7. CLASSE Object E ARRAYS

PROGRAMAÇÃO ORIENTADA AOS OBJETOS

Desenvolvido por:

Carlos Urbano
Catarina Reis
José Magno
Marco Ferreira
Ricardo Antunes

ÍNDICE

7.1. CLASSE Object

`Object` é a classe raiz da hierarquia das classes do java

Qualquer classe em java estende de `Object` mesmo que tal não tenha sido declarado explicitamente

Assim, qualquer instância de uma classe é um `Object`, herdando os métodos definidos nessa superclasse

7.1. CLASSE Object

Entre outros, estão definidos na classe **Object** os métodos:

```
public boolean equals(Object object)

public int hashCode()

public String toString()
```

7.1. CLASSE Object


```
public boolean equals(Object obj)
```

- Verifica a igualdade entre o objeto referenciado por `this` e qualquer outro objeto referenciado por `obj`, devolvendo `true` neste caso e `false` caso contrário

- Por omissão:

```
public boolean equals(Object obj) {
    return (this == obj);
}
```

7.1. CLASSE Object

equals() DA CLASSE Aula

```
public abstract class Sala<TAula extends Aula> extends Descritor implements
                                        RepositorioAulas<TAula>, AssociavelAulas<TAula> {
    protected GestorAulas<TAula> gestorAulas;
    private boolean aberta;
    …
    @Override
    public boolean equals(Object o) {
        if (this == o) {
            return true;
        }
        if (!(o instanceof Sala)) {
            return false;
        }
        if (!super.equals(o)) {
            return false;
        }
        Sala<?> sala = (Sala<?>) o;
        return aberta == sala.aberta &&
               Objects.equals(gestorAulas, sala.gestorAulas);
    }
    …
}
```

7.1. CLASSE Object


```
public int hashCode()
```

- Esse método devolve um inteiro para o objeto referenciado por `this`
- Utilizado na implementação de várias `Collections`

- Por omissão:

```
public native int hashCode();
```

7.1. CLASSE Object

hashCode() DA CLASSE Pessoa

```
public abstract class Pessoa extends Identificador implements
                                        RepositorioAulas<Aula>, AssociavelAulas<Aula> {
    protected GestorAulas<Aula> gestorAulas;
    …

    @Override
    public int hashCode() {

        return Objects.hash(super.hashCode(), gestorAulas);
    }
    …
}
```

7.1. CLASSE Object


```
public String toString()
```

- Por omissão, devolve uma **String** com o package, nome da classe e um valor hexadecimal que representa o objeto

- Por omissão:

```
public String toString() {
    return getClass().getName() + "@" +
                    Integer.toHexString(hashCode());
}
```

7.1. CLASSE Object

toString() DA CLASSE Aluno

```
public class Aluno extends Pessoa {
    …

    @Override
    public String toString() {
        final StringBuilder sb = new StringBuilder("Aluno{");
        sb.append("gestorAulas=").append(gestorAulas);
        sb.append(", numero=").append(numero);
        sb.append(", nome='").append(nome).append('\'');
        sb.append('}');
        return sb.toString();
    }
}
```

DEVE TER-SE EM ATENÇÃO A RECURSIVIDADE QUANDO EXISTEM REFERÊNCIAS CIRCULARES

7.2. ARRAYS

Em Java, os *arrays* são tratados como objetos. Assim, é necessário:

- **Declarar uma variável do tipo *array***
- **Instanciar o *array* utilizando o operador `new`**

Exemplos de declarações:

Dimensão	Sintaxe	Exemplos
1	`Tipo[] nomeDoArray;` ou `Tipo nomeDoArray[];`	`int[] arrayInteiros;` `Complexo[] arrayComplexos;`
2	`Tipo[] nomeDoArray;` ou `Tipo nomeDoArray[][];`	`int[][] matrizInteiros;` `Complexo[][] matrizComplexos;`
N	`Tipo[]…[] nomeDoArray;` ou `Tipo nomeDoArray[]…[];`	`int[]…[] matrizInteiros;` `Complexo matrizComplexos[]…[];`

7.2. ARRAYS

Ao contrário da linguagem C, na declaração de *arrays* em Java não se especifica o tamanho de cada dimensão

- **O tamanho de cada dimensão é indicado quando se instancia o *array***
- **O tamanho máximo da dimensão pode ser qualquer expressão do tipo inteiro positivo**

Exemplos de instanciações:

Dim.	Sintaxe	Exemplos
1	`new Tipo[tamanhoD1];`	`arrayInteiros = new int[23]` `arrayComplexos = new Complexo[5];`
2	`new Tipo[tamanhoD1][tamanhoD2];`	`matrizInteiros = new int[2][5];` `matrizComplexos = new Complexo[20][4];`
N	`new Tipo[tamanhoD1]…[tamanhoDn];`	`multiInteiros = new int[3]…[10] ;` `multiComplexos = new Complexo[10]…[50];`

7.2. ARRAYS

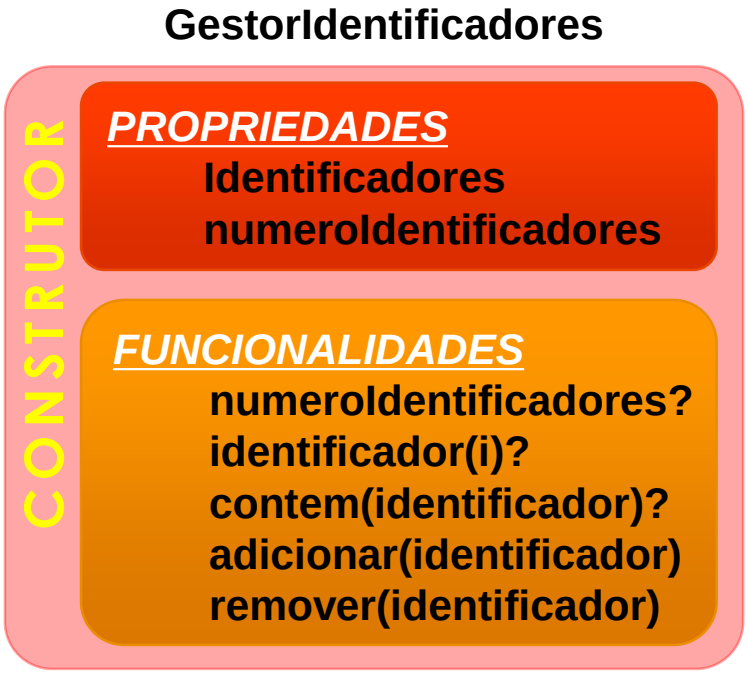
É possível criar um array a partir de um conjunto de valores iniciais

Neste caso o tamanho do array é determinado pelo número de valores iniciais

Exemplos de instanciações:

```
int[] notas = {10, 15, 2, 17, 8};

double[][] bidimensional = {{1, 2, 3}, {4, 5, 6}};

Complexo[] complexos  = {new Complexo(0, 1),
                         null,
                         new Complexo(2, 3)
                         };
```

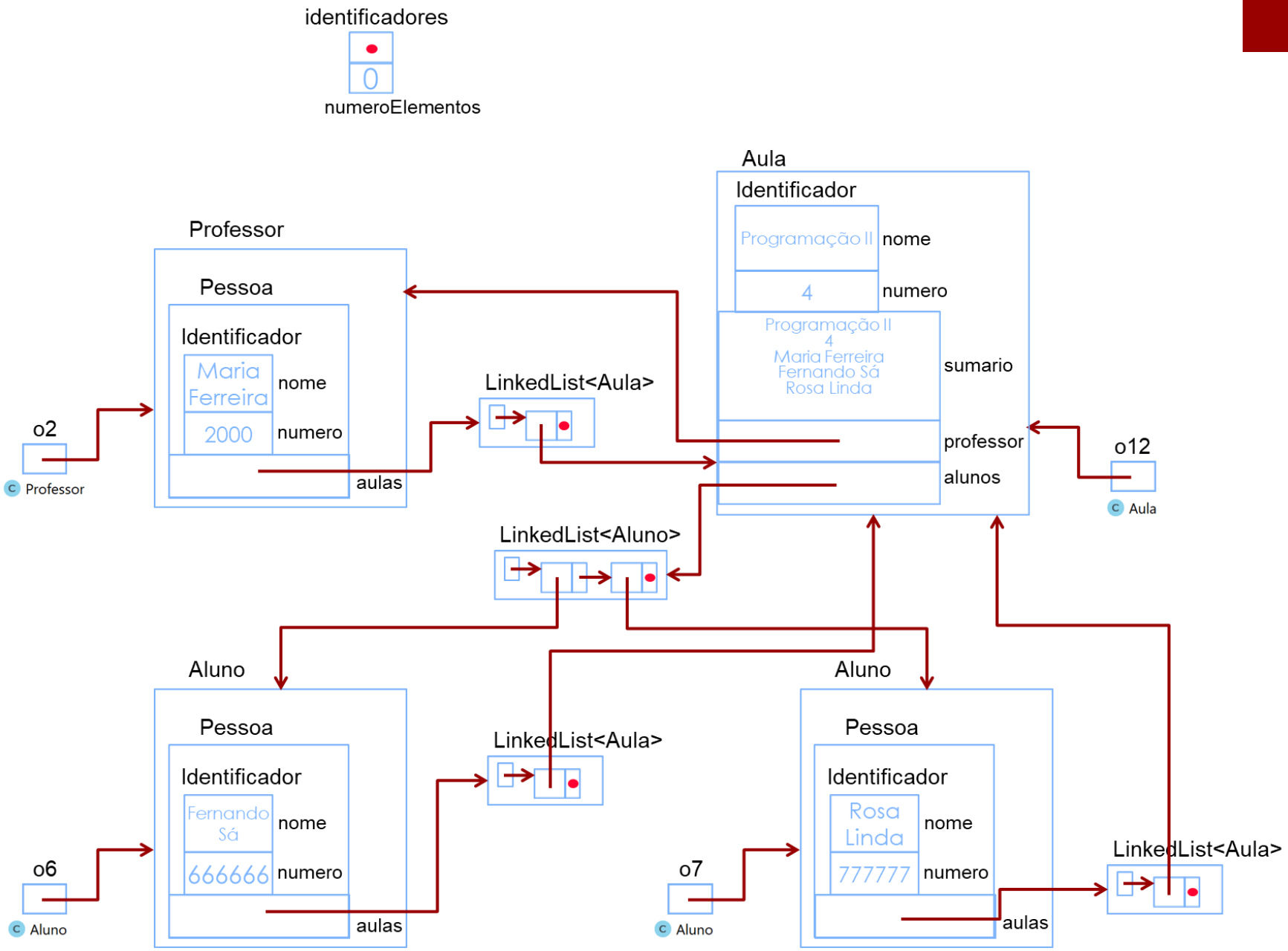
VAMOS CONSIDERAR A CLASSE `GestorIdentificadores` QUE EFETUA A GESTÃO DE UM ARRAY DE IDENTIFICADORES



DECLARAÇÃO DE UM ARRAY UNIDIMENSIONAL DE Identificadores

```
class GestorIdentificadores {
    private Identificador[] identificadores;
    private int numeroIdentificadores;

    …
}
```

7.2. ARRAYS

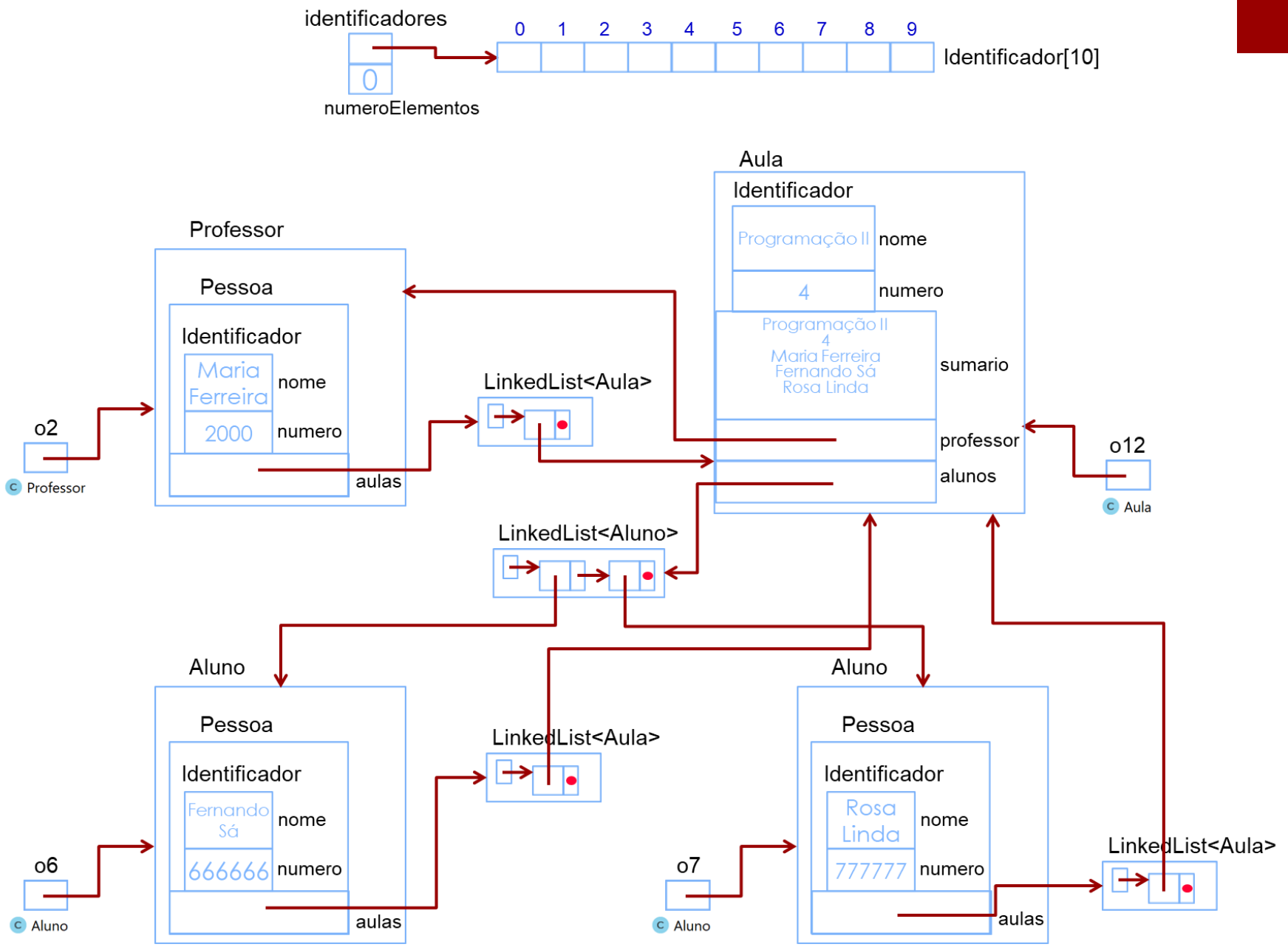

identificadores
0
numeroElementos
Aula
Identificador
Programação II
nome
4
numero
Programação II
4
Maria Ferreira
Fernando Sá
Rosa Linda
sumario
professor
alunos
Professor
Pessoa
Identificador
Maria Ferreira
nome
2000
numero
aulas
LinkedList<Aula>
o2
Professor
o12
Aula
LinkedList<Aluno>
Aluno
Pessoa
Identificador
Fernando Sá
nome
666666
numero
aulas
LinkedList<Aula>
o6
Aluno
Aluno
Pessoa
Identificador
Rosa Linda
nome
777777
numero
aulas
LinkedList<Aula>
o7
Aluno

CRIAÇÃO DE UM ARRAY UNIDIMENSIONAL DE Identificadores

```
class GestorIdentificadores {
    private Identificador[] identificadores;
    private int numeroIdentificadores;

    …

    public GestorIdentificadores(int numeroMaximoIdentificadores) {
        identificadores = new Identificador[numeroMaximoIdentificadores];
        numeroIdentificadores = 0;
    }
}
```

7.2. ARRAYS


```
public Main() {
    GestorIdentificadores gestorIdentificadores =
                                        new GestorIdentificadores(10);
}
```



ADIÇÃO DE UM IDENTIFICADOR

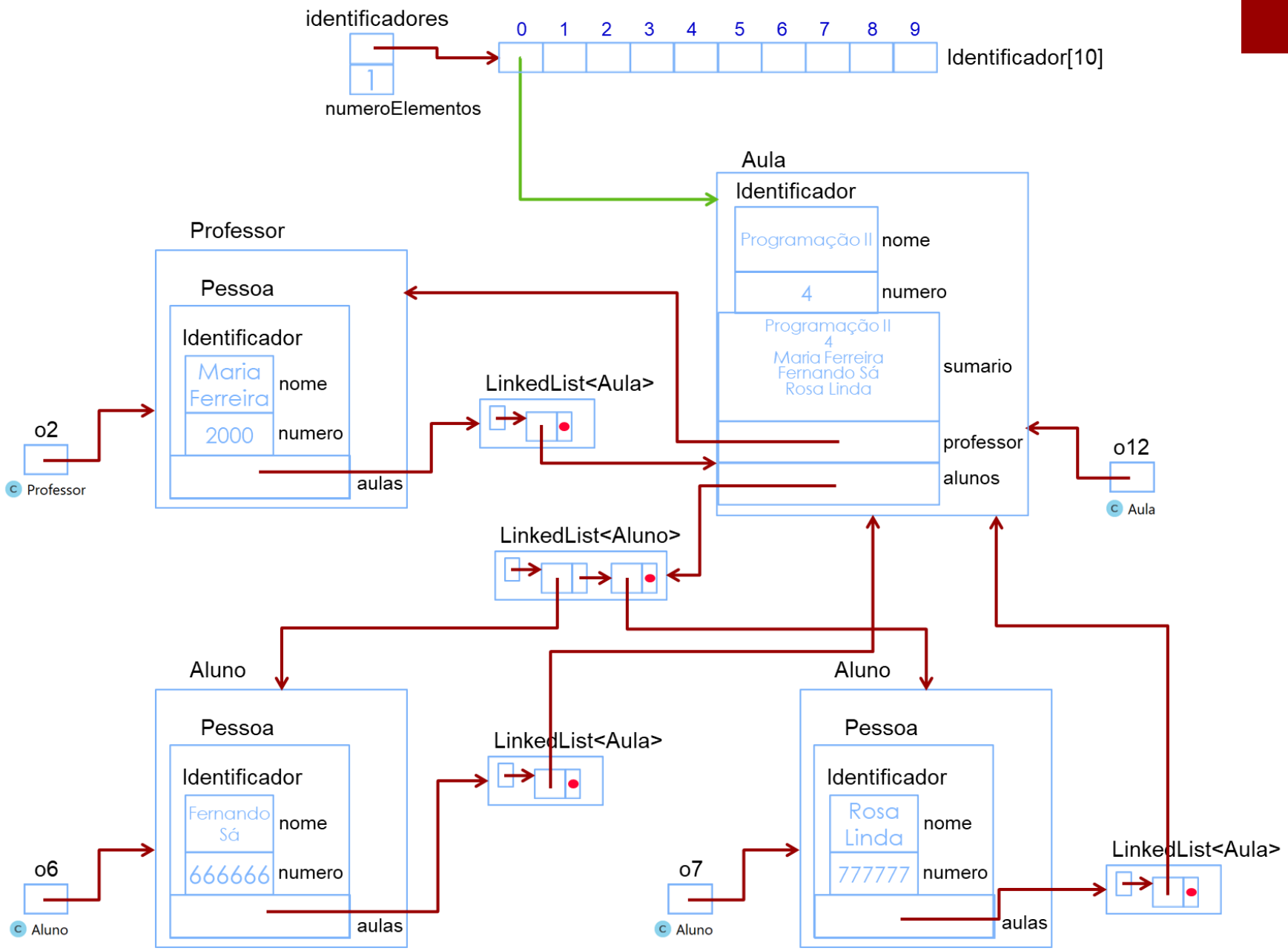
```
class GestorIdentificadores {
    private Identificador[] identificadores;
    private int numeroIdentificadores;

    …

    public void adicionar(Identificador identificador) {
        if (numeroIdentificadores >= identificadores.length ||
                    contem(identificador)) {
            return;
        }
        identificadores[numeroIdentificadores++] = identificador;
    }
}
```

`identificadores.length` DEVOLVE O NÚMERO DE ELEMENTOS PARA O QUAL O ARRAY FOI CRIADO

7.2. ARRAYS

```
public Main() {
    …
    gestorIdentificadores.adicionar(o12);
}
```

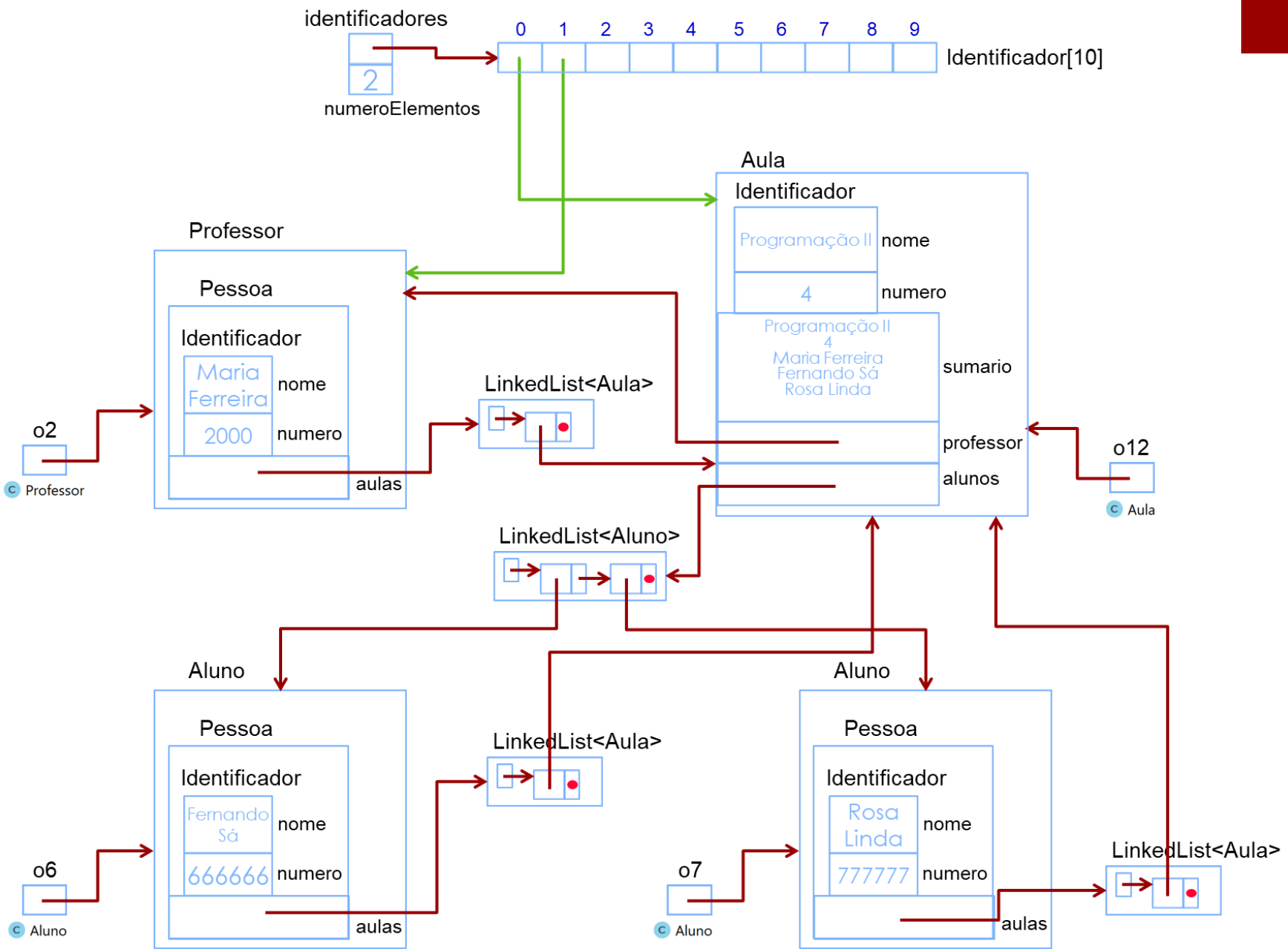
Main



7.2. ARRAYS

```
public Main() {
    …
    gestorIdentificadores.adicionar(o2);
}
```

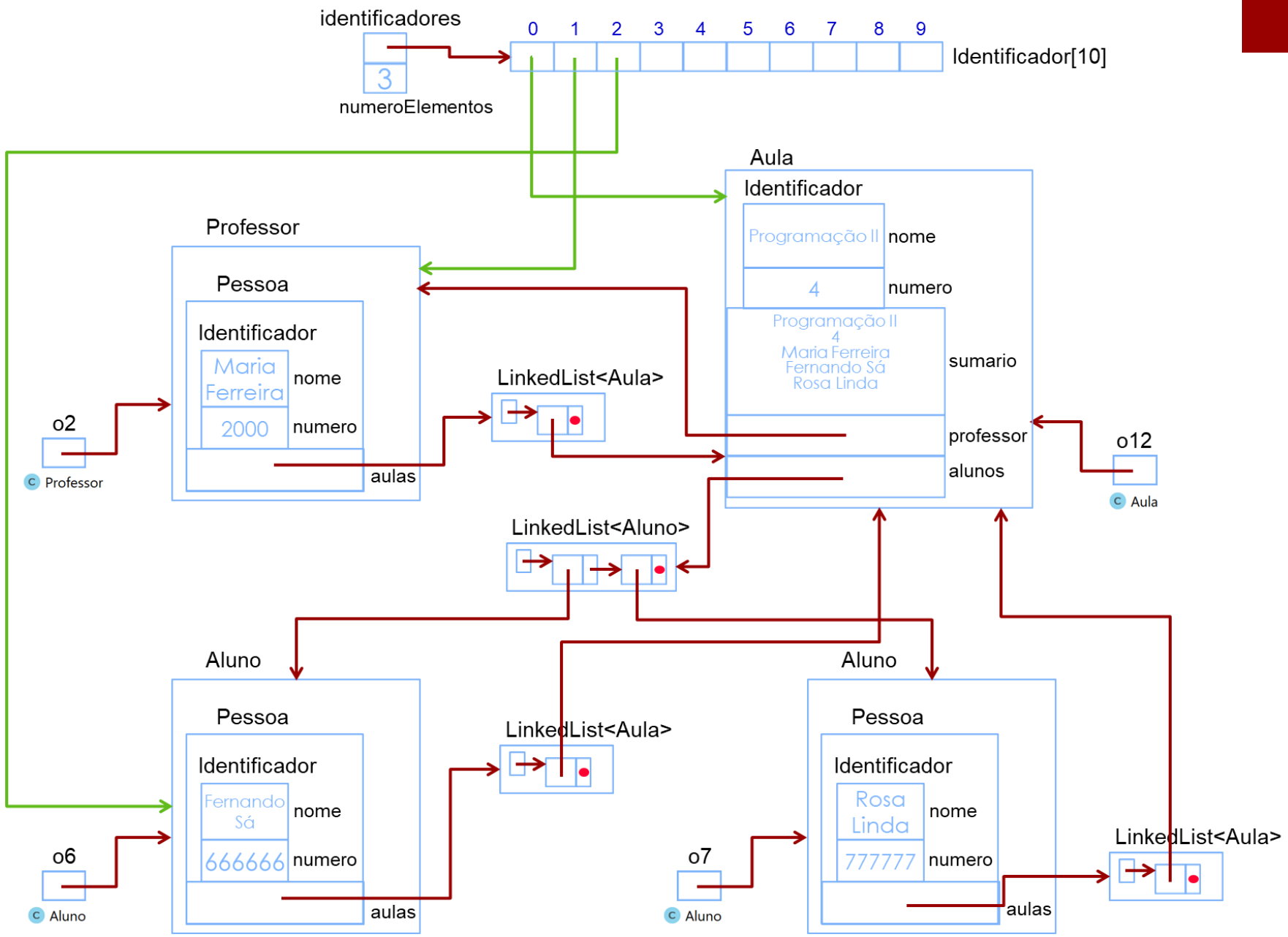
Main



7.2. ARRAYS

```
public Main() {
    …
    gestorIdentificadores.adicionar(o6);
}
```

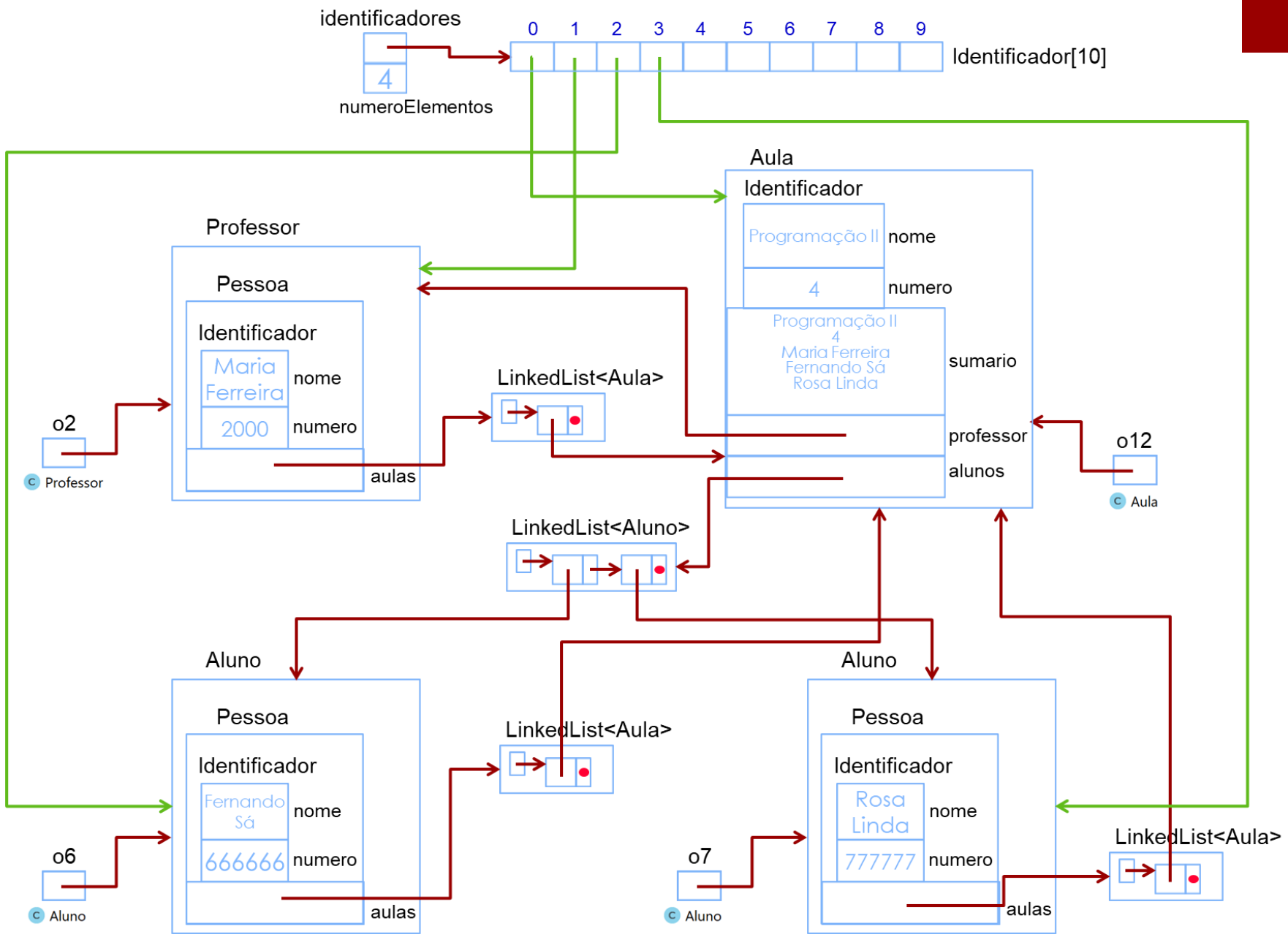
Main



7.2. ARRAYS

```
public Main() {
    …
    gestorIdentificadores.adicionar(o7);
}
```

Main



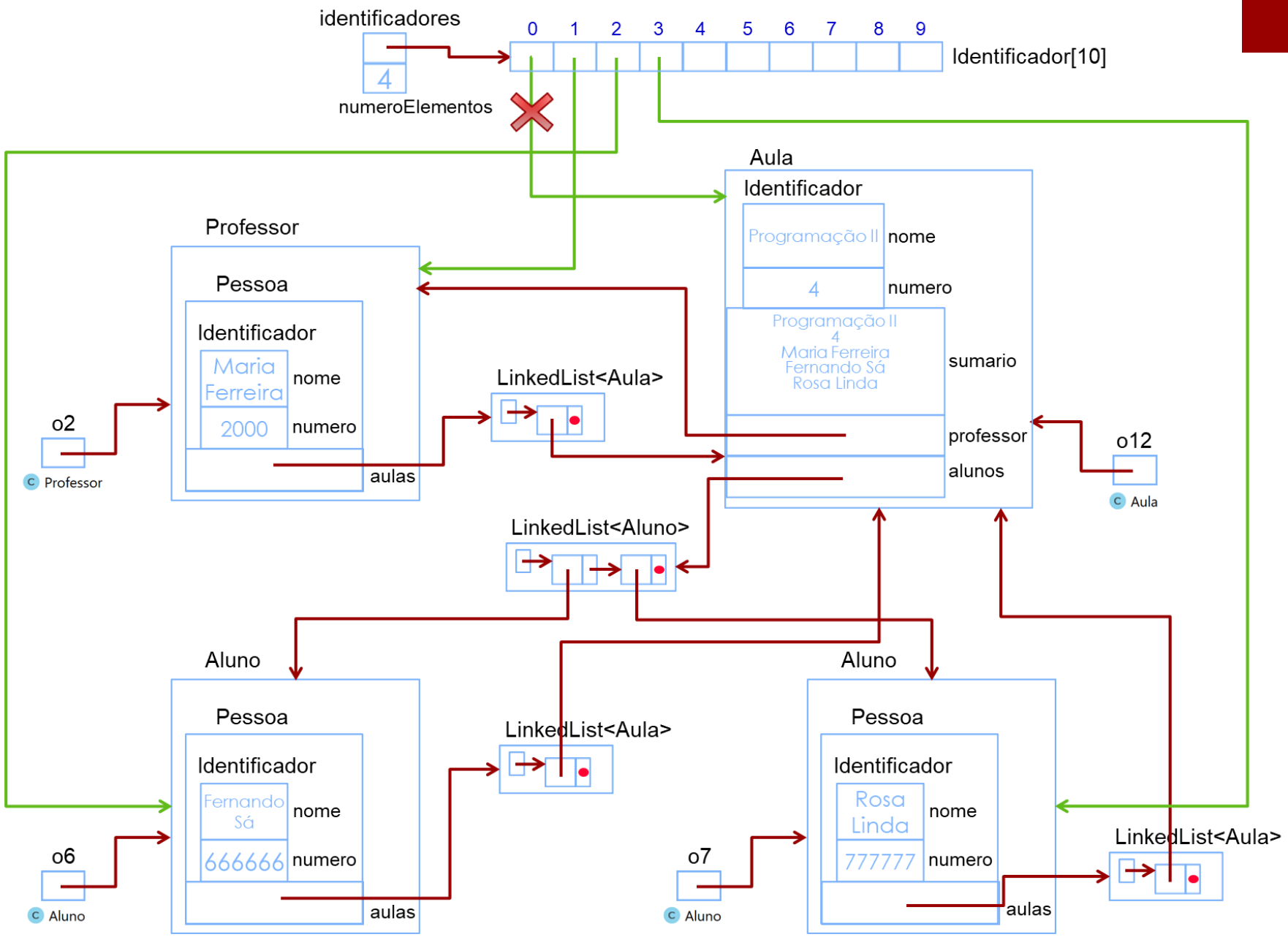
REMOÇÃO DE UM IDENTIFICADOR

```
class GestorIdentificadores {
    private Identificador[] identificadores;
    private int numeroIdentificadores;

    …

    public void remover(Identificador identificador) {
        for (int i = 0; i < identificadores.length; i++) {
            if (identificadores[i].equals(identificador)) {
                for (int j = i; j < numeroIdentificadores - 1; j++) {
                    identificadores[j] = identificadores[j + 1];
                }
                numeroIdentificadores--;
                return;
            }
        }
    }
}
```

7.2. ARRAYS

```
public Main() {
    …
    gestorIdentificadores.remover(o12);
}
```

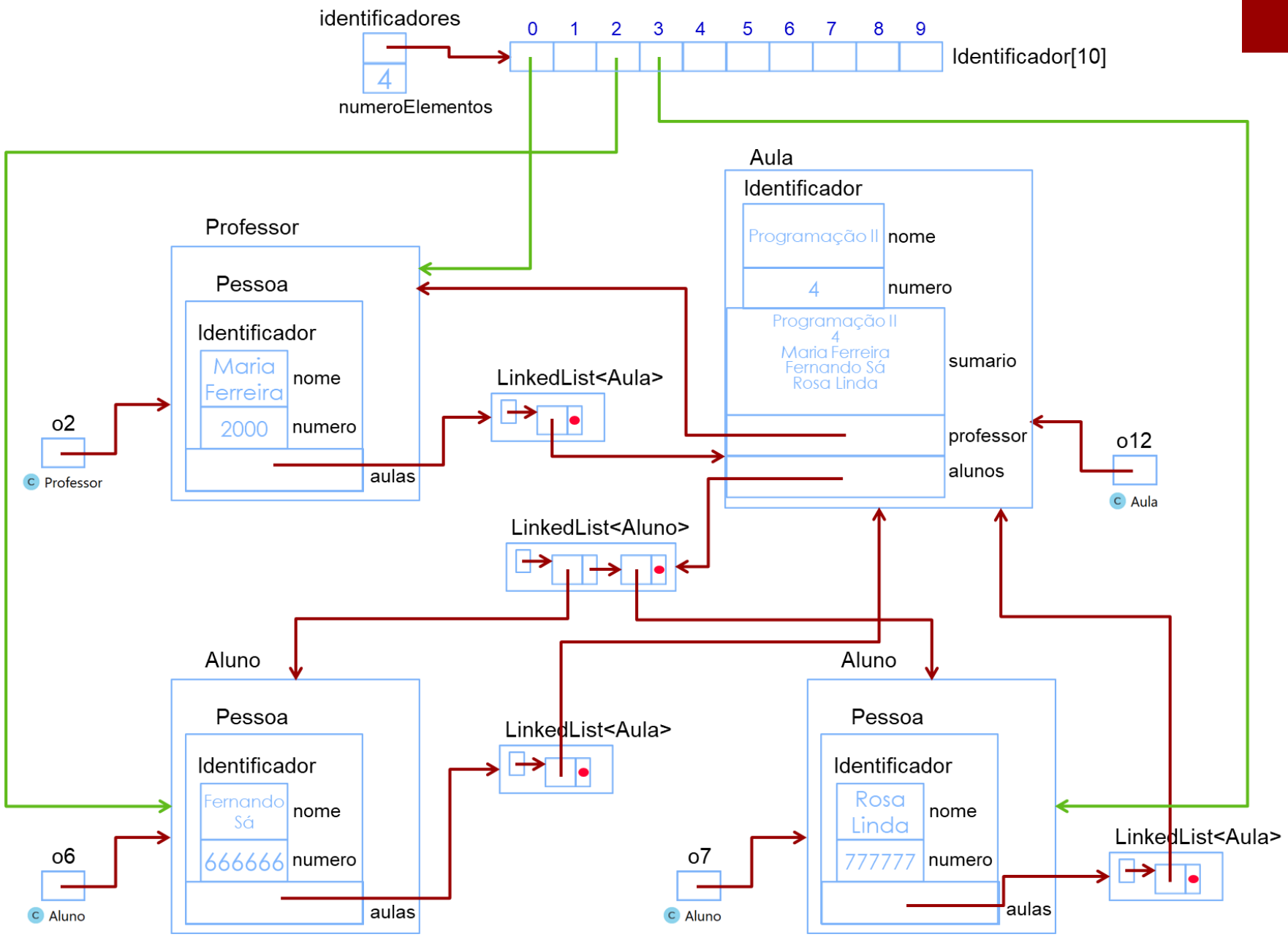
Main



7.2. ARRAYS

```
public Main() {
    …
    gestorIdentificadores.remover(o12);
}
```

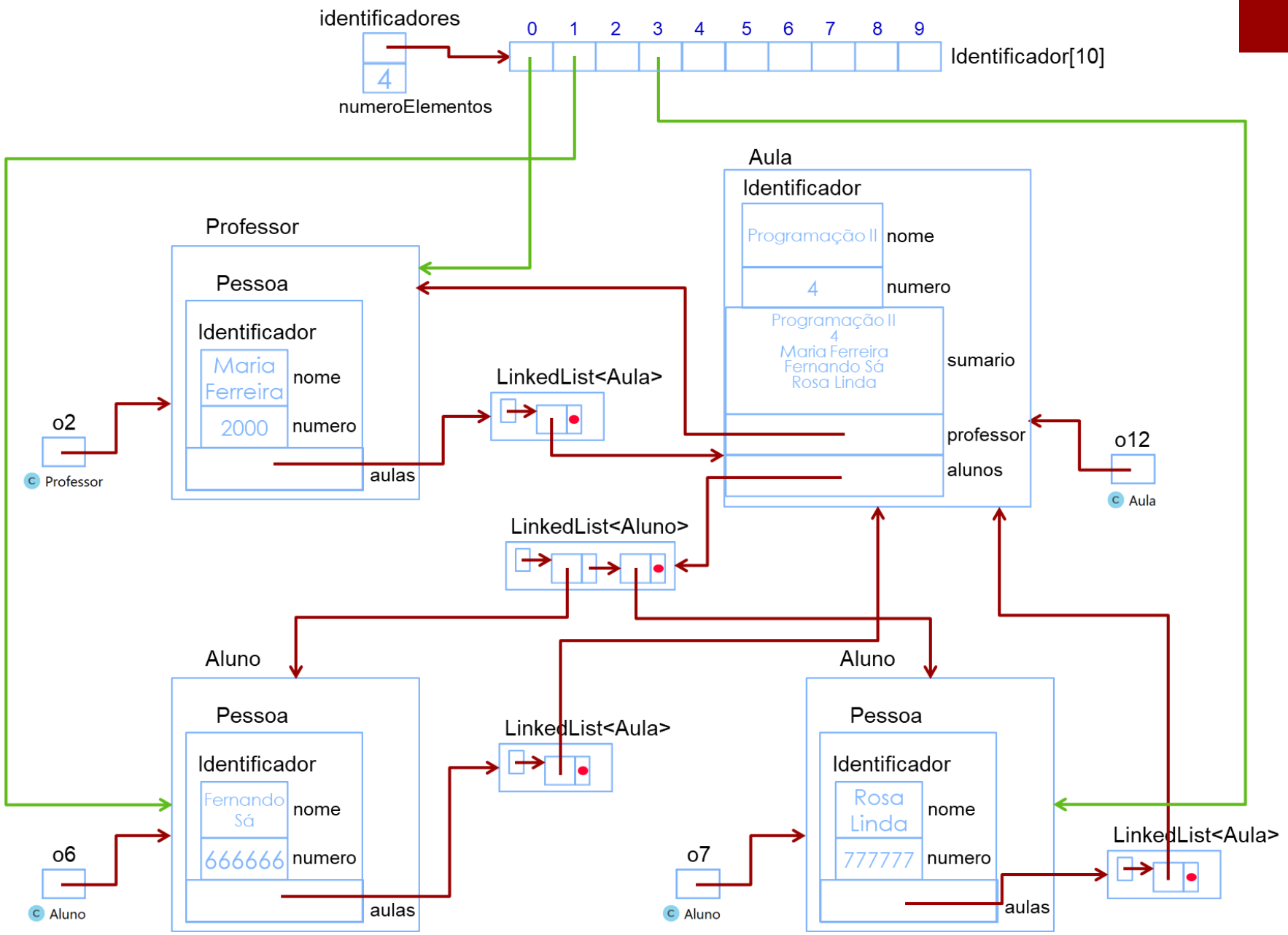
Main



7.2. ARRAYS

```
public Main() {
    …
    gestorIdentificadores.remover(o12);
}
```

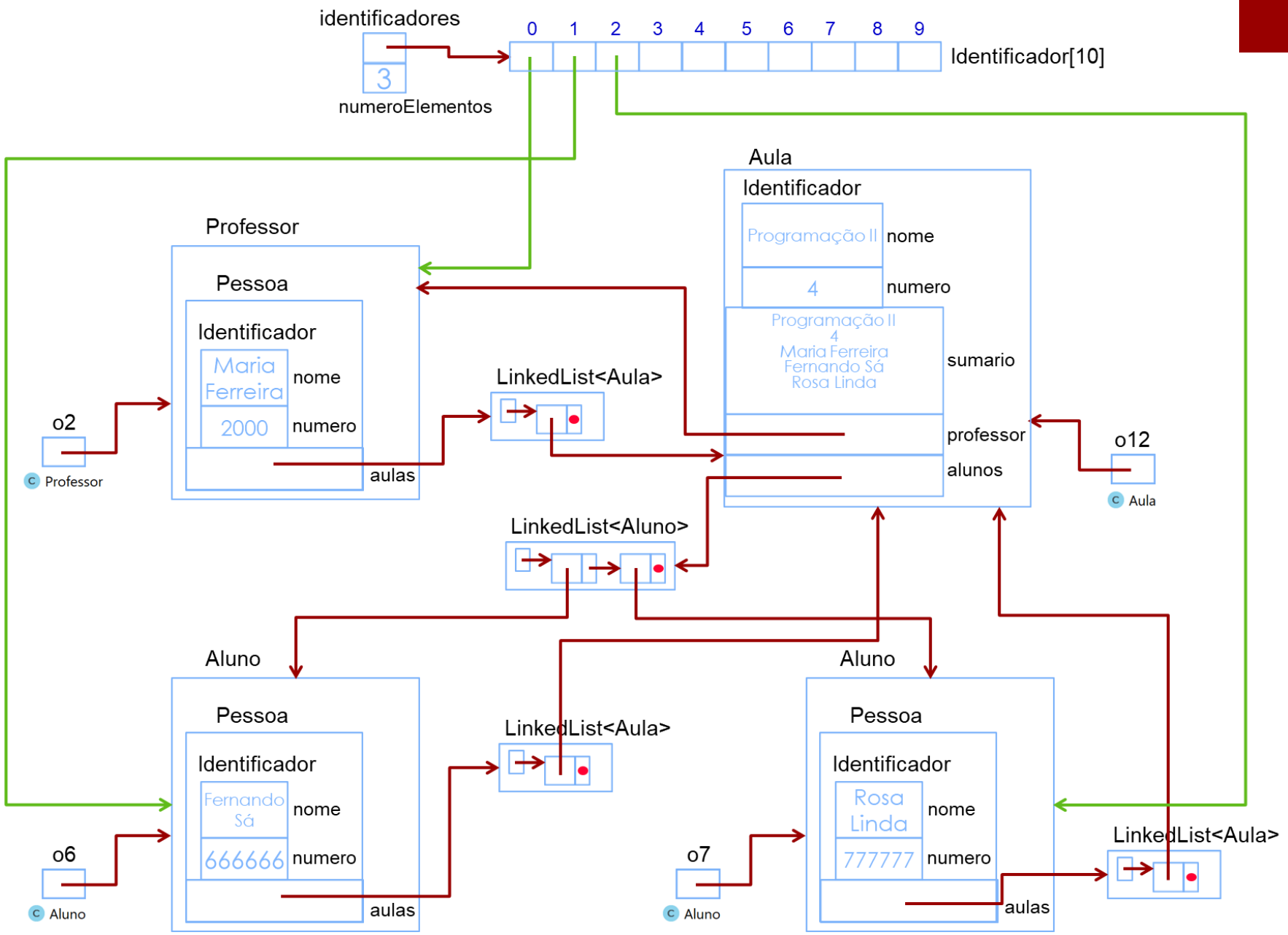
Main



7.2. ARRAYS

```
public Main() {
    …
    gestorIdentificadores.remover(o12);
}
```

Main



DEVOLUÇÃO DO NÚMERO IDENTIFICADORES DE UM GESTOR

```
class GestorIdentificadores {
    private Identificador[] identificadores;
    private int numeroIdentificadores;

    …

    public int getNumeroIdentificadores() {
        return numeroIdentificadores;
    }
}
```

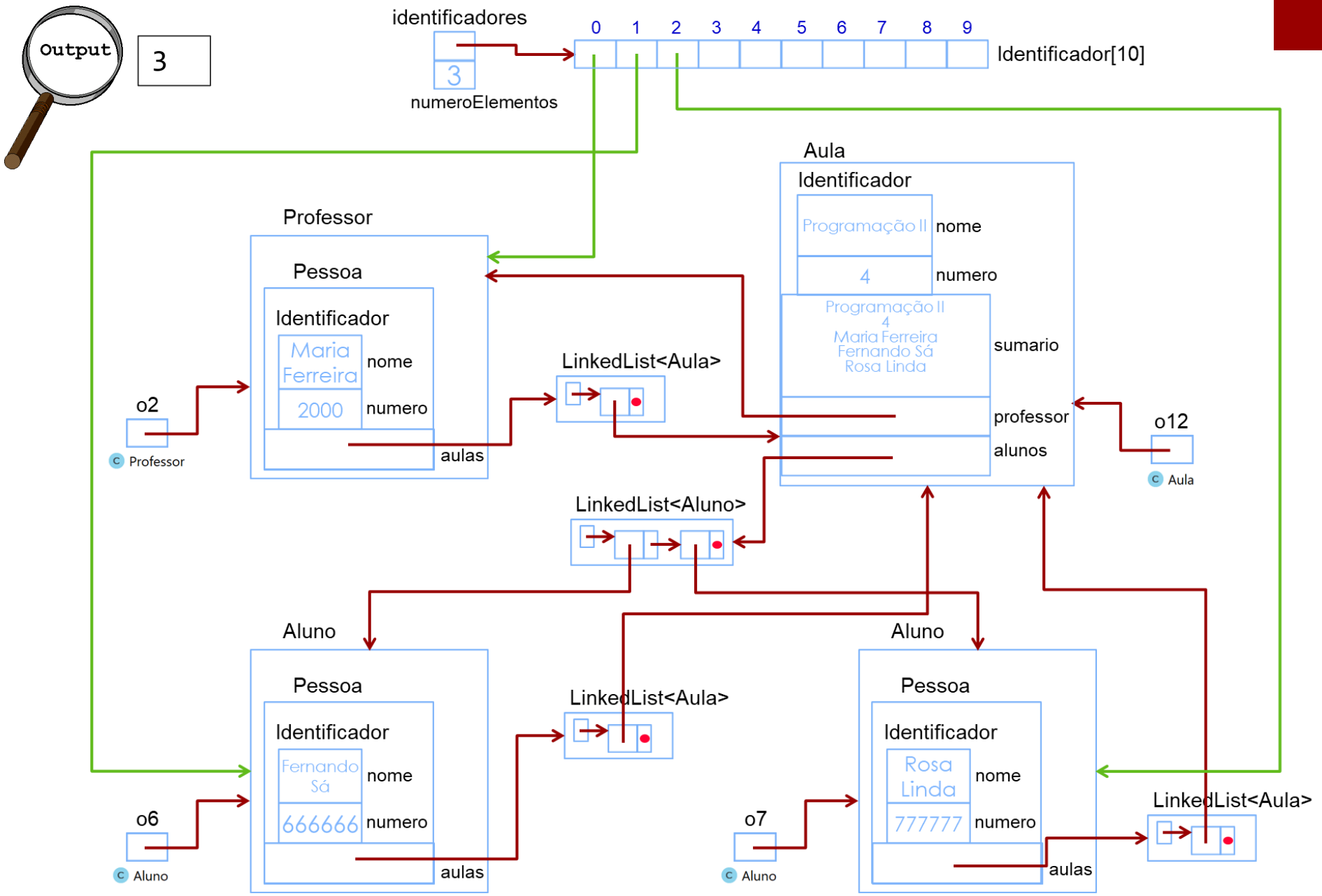
7.2. ARRAYS

```
public Main() {
    System.out.println(gestorIdentificadores.getNumeroIdentificadores());
}
```

Main

Output

3



DEVOLUÇÃO DO IDENTIFICADOR DE UM DETERMINADO ÍNDICE

```
class GestorIdentificadores {
    private Identificador[] identificadores;
    private int numeroIdentificadores;

    …

    public Identificador getIdentificador(int indice) {
        return indice < 0 || indice >= numeroIdentificadores ?
                null : identificadores[indice];
    }
}
```

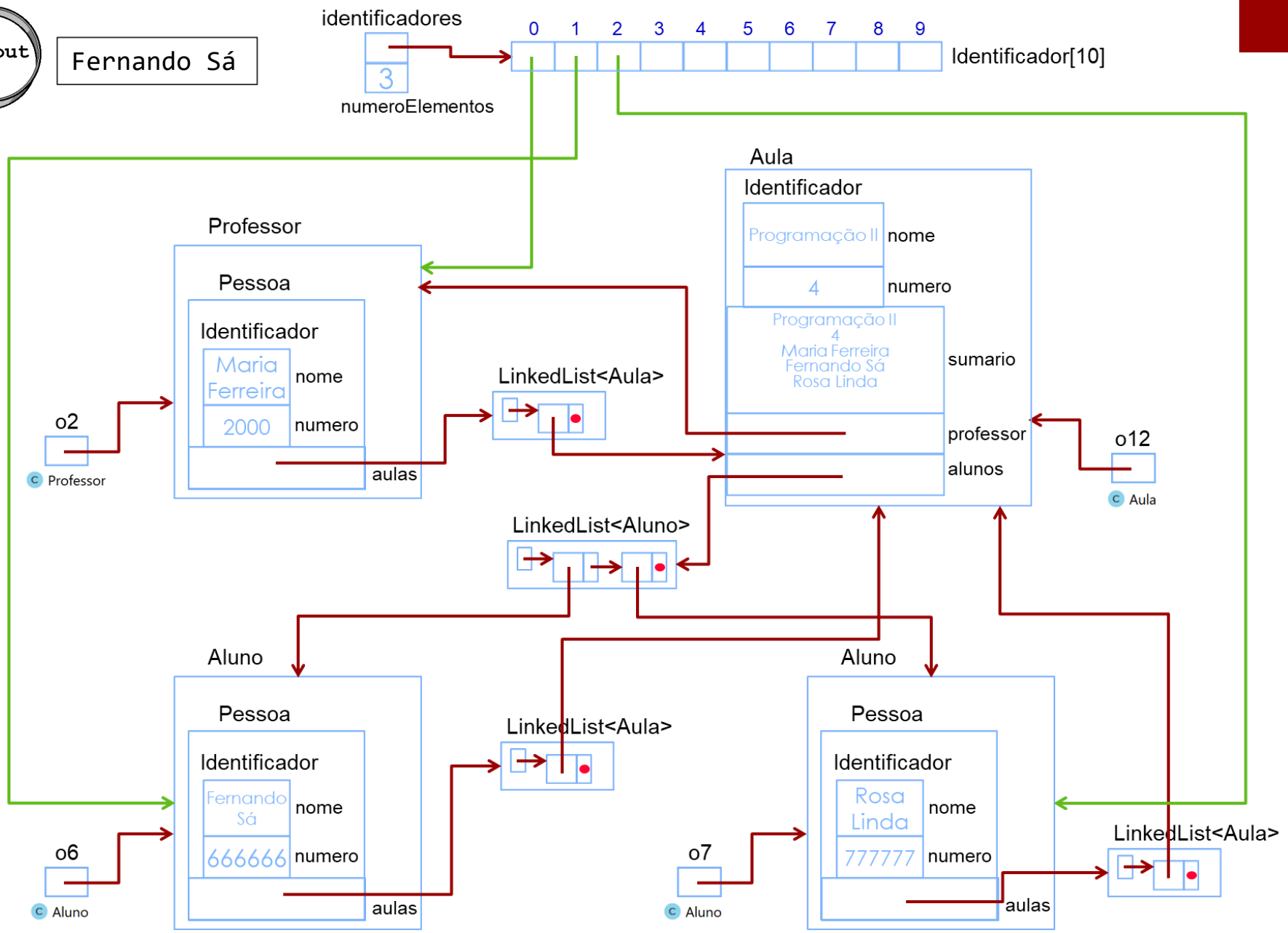
7.2. ARRAYS

```
public Main() {
    System.out.println(gestorIdentificadores.getIdentificador(1).getNome());
}
```

Main

Output

Fernando Sá



VERIFICAÇÃO SE UM IDENTIFICADOR ESTÁ CONTIDO NUM GESTOR

```
class GestorIdentificadores {
    private Identificador[] identificadores;
    private int numeroIdentificadores;

    …

     public boolean contem(Identificador identificador) {
        for (int i = 0; i < identificadores.length; i++) {
            if (identificadores[i].equals(identificador)) {
                return true;
            }
        }
        return false;
    }
}
```

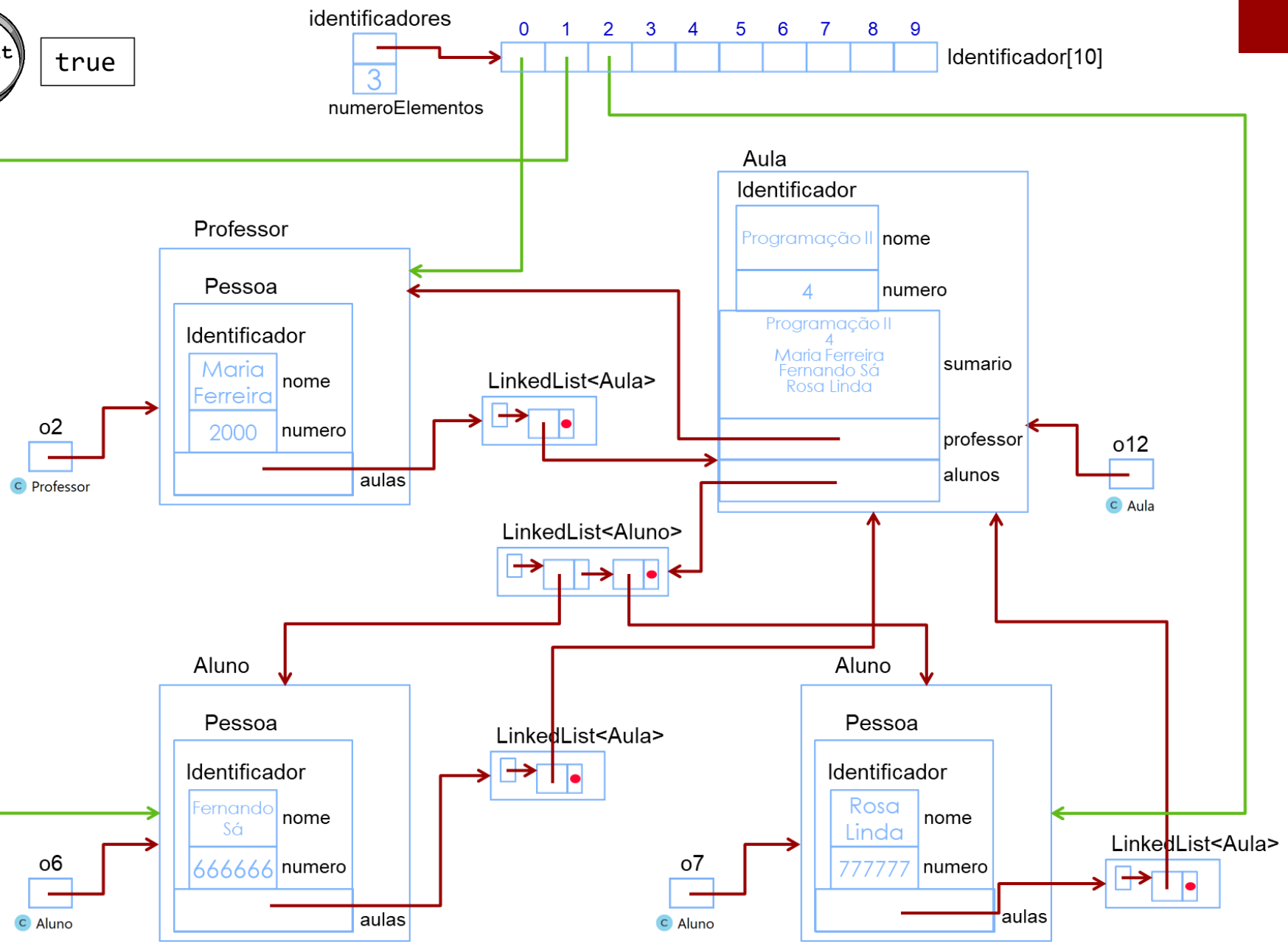
7.2. ARRAYS


```
public Main() {
    System.out.println(gestorIdentificadores.contem(o6));
}
```

Main

Output

```
true
```



PERCORRER QUANDO TODOS OS ELEMENTOS SÃO NÃO NULOS

```
class GestorIdentificadores {
    private Identificador[] identificadores;
    private int numeroIdentificadores;

    …

    public void percorrerArrayDeElementosNaoNulos() {
        for (Identificador identificador : identificadores) {
            System.out.println(identificador);
        }
    }
}
```

PERCORRER QUANDO TODOS OS ELEMENTOS SÃO NÃO NULOS

```
class GestorIdentificadores {
    private Identificador[] identificadores;
    private int numeroIdentificadores;

    …

    public void percorrerArrayDeElementosNaoNulos() {
        for (Identificador identificador : identificadores) {
            System.out.println(identificador);
        }
    }
}
```

ATENÇÃO: CASO UM ELEMENTO SEJA NULO IRÁ OCORRER UMA EXCEÇÃO "NullPointerException"

7.2. ARRAYS

EM JAVA, UM ARRAY COM DIMENSÃO SUPERIOR A 1 É CONSIDERADO UM ARRAY DE ARRAYS.

EXEMPLO:

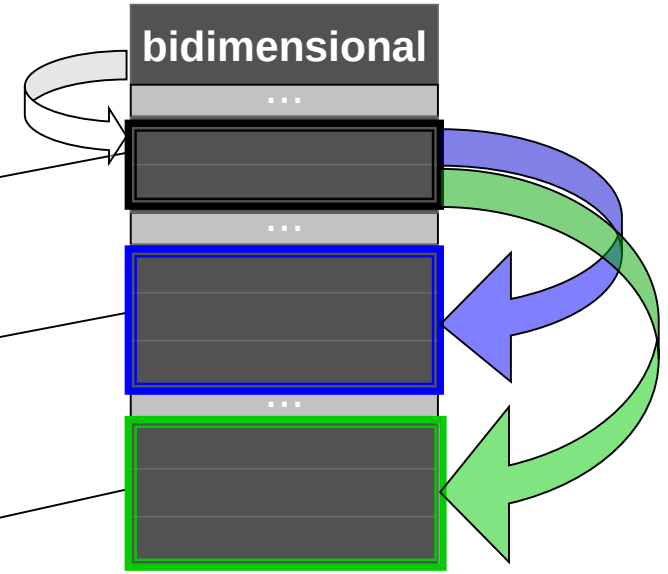
UM ARRAY BIDIMENSIONAL 2X3 É CONSIDERADO COMO SENDO UM ARRAY DE 2 ELEMENTOS, SENDO CADA ELEMENTO UM ARRAY DE TAMANHO 3

```
Complexo[][] bidimensional = new Complexo[2][3];
```

OBJETO DO TIPO COMPLEXO[][] DE DIMENSÃO 2 E COM 2 ELEMENTOS

OBJETO DO TIPO COMPLEXO[] DE DIMENSÃO 1 E COM 3 ELEMENTOS

OBJETO DO TIPO COMPLEXO[] DE DIMENSÃO 1 E COM 3 ELEMENTOS



7.2. ARRAYS

O TAMANHO DO ARRAY NÃO PODE SER ALTERADO APÓS A INSTANCIAÇÃO

PARA SABER QUAL O TAMANHO DO ARRAY UTILIZA-SE O ATRIBUTO ESPECIAL **length** (ATRIBUTO SÓ DE LEITURA - **final**)

```
public ArraysDemo1() {
    int[][] bidimensional = new int[3][4];

    int linhas = bidimensional.length;
    int colunas = bidimensional[0].length;

    for (int i = 0; i < linhas; i++) {
        for (int j = 0; j < colunas; j++) {
            bidimensional[i][j] = (int) (Math.random() * 100 % 100);
        }
    }
}
```

INDICA O TAMANHO DA 1ª DIMENSÃO

7.2. ARRAYS

O TAMANHO DO ARRAY NÃO PODE SER ALTERADO APÓS A INSTANCIAÇÃO

PARA SABER QUAL O TAMANHO DO ARRAY UTILIZA-SE O ATRIBUTO ESPECIAL **length** (ATRIBUTO SÓ DE LEITURA - **final**)

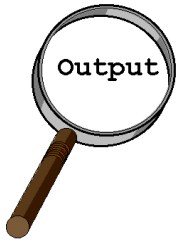
```
public ArraysDemo1() {
    int[][] bidimensional = new int[3][4];

    int linhas = bidimensional.length;
    int colunas = bidimensional[0].length;

    for (int i = 0; i < linhas; i++) {
        for (int j = 0; j < colunas; j++) {
            bidimensional[i][j] = (int) (Math.random() * 100 % 100);
        }
    }
}
```

INDICA O TAMANHO DA 1ª DIMENSÃO

INDICA O TAMANHO DO 1º ARRAY DA 2ª DIMENSÃO

7.2. ARRAYS


```
public ArraysDemo2() {
    int[][] matriz = {{1, 2}, {3, 4, 5}, {6, 7, 8}};

    for (int i = 0; i < matriz.length; i++) {
        for (int j = 0; j < matriz[0].length; j++) {
            System.out.print(matriz[i][j] + "\t");
        }
        System.out.println();
    }
}
```




```
1    2
3    4
6    7
```

O RESULTADO DO EXEMPLO ANTERIOR NÃO É O ESPERADO

ONDE ESTÁ O ERRO?

7.2. ARRAYS

ESTRUTURA DE COLEÇÃO MAIS BÁSICA EM JAVA E, EMBORA SEJA ESSENCIAL QUALQUER PROGRAMADOR SABER COMO A UTILIZAR, NEM SEMPRE É A MELHOR SOLUÇÃO

7.2. ARRAYS

VANTAGENS:

- NÃO TEM MECANISMOS COMPLICADOS PARA APRENDER – O ACESSO É DIRETO - A SINTAXE DOS PARÊNTESIS RETOS `[]` E A VARIÁVEL DE INSTÂNCIA `length`
- UM ARRAY UNIDIMENSIONAL GUARDA OS SEUS CONTEÚDOS NUM ESPAÇO CONTÍGUO EM MEMÓRIA, EM CERTAS SITUAÇÕES QUE EXIGEM "ALTA PERFORMANCE" PODE SER EXTREMAMENTE RÁPIDO

7.2. ARRAYS

DESVANTAGENS:

- TAMANHO FIXO, O QUE SIGNIFICA QUE É LIMITATIVO USAR EM SITUAÇÕES EM QUE O NÚMERO DE ELEMENTOS NÃO É PREVISÍVEL À PARTIDA, OU QUANDO TEM QUE REMOVER UM ELEMENTO
- NÃO TEM MÉTODOS PRÓPRIOS PARA, POR EXEMPLO, FAZER TAREFAS COMUNS TAIS COMO DEVOLVER UMA SUBSEQUÊNCIA DOS VALORES ENTRE CERTAS POSIÇÕES