ENGENHARIA DE SOFTWARE

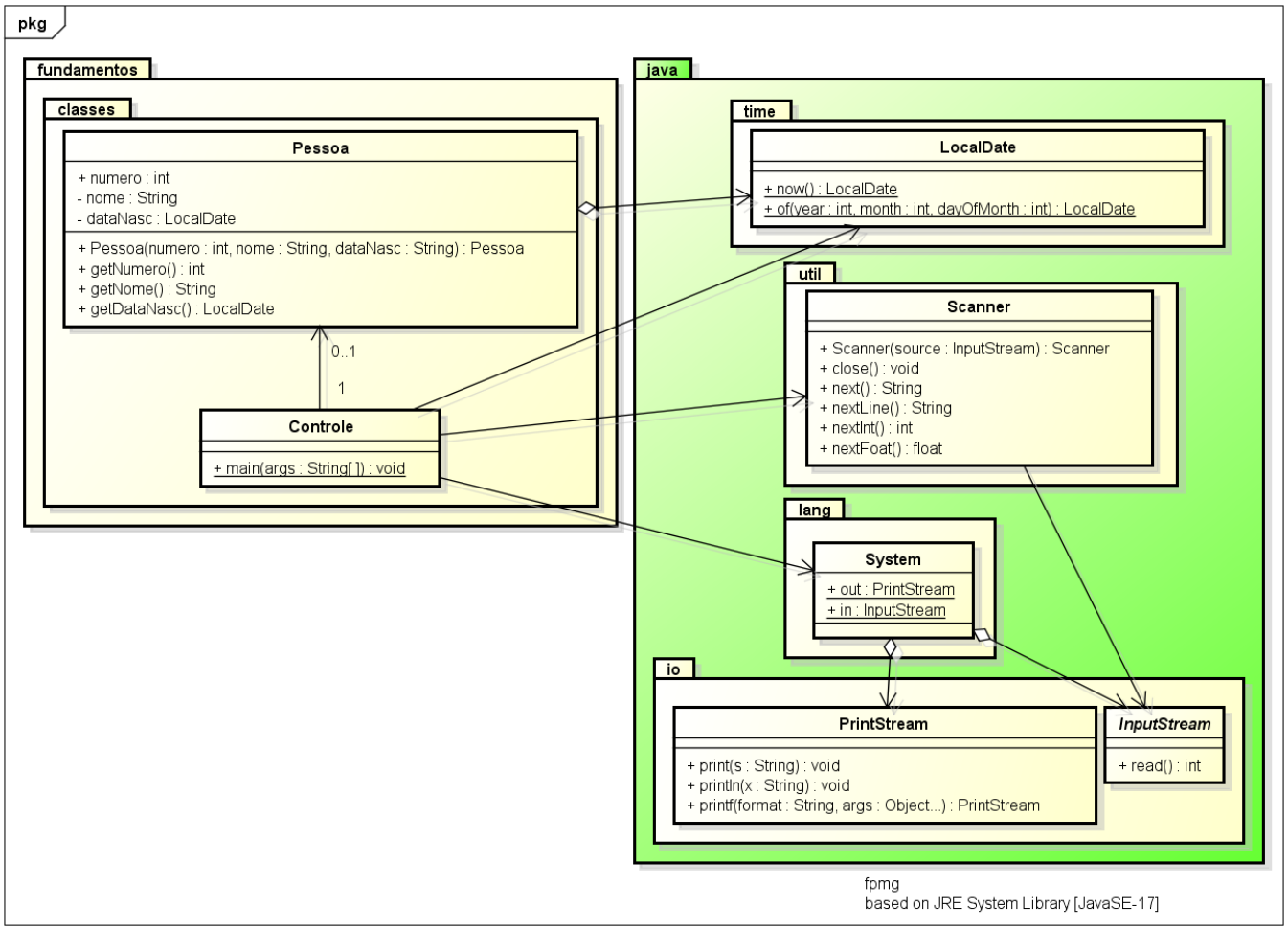
Exemplos de Diagrama de classes, Diagrama de sequência e implementação em Java

Francisco Pedro Morais Gonçalves

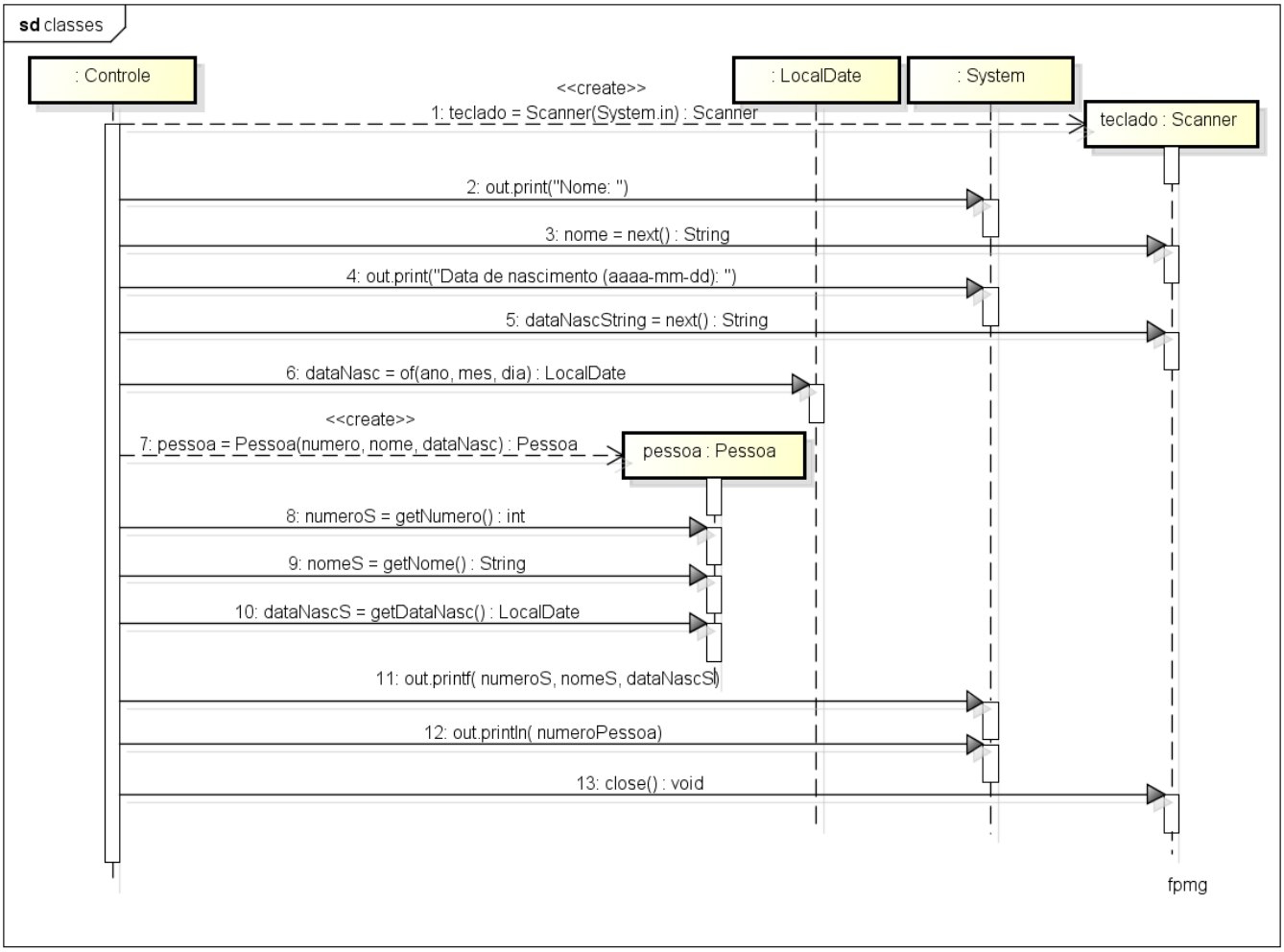
Luanda, Abril de 2024

1. CLASSES, ATRIBUTOS E MÉTODOS

1.1. Diagrama de Classes (com algumas classes da JRE System Library)



1.2. Diagrama de Sequência



1.3. Código Fonte

1.3.1. Classe Controle

```
1   /**
2    * FUNDAMENTOS DE ENGENHARIA DE SOFTWARE
3    * - Classes
4    * - Atributos: visibilidade, nome, e tipo
5    * - Métodos: visibilidade, nome, parâmetros e tipo-de-retorno
6    *
7    * @author Francisco Pedro Morais Gonçalves
8    * https://github.com/fgonca
9    */
10
11  package fundamentos.classes; // pacote
12
13  import java.time.LocalDate; // reutilização da classe LocalDate do pacote java.time
14  import java.util.Scanner; // reutilização da classe Scanner do pacote java.util
15
16  //Declaração de classe
17  public class Controle
```

```
18  {
19        // declaração do método main
20        public static void main(String[] args)
21        {
22              /*
23               * 1- criar um teclado:
24               * (invocação dum método constructor da classe Scanner com o parâmetro System.in,
25               * seguida da atribuição duma referência do resultado - um objecto de tipo
26               *  Scanner - à variável teclado)
27               */
28              Scanner teclado = new Scanner(System.in);
29
30              /*
31               * 2- pedir ao usuário o nome:
32               * (invocação do método print do objecto referenciado pelo atributo out da classe
33               * System, com o parâmetro "Digite o nome: ")
34               */
35              System.out.print("Nome: ");
36
37              /*
38               * 3- ler o nome:
39               * (invocação do método next do objecto referenciado pela variável teclado,
40               * seguida da atribuição duma referência do resultado da invocação - um objecto
41               * de tipo String - à variável nome)
42               */
43              String nome = teclado.next();
44
45              // 4- pedir ao usuário a data de nascimento
46              System.out.print("Data de nascimento (aaaa-mm-dd): ");
47
48              // 5- ler a data de nascimento
49              String dataNascString = teclado.next();
50
51              // tranformar a data de nascimento
52              String[] dataNascStringArray = dataNascString.split("-"); //separar dia, mês e ano
53              int ano = Integer.parseInt(dataNascStringArray[0]); // converter String para int
54              int mes = Integer.parseInt(dataNascStringArray[1]); // converter String para int
55              int dia = Integer.parseInt(dataNascStringArray[2]); // converter String para int
56
57              /*
58               * 6- definir a data de nascimento:
59               * (invocação o método of da classe LocalDate, com os parâmetros ano, mes e dia,
60               * seguida da atribuição duma referência do resultado da invocação - um objecto
61               * de tipo LocalDate - à variável dataNasc)
62               */
63              LocalDate dataNasc = LocalDate.of(ano, mes, dia);
64
65              /*
66               * definir o número:
67               * (atribuição dum número inteiro à variável numero)
68               */
69              int numero = 1;
70
71              /*
72               * 7- criar uma pessoa:
73               * (invocação dum método constructor da classe Pessoa com os parâmetros numero,
74               * nome e dataNasc, seguida da atribuição duma referência do resultado da
75               * invocação - um objecto de tipo Pessoa - à variável pessoa)
76               */
```

```
77              Pessoa pessoa = new Pessoa(numero, nome, dataNasc);
78
79              /*
80               * 8- obter o número da pessoa:
81               * invocação do método getNumero do objecto referenciado pela variável pessoa,
82               * seguida da atribuição do resultado da invocação - um número inteiro - à
83               * variável numeroS
84               */
85              int numeroS = pessoa.getNumero();
86
87              /*
88               * 9- obter o nome da pessoa:
89               * invocação do método getNome do objecto referenciado pela variável pessoa,
90               * seguida da atribuição do resultado da invocação - um objecto
91               * de tipo String - à variável nomeS
92               */
93              String nomeS = pessoa.getNome();
94
95              // 10- obter o nome da pessoa:
96              LocalDate dataNascS = pessoa.getDataNasc();
97
98              // 11- apresentar os dados da pessoa
99              System.out.printf("Número: %d, %s, %s\n", numeroS, nomeS, dataNascS);
100
101             /*
102              * obter o número da pessoa:
103              * (atribuição do valor do atributo numero do objecto referenciado pela variável
104              * pessoa - um número inteiro - à variável numeroPessoa)
105              */
106             int numeroPessoa = pessoa.numero;
107
108             // 12- apresentar o número da pessoa
109             System.out.println("Número: " + numeroPessoa);
110
111             /*
112              * 13- fechar o teclado:
113              * (invocação do método close do objecto referenciado pela variável teclado)
114              */
115             teclado.close();
116         }
117 }
```

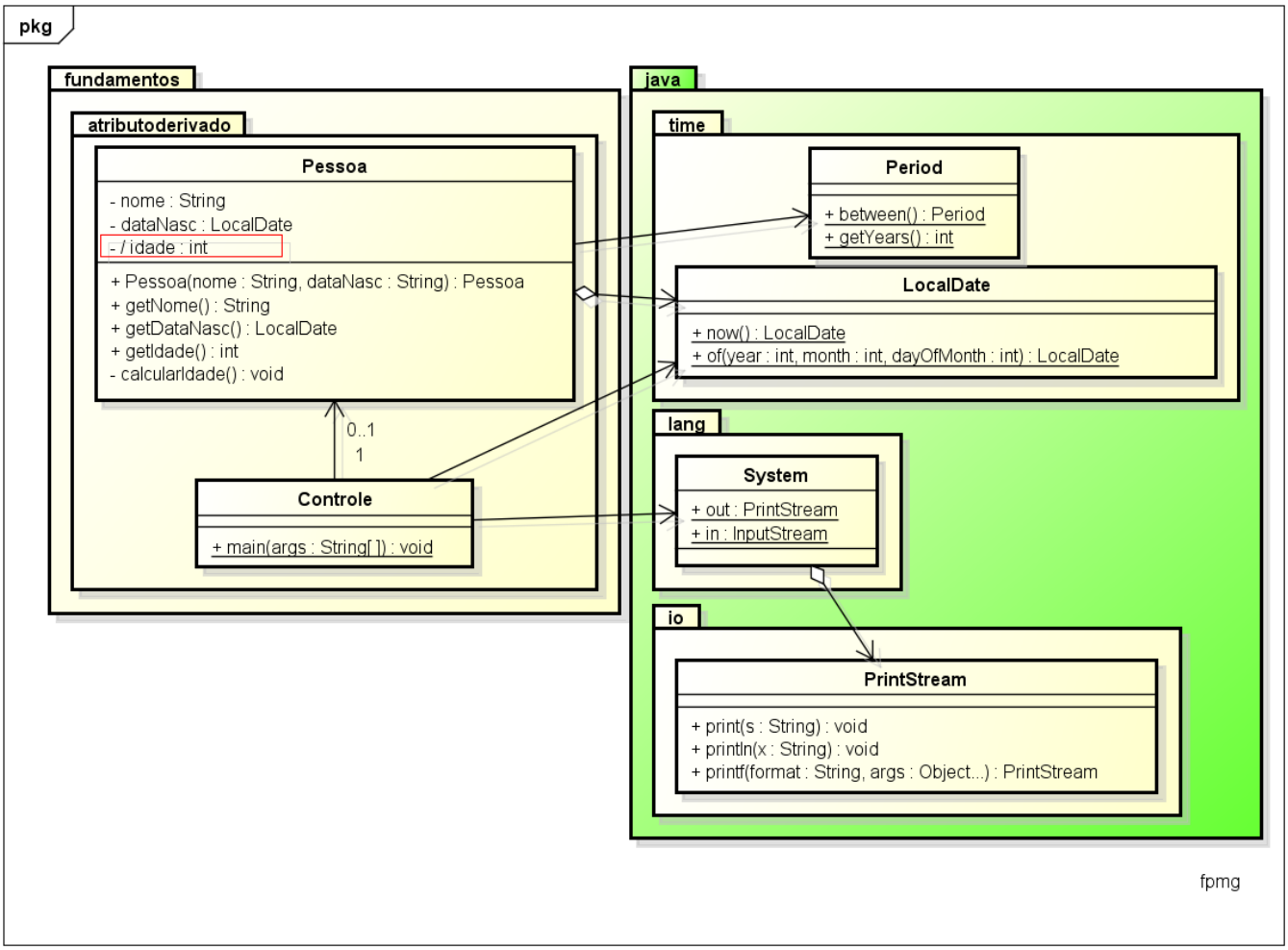
1.3.2. Classe Pessoa

```
1   /**
2    * FUNDAMENTOS DE ENGENHARIA DE SOFTWARE
3    * - Classes
4    * - Atributos: visibilidade, nome, e tipo
5    * - Métodos: visibilidade, nome, parâmetros e tipo-de-retorno
6    *
7    * @author Francisco Pedro Morais Gonçalves
8    * https://github.com/fgonca
9    */
10
11  package fundamentos.classes; // pacote
12
13  import java.time.LocalDate; // reutilização da classe LocalDate do pacote java.time
14
15  // Declaração de classe
```

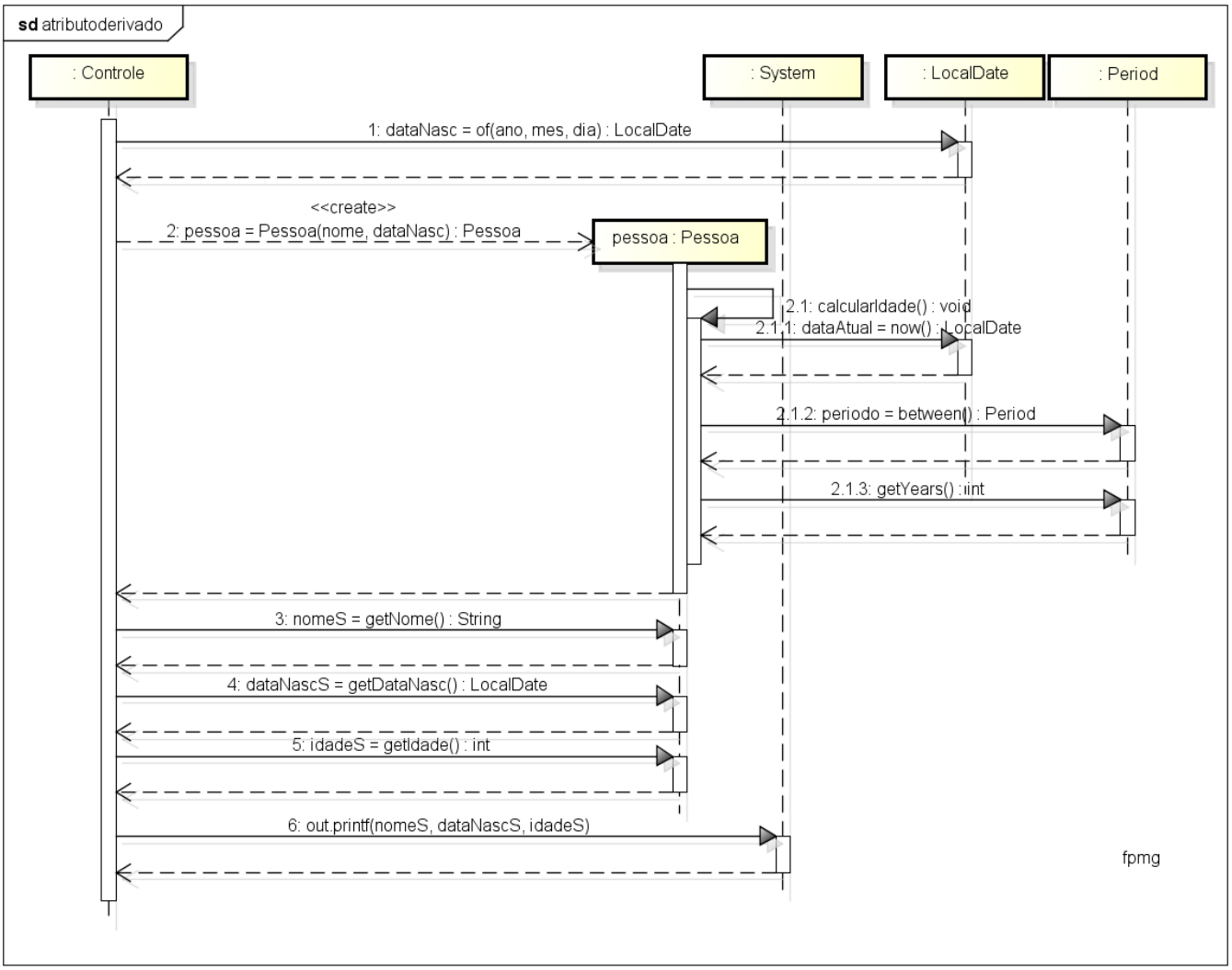
```
16  class Pessoa
17  {
18        public int numero; // declaracão de atributo
19        private String nome; // declaracão de atributo
20        private    LocalDate dataNasc; // declaracão de atributo
21
22        // declaração de método construtor
23        public Pessoa(int numero, String nome, LocalDate dataNasc)
24        {
25              this.numero = numero;  // atribuir o valor do parâmetro numero ao atributo numero
26              this.nome = nome; // atribuir o valor do parâmetro nome ao atributo nome
27              this.dataNasc = dataNasc; // ...
28        }
29
30        // declaração de método
31        public int getNumero() {
32              return numero;
33        }
34
35        // declaração de método
36        public String getNome() {
37              return nome;
38        }
39
40        // declaração de método
41        public LocalDate getDataNasc() {
42              return dataNasc;
43        }
44  }
```

2. ATRIBUTO DERIVADO

2.1. Diagrama de Classes (com algumas classes da JRE System Library)



2.2. Diagrama de Sequência



2.3. Código Fonte

2.3.1. Classe Controle

```
1   /**
2    * FUNDAMENTOS DE ENGENHARIA DE SOFTWARE
3    * - Atributo Derivado
4    *
5    * @author Francisco Pedro Morais Gonçalves
6    * https://github.com/fgonca
7    */
8
9
10  package fundamentos.atributoderivado;
11
12  import java.time.LocalDate;
13
14  public class Controle
15  {
```

```
16
17        public static void main(String[] args)
18        {
19
20              // 1- definir a data de nascimento
21              LocalDate dataNasc = LocalDate.of(2001, 01, 01);
22
23              // 2- criar uma pessoa
24              Pessoa pessoa = new Pessoa("Abel", dataNasc);
25
26              // 3- obter o nome da pessoa
27              String nomeS = pessoa.getNome();
28
29              // 4- obter a data de nascimento da pessoa
30              LocalDate dataNascS = pessoa.getDataNasc();
31
32              // 5- obter a data de nascimento da pessoa
33              int idadeS = pessoa.getIdade();
34
35              // 6- apresentar os dados da pessoa
36              System.out.printf("%s nasceu em %s; %d anos de idade.", nomeS, dataNascS, idadeS);
37        }
38  }
```

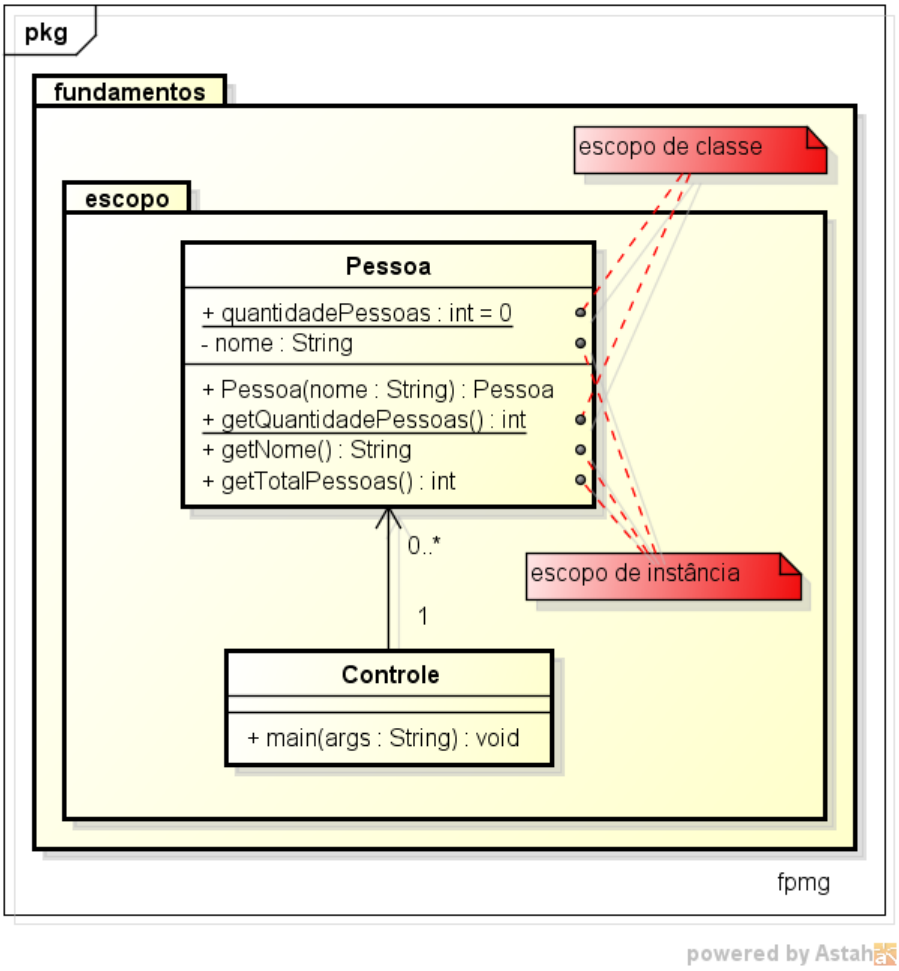
2.3.2. Classe pessoa

```
1   /**
2    * FUNDAMENTOS DE ENGENHARIA DE SOFTWARE
3    * - Atributo Derivado
4    *
5    * @author Francisco Pedro Morais Gonçalves
6    * https://github.com/fgonca
7    */
8
9   package fundamentos.atributoderivado;
10
11  import java.time.LocalDate;
12  import java.time.Period;
13
14  public class Pessoa
15  {
16        private String nome;
17        private   LocalDate dataNasc;
18        private int idade; // atributo derivado
19
20        // método construtor
21        public Pessoa(String nome, LocalDate dataNasc)
22        {
23              this.nome = nome;
24              this.dataNasc = dataNasc;
25              this.calcularIdade();
26        }
27
28        public String getNome() {
29              return nome;
30        }
31
32        public LocalDate getDataNasc() {
33              return dataNasc;
```

```
34          }
35
36          public int getIdade() {
37                return idade;
38          }
39
40          // declaração do método privado que acha o valor do atributo derivado idade
41          private void calcularIdade()
42          {
43
44                // 1- determina a data actual
45                LocalDate dataAtual= LocalDate.now();
46
47                // 2- determina a período entre as datas
48                Period periodo = Period.between(this.dataNasc, dataAtual);
49
50                // 3- obtem a idade da pessoa
51                this.idade = periodo.getYears();
52          }
53    }
```

3. ESCOPO DE ATRIBUTOS E DE MÉTODOS

3.1. Diagrama de classes



3.2. Código Fonte

3.2.1. Classe controle

```
1   /**
2    * FUNDAMENTOS DE ENGENHARIA DE SOFTWARE
3    * - Escopo de atributos e de métodos
4    *
5    * @author Francisco Pedro Morais Gonçalves
6    * https://github.com/fgonca
7    */
8   package fundamentos.escopo;
9
10  public class Controle
11  {
12
13      public static void main(String[] args)
14      {
```

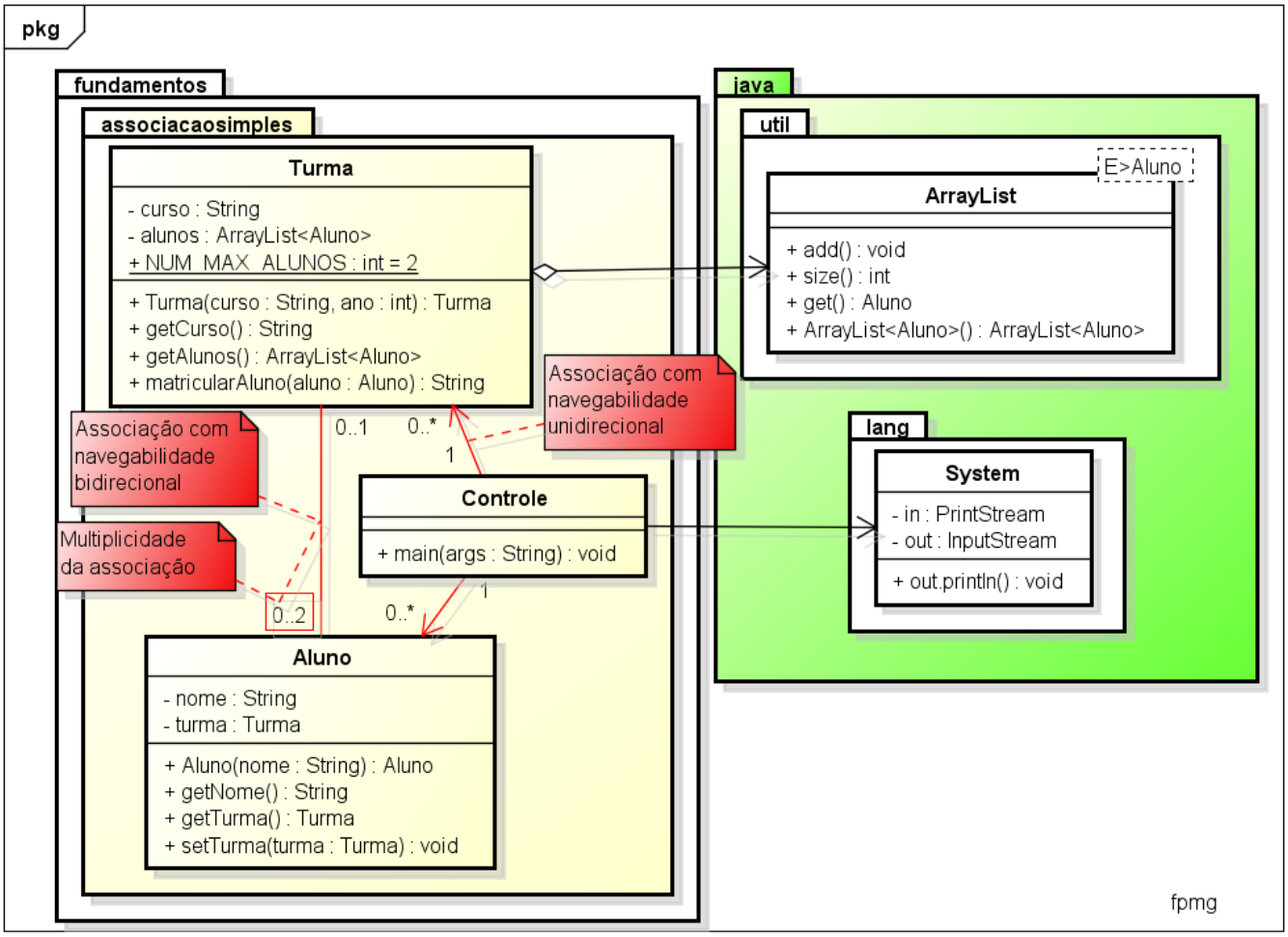
```
15              int qtdPessoasInicial = Pessoa.getQuantidadePessoas();
16              System.out.println("Quantidade inicial: " + qtdPessoasInicial);
17
18              Pessoa pessoa1 = new Pessoa("Abel");
19              System.out.println(pessoa1.getNome());
20
21              Pessoa pessoa2 = new Pessoa("Bela");
22              System.out.println(pessoa2.getNome());
23
24              int qtdPessoasFinal = Pessoa.getQuantidadePessoas();
25
26              System.out.println("Método getQuantidadePessoas: " + qtdPessoasFinal);
27              System.out.println("Método getTotalPessoas: " + pessoa2.getTotalPessoas());
28              System.out.print("Atributo quantidadePessoas: " + Pessoa.quantidadePessoas);
29      }
30  }
```

3.2.2. Classe Pessoa

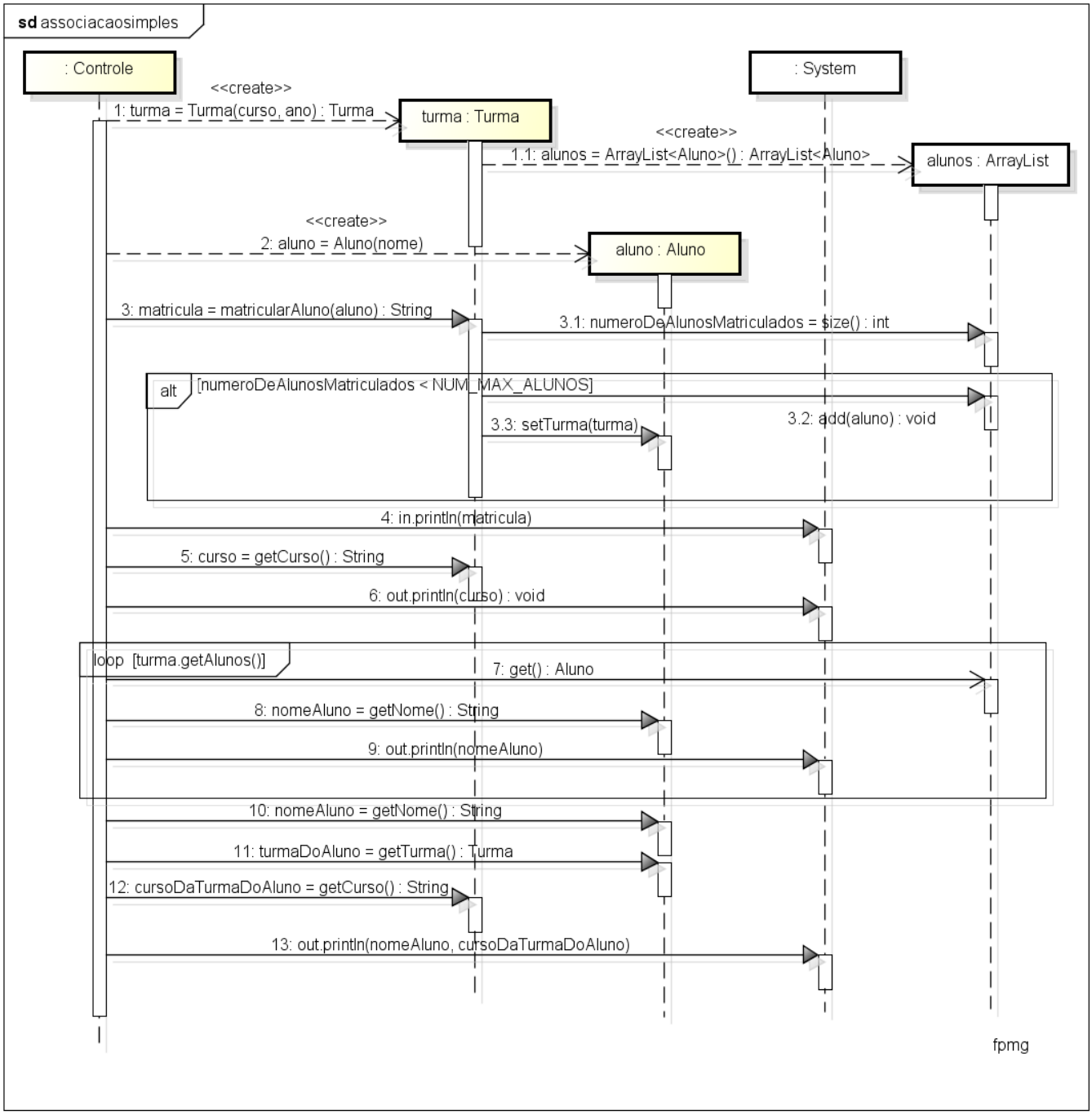
```
1   /**
2    * FUNDAMENTOS DE ENGENHARIA DE SOFTWARE
3    * - Escopo de atributos e de métodos
4    *
5    * @author Francisco Pedro Morais Gonçalves
6    * https://github.com/fgonca
7    */
8
9   package fundamentos.escopo;
10
11  public class Pessoa
12  {
13      public static int quantidadePessoas = 0; //atributo com escopo de classe (estático)
14      private String nome; // atributo com escopo de instância
15
16      public Pessoa(String nome)
17      {
18          this.nome = nome;
19          quantidadePessoas +=1; // incrementa a quantidade de pessoas
20      }
21
22      // método com escopo de classe (estático)
23      public static int getQuantidadePessoas()
24      {
25          return quantidadePessoas;
26      }
27
28      // método com escopo de instância
29      public String getNome()
30      {
31          return nome;
32      }
33
34      // método com escopo de instância
35      public int getTotalPessoas()
36      {
37          return quantidadePessoas;
38      }
39  }
```

4. ASSOCIAÇÃO SIMPLES: MULTIPLICIDADE E NAVEGABILIDADE

4.1. Diagrama de classes



4.2. Diagrama de sequência



4.3. Código Fonte

4.3.1. Classe Controle

```
1   /**
2    * FUNDAMENTOS DE ENGENHARIA DE SOFTWARE
3    * - Associação Simples
4    *         - Multiplicidade
5    *         - Navegabilidade
6    *
7    * @author Francisco Pedro Morais Gonçalves
8    * https://github.com/fgonca
```

```
 */

package fundamentos.associacaosimples;

public class Controle
{
     public static void main(String[] args)
     {
          /*
           * 1- criar uma turma:
           * o valor da variável turma é uma referência a um objecto do tipo Turma
           */
          Turma turma = new Turma("Matemática");

          /*
           * 2- criar um aluno:
           * o valor da variável aluno é uma referência a um objecto do tipo Aluno
           */
          Aluno aluno = new Aluno("Abel");

          // 3- matricular aluno
          String matricula = turma.matricularAluno(aluno);

          // 4- apresentar resultado da matrícula
          System.out.println(matricula);

          // criar mais um aluno, matriculá-lo e apresentar o resultado da matrícula
          Aluno aluno2 = new Aluno("Bela");
          System.out.println(turma.matricularAluno(aluno2));

          // 5- obter o curso
          String curso = turma.getCurso();

          // 6- apresentar o curso da turma
          System.out.println("\nCurso: " + curso);

          // 7- buscar cada aluno...
          for (Aluno alunoDaTurma: turma.getAlunos())
          {
               // 8- obter o nome do aluno matriculado na turma
               String nomeAluno = alunoDaTurma.getNome();

               // 9- apresentar o nome do aluno matriculado na turma
               System.out.println("  - " + nomeAluno);
          }

          // 10- obter o nome do aluno
          String nomeAluno = aluno.getNome();

          // 11- obter a turma do aluno
          Turma turmaDoAluno = aluno.getTurma();

          // 12- obter o curso da turma do aluno
          String cursoDaTurmaDoAluno = turmaDoAluno.getCurso();

          // 13- apresentar os dados do aluno
          System.out.printf("\n%s, %s\n", nomeAluno, cursoDaTurmaDoAluno);

          // obter os dados do aluno2 e apresertá-los
```

```
68              String nomeAluno2 = aluno2.getNome();
69              Turma turmaAluno2 = aluno2.getTurma();
70              String cursoTurAluno2 = turmaAluno2.getCurso();
71              System.out.printf("%s, %s\n", nomeAluno2, cursoTurAluno2);
72
73              // tentar criar mais um aluno, matriculá-lo e apresentar o resultado da matrícula
74              Aluno aluno3 = new Aluno("Carlos");
75              System.out.println("\n" + turma.matricularAluno(aluno3));
76
77          }
78      }
```

4.3.2. Classe Turma

```
1   /**
2    * FUNDAMENTOS DE ENGENHARIA DE SOFTWARE
3    * - Associação Simples
4    *         - Multiplicidade
5    *         - Navegabilidade
6    *
7    * @author Francisco Pedro Morais Gonçalves
8    * https://github.com/fgonca
9    */
10
11  package fundamentos.associacaosimples;
12
13  import java.util.ArrayList;
14
15  public class Turma
16  {
17      private String curso;
18      private ArrayList<Aluno> alunos; // referência aos alunos da turma
19      public static final int  NUM_MAX_ALUNOS = 2; // constante
20
21      public Turma(String curso)
22      {
23          this.curso = curso;
24          this.alunos = new ArrayList<Aluno>();
25      }
26
27      // obter o curso
28      public String getCurso()
29      {
30          return curso;
31      }
32
33      // obter os alunos
34      public ArrayList<Aluno> getAlunos()
35      {
36          return alunos;
37      }
38
39      public String matricularAluno(Aluno aluno)
40      {
41          // 1- obter o número de alunos matriculados
42          int numeroDeAlunosMatriculados = alunos.size();
43
44          if(numeroDeAlunosMatriculados < NUM_MAX_ALUNOS)
```

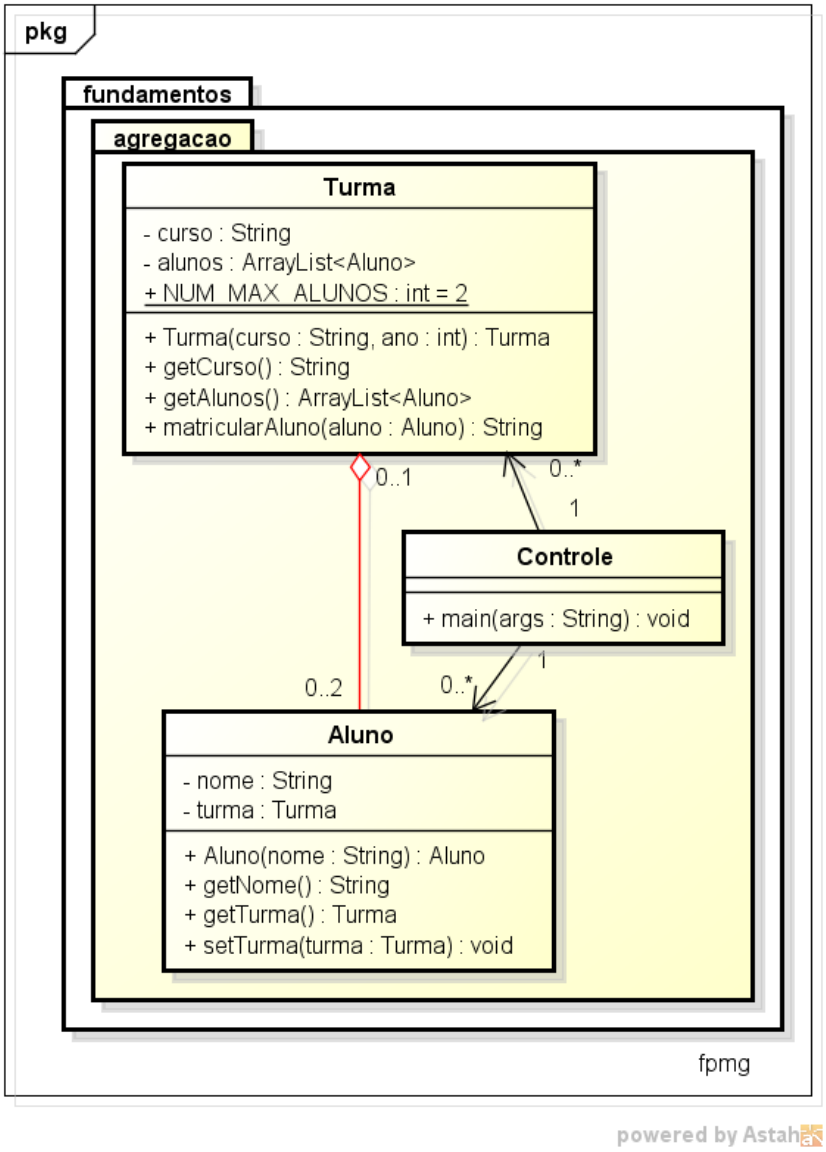
```
45              {
46                   // 2- adicionar aluno
47                   alunos.add(aluno);
48
49                   // 3- definir a turma do aluno
50                   aluno.setTurma(this);
51
52                   return "Aluno matriculado!";
53              }
54
55              return "Aluno não matriculado! Turma cheia";
56         }
57    }
```

4.3.3. Classe Aluno

```
1    /**
2     * FUNDAMENTOS DE ENGENHARIA DE SOFTWARE
3     * - Associação Simples
4     *         - Multiplicidade
5     *         - Navegabilidade
6     *
7     * @author Francisco Pedro Morais Gonçalves
8     * https://github.com/fgonca
9     */
10
11   package fundamentos.associacaosimples;
12
13   class Aluno
14   {
15        private String nome;
16        private Turma turma; // referência à turma do aluno
17
18        // criar Aluno: método construtor
19        public Aluno(String nome)
20        {
21             this.nome = nome;
22        }
23
24        // obter o nome
25        public String getNome()
26        {
27             return nome;
28        }
29
30        // obter a turma
31        public Turma getTurma()
32        {
33             return turma;
34        }
35
36        // definir a turma
37        public void setTurma(Turma turma)
38        {
39             this.turma = turma;
40        }
41   }
```

5. AGREGAÇÃO

5.1. Diagrama de Classes

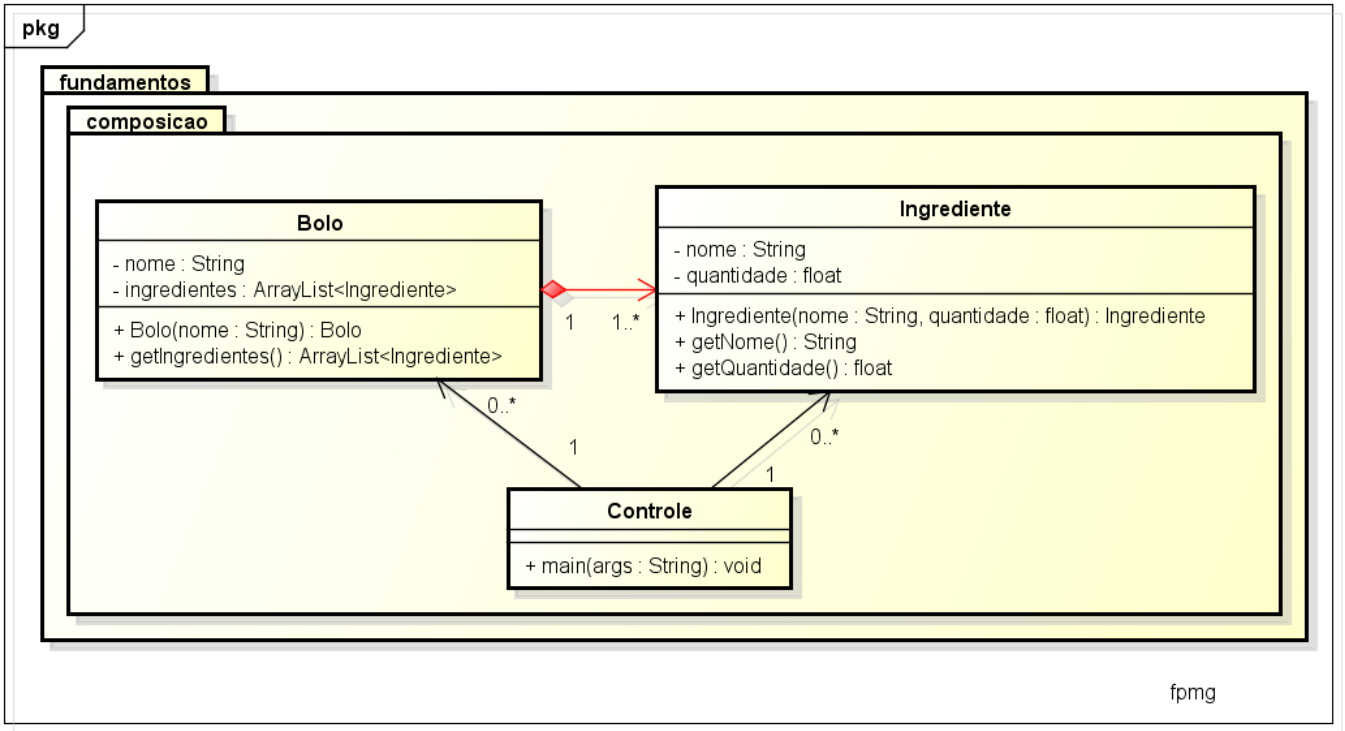


5.2. Diagrama de Sequência e Código Fonte

Idênticos aos da Associação Simples

6. COMPOSICAO

6.1. Diagrama de Classes



6.2. Código Fonte

6.2.1. Classe Controle

```
1   /**
2    * FUNDAMENTOS DE ENGENHARIA DE SOFTWARE
3    * - Composição
4    *
5    * @author Francisco Pedro Morais Gonçalves
6    * https://github.com/fgonca
7    */
8
9   package fundamentos.composicao;
10
11  public class Controle
12  {
13
14      public static void main(String[] args)
15      {
16
17          Bolo bolo= new Bolo ("Bolo de leite");
18          System.out.println("Pediu um "+bolo.getNome());
19          System.out.printf("Os ingredientes do %s são:\n", bolo.getNome());
20          for(int i = 0; i < bolo.getIngredientes().size(); i++)
```

```
21              {
22                   String nome = bolo.getIngredientes().get(i).getNome();
23                   float quantidade = bolo.getIngredientes().get(i).getQuantidade();
24                   System.out.printf(i+1+"- %s, %.2f;\n", nome, quantidade);
25              }
26         }
27    }
```

6.2.2. Classe Bolo

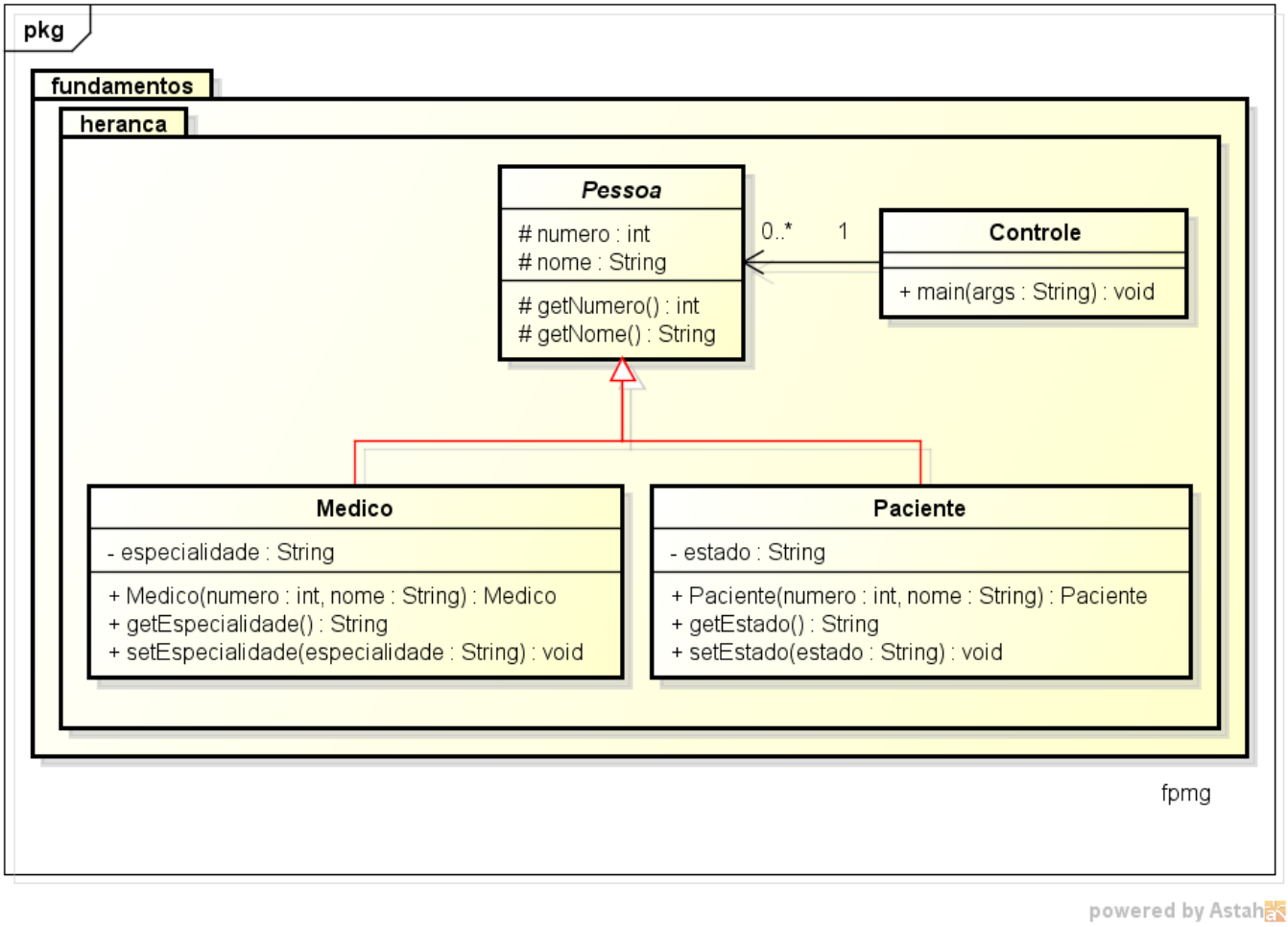
```
1    /**
2     * FUNDAMENTOS DE ENGENHARIA DE SOFTWARE
3     * - Composição
4     *
5     * @author Francisco Pedro Morais Gonçalves
6     * https://github.com/fgonca
7     */
8
9    package fundamentos.composicao;
10
11   import java.util.ArrayList;
12
13   public class Bolo
14   {
15        private String nome;
16        private ArrayList<Ingrediente> ingredientes= new ArrayList<Ingrediente>();
17
18        public Bolo(String nome)
19        {
20             this.nome = nome;
21             if(nome=="Bolo de leite" || nome=="bolo de leite")
22             {
23                  // os ingredientes são criados dentro da classe Bolo
24                  Ingrediente trigo = new Ingrediente("trigo", 1);
25                  Ingrediente leite = new Ingrediente("leite", 0.5f);
26                  Ingrediente fermento = new Ingrediente("fermento", 0.1f);
27                  Ingrediente ovo = new Ingrediente("ovo", 3);
28                  Ingrediente acucar = new Ingrediente("açúcar", 0.5f);
29
30                  this.ingredientes.add(trigo);
31                  this.ingredientes.add(leite);
32                  this.ingredientes.add(fermento);
33                  this.ingredientes.add(ovo);
34                  this.ingredientes.add(acucar);
35             }
36        }
37
38        public String getNome()
39        {
40             return nome;
41        }
42
43        public ArrayList<Ingrediente> getIngredientes()
44        {
45             return ingredientes;
46        }
47   }
```

6.2.3. Classe Ingrediente

```java
/**
 * FUNDAMENTOS DE ENGENHARIA DE SOFTWARE
 * - Composição
 *
 * @author Francisco Pedro Morais Gonçalves
 * https://github.com/fgonca
 */

package fundamentos.composicao;

public class Ingrediente
{
	private String nome;
	private float quantidade;

	public Ingrediente(String nome, float quantidade)
	{
		this.nome = nome;
		this.quantidade = quantidade;
	}

	public String getNome()
	{
		return nome;
	}

	public float getQuantidade()
	{
		return quantidade;
	}
}
```

7. HERANÇA

7.1. Diagrama de Classes



7.2. Código Fonte

7.2.1. Classe Controle

```
1   /**
2    * FUNDAMENTOS DE ENGENHARIA DE SOFTWARE
3    * - Herança
4    *
5    * @author Francisco Pedro Morais Gonçalves
6    * https://github.com/fgonca
7    */
8
9   package fundamentos.heranca;
10
11  public class Controle
12  {
13
```

```
14        public static void main(String[] args)
15        {
16
17              // instanciar paciente
18              Paciente paciente = new Paciente(1, "Abel");
19              paciente.setEstado("bem");
20
21              // obter dados do paciente
22              int numeroP = paciente.getNumero(); // utilização de método herdado
23              String nomeP = paciente.getNome(); // utilização de método herdado
24              String estadoP= paciente.getEstado();
25
26              // exibir dados do paciente
27              String formularioP = "O(A) paciente %d, %s, está %s.\n";
28              System.out.printf(formularioP, numeroP, nomeP, estadoP);
29
30              // instanciar médico
31              Medico medico = new Medico(2, "Bela");
32              medico.setEspecialidade("Fisioterapia");
33
34              // obter dados do médico
35              int numeroM = medico.getNumero(); // utilização de método herdado
36              String nomeM = medico.getNome(); // utilização de método herdado
37              String estadoM=  medico.getEspecialidade().toLowerCase();
38
39              // exibir dados do médico
40              String formularioM = "O(A) médico(a) %d, %s, é especialista em %s.\n";
41              System.out.printf(formularioM, numeroM, nomeM, estadoM);
42        }
43  }
```

7.2.2. Classe Pessoa

```
1   /**
2    * FUNDAMENTOS DE ENGENHARIA DE SOFTWARE
3    * - Herança
4    *
5    * @author Francisco Pedro Morais Gonçalves
6    * https://github.com/fgonca
7    */
8
9   package fundamentos.heranca;
10
11  //Superclasse abstracta
12  public abstract class Pessoa
13  {
14        protected int numero;
15        protected String nome;
16
17        protected int getNumero()
18        {
19              return numero;
20        }
21
22        protected String getNome()
23        {
24              return nome;
25        }
26  }
```

7.2.3. Classe Médico

```java
/**
 * FUNDAMENTOS DE ENGENHARIA DE SOFTWARE
 * - Herança
 *
 * @author Francisco Pedro Morais Gonçalves
 * https://github.com/fgonca
 */

package fundamentos.heranca;

//Declaração de subclasse
public class Medico extends Pessoa
{
	private String especialidade;

	public Medico(int numero, String nome)
	{
		this.numero = numero;
		this.nome = nome;
	}

	public String getEspecialidade()
	{
		return especialidade;
	}

	public void setEspecialidade(String especialidade)
	{
		this.especialidade = especialidade;
	}
}
```

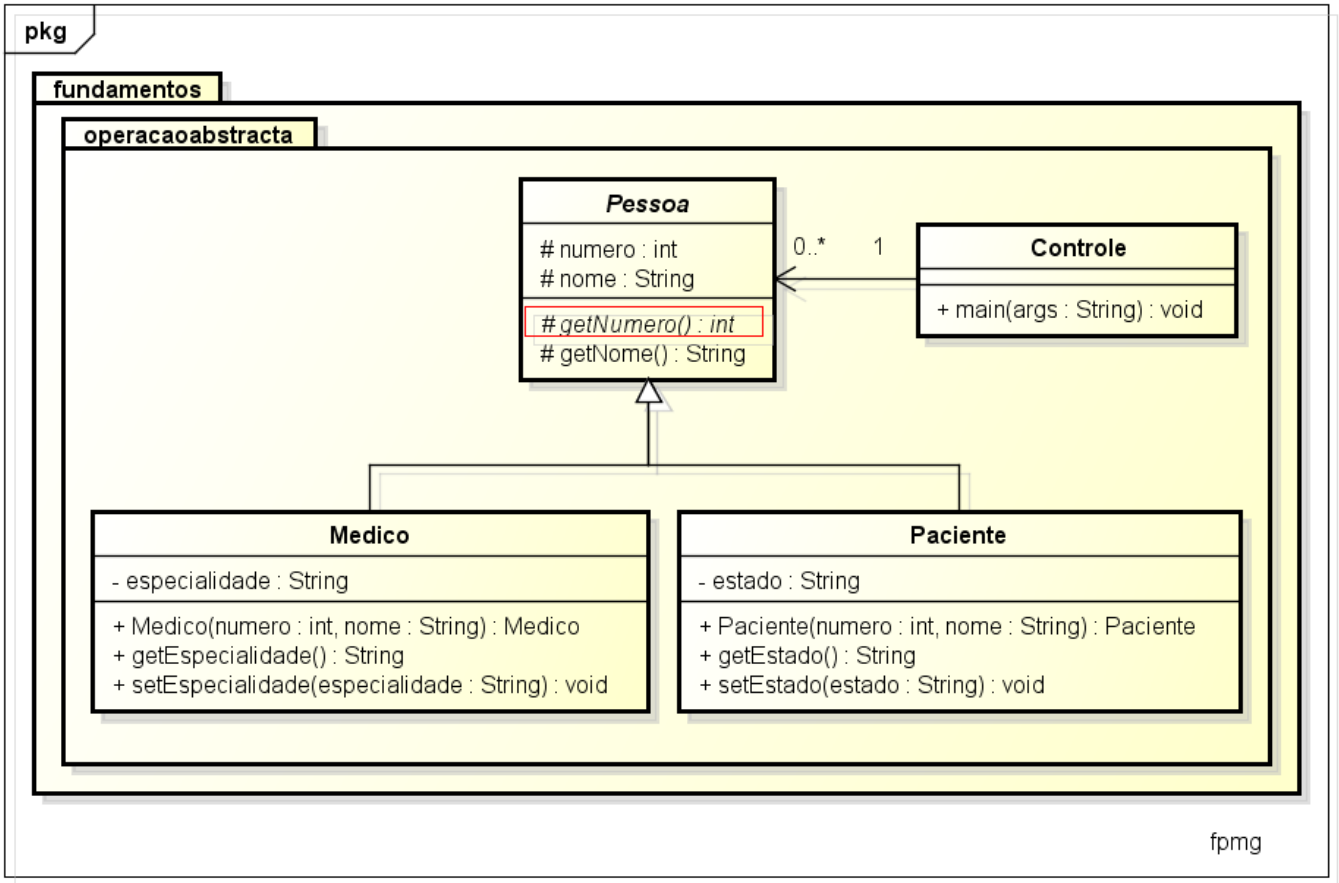
7.2.4. Classe Paciente

```java
/**
 * FUNDAMENTOS DE ENGENHARIA DE SOFTWARE
 * - Herança
 *
 * @author Francisco Pedro Morais Gonçalves
 * https://github.com/fgonca
 */

package fundamentos.heranca;

//Declaração de subclasse
public class Paciente extends Pessoa
{
	private String estado;

	public Paciente(int numero, String nome)
	{
		this.numero = numero;
		this.nome = nome;
	}
```

```
21
22          public String getEstado()
23          {
24               return estado;
25          }
26
27          public void setEstado(String estado)
28          {
29               this.estado = estado;
30          }
31     }
```

8. OPERAÇÃO ABSTRACTA

8.1. Diagrama de Classes



8.2. Código Fonte

8.2.1. Classe Controle

```
1   /**
2    * FUNDAMENTOS DE ENGENHARIA DE SOFTWARE
3    * - Operação Abstracta
4    *
5    * @author Francisco Pedro Morais Gonçalves
6    * https://github.com/fgonca
7    */
8
9   package fundamentos.operacaoabstrata;
10
11  public class Controle
12  {
13
14      public static void main(String[] args)
15      {
```

```
16
17              // instanciar paciente
18              Paciente paciente = new Paciente(1, "Abel");
19              paciente.setEstado("bem");
20
21              // obter dados do paciente
22              int numeroP = paciente.getNumero();
23              String nomeP = paciente.getNome();
24              String estadoP= paciente.getEstado();
25
26              // exibir dados do paciente
27              String formularioP = "O(A) paciente %d, %s, está %s.\n";
28              System.out.printf(formularioP, numeroP, nomeP, estadoP);
29
30              // instanciar médico
31              Medico medico = new Medico(2, "Bela");
32              medico.setEspecialidade("Fisioterapia");
33
34              // obter dados do médico
35              int numeroM = medico.getNumero();
36              String nomeM = medico.getNome();
37              String estadoM=  medico.getEspecialidade().toLowerCase();
38
39              // exibir dados do médico
40              String formularioM = "O(A) médico(a) %d, %s, é especialista em %s.\n";
41              System.out.printf(formularioM, numeroM, nomeM, estadoM);
42      }
43  }
```

8.2.2. Classe Pessoa

```
1   /**
2    * FUNDAMENTOS DE ENGENHARIA DE SOFTWARE
3    * - Operação Abstracta
4    *
5    * @author Francisco Pedro Morais Gonçalves
6    * https://github.com/fgonca
7    */
8
9   package fundamentos.operacaoabstrata;
10
11  //Declaração de superclasse abstracta
12  public abstract class Pessoa
13  {
14      protected int numero;
15      protected String nome;
16
17      // declaração de operação abstracta (método abstracto)
18      protected abstract int getNumero();
19
20      protected String getNome()
21      {
22          return nome;
23      }
24  }
```

8.2.3. Classe Medico

```
1   /**
2    * FUNDAMENTOS DE ENGENHARIA DE SOFTWARE
```

```
 * - Operação Abstracta
 *
 * @author Francisco Pedro Morais Gonçalves
 * https://github.com/fgonca
 */

package fundamentos.operacaoabstrata;

//declaração de subclasse
public class Medico extends Pessoa
{
	private String especialidade;

	public Medico(int numero, String nome)
	{
		this.numero = numero;
		this.nome = nome;
	}

	public String getEspecialidade()
	{
		return especialidade;
	}

	public void setEspecialidade(String especialidade)
	{
		this.especialidade = especialidade;
	}

	// implemetação do método abstracto declarado pela superclasse
	protected int getNumero()
	{
		return numero;
	}
}
```

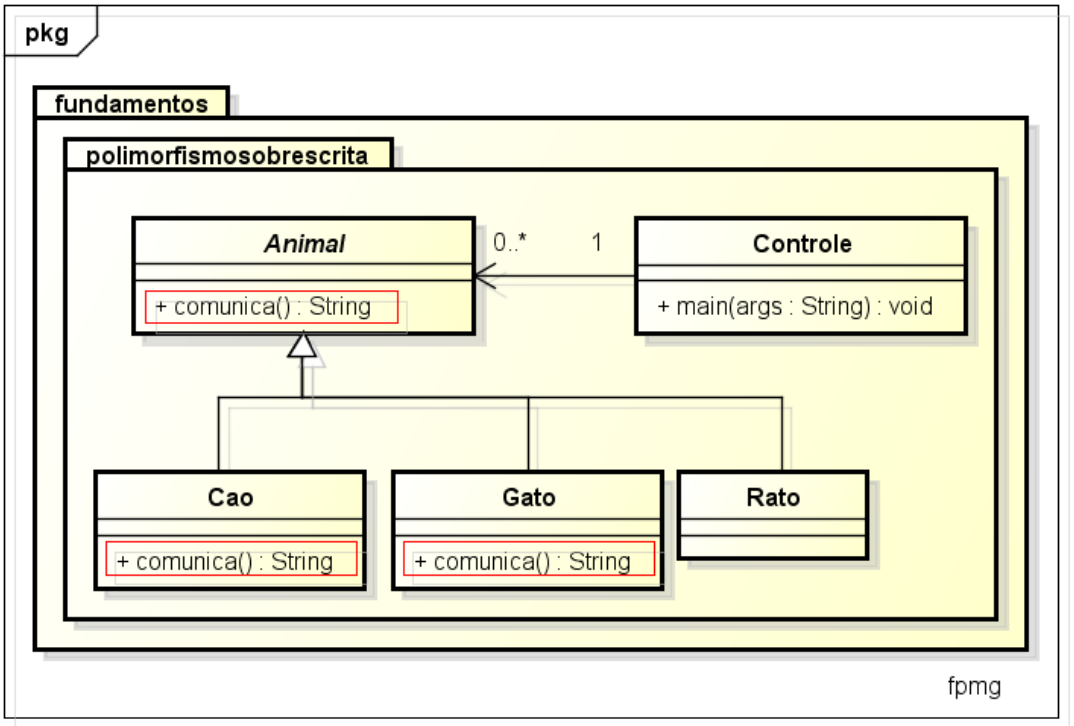
8.2.4. Classe Paciente

```
/**
 * FUNDAMENTOS DE ENGENHARIA DE SOFTWARE
 * - Operação Abstracta
 *
 * @author Francisco Pedro Morais Gonçalves
 * https://github.com/fgonca
 */

package fundamentos.operacaoabstrata;

//declaração de subclasse
public class Paciente extends Pessoa
{
	private String estado;

	public Paciente(int numero, String nome)
	{
		this.numero = numero;
		this.nome = nome;
	}

```

```
22          public String getEstado()
23          {
24               return estado;
25          }
26
27          public void setEstado(String estado)
28          {
29               this.estado = estado;
30          }
31
32          // implemetação do método abstracto declarado pela superclasse
33          public int getNumero()
34          {
35               return numero;
36          }
37     }
```

9. POLIMORFISMO E SOBRESCRITA

9.1. Diagrama de Classes



9.2. Código Fonte

9.2.1. Classe Controle

```
1   /**
2    * FUNDAMENTOS DE ENGENHARIA DE SOFTWARE
3    * - Polimorfismo
4    * - Sobrescrita
5    * @author Francisco Pedro Morais Gonçalves
6    * https://github.com/fgonca
7    */
8
9   package fundamentos.polimorfismosobrescrita;
10
11  public class Controle
12  {
13       public static void main(String[] args)
14       {
15            Cao cao = new Cao();
16            System.out.println("Cão "+cao.comunicar());;
17
18            Gato gato = new Gato();
19            System.out.println("Gato "+gato.comunicar());
20
```

```
21              Rato rato = new Rato();
22              System.out.println("Rato "+rato.comunicar());
23          }
24      }
```

9.2.2. Classe Animal

```
1   package fundamentos.polimorfismosobrescrita;
2
3   // Superclasse
4   public abstract class Animal
5   {
6       protected String comunicar()
7       {
8           return "Emite um som"; // uma das formas de comunicar
9       }
10  }
```

9.2.3. Classe Cao

```
1   package fundamentos.polimorfismosobrescrita;
2
3   public class Cao extends Animal
4   {
5       /*
6        * Sobrescreve o método comunica da superclasse
7        */
8       @Override // anotação de sobrescrita
9       public String comunicar()
10      {
11          return "ladra"; // uma das formas de comunicar
12      }
13  }
14
```

9.2.4. Classe Gato

```
1   package fundamentos.polimorfismosobrescrita;
2
3   public class Gato extends Animal
4   {
5       /*
6        * Sobrescreve o método comunica da superclasse
7        */
8       @Override // anotação de sobrescrita
9       public String comunicar()
10      {
11          return "mia"; // uma das formas de comunicar
12      }
13  }
```

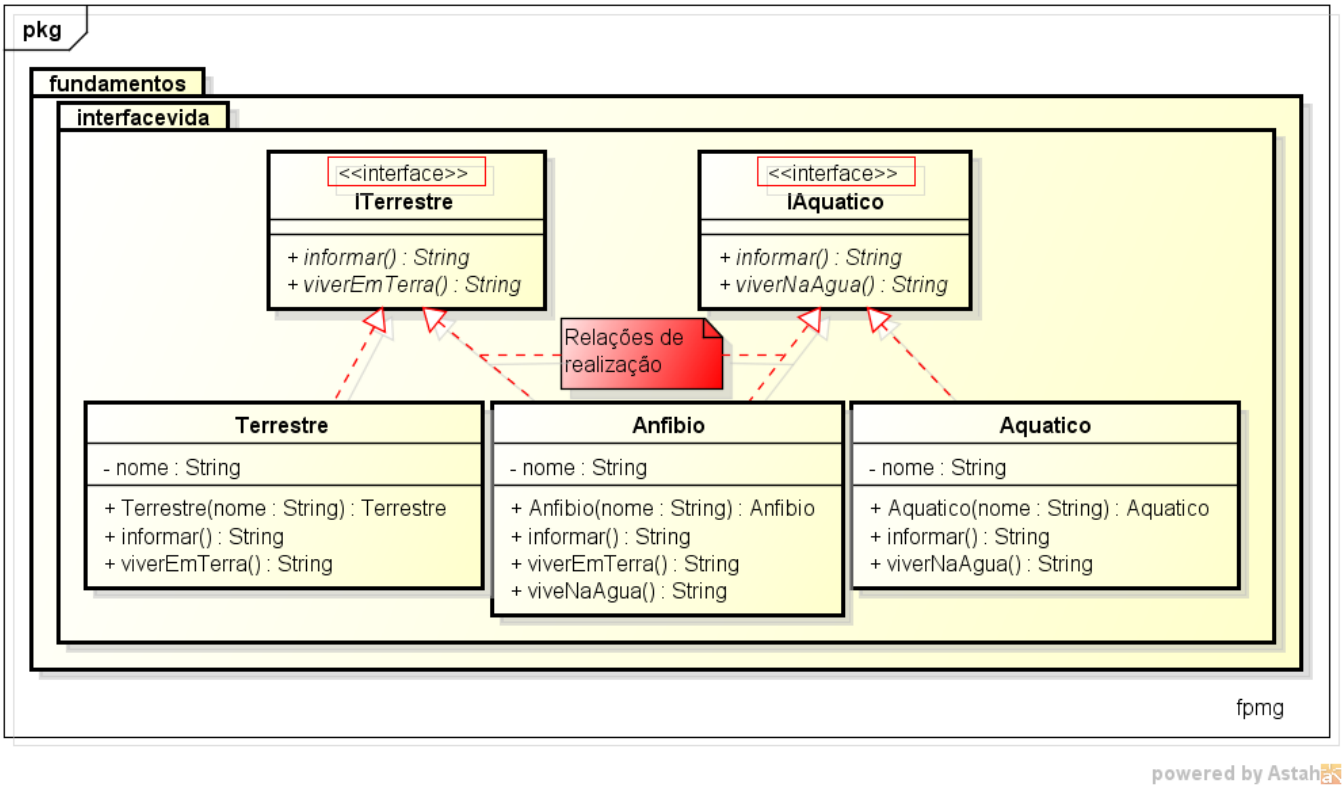
9.2.5. Classe Rato

```
1   package fundamentos.polimorfismosobrescrita;
2
3   public class Rato extends Animal
4   {
5
```

```
6     }
```

10. INTERFACE

10.1. Diagrama de Classes



powered by Astah

10.2. Código Fonte

10.2.1. Classe Controle

```
1   /**
2    * FUNDAMENTOS DE ENGENHARIA DE SOFTWARE
3    * - Interface
4    *
5    * @author Francisco Pedro Morais Gonçalves
6    * https://github.com/fgonca
7    */
8
9   package fundamentos.interfacebase;
10
11  public class Controle
12  {
13
14      public static void main(String[] args)
15      {
16
17          Terrestre cao = new Terrestre("Cão");
18          System.out.printf("%s %s.\n", cao.informar(), cao.viverEmTerra());
19
20          Aquatico peixe = new Aquatico("Peixe");
```

```
21              System.out.printf("%s %s.\n", peixe.informar(), peixe.viverNaAgua());
22
23              Anfibio sapo = new Anfibio("Sapo");
24              System.out.printf("%s %s e %s.", sapo.informar(), sapo.viverEmTerra(),
25                        sapo.viverNaAgua());
26        }
27  }
```

10.2.2. Interface ITerrestre

```
1   package fundamentos.interfacevida;
2
3   // Declaração de interface
4   public interface ITerrestre
5   {
6         String informar();
7         String viverEmTerra();
8   }
```

10.2.3. Interface IAquatico

```
1   package fundamentos.interfacevida;
2
3   // Declaração de interface
4   public interface IAquatico
5   {
6         String informar();
7         String viverNaAgua();
8   }
```

10.2.4. Classe Terrestre

```
1   package fundamentos.interfacevida;
2
3   // Declaração de realização/implementação de uma interface
4   public class Terrestre implements ITerrestre
5   {
6         private String nome;
7
8         public Terrestre(String nome)
9         {
10              this.nome = nome;
11        }
12
13        // a classe é obrigada a implementar os métodos da interface
14        public String informar()
15        {
16              return nome;
17        }
18
19        // a classe é obrigada a implementar os métodos da interface
20        public String viverEmTerra()
21        {
22              return "vive em terra";
23        }
24  }
```

10.2.5. Classe Aquatico

```
1  package fundamentos.interfacevida;
2
3  // Declaração de realização/implementação de uma interface
4  public class Aquatico implements IAquatico
5  {
6        private String nome;
7
8        public Aquatico(String nome)
9        {
10                this.nome = nome;
11       }
12
13        // a classe é obrigada a implementar os métodos da interface
14       public String informar()
15       {
16            return nome;
17       }
18
19        // a classe é obrigada a implementar os métodos da interface
20       public String viverNaAgua()
21       {
22            return "vive na água";
23       }
24
25 }
```

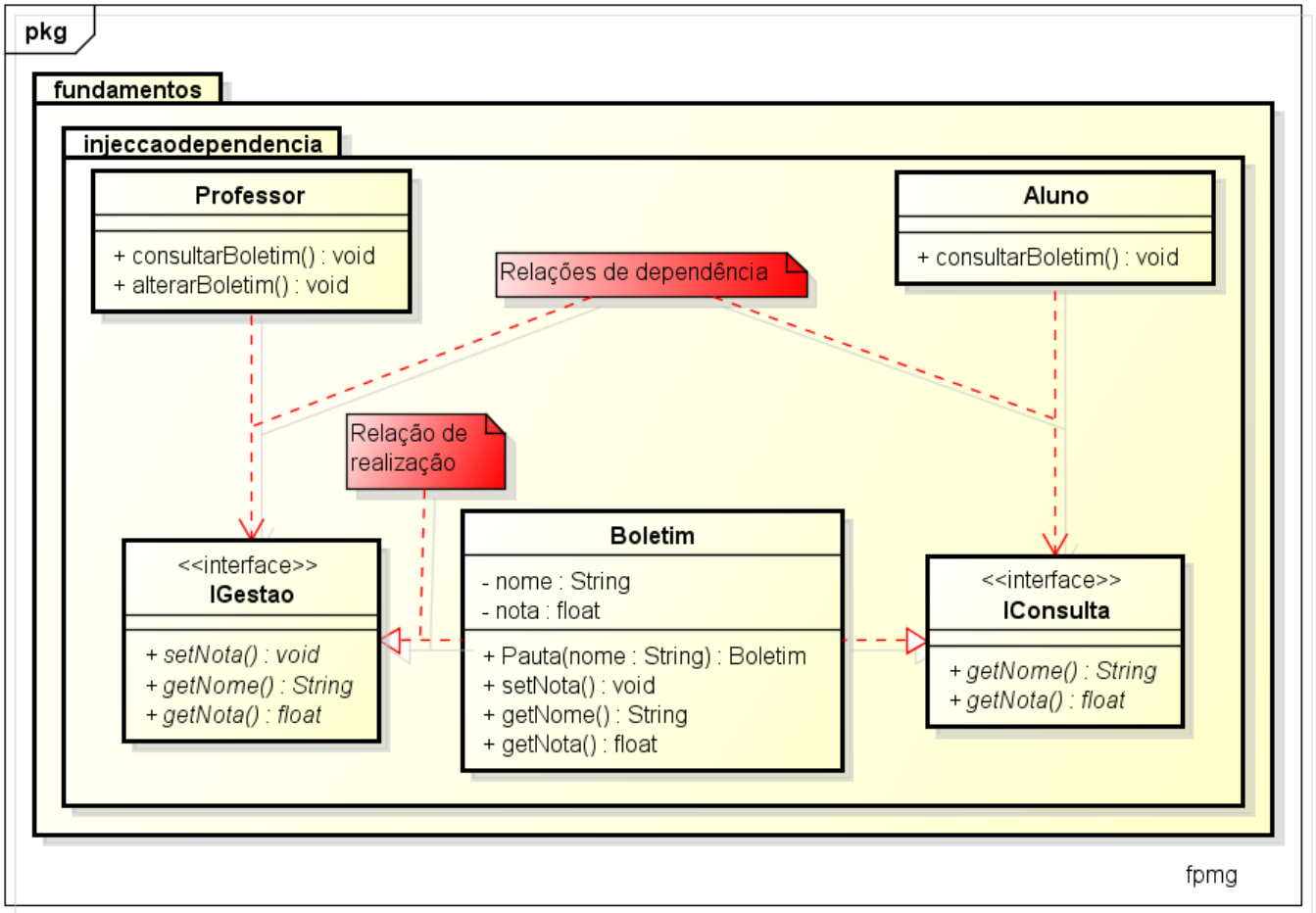
10.2.6. Classe Anfibio

```
1  package fundamentos.interfacevida;
2
3  // Declaração de realização/implementação de duas interfaces
4  public class Anfibio implements ITerrestre, IAquatico
5  {
6        private String nome;
7
8        public Anfibio(String nome)
9        {
10                this.nome = nome;
11       }
12
13        // a classe é obrigada a implementar os métodos da interface
14       public String informar()
15       {
16            return nome;
17       }
18
19        // a classe é obrigada a implementar os métodos da interface
20       public String viverNaAgua()
21       {
22            return "vive na água";
23       }
24
25        // a classe é obrigada a implementar os métodos da interface
26       public String viverEmTerra()
27       {
28            return "vive em terra";
```

```
29          }
30      }
```

11. INJECÇÃO DE DEPENDÊNCIA

11.1. Diagrama de Classe



11.1. Código Fonte

11.1.1. Classe Controle

```
1   /**
2    * FUNDAMENTOS DE ENGENHARIA DE SOFTWARE
3    * - Injecção de Dependência
4    *
5    * @author Francisco Pedro Morais Gonçalves
6    * https://github.com/fgonca
7    */
8
9   package a11_injeccao_dependencia;
10
11  public class Controle {
12
13      public static void main(String[] args) {
14
```

```
15            // criar a dependência
16            Boletim boletim = new Boletim("Abel");
17
18            // injectar a dependência
19            Professor professor = new Professor(boletim);
20
21            professor.apresentarBoletim();
22            professor.alterarBoletim(19);
23            professor.apresentarBoletim();
24
25            // injectar a dependência
26            Aluno aluno = new Aluno(boletim);
27
28            aluno.apresentarBoletim();
29      }
30  }
```

11.1.2. Interface Consula

```
1   package a11_injeccao_dependencia;
2
3   public interface IConsulta
4   {
5       String getNome();
6       float getNota();
7   }
```

11.1.3. Interface Gestao

```
1   package a11_injeccao_dependencia;
2
3   public interface IGestao
4   {
5       String getNome();
6       float getNota();
7       void setNota(float nota);
8   }
```

11.1.4. Classe Boletim

```
1   /**
2    * FUNDAMENTOS DE ENGENHARIA DE SOFTWARE
3    * - Injecção de Dependência
4    *
5    * @author Francisco Pedro Morais Gonçalves
6    * https://github.com/fgonca
7    */
8
9   package a11_injeccao_dependencia;
10
11  public class Boletim implements IGestao, IConsulta
12  {
13      private String nome;
14      private float nota = 15.5f;
15
16      public Boletim(String nome)
17      {
18          this.nome = nome;
```

```
19        }
20
21        public String getNome() {
22              return nome;
23        }
24
25        public float getNota() {
26              return nota;
27        }
28
29        public void setNota(float nota) {
30              this.nota = nota;
31        }
32  }
33
```

11.1.5. Classe Aluno

```
1   /**
2    * FUNDAMENTOS DE ENGENHARIA DE SOFTWARE
3    * - Injecção de Dependência
4    *
5    * @author Francisco Pedro Morais Gonçalves
6    * https://github.com/fgonca
7    */
8
9   package a11_injeccao_dependencia;
10
11  public class Aluno
12  {
13        private IGestao iGestao;
14        public Aluno(IGestao iGestao) // receber injecção de dependência
15        {
16              this.iGestao = iGestao;
17        }
18
19        public void apresentarBoletim()
20        {
21              String nome = this.iGestao.getNome();
22              float nota = this.iGestao.getNota();
23
24              System.out.println(nome + ": " + nota);
25        }
26  }
```

11.1.6. Classe Professor

```
1   /**
2    * FUNDAMENTOS DE ENGENHARIA DE SOFTWARE
3    * - Injecção de Dependência
4    *
5    * @author Francisco Pedro Morais Gonçalves
6    * https://github.com/fgonca
7    */
8
9   package a11_injeccao_dependencia;
10
11  public class Professor
12  {
```

```
13        private IGestao iGestao;
14        public Professor(IGestao iGestao) // receber injecção de dependência
15        {
16              this.iGestao = iGestao;
17        }
18
19        public void apresentarBoletim()
20        {
21              String nome = this.iGestao.getNome();
22              float nota = this.iGestao.getNota();
23
24              System.out.println(nome + ": " + nota);
25        }
26
27        public void alterarBoletim(float nota)
28        {
29              this.iGestao.setNota(nota);
30        }
31  }
```