INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
BAHIA

Collections Framework

Conjunto: java.util.Set

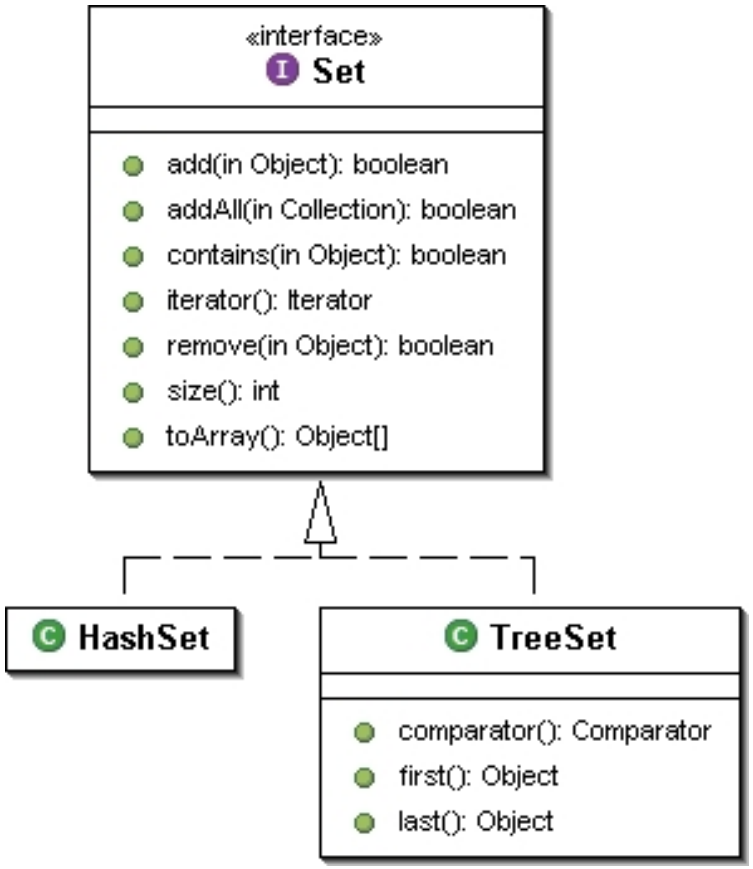
- Um **conjunto (Set)** funciona de forma **análoga** aos **conjuntos da matemática**, ele é uma coleção que **não permite elementos duplicados**.

- Outra característica **fundamental** dele é o fato de que a **ordem** em que os elementos são armazenados pode **não ser a ordem** na qual eles **foram inseridos** no conjunto

Conjunto: java.util.Set

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA BAHIA

- **Um conjunto** é representado **pela interface Set** e tem como suas **principais implementações** as classes **HashSet**, **LinkedHashSet** e **TreeSet**:



Conjunto: java.util.Set

- O código a seguir **cria um conjunto** e adiciona diversos elementos, e alguns repetidos:

```
Set<String> cargos = new HashSet<String>();

cargos.add("Gerente");
cargos.add("Diretor");
cargos.add("Presidente");
cargos.add("Secretária");
cargos.add("Funcionário");
cargos.add("Diretor"); // repetido!

// imprime na tela todos os elementos
System.out.println(cargos);
```

- Aqui, o **segundo Diretor** não será **adicionado** e o método add lhe retornará **false**.

Percorrer Colleções

- Como **percorrer** os **elementos** de uma **coleção**?

- **Se for uma lista**, podemos sempre utilizar um **laço for**, invocando o método **get** para cada elemento. Mas e se a coleção **não permitir indexação**?

- Por exemplo, um **Set não possui** um **método** para pegar o primeiro, o segundo ou o quinto elemento do conjunto, já que **um conjunto não possui o conceito de "ordem"**

Percorrer Colleções

- Podemos usar o **enhanced-for** (o "foreach") do Java 5 para **percorrer qualquer Collection** sem nos preocupar com isso;

```
Set<String> conjunto = new HashSet<String>();

conjunto.add("java");
conjunto.add("vraptor");
conjunto.add("scala");

for (String palavra : conjunto) {
    System.out.println(palavra);
```

- Em que **ordem** os **elementos** serão **acessados**?
- Numa **lista**, os elementos aparecerão **de acordo** com o **índice** em que **foram inseridos**, isto é, de acordo com o que foi pré-determinado.
- Já no **Set depende** da **implementação** usada;

Map

Mapas - Map

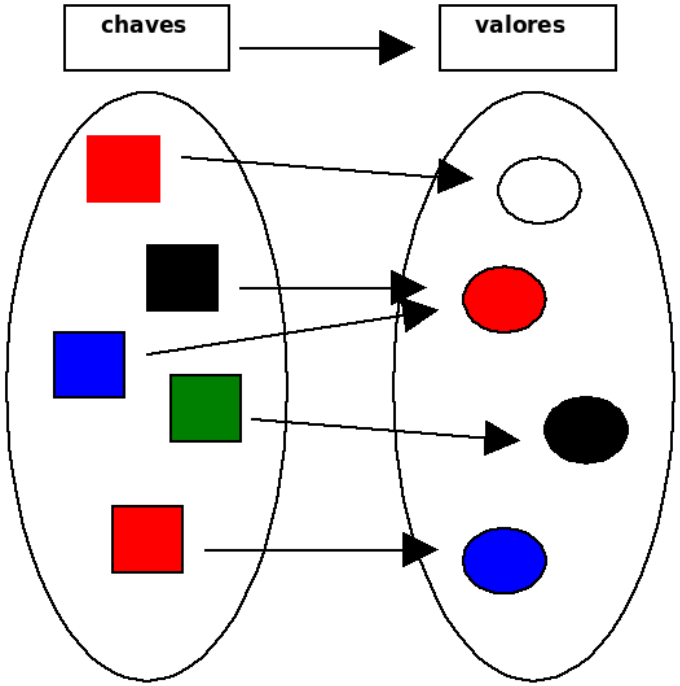
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA BAHIA

- Muitas vezes queremos **buscar rapidamente** um objeto dado alguma **informação sobre ele;**
- Um exemplo seria, dada a **placa do carro**, obter todos os **dados do carro**.
- Poderíamos utilizar uma lista para isso e percorrer **todos os seus elementos**, mas isso pode ser **péssimo para a performance**, mesmo para listas não muito grandes.
- **Um mapa é composto** por um conjunto de **associações** entre um **objeto chave** a um **objeto valor.**

Mapas - Map

- **java.util.Map** é um **mapa**, pois é possível usá-lo para mapear **uma chave a um valor**

- Exemplo: mapeie à chave "**rua**" ao valor "**Vergueiro**". Semelhante a associações de palavras que podemos fazer em um dicionário.



Possíveis ações em um mapa:

Mapeie uma chave a um valor
O que está mapeado na chave X?
Remapeie uma certa chave
Quero o conjunto de chaves.
Quero o conjunto de valores.
Desmapeie a chave X.

Mapas - Map

- O **método put(Object, Object)** da interface **Map** recebe a **chave e o valor** de uma nova associação.
- Para saber o que está associado a um **determinado objeto-chave**, passa-se esse objeto no método **get(Object)**
- Essas são as **duas operações principais** e mais frequentes **realizadas** sobre um **mapa**.

Mapas - Map

- Exemplo: criamos **duas contas** correntes e as colocamos em um mapa **associando-as** aos seus donos.

```
ContaCorrente c1 = new ContaCorrente();
c1.deposita(10000);

ContaCorrente c2 = new ContaCorrente();
c2.deposita(3000);

// cria o mapa
Map<String, ContaCorrente> mapaDeContas = new HashMap<String, ContaCorrente>();

// adiciona duas chaves e seus respectivos valores
mapaDeContas.put("diretor", c1);
mapaDeContas.put("gerente", c2);

// qual a conta do diretor? (sem casting!)
ContaCorrente contaDoDiretor = mapaDeContas.get("diretor");
System.out.println(contaDoDiretor.getSaldo());
```

- Um **mapa é muito usado para "indexar"** objetos de acordo com **determinado critério**, para podermos buscar esse objetos rapidamente

Mapas - Map

- Assim como as **coleções**, **Map** trabalha diretamente com **Objects** (tanto na chave quanto no valor), o que tornaria necessário o **casting** no momento que **recuperar** elemento;

- Usando os **generics**, como fizemos aqui, **não precisamos** mais do **casting**.

- Suas **principais** implementações são o **HashMap, o TreeMap e o Hashtable**.