

Aula 12

Coleções

Programação III

Prof. Augusto César Oliveira

augusto.oliveira@unicap.br

Na aula passada...

- Amadurecer o conceito de polimorfismo na programação orientada a objetos;
- Entender como o polimorfismo permite que objetos de diferentes classes sejam tratados de maneira uniforme;
- Compreender o conceito de interfaces e seu papel na programação orientada a objetos;
- Aprender a criar e implementar interfaces em Java.

O objetivo da aula de hoje...

- Compreender a importância das coleções;
- Conhecer os principais tipos de coleções;
- Aprender a criar e inicializar coleções;
- Inserir, recuperar e remover elementos em coleções;
- Ordenar elementos em coleções.

Contextualizando...

- Em Java é possível criar **arrays de tamanho fixos** de itens de **tipos primitivos** ou **objetos personalizados**.

```
// declaração de array  
int[] x, y;  
int z[];  
  
// criação de 10 posições  
x = new int[10];
```

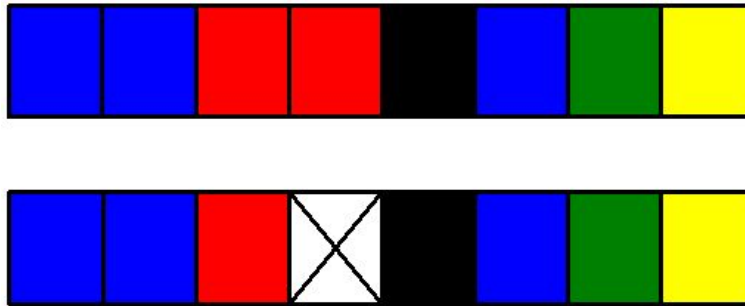
Problema!

- Depois de criado, o **array não pode ser redimensionado**.
- Uma alternativa para isso é **copiar os elementos de um array para outro**.

```
int[] x = {1,2,3};  
int[] y = new int[5];  
System.arraycopy(x, 0, y, 0, 3);  
  
for (int i = 0; i < y.length; i++) {  
    System.out.println(y[i]);  
} // será impresso 1,2,3,0 e 0
```

Outras dificuldades que arrays apresentam

- Não conseguimos saber **quantas posições do array já foram utilizadas** sem antes criar **métodos auxiliares de verificação**.



Retire a quarta Conta

`conta[3] = null;`

2. Coleções

Coleções



O que é uma "coleção"?

- Uma **coleção** em Java se refere a um **grupo de elementos** relacionados que são **armazenados e manipulados em conjunto**.
- As coleções são utilizadas para: **armazenar, recuperar, manipular dados e percorrer dados**.
- Exemplos de "coleções" do mundo real:
 1. Uma mão de baralho (coleção de cartas);
 2. Um *mail folder* (uma coleção de e-mails);
 3. Uma agenda telefônica (coleção de nomes e telefones).

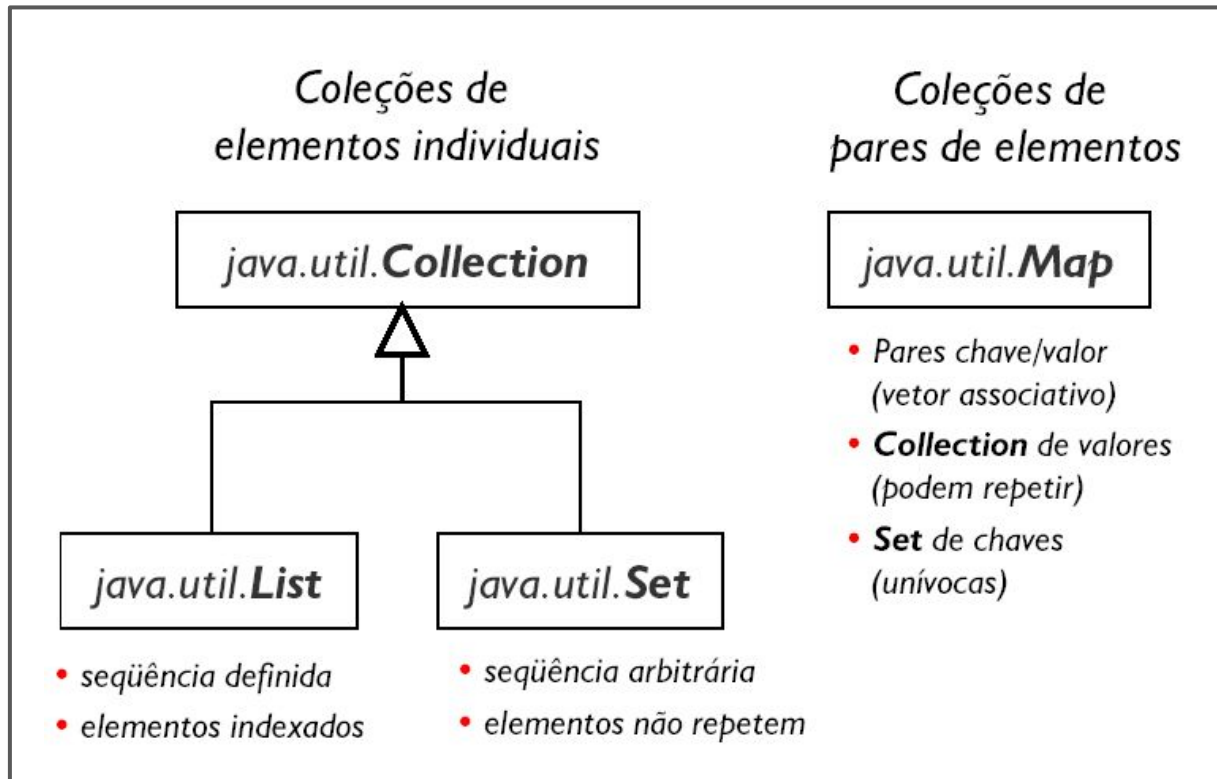
Java Collections Framework

- O "Java Collection Framework" (ou Estrutura de Coleções Java) é uma **biblioteca padrão** em Java que fornece um **conjunto de interfaces, classes e algoritmos** para manipular e armazenar **coleções de objetos**.
- A estrutura de coleções está definida no pacote `java.util` e foi introduzida na versão 1.2 do Java.
- Não é necessário reinventar a roda!

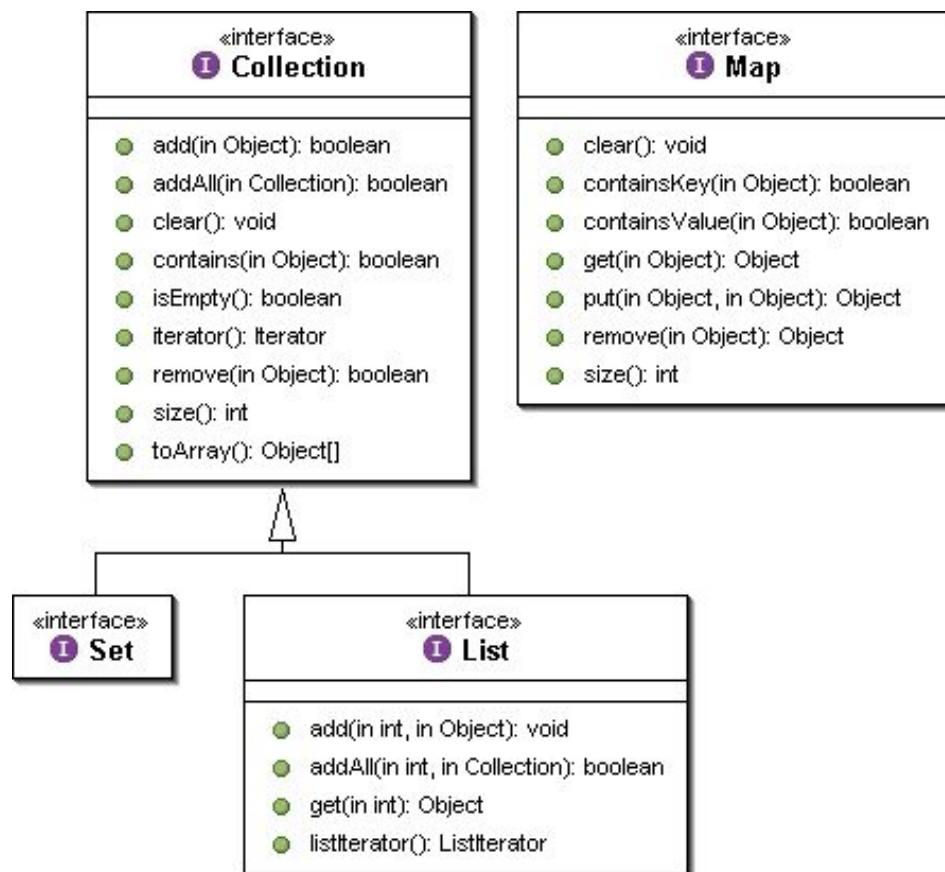
Vantagens do uso de "coleções"

1. Reduzem o esforço de programação;
2. Aumentam a velocidade e a qualidade do programa;
3. Permitem a interoperabilidade entre apis não relacionadas;
4. Reduzem o esforço para aprender e utilizar novas apis;
5. Reduzem o esforço para se projetar novas apis;
6. Estimulam o reúso de software.

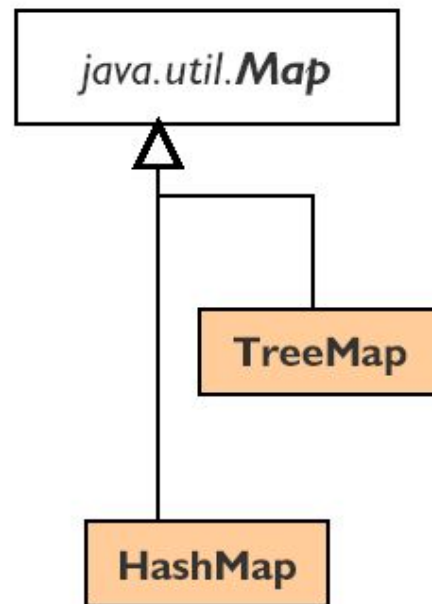
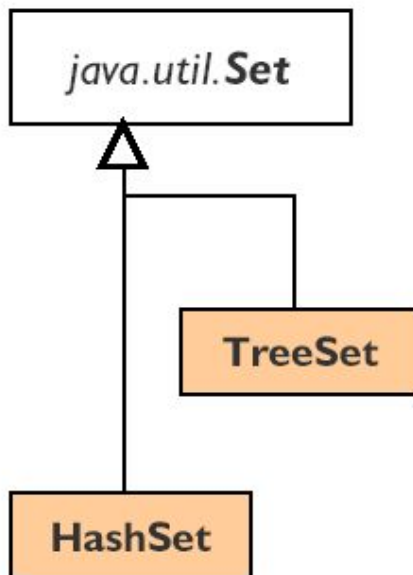
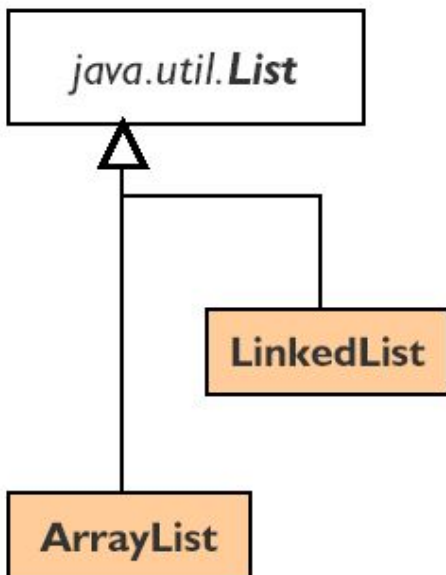
Interfaces de "coleções"



Métodos das interfaces "Collection" e "Map"



Classes concretas



3.

Interface "Set"

Coleções


A interface "Set"

- Estende de: `Collection`;
- **Não aceita elementos repetidos;**
- **Não garante a ordem entre os elementos;**
- Não implementa novos métodos.

Exemplo: classe "HashSet"

```
import java.util.HashSet;
public class Set {
    public static void main(String[] args) {

        HashSet<String> cars = new HashSet<String>();
        cars.add("Volvo");
        cars.add("BMW");
        cars.add("Ford");
        cars.add("BMW"); // Iteração
        cars.add("Mazda");
        cars.add("Volvo");
        System.out.println(cars);
    }
}
```



The screenshot shows an IDE window with tabs for 'Console', 'Problems', and 'Javadoc'. The 'Console' tab is active, displaying the output of the Java application: '<terminated> Set [Java Application] C:\Users\g...' followed by the set contents '[Volvo, Mazda, Ford, BMW]'. The text is color-coded, with 'Set' in blue, '[Java Application]' in green, and the set elements in black.

4.

Interface "List"

Coleções



A interface "List"

- Estende de: `Collection`;
- **Aceita elementos repetidos**;
- Mantém uma **ordenação específica** entre os elementos (**sequência**).
- Ela resolve todos os problemas os quais levantamos em relação à array (**busca**, **remoção**, **tamanho infinito**, etc.)

Exemplo: classe "ArrayList"

```
import java.util.ArrayList;

public class List {

    public static void main(String[] args) {
        ArrayList<String> nomes = new ArrayList();
        nomes.add("João");
        nomes.add("Maria");
        nomes.add("Eduardo");
        nomes.add("Silvana");
        nomes.add("Mário");

        System.out.println("Lista de nomes: " + nomes);
    }
}
```

Exemplo: classe "ArrayList"

```
import java.util.ArrayList;

public class List {

    public static void main(String[] args) {
        ArrayList<String> nomes = new ArrayList();
        nomes.add("João");
        nomes.add("Maria");
        nomes.add("Eduardo");
        nomes.add("Silvana");
        nomes.add("Mário");

        System.out.println("Lista de nomes: " + nomes);
    }
}
```

5.

Interface "Map"

Coleções



A interface "Map"

- Representada através de **pares chave-valor**, onde cada chave é mapeado para um valor correspondente;
- **Não possui chaves duplicadas;**
- Cada **chave** leva **somente um elemento**.

Exemplo: classe "HashMap"

```
import java.util.HashMap;

public class Ex_Map {

    public static void main(String[] args) {

        HashMap<String, Integer> agenda = new HashMap<String, Integer>();
        agenda.put("Luma", 11222);
        agenda.put("Alex", 22333);
        agenda.put("Andrea", 3344);

        System.out.println(agenda);
    }
}
```

Exemplo: classe "HashMap"

```
import java.util.HashMap;  
  
public class Ex_Map {  
  
    public static void main(String[] args) {
```

```
        HashMap<String, Integer>  
        agenda = new HashMap<>();  
        agenda.put("Luma", 11222);  
        agenda.put("Alex", 22333);  
        agenda.put("Andrea", 33444);
```

```
        System.out.println(agenda);  
    }  
}
```



6.

Outras classes

Coleções



Exemplo: classe "SortedSet"

```
import java.util.TreeSet;
import java.util.SortedSet;

public class Ex_SortedSet {

    public static void main(String[] args) {

        SortedSet<Integer> jogo = new TreeSet<Integer>();
        jogo.add(10);
        jogo.add(1);
        jogo.add(55);
        jogo.add(12);
        jogo.add(4);
        System.out.println(jogo);
    }
}
```

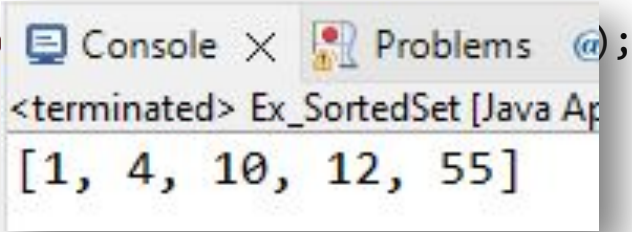
Exemplo: classe "SortedSet"

```
import java.util.TreeSet;
import java.util.SortedSet;

public class Ex_SortedSet {

    public static void main(String[] args) {

        SortedSet<Integer> jogo = new TreeSet<>();
        jogo.add(10);
        jogo.add(1);
        jogo.add(55);
        jogo.add(12);
        jogo.add(4);
        System.out.println(jogo);
    }
}
```



Exemplo: classe "SortedMap"

```
import java.util.SortedMap;
import java.util.TreeMap;

public class Ex_SortedMap {

    public static void main(String[] args) {

        SortedMap<Integer, String> sm = new TreeMap<Integer, String>();
        sm.put(2, "practice");
        sm.put(3, "quiz");
        sm.put(5, "code");
        sm.put(4, "contribute");
        sm.put(1, "geeksforgeeks");

        System.out.println(sm);

    }
}
```

Exemplo: classe "SortedMap"

```
import java.util.SortedMap;
import java.util.TreeMap;

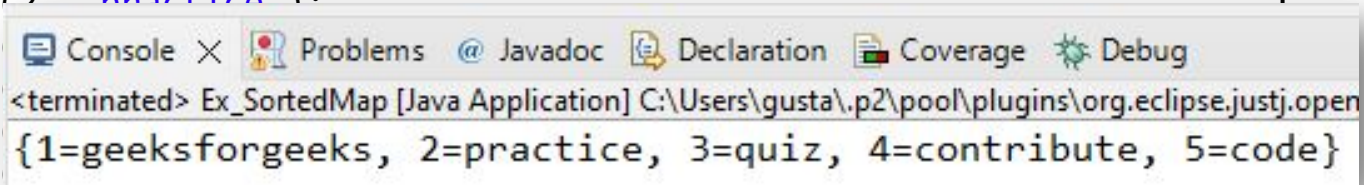
public class Ex_SortedMap {

    public static void main(String[] args) {

        SortedMap<Integer, String> sm = new TreeMap<Integer, String>();
        sm.put(1, "geeksforgeeks");
        sm.put(2, "practice");
        sm.put(3, "quiz");
        sm.put(4, "contribute");
        sm.put(5, "code");

        System.out.println(sm);

    }
}
```



The screenshot shows the Eclipse IDE interface with a console window open. The console displays the output of the Java application: {1=geeksforgeeks, 2=practice, 3=quiz, 4=contribute, 5=code}. The console window also shows the title bar with tabs for Console, Problems, Javadoc, Declaration, Coverage, and Debug.

Quando usar cada uma delas?

- **Set**: para coleções que **não** aceitam elementos duplicados.
- **List**: se existem **entradas duplicadas**;
- **Map**: para coleções compostas por **pares chave/valor**.

7.

Considerações finais

Coleções



O que aprendemos hoje?

- Compreender a importância das coleções;
- Conhecer os principais tipos de coleções;
- Aprender a criar e inicializar coleções;
- Inserir, recuperar e remover elementos em coleções;
- Ordenar elementos em coleções.

Próxima aula...

-

8.

Exercício de fixação

Teams



Coleções

- **Link da atividade**: clique aqui.

Aula 12

Coleções

Programação III

Prof. Augusto César Oliveira

augusto.oliveira@unicap.br