Programação Orientada a Objetos

Aula 4 –Composição e coleção

Ana Patrícia F. Magalhães Mascarenhas

anapatriciamagalhaes@gmail.com

apmagalhaes@uneb.br

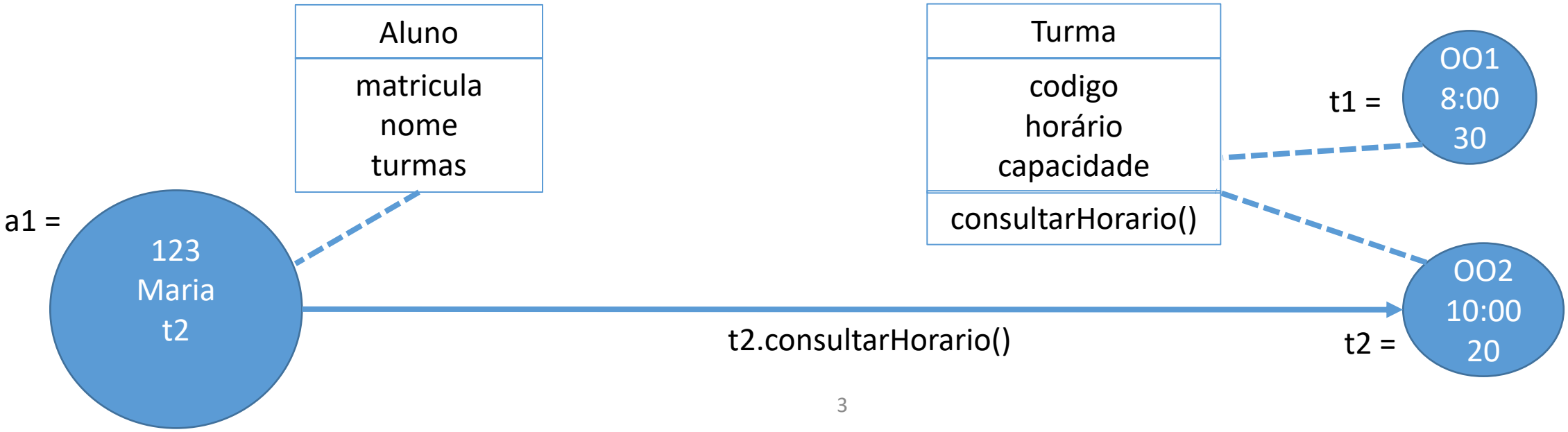
Plano de Aula

- Objetivo
  - Entender como funciona um sistema com várias classes
  - Entender a troca de mensagens entre objetos
  - Construir programas com várias classes
  - Trabalhar com coleção

- Bibliografia básica
  - Santos, Rafael. Programação Orientada a Objetos com Java. Ed. Campus, 2003
  - Livro: SEPE, A.; MAITINO, R. N. Programação orientada a objetos. Londrina: Editora e Distribuidora Educacional AS, 2017. 176p.

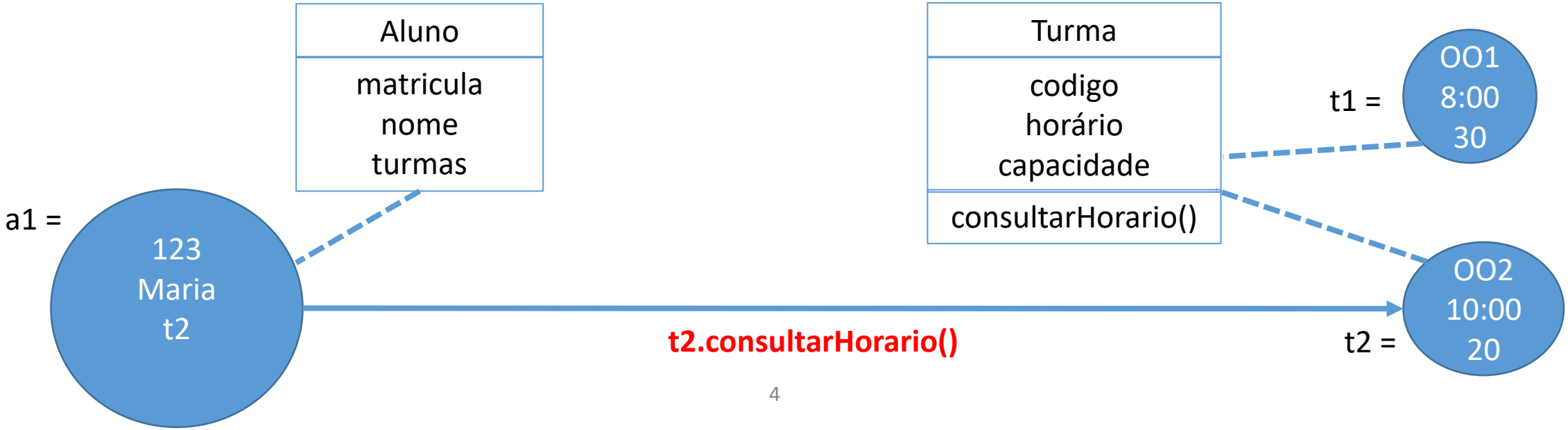
Entendendo mensagens

- Quando construímos um sistema OO temos muitas classes.
- A partir destas classes objetos serão instanciados e trocarão mensagens.
- Uma mensagem pode ser vista como a chamada de um método.



Sistemas OO

- A troca de mensagens entre objetos requer que o objeto emissor da mensagem conheça a referência do objeto receptor da mensagem.
  - No exemplo, o objeto a1 possui a referência do objeto t2 no atributo turmas.
  - Em Java enviamos uma mensagem usando a referência do objeto desejado "." nome do método a ser chamado.



Continuando o exercício anterior

1) Para o cenário da farmácia, considere que os medicamentos são produzidos por um laboratório. Crie uma classe Laboratório com os seguintes atributos cnpj, nome, email, telefone e percentual de lucro. Em seguida construa os seguintes métodos:

   - Método para criar um laboratório informando todos os dados;
   - Métodos get e set para o atributo percentual de lucro;

   Agora vamos modificar a classe Medicamento para acrescentar um atributo que indique qual o laboratório que fabrica o medicamento.

   Em seguida modifique o método que calcula o preço de um medicamento considerando o lucro de 30%. Agora você precisa acessar utilizar o percentual de lucro correspondente ao laboratório que fabrica o medicamento.

   Para finalizar vamos criar um método para verificar se um medicamento pode ser substituído por outro medicamento. Neste caso eles devem conter o mesmo princípio ativo.

Vetores

Um vetor (array) é uma seqüência de objetos ou valores de tipos primitivos, todos do mesmo tipo e combinados sob um único identificador.

Vetores são estáticos. O seu tamanho é definido no momento da sua criação.

Em Java, vetores são objetos. Na prática, eles herdam de Object.

Arrays possuem um atributo público que informa o seu tamanho: **length**.

Arrays em Java iniciam na posição (índice) 0.

Vetores

- **Declaração**: tipo[] identificador;

  Exemplos: int[] vet;

  Button[] b;

Declaração de um vetor de tipo primitivo.

Declaração de um vetor de objeto.

- **Construção:** identificador = new tipo[tamanho]

  Exemplo: vet = new int[12];

  b = new Button[10];

- **Inicialização:**

  int[] vet = {1,2,3,4};

  String[] Mes = {"JAN", "FEV", "MAR", "ABR", "MAI", "JUN",
  "JUL", "AGO","SET", "OUT", "NOV", "DEZ"};

Observe que um vetor pode ser declarado, construído e inicializado ao mesmo tempo.

Exemplo de Vetores

Esse exemplo mostra os meses e a quantidade de dias que cada mês possui.

```
 class DiasDosMeses
{  public static void main(String[] arg)
   {   int[] maxDiasMes = new int[12];
       String[] nomeMes = {"JAN", "FEV", "MAR", "ABR", "MAI", "JUN",
                                          "JUL", "AGO","SET", "OUT", "NOV", "DEZ"};
       for(int i=0; i < maxDiasMes.length; i++)
       {  if(((i+1 < 8) && ((i+1)%2==1)) || ((i+1 >= 8) && ((i+1)%2==0)))
             maxDiasMes[i] = 31;
          else
             maxDiasMes[i] = 30;
       }
       maxDiasMes[1] = 28;
       for(int i=0;i<12;i++)
           System.out.println(nomeMes[i]+":"+ maxDiasMes[i]);
   }
}
```

Matrizes

- Os arrays multidimensionais funcionam de forma análoga aos arrays dimensionais.
- Cada dimensão é representada por um par de colchetes [].
- A propriedade length, quando associada a matriz, retorna o número de linhas
- A propriedade length, quando associada a uma linha, retorna o número de colunas

Ex.:

```
int[][] x = new int[3][5];
int y=x.length; y terá o valor 3
int w=x[0].length;       w terá o valor 5
```

Matrizes

- Os arrays multidimensionais funcionam de forma análoga aos arrays dimensionais.
- Cada dimensão é representada por um par de colchetes [].

Exemplo de Matrizes

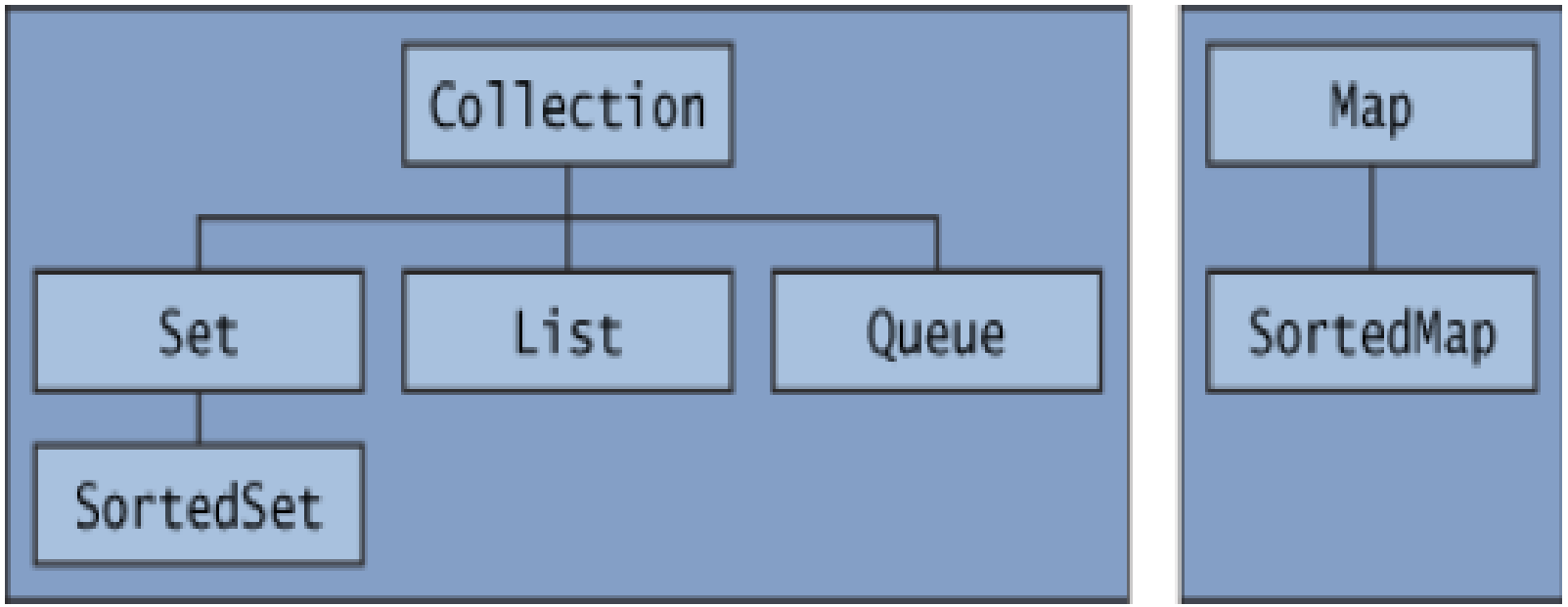
```
class ManipulaMatriz
 {   public static void main(String args[])
     {    int[][] Mat = new int[5][2];
          for( int i=0; i < Mat.length; i++)
          {   for( int j=0; j < Mat[0].length; j++)
              {  Mat[i][j]= (i*2)+j;
                   System.out.print(" Mat[ "+ i +" ] " +"[ " + j + " ] = " + Mat[i][j]);
              }
              System.out.println(“ ");
          }
     }
 }
```

Declaração e construção de uma matriz.

Coleções

- Objeto utilizado para armazenar vários outros objetos
- Pacote Java.util
- Interface Collection

Java Collection Framework



Interface Collection

- **public interface Collection<E> extends Iterable<E> {**
- //Operações básicas
- **int size();**
- **boolean isEmpty();**
- **boolean contains(Object element);**
- **boolean add(E element);**
- **boolean remove(Object element);**
- Iterator iterator();

- //Operações na coleção
- **boolean containsAll(Collection<?> c);**
- **boolean addAll(Collection<? extends E> c);**
- **boolean removeAll(Collection<?> c);**
- **boolean retainAll(Collection<?> c);**
- **void clear();**

- }

Conjuntos

- Modela um conjunto, portanto não há elementos repetidos
- Pode estar ordenado ou não
- Tipos:
  - **HashSet:** implementação de Set, modela conjuntos não ordenados (usa tabela hash);
  - **TreeSet:** implementado com árvore
  - **LinkedHashSet:** implementado com tabela hash e lista encadeada

Conjuntos (2)

- Na inicialização é informado o tipo do elemento

```
public static void main(String[] args) {
Set<String> nomes = new HashSet<String>();
nomes.add("Joao");
nomes.add("Jose");
nomes.add("Maria");
nomes.add("Bianca");
System.out.println("Qtd elementos: "+nomes.size());
if (nomes.contains("Maria"))
System.out.println("Contém Maria");
}
```

Conjuntos (3)

- Iterator
  - Acessar cada elemento
  - *boolean hasNext() - informa se ainda há*
  - *elementos a serem "visitados"*
  - *<T> next() - retorna o próximo elemento a ser*
  - *visitado*

- For-each
  - *for (String nome: nomes) { ... }*

```
public static void main(String[] args) {
Collection<String> nomes = new TreeSet<String>();
...
System.out.println("Qtd elementos: "+nomes.size());
Iterator<String> iterator = nomes.iterator();
while (iterator.hasNext()){
   System.out.println(iterator.next());
}
}
```

```
public static void main(String[] args) {
Collection<String> nomes = new HashSet<String>();
...
System.out.println("Qtd elementos: "+nomes.size());
for (String n : nomes) {
System.out.println(n);
}
}
```

Listas

- List: modela listas de dados, onde os elementos (repetidos ou não) estão ordenados;
- Collection Ordenada
- 2 tipos
  - **ArrayList: List implementado com arrays**
  - LinkedList: é uma Lista, onde os elementos estão ligados. Tem uma inserção e deleção muito mais rápidos que ArrayList.

```
public static void main(String[] args) {
ArrayList<String> nomes = new
ArrayList<String>();
nomes.add("João");
nomes.add("José");
nomes.add("Maria");
nomes.add("Bianca");
System.out.println("Qtd elementos:
"+nomes.size());
String string = nomes.get(3);
System.out.println(string);
}
```

Exercício

Crie uma classe ligação telefônica. Uma ligação telefônica possui como atributos o número do telefone que a originou, o nome da localidade de origem, o número e o local de destino da ligação, o valor total da ligação, o momento de início e o momento de término da ligação. Para representar um momento, crie uma classe de nome Tempo. Esta classe representará uma hora, minuto e segundo. A classe que representa a ligação telefônica forneça os seguintes métodos:

- Método que permita criar uma ligação fornecendo o momento do inicio, o local e o número de origem e o local e o número de destino da ligação.
- Método que calcule o valor da ligação. O valor da ligação será correspondente a R$ 1.00 por minuto. Mesmo que o usuário fale por 30s será cobrado um minuto. Divida as responsabilidades e construa os métodos nas classes mais apropriadas.
- Método que receba como parâmetro um número de telefone e informe se a ligação foi originada ou se destinava ao número informado. Exemplo: para uma ligação originada do número 99999999 e que se destinava ao número 2222222. O método deve retornar positivamente a mensagem que pergunta se 99999999 é um telefone envolvido e a mensagem que pergunta se 2222222 é um telefone envolvido e negativamente para qualquer outra.

Exercícios para casa

Lista de exercícios nr. 3