Linguagem de Programação I

Aula 8
Coleções de Objetos Genéricas – Classes e Métodos

l.bertholdo@ifsp.edu.br

Conteúdo

- Coleções de Objetos
  - Listas de Objetos
    - ArrayList
    - LinkedList
  - Conjuntos de Objetos
    - HashSet
    - TreeSet
  - Mapas de Objetos

l.bertholdo@ifsp.edu.br

Coleções de Objetos

- Arrays são muito úteis, porém têm algumas limitações:
  - Não é possível aumentar ou diminuir seu tamanho.
  - A ordenação dos elementos de um array não é simples.
  - Seus índices precisam ser números inteiros para que os dados possam ser acessados.
- As coleções visam facilitar o uso de estruturas de dados como listas, filas, pilhas, conjuntos e outras, tornando sua manipulação muito mais simples e transparente.
- As coleções de objetos são implementadas por meio de interfaces e classes presentes no pacote **java.util** da API Java, o qual deve ser importado pelas classes que as utilizarem.

Coleções de Objetos

- As antigas coleções da API Java não eram **genéricas**, pois eram capazes de armazenar apenas instâncias da classe **Object**.

- Com isso, ao armazenar um elemento na coleção, essas classes precisavam converter o tipo de dado do elemento para o tipo **Object** e, ao recuperá-los, precisam fazer a operação inversa.

- As atuais coleções também armazenam instâncias da classe **Object**, porém também permitem armazenar outros tipos de objetos. Por isso, são consideradas “coleções genéricas”.

```
ArrayList a1 = new ArrayList(); // Armaneza objetos do tipo Object.
ArrayList<String> a2 = new ArrayList<>(); // Armaneza objetos do tipo String.
ArrayList<Integer> a3 = new ArrayList<>(); // Armaneza objetos do tipo Integer.
```

l.bertholdo@ifsp.edu.br

Coleções de Objetos

- As classes da API Java que representam coleções armazenam somente objetos.

- Assim, para que uma coleção possa armazenar um tipo de dado nativo (int, float, double, char etc), é preciso associá-la à classe da API Java que equivale a este tipo de dado.

```
ArrayList<Integer> a3 = new ArrayList<>(); // Armaneza objetos do tipo Integer.
ArrayList<int> a4 = new ArrayList<>(); // -> Erro, pois "int" é um tipo nativo.
```

l.bertholdo@ifsp.edu.br

Coleções de Objetos

- Cada tipo nativo tem sua respectiva classe, capaz de converter e manipular valores do seu tipo correspondente.

Tipo Nativo	Classe Equivalente
int	Integer
float	Float
double	Double
long	Long
char	Character
boolean	Boolean

- Isso é possível graças aos mecanismos ***autoboxing*** *e* ***autounboxing***, que permitem atribuir tipos nativos a instâncias de classes e vice-versa.

```
Integer x = 2; // autoboxing
int y = x;     // autounboxing
x = y;         // autoboxing
```

l.bertholdo@ifsp.edu.br

Listas de Objetos

- Listas de objetos funcionam de forma similiar a um array comum, porém com recursos adicionais e a diferença de que a alocação dos elementos na memória é feita dinamicamente.
- As listas de objetos são encapsuladas por classes que implementam a interface **List**, a qual declara métodos específicos para manipulação de listas.
- Principais características:
  - Ao criar uma lista, seu tamanho não precisa ser especificado, já que ele aumenta ou diminui conforme os elementos são inseridos ou removidos da coleção.
  - Nas listas de objetos os elementos têm posições definidas e podem se repetir.

l.bertholdo@ifsp.edu.br

Listas de Objetos

- Há duas classes que implementam a interface **List** visando a representação de listas:
  - **ArrayList** – Possui um mecanismo mais rápido que um array comum, exceto para as operações de inserção e remoção.
  - **LinkedList** – Possui um mecanismo mais rápido para operações de inserção e remoção, porém é mais lento para acesso sequencial aos elementos da lista.
- Ao criar uma instância destas classes, é preciso definir o tipo de objeto (dado) que a lista irá armazenar. Por exemplo:
  ```
  ArrayList<Integer> lista = new ArrayList<>();
  ```

ArrayList e LinkedList

- Principais métodos:
  - **add:** Adiciona um elemento no final ou em uma dada posição da lista, considerando que a 1ª posição é zero.

    Sintaxe: <nome_lista>.add(<elemento>)
    Sintaxe: <nome_lista>.add(<posição>, <elemento>)
  - **remove:** Remove um elemento da lista por meio de sua posição ou por meio da 1ª ocorrência do elemento na lista.

    Sintaxe: <nome_lista>.remove(<posição>)
    Sintaxe: <nome_lista>.remove(<elemento>)
  - **get:** Retorna um elemento por meio de sua posição na lista.

    Sintaxe: <nome_lista>.get(<posição>)
  - **size:** Retorna quantos elementos a lista possui.

    Sintaxe: <nome_lista>.size()

ArrayList e LinkedList

```java
public class Exemplo_ArrayList {
    public static void main(String[] args) {
        // Cria uma lista de objetos ArrayList vazia.
        ArrayList<String> alunos = new ArrayList<>();

        //Adiciona os elementos à lista.
        alunos.add("João");
        alunos.add("Maria");
        alunos.add("Pedro");
        alunos.add("Carlos");
        // Adicionando o elemento "João" novamente, agora na posição 2.
        alunos.add(2, "João");
        System.out.println(alunos); // Imprime os elementos da lista.

        alunos.remove("Pedro"); // Remove o elemento "Pedro" da lista.
        alunos.remove(0); // Remove o elemento que está na posição zero na lista.
        System.out.println(alunos); // Imprime os elementos da lista.

        // Obtém o elemento que está na posição 1 na lista.
        System.out.println("2º aluno: " + alunos.get(1));

        // Retorna o número de elementos contidos na lista.
        System.out.println("Número de alunos: " + alunos.size());
    }
}
```

```
[João, Maria, João, Pedro, Carlos]
[Maria, João, Carlos]
2º aluno: João
Número de alunos: 3
```

ArrayLists e LinkedLists também podem ser inicializados já em sua declaração por meio da classe **Arrays**:

```java
ArrayList<String> alunos = new ArrayList<>(Arrays.asList("João", "Maria", "Pedro"));
```

ArrayList e LinkedList

- Principais métodos:
  - **set:** Substitui um elemento em uma dada posição da lista, considerando que a 1ª posição é zero.
    Sintaxe: <nome_lista>.set(<posição>, <novo_elemento>)
  - **contains:** Verifica se uma lista contém um determinado elemento.
    Sintaxe: <nome_lista>.contains(<elemento>)
  - **indexOf:** Retorna a posição da 1ª ocorrência do elemento. Se o elemento não existir, retorna -1.
    Sintaxe: <nome_lista>.indexOf(<elemento>)
  - **lastIndexOf:** Retorna a posição da última ocorrência do elemento. Se o elemento não existir, retorna -1.
    Sintaxe: <nome_lista>.lastIndexOf(<elemento>)
  - **isEmpty:** Verifica se uma lista está vazia.
    Sintaxe: <nome_lista>.isEmpty()

ArrayList e LinkedList

```java
public class Exemplo_ArrayList2 {
    public static void main(String[] args) {
        // Cria uma lista de objetos ArrayList vazia.
        ArrayList<String> alunos = new ArrayList<>();

        //Adiciona os elementos à lista.
        alunos.add("João");
        alunos.add("Maria");
        alunos.add("Pedro");
        alunos.add("Carlos");

        alunos.set(2, "Julia"); // Substitui o elemento "Pedro" por "Julia".
        System.out.println(alunos); // Imprime os elementos da lista.

        if (alunos.contains("Julia")) // Verifica se a lista contém o elemento "Julia".
            System.out.println("A lista contém a aluna Julia.");          "Julia"

        // Retorna a posição da 1ª ocorrência do elemento "Maria".
        System.out.println("Posição da 1ª ocorrência de 'Maria': " + alunos.indexOf("Maria"));

        // Retorna a posição da última ocorrência do elemento "Carlos".
        System.out.println("Posição da última ocorrência de 'Carlos': " + alunos.lastIndexOf("Carlos"));

        if (alunos.isEmpty()) // Verifica se a lista está vazia.
            System.out.println("A lista está vazia.");
        else
            System.out.println("A lista não está vazia.");
    }
}
```

```
[João, Maria, Julia, Carlos]
A lista contém a aluna Julia.
Posição da 1ª ocorrência de 'Maria': 1
Posição da última ocorrência de 'Carlos': 3
A lista não está vazia.
```

LinkedList – Filas e Pilhas

- A classe **LinkedList** possui métodos adicionais que permitem simular estruturas de dados como filas e pilhas.
  - **Fila** – Estrutura de dado onde o 1º elemento a entrar é o 1º a sair (*FIFO – first-in, first-out*). É simulada por meio de métodos para inserção no fim e remoção no início de uma lista.
  - **Pilha** – Estrutura de dado onde o último elemento a entrar é o 1º a sair (*LIFO – last-in, first-out*). É simulada por meio de métodos para inserção e remoção no fim de uma lista.

l.bertholdo@ifsp.edu.br

LinkedList – Filas e Pilhas

- Principais métodos:
  - **addFirst:** Insere um elemento na 1ª posição de uma lista.
    Sintaxe: <nome_lista>.addFirst(<elemento>)
  - **addLast:** Insere um elemento na última posição de uma lista.
    Sintaxe: <nome_lista>.addLast(<elemento>)
  - **pollFirst:** Remove o elemento que está na 1ª posição de uma lista.
    Sintaxe: <nome_lista>.pollFirst()
  - **pollLast:** Remove o elemento que está na última posição de uma lista.
    Sintaxe: <nome_lista>.pollLast()
  - **peekFirst:** Retorna o elemento que está na 1ª posição de uma lista.
    Sintaxe: <nome_lista>.peekFirst()
  - **peekLast:** Retorna o elemento que está na última posição de uma lista.
    Sintaxe: <nome_lista>.peekLast()

l.bertholdo@ifsp.edu.br

LinkedList – Fila

```java
public class Exemplo_LinkedList {
    public static void main(String[] args) {
        // Cria uma fila de objetos LinkedList vazia.
        LinkedList<String> fila = new LinkedList<>();

        //Adiciona os elementos à fila.
        fila.add("João");
        fila.add("Maria");
        fila.add("Pedro");
        fila.add("Carlos");

        // Adiciona o elemento "Julia" na última posição da fila.
        fila.addLast("Julia");
        System.out.println(fila); // Imprime os elementos da fila.

        // Retorna o 1º elemento da fila.
        System.out.println("1º elemento da fila: " + fila.peekFirst());

        fila.pollFirst(); // Remove o 1º elemento da fila.
        System.out.println(fila); // Imprime os elementos da fila.
    }
}
```

```
[João, Maria, Pedro, Carlos, Julia]
1º elemento da fila: João
[Maria, Pedro, Carlos, Julia]
```

LinkedList – Pilha

```java
public class Exemplo_LinkedList2 {
    public static void main(String[] args) {
        // Cria uma pilha de objetos LinkedList vazia.
        LinkedList<String> pilha = new LinkedList<>();

        //Adiciona os elementos à pilha.
        pilha.add("João");
        pilha.add("Maria");
        pilha.add("Pedro");
        pilha.add("Carlos");

        // Adiciona o elemento "Julia" no topo da pilha (última posição).
        pilha.addLast("Julia");
        System.out.println(pilha); // Imprime os elementos da pilha.

        // Retorna o elemento que está no topo da pilha (último elemento).
        System.out.println("Elemento que está no topo da pilha: " + pilha.peekLast());

        pilha.pollLast(); // Remove o elemento que está no topo da pilha (último elemento).
        System.out.println(pilha); // Imprime os elementos da pilha.
    }
}
```

```
[João, Maria, Pedro, Carlos, Julia]
Elemento que está no topo da pilha: Julia
[João, Maria, Pedro, Carlos]
```

l.bertholdo@ifsp.edu.br

Conjuntos de Objetos

- Os conjuntos de objetos são encapsulados por classes que implementam a interface **Set**, a qual declara métodos específicos para manipulação de conjuntos.

- Principais características:

  - Ao criar um conjunto, seu tamanho não precisa ser especificado, já que ele aumenta ou diminui conforme os elementos são inseridos ou removidos.

  - Em um conjunto de objetos não são permitidos objetos duplicados.

Conjuntos de Objetos

- Há duas classes que implementam a interface **Set** visando a representação de conjuntos:
  - **HashSet** – Possui um mecanismo mais rápido para operações de alteração em conjuntos, porém não estabelece uma ordem crescente ou descrescente entre seus elementos.
  - **TreeSet** – Possui um mecanismo mais lento para operações de inserção e remoção em conjuntos, porém mantém a ordenação crescente de seus elementos.
- Ao criar uma instância destas classes, é preciso definir o tipo de objeto (dado) que o conjunto irá armazenar. Por exemplo:

```
HashSet<Integer> conjunto = new HashSet<>();
```

HashSet e TreeSet

- Principais métodos:
  - **add:** Adiciona um elemento ao conjunto.
    Sintaxe: <nome_conjunto>.add(<elemento>)
  - **remove:** Remove um elemento do conjunto.
    Sintaxe: <nome_conjunto>.remove(<elemento>)
  - **isEmpty:** Verifica se um conjunto está vazio.
    Sintaxe: <nome_conjunto>.isEmpty()
  - **contains:** Verifica se um conjunto contém um determinado elemento.
    Sintaxe: <nome_conjunto>.contains(<elemento>)
  - **size:** Retorna quantos elementos o conjunto possui.
    Sintaxe: <nome_conjunto>.size()
  - **clear:** Remove todos os elementos do conjunto.
    Sintaxe: <nome_conjunto>.clear()

l.bertholdo@ifsp.edu.br

HashSet

```
public class Exemplo_TreeSet {
    public static void main(String[] args) {
        // Cria um conjunto de objetos TreeSet vazio.
        HashSet<String> alunos = new HashSet<>();

        //Adiciona os elementos ao conjunto.
        alunos.add("João");
        alunos.add("Maria");
        alunos.add("Pedro");
        alunos.add("João"); // Tentando adicionar o elemento "João" novamente.
        alunos.add("Carlos");
        System.out.println(alunos); // Imprime os elementos do conjunto.

        alunos.remove("Pedro"); // Remove o elemento "Pedro" do conjunto.

        if (alunos.isEmpty()) // Verifica se o conjunto esá vazio.
            System.out.println("O conjunto está vazio.");
        else
            System.out.println("O conjunto não está vazio.");

        if (alunos.contains("Maria")) // Verifica se o conjunto contém o elemento "Maria".
            System.out.println("O conjunto contém a aluna Maria.");

        // Retorna o número de elementos contidos no conjunto.
        System.out.println("Número de alunos: " + alunos.size());

        System.out.println(alunos); // Imprime os elementos do conjunto.
    }
}
```

```
[João, Pedro, Maria, Carlos]
O conjunto não está vazio.
O conjunto contém a aluna Maria.
Número de alunos: 3
[João, Maria, Carlos]
```

Os alunos não são armazenados em ordem alfabética no conjunto.

HashSets e TreeSets também podem ser inicializados já em sua declaração por meio da classe **Arrays**:

```
HashSet<String> alunos = new HashSet<>(Arrays.asList("João", "Maria", "Pedro"));
```

TreeSet

```java
public class Exemplo_TreeSet {
    public static void main(String[] args) {
        // Cria um conjunto de objetos TreeSet vazio.
        TreeSet<String> alunos = new TreeSet<>();

        //Adiciona os elementos ao conjunto.
        alunos.add("João");
        alunos.add("Maria");
        alunos.add("Pedro");
        alunos.add("João"); // Tentando adicionar o elemento "João" novamente.
        alunos.add("Carlos");
        System.out.println(alunos); // Imprime os elementos do conjunto.

        alunos.remove("Pedro"); // Remove o elemento "Pedro" do conjunto.

        if (alunos.isEmpty()) // Verifica se o conjunto esá vazio.
            System.out.println("O conjunto está vazio.");
        else
            System.out.println("O conjunto não está vazio.");

        if (alunos.contains("Maria")) // Verifica se o conjunto contém o elemento "Maria".
            System.out.println("O conjunto contém a aluna Maria.");

        // Retorna o número de elementos contidos no conjunto.
        System.out.println("Número de alunos: " + alunos.size());

        System.out.println(alunos); // Imprime os elementos do conjunto.
    }
}
```

```
[Carlos, João, Maria, Pedro]
O conjunto não está vazio.
O conjunto contém a aluna Maria.
Número de alunos: 3
[Carlos, João, Maria]
```

Os alunos são armazenados em ordem alfabética no conjunto.

l.bertholdo@ifsp.edu.br

TreeSet

- Outros métodos:
  - **first:** Retorna o 1º elemento do conjunto.
    Sintaxe: <nome_conjunto>.first()
  - **last:** Retorna o último elemento do conjunto.
    Sintaxe: <nome_conjunto>.last()
  - **headSet:** Retorna os elementos antecessores de um dado elemento. **Sintaxe:** <nome_conjunto>.headSet(<elemento>)
  - **tailSet:** Retorna um dado elemento e seus elementos sucessores. **Sintaxe:** <nome_conjunto>.tailSet(<elemento>)
  - **subSet:** Retorna um subconjunto que vai do 1º elemento informado até o antecessor do 2º elemento informado.
    Sintaxe: <nome_conjunto>.subSet(<1º elemento>, <2º elemento>)

TreeSet

```java
public class Exemplo_TreeSet2 {
    public static void main(String[] args) {
        // Cria um conjunto de objetos TreeSet vazio.
        TreeSet<String> alunos = new TreeSet<>();

        //Adiciona os elementos ao conjunto.
        alunos.add("João");
        alunos.add("Maria");
        alunos.add("Pedro");
        alunos.add("Carlos");
        alunos.add("Raquel");
        System.out.println(alunos); // Imprime os elementos do conjunto.

        // Imprime o 1º elemento do conjunto.
        System.out.println("1º aluno do conjunto: " + alunos.first());

        // Imprime o último elemento do conjunto.
        System.out.println("Último aluno do conjunto: " + alunos.last());

        // Imprime os elementos antecessores do elemento "Maria".
        System.out.println("Alunos antecessores de Maria: " + alunos.headSet("Maria"));

        // Imprime o elemento "Maria" e seus sucessores.
        System.out.println("Maria e seus alunos sucessores: " + alunos.tailSet("Maria"));

        // Imprime o subconjunto que vai do elemento "João" até o antecessor de "Pedro".
        System.out.println("Subconjunto: " + alunos.subSet("João", "Pedro"));
    }
}
```

```
[Carlos, João, Maria, Pedro, Raquel]
1º aluno do conjunto: Carlos
Último aluno do conjunto: Raquel
Alunos antecessores de Maria: [Carlos, João]
Maria e seus alunos sucessores: [Maria, Pedro, Raquel]
Subconjunto: [João, Maria]
```

l.bertholdo@ifsp.edu.br

HashSet e TreeSet

- As classes **HashSet** e **TreeSet** também têm métodos para realizar operações matemáticas com conjuntos como **união**, **intersecção** e **diferença** entre conjuntos.
  - **addAll (união)** – Junta os elementos de dois conjuntos, descartando os elementos repetidos. O conjunto que chama o método é alterado passando a conter os elementos dos dois conjuntos.

    Sintaxe: <nome_conjunto1>.addAll(<nome_conjunto2>)
  - **retainAll (intersecção)** – Retorna os elementos que dois conjuntos têm em comum. O conjunto que chama o método é alterado, passando a conter apenas os elementos em comum.

    Sintaxe: <nome_conjunto1>.retainAll(<nome_conjunto2>)

HashSet e TreeSet

- **removeAll (diferença entre conjuntos)** – Retorna os elementos de um conjunto que não estão no outro conjunto. O conjunto que chama o método é alterado passando a conter apenas os elementos que não estão no outro conjunto.

  Sintaxe: <nome_conjunto1>.removeAll(<nome_conjunto2>)

- **containsAll:** Este método não modifica o conjunto que o chama. Ele apenas verifica se o conjunto passado como argumento está contido no conjunto que chamou o método.

  Sintaxe: <nome_conjunto1>.containsAll(<nome_conjunto2>)

HashSet e TreeSet

- Operação de União – Exemplo

```
public class Exemplo_TreeSet3 {
    public static void main(String[] args) {
        String[] c = {"João", "Pedro", "Paulo", "Maria", "Carlos"};
        TreeSet<String> clientes = new TreeSet<>();

        String[] f = {"Ana", "Paulo", "Antonio", "Isabel", "Maria"};
        TreeSet<String> fornecedores = new TreeSet<>();

        for (String i : c) //Adiciona os elementos ao conjunto de clientes.
            clientes.add(i);

        for (String i : f) //Adiciona os elementos ao conjunto de fornecedores.
            fornecedores.add(i);

        System.out.println("Clientes: " + clientes);
        System.out.println("Fornecedores: " + fornecedores);

        clientes.addAll(fornecedores);
        System.out.println("Clientes U Fornecedores: " + clientes);
    }
}
```

```
Clientes: [Carlos, João, Maria, Paulo, Pedro]
Fornecedores: [Ana, Antonio, Isabel, Maria, Paulo]
Clientes U Fornecedores: [Ana, Antonio, Carlos, Isabel, João, Maria, Paulo, Pedro]
```

HashSet e TreeSet

- Operação de Intersecção – Exemplo

```
public class Exemplo_TreeSet4 {
    public static void main(String[] args) {
        String[] c = {"João", "Pedro", "Paulo", "Maria", "Carlos"};
        TreeSet<String> clientes = new TreeSet<>();

        String[] f = {"Ana", "Paulo", "Antonio", "Isabel", "Maria"};
        TreeSet<String> fornecedores = new TreeSet<>();

        for (String i : c) //Adiciona os elementos ao conjunto de clientes.
            clientes.add(i);

        for (String i : f) //Adiciona os elementos ao conjunto de fornecedores.
            fornecedores.add(i);

        System.out.println("Clientes: " + clientes);
        System.out.println("Fornecedores: " + fornecedores);

        clientes.retainAll(fornecedores);
        System.out.println("Clientes ∩ Fornecedores: " + clientes);
    }
}
```

```
Clientes: [Carlos, João, Maria, Paulo, Pedro]
Fornecedores: [Ana, Antonio, Isabel, Maria, Paulo]
Clientes ∩ Fornecedores: [Maria, Paulo]
```

l.bertholdo@ifsp.edu.br

HashSet e TreeSet

- Operação de Diferença entre Conjuntos – Exemplo

```
public class Exemplo_TreeSet5 {
    public static void main(String[] args) {
        String[] c = {"João", "Pedro", "Paulo", "Maria", "Carlos"};
        TreeSet<String> clientes = new TreeSet<>();

        String[] f = {"Ana", "Paulo", "Antonio", "Isabel", "Maria"};
        TreeSet<String> fornecedores = new TreeSet<>();

        for (String i : c) //Adiciona os elementos ao conjunto de clientes.
            clientes.add(i);

        for (String i : f) //Adiciona os elementos ao conjunto de fornecedores.
            fornecedores.add(i);

        System.out.println("Clientes: " + clientes);
        System.out.println("Fornecedores: " + fornecedores);

        clientes.removeAll(fornecedores);
        System.out.println("Clientes - Fornecedores: " + clientes);
    }
}
```

```
Clientes: [Carlos, João, Maria, Paulo, Pedro]
Fornecedores: [Ana, Antonio, Isabel, Maria, Paulo]
Clientes - Fornecedores: [Carlos, João, Pedro]
```

l.bertholdo@ifsp.edu.br

HashSet e TreeSet

- Verificação de Subconjuntos – Exemplo

```
public class Exemplo_TreeSet6 {
    public static void main(String[] args) {
        String[] c1 = {"João", "Paulo", "Antonio", "Maria"};
        TreeSet<String> conjunto1 = new TreeSet<>();

        String[] c2 = {"Paulo", "Maria"};
        TreeSet<String> conjunto2 = new TreeSet<>();

        for (String i : c1) //Adiciona os elementos ao conjunto 1.
            conjunto1.add(i);

        for (String i : c2) //Adiciona os elementos ao conjunto 2.
            conjunto2.add(i);

        System.out.println("Conjunto 1: " + conjunto1);
        System.out.println("Conjunto 2: " + conjunto2);

        if (conjunto1.containsAll(conjunto2))
            System.out.println("O conjunto 1 contém o conjunto 2.");
        else
            System.out.println("O conjunto 1 não contém o conjunto 2.");
    }
}
```

```
Conjunto 1: [Antonio, João, Maria, Paulo]
Conjunto 2: [Maria, Paulo]
O conjunto 1 contém o conjunto 2.
```

l.bertholdo@ifsp.edu.br

Mapas de Objetos

- Mapas de objetos são similiares às listas de objetos, com a diferença de que suas posições não precisam ser representadas por números inteiros.
- Cada elemento de um mapa é representado por um par de objetos denominados **chave** e **valor**.
- Principais características:
  - Ao criar um mapa, seu tamanho não precisa ser especificado, já que ele aumenta ou diminui conforme os elementos são inseridos ou removidos da coleção.
  - Da mesma forma que nas listas, nos mapas de objetos os elementos podem se repetir. Porém, não são permitidas chaves repetidas.

l.bertholdo@ifsp.edu.br

Mapas de Objetos

- Os mapas de objetos são encapsulados por classes que implementam a interface **Map**, a qual declara métodos específicos para manipulação de mapas.
- Há duas classes que implementam a interface **Map** visando a representação de mapas:
  - **HashMap** – Possui um mecanismo mais rápido nas operações de inserção e recuperação de objetos.
  - **TreeMap** – Possui um mecanismo mais lento, porém mantém a ordem das chaves de seus elementos.
- Ao criar uma instância destas classes, é preciso definir os tipos de objetos (dados) da chave e do valor a serem armazenados.
  Por exemplo: `HashMap<Integer, String> mapa = new HashMap<>();`

l.bertholdo@ifsp.edu.br

Referências

- H. M. Deitel; P. J. Deitel; Java Como Programar – 4ª Edição. Bookman, 2003.
- Rafael Santos; Introdução à Programação Orientada a Objetos usando Java – 2ª edição. Rio de Janeiro: Elsevier, 2013.
- http://docs.oracle.com/javase/8/docs/api/index.html