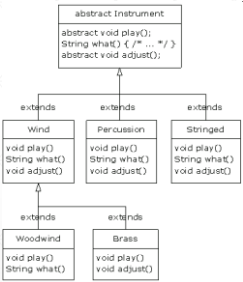
```
1 package br.com.fatecpg.heranca;
2
3 import java.awt.Graphics;
4
5 class Retângulo extends Forma {
6     public int base;
7     public int altura;
8     void Desenha (Graphics g, int x, int y) {
9         g.drawRect(x, y, x + base, y + altura);
10    }
11    double getArea() {
12        return base * altura;
13    }
14    @Override
15    void ImprimeArea() {
16        super.ImprimeArea();
17    }
18 }
```

### Sobrecarga (Overload)

- Quando há 2 ou mais métodos com o mesmo nome, mas com assinatura (e comportamento) diferente, declarados na mesma classe;
- Também conhecido como overload ou polimorfismo horizontal.

### Classes abstratas

- São classes que abstraem um conjunto de características que podem ou não serem implementadas;
- Não podem ser instanciadas;
- “A implementamos quando sabemos o que fazer mas não como fazer”.



### Exemplo

```
package br.com.fatecpg.heranca;

import java.awt.Graphics;

abstract class Forma {
    abstract void Desenha(Graphics g, int x, int y);
    abstract double getArea();
}

class Quadrado extends Forma {
    public int lado;
    void Desenha(Graphics g, int x, int y) {
        g.drawRect(x, y, x + lado, y + lado);
    }
    double getArea() {
        return lado * lado;
    }
}

class Retângulo extends Forma {
    public int base;
    public int altura;
    void Desenha(Graphics g, int x, int y) {
        g.drawRect(x, y, x + base, y + altura);
    }
    double getArea() {
        return base * altura;
    }
}
```

### Upcasting e Downcasting

- Upcasting é quando transformamos (através de uma atribuição) um tipo de baixo para cima, na hierarquia;
- Downcasting é quando transformamos (também através de uma atribuição) um tipo de cima para baixo na hierarquia.

### Exemplo de Upcasting

```
1 package br.com.fatecpg.heranca;
2
3 public class Main {
4     public static void main(String[] args) {
5         Quadrado q1 = new Quadrado();
6         q1.lado = 15;
7
8         Forma f = q1;
9
10        System.out.println("Área da Forma: " + f.getÁrea());
11    }
12 }
```

Output - JavaApplication1 (run) #10

```
init:
deps-jar:
Compiling 1 source file to C:\Documents and Settings\user\workspace\JavaApplication1\out\production\JavaApplication1:
run:
Área da Forma: 225.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

### Exemplo de Downcasting

```
1 package br.com.fatecpg.heranca;
2
3 public class Main {
4     public static void main(String[] args) {
5         Forma f = new Quadrado();
6         Quadrado q = (Quadrado)f;
7         q.lado = 15;
8         System.out.println("Área do quadrado: " + q.getÁrea());
9     }
10 }
```

Output - JavaApplication1 (run) #10

```
init:
deps-jar:
Compiling 1 source file to C:\Documents and Settings\user\workspace\JavaApplication1\out\production\JavaApplication1:
run:
Área do quadrado: 225.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

### Interface

- Resolve (em partes) o problema da falta de Herança Múltipla;
- Uma classe pode implementar mais de uma interface;
- Modela um comportamento esperado;
- Na prática, uma interface assemelha-se a uma classe abstrata que contém apenas métodos abstratos.

### Exemplo

```
1 package br.com.fatecpg.heranca;
2
3 public interface Cálculos {
4     public abstract double getÁrea();
5     public abstract double getPerímetro();
6 }
7
```

```
1 package br.com.fatecpg.heranca;
2
3 import java.awt.Graphics;
4
5 public interface Desenho {
6     abstract void Desenha (Graphics g, int x, int y);
7 }
```

```
1 package br.com.fatecpg.heranca;
2
3 abstract class Forma implements Cálculos, Desenho {
4     void ImprimeÁrea() {
5         System.out.println("Área da Forma: " + this.getÁrea());
6     }
7 }
```



## Exemplo (Classe Quadrado)

```
1 package br.com.fatecpg.heranca;
2
3 import java.awt.Graphics;
4
5 class Quadrado extends Forma{
6     public int lado;
7     public void Desenha(Graphics g, int x, int y) {
8         g.drawRect(x, y, x + lado, y + lado);
9     }
10    public double getArea() {
11        return lado * lado;
12    }
13    @Override
14    void ImprimeArea() {
15        System.out.println("Área do quadrado: " + this.getArea());
16    }
17    public double getPerimetro() {
18        return lado * 4;
19    }
20 }
```

## Associação

- Vínculo que permite que objetos de uma ou mais classes se relacionem;
- É possível que um objeto convoque comportamentos e estados de outros objetos;
- As associações podem ser:
  - Unárias: Entre objetos de uma mesma classe;
  - Múltiplas: Entre mais de dois objetos.

## Associação

- Cada associação possui características de:
  - Cardinalidade ou multiplicidade - determina quantos objetos no sistema são possíveis em cada vértice da associação;
  - Navegação - se é possível para cada objeto acessar outro objeto da mesma associação.

## Exemplo de associação

- Basicamente podemos ter:
  - Agregação: quando duas ou mais classes formam outra classe. Neste caso, a ausência de uma delas não compromete a existência da classe resultante da associação.
  - Composição: também ocorre quando duas ou mais classes compõem outra classe. Neste caso, porém, a ausência de uma das classes formadoras compromete a existência da classe resultante da associação, impedindo sua utilização.

```
package RegraNegocio;

public class Aula {
    private Aluno _aluno;
    private Materia _materia;
}
```

```
package RegraNegocio;

public class Carro{
    private Chassi _chassi;
    private Motor _motor;
}
```

## Exemplo prático

```
class Data{
    public int dia;
    public int mes;
    public int ano;
    public String getData(){
        return this.dia + "/" + this.mes + "/" + this.ano;
    }
}

class Pessoa{
    public String Nome;
    public Data nascimento = new Data();
}

public class Main {
    public static void main(String[] args) {
        // TODO code application logic here
        Pessoa p1 = new Pessoa();
        p1.Nome = "Roberval";
        p1.nascimento.ano = 2000;
        p1.nascimento.mes = 1;
        p1.nascimento.dia = 1;
        System.out.println(p1.Nome + " nasceu em " + p1.nascimento.getData());
    }
}
```

Output - JavaApplication24 (run)

init:  
deps:jar:  
Compiling 1 source file to C:\Documents and Settings\user\workspace\JavaApplication24\out-classes:  
compile:  
run:  
Roberval nasceu em 1/1/2000  
BUILD SUCCESSFUL (total time: 0 seconds)

## Outro exemplo

```
class Temperatura{
    public double grausCelsius;
    public double getCel(){return this.grausCelsius;}
    public double getFar(){return (9 / 5) * this.grausCelsius + 32;}
}

class CâmaraTérmica{
    public Temperatura minTemp = new Temperatura();
    public Temperatura maxTemp = new Temperatura();
    public CâmaraTérmica(int minTemp, int maxTemp){
        this.minTemp.grausCelsius = minTemp;
        this.maxTemp.grausCelsius = maxTemp;
    }
}

public class Main {
    public static void main(String[] args) {
        // TODO code application logic here
        CâmaraTérmica c12 = new CâmaraTérmica(56, 120);
        System.out.println("A temperatura da câmara térmica deve ficar entre "
            + c12.minTemp.getCel() + "°C e " + c12.maxTemp.getCel() + "°C");
        System.out.println("A temperatura da câmara térmica deve ficar entre "
            + c12.minTemp.getFar() + "°F e " + c12.maxTemp.getFar() + "°F");
    }
}
```

Output - JavaApplication24 (run)

init:  
deps:jar:  
Compiling 1 source file to C:\Documents and Settings\user\workspace\JavaApplication24\out-classes:  
compile:  
run:  
A temperatura da câmara térmica deve ficar entre 56.0°C e 120.0°C  
A temperatura da câmara térmica deve ficar entre 88.0°F e 182.0°F  
BUILD SUCCESSFUL (total time: 0 seconds)