Programação Orientada por Objetos

Iteradores

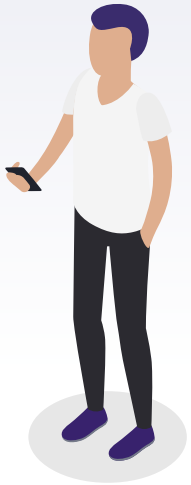
Prof. Cédric Grueau
Prof. José Sena Pereira
Departamento de Sistemas e Informática
Escola Superior de Tecnologia de Setúbal
Instituto Politécnico de Setúbal
2022/2023

Sumário

- Iterar sobre uma coleção
- Remover elementos de uma coleção
- Método **`equals`**
- Exemplo da agenda de contatos

A aplicação PlayList - continuação

- Uma aplicação CityAssets que permita gerir os bens de uma câmara Municipal
  - Deve poder:
    - Mostrar o número de bens
    - Calcular o custo mensal de manutenção dos bens
    - Apresentar a informação de um dado bem



Implementação parcial da classe **Assets**

```
import java.util.ArrayList;

public class Assets
{
    ArrayList<Asset> assets;

public Assets(Asset[] assets)
    {
     this.assets = new ArrayList<Asset>();
        for(Asset a : assets){
            this.assets.add(a);
        }
    }

 public String getInfo(int indice)
    {
      return assets.get(indice).toString();
    }

 public int size(){
    return assets.size();
    }

    public ArrayList<Asset> getAssets(){
    }

    public void removeAssets(int number){

    }
}
```

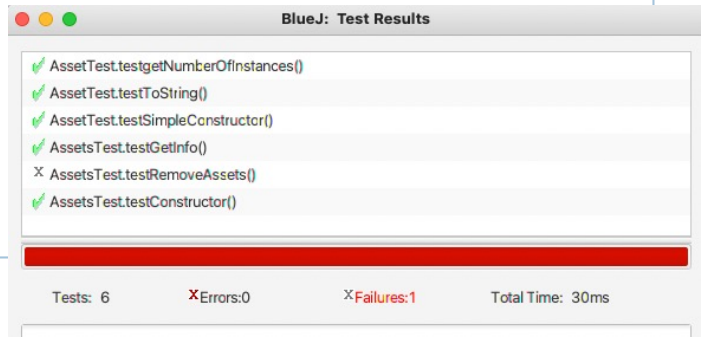
Teste de **removeAssets(Asset[] assets)**

```
public class AssetsTest
{
   private static final Asset[] SOME_ASSETS = {new Asset(130.0,0.0,10),new Asset(3500.0,250.0,5),
                          new Asset(39000.0,1000.0,1)};
    private static final Asset[] ASSETS_TO_REMOVE = {
        new Asset(130.0,0.0,10),new Asset(39000.0,1000.0,1)};
    Assets assets;

    @Test
    public void testRemoveAssets() {
        System.out.println("Teste de remoção de bens");
        assets.removeAssets(ASSETS_TO_REMOVE);
        assertEquals(1, assets.size());
    }
```

Declaramos variáveis de instância para o novo teste.

Executamos o teste: falha pela falta de implementação do método **removeAssets**



Implementação de **`removeAssets(Asset[] assets)`**

- Como definir que dois bens são idênticos?
  - **`==`** serve para verificar se os dois objetos têm a mesma referência
- Como definir o método **`equals()`**,tal como usamos até agora com o tipo **`String`**?
  - `var1.equals(var2)`
  - `asset1.equals(asset2)`

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

É suficiente?

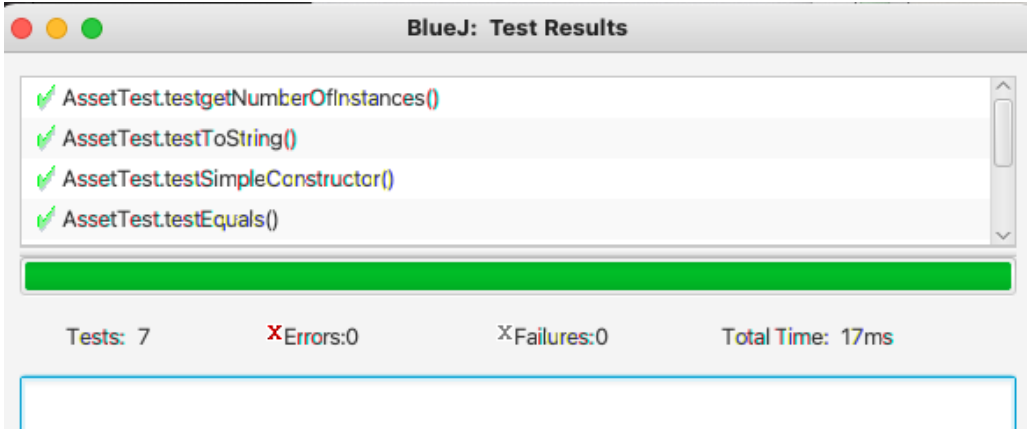
Método equals()

```
public boolean equals(Object obj) {
        if (this == obj) {
                return true; // Igualdade de referência.
        }
        if (!(obj instanceof Asset)) {
                return false; // Não é o mesmo tipo.
        }
        // Aceder aos campos do outro bem para verificar o estado.
        Asset other = (Asset) obj;
        return value == other.value &&
                manutentionCost == other.manutentionCost &&
                numberOfInstances == other. numberOfInstances;
}
```

Testar o método **equals()**

```
@Test
    public void testEquals(){
        Asset asset3 = new Asset(6700.0,29.0,3);
        assertEquals(asset3,asset2);

    }
```



Implementação de **removeAssets(Asset[] assets)**

```
public void removeAssets(Asset[] assets){
        for(int i = 0; i < assets.length; i++){
            for(int j = 0; j < this.assets.size(); j++){
                if (assets[i].equals(this.assets.get(j))){
                    this.assets.remove(this.assets.get(j));
                }
            }
        }
    }
```

- **Não se deve eliminar elementos de uma coleção desta forma. Não é seguro.**

Percorrer coleções com Iteradores

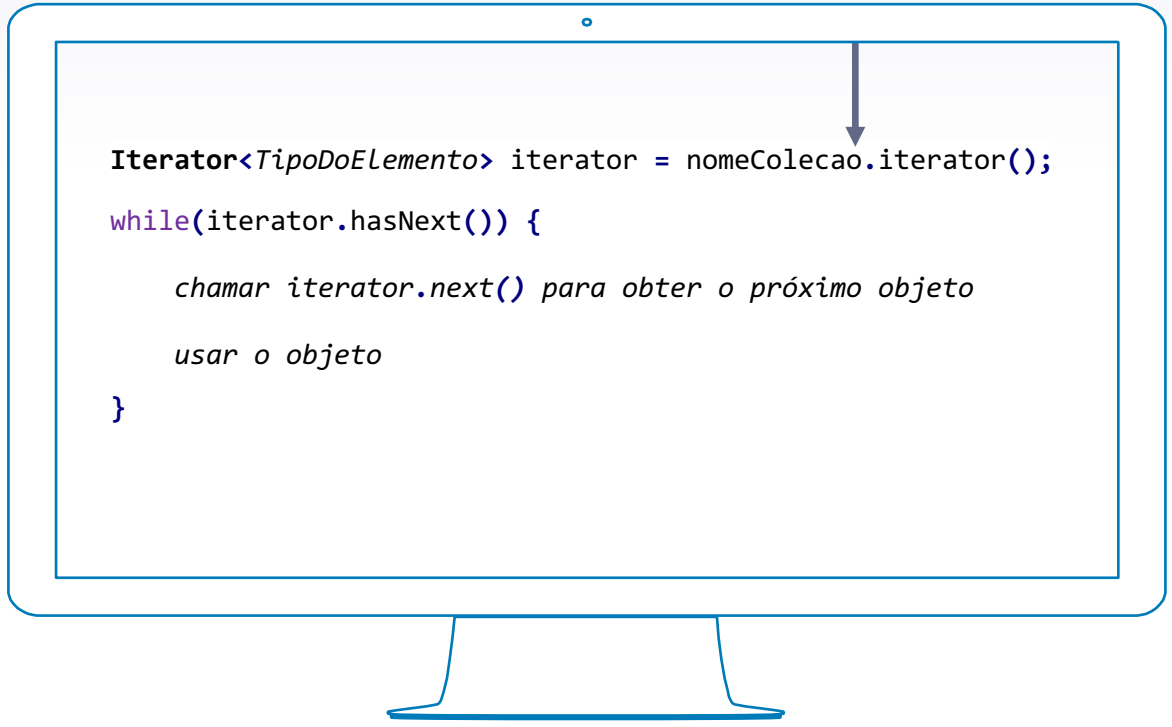
- Os objetos do tipo **`Iterator`** são utilizados para percorrer os vários elementos de uma coleção.
  - É possível obter estes objetos através do método **`iterator()`** existente na coleção. Todas as coleções do Java têm este método.
  - O objeto **`iterator`** possui três métodos:
    - **`hasNext()`** – diz se existe o próximo elemento.
    - **`next()`** – devolve o próximo elemento
    - **`remove()`** – remove o último elemento devolvido. Algumas coleções não permitem a utilização deste método.
  - Tal como nas coleções o tipo **`Iterator`** é genérico necessitando da informação de qual é o tipo que se está a iterar (o mesmo que é usado como elemento da coleção)
    - Ex: **`Iterator<Student> iterator`**

Percorrer coleções com Iteradores

- Utilização do objeto **Iterator**

Retorna o objeto **Iterator**

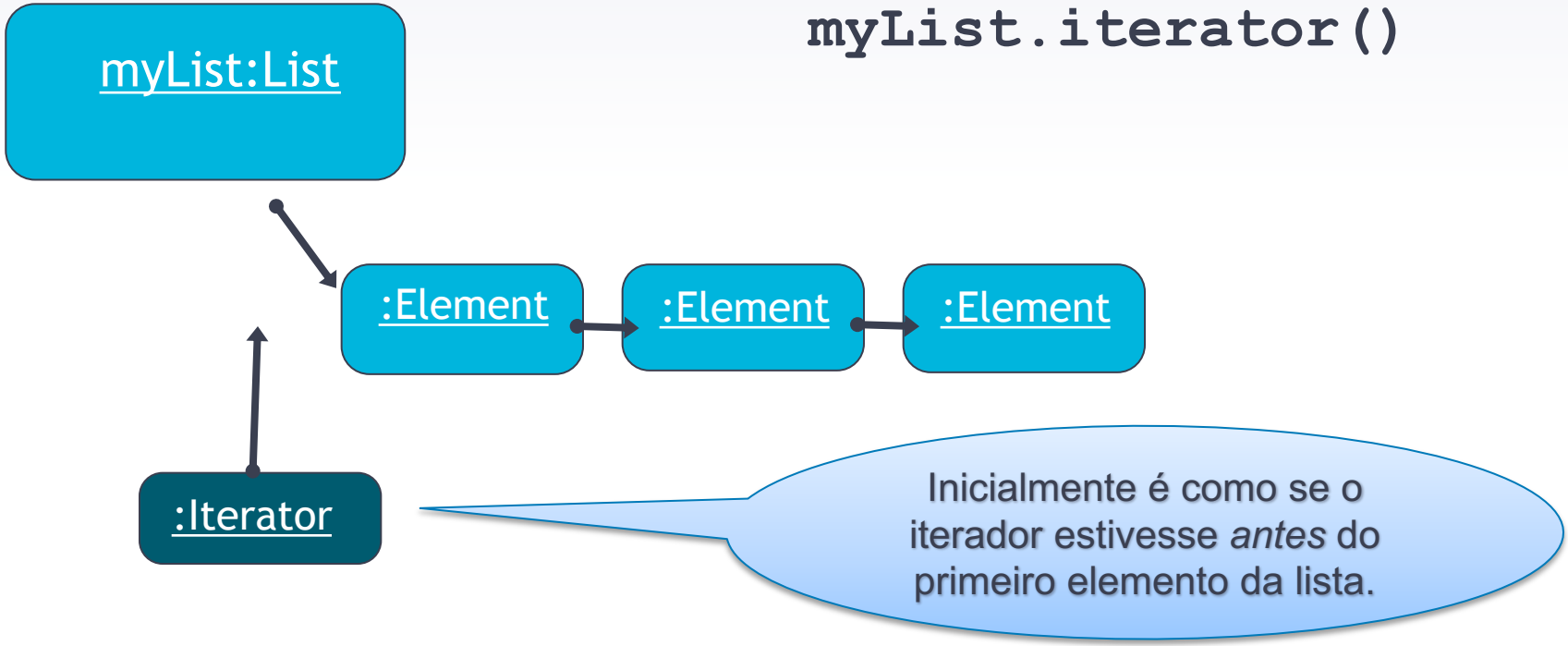
em **java.util.Iterator**



Percorrer coleções com Iteradores

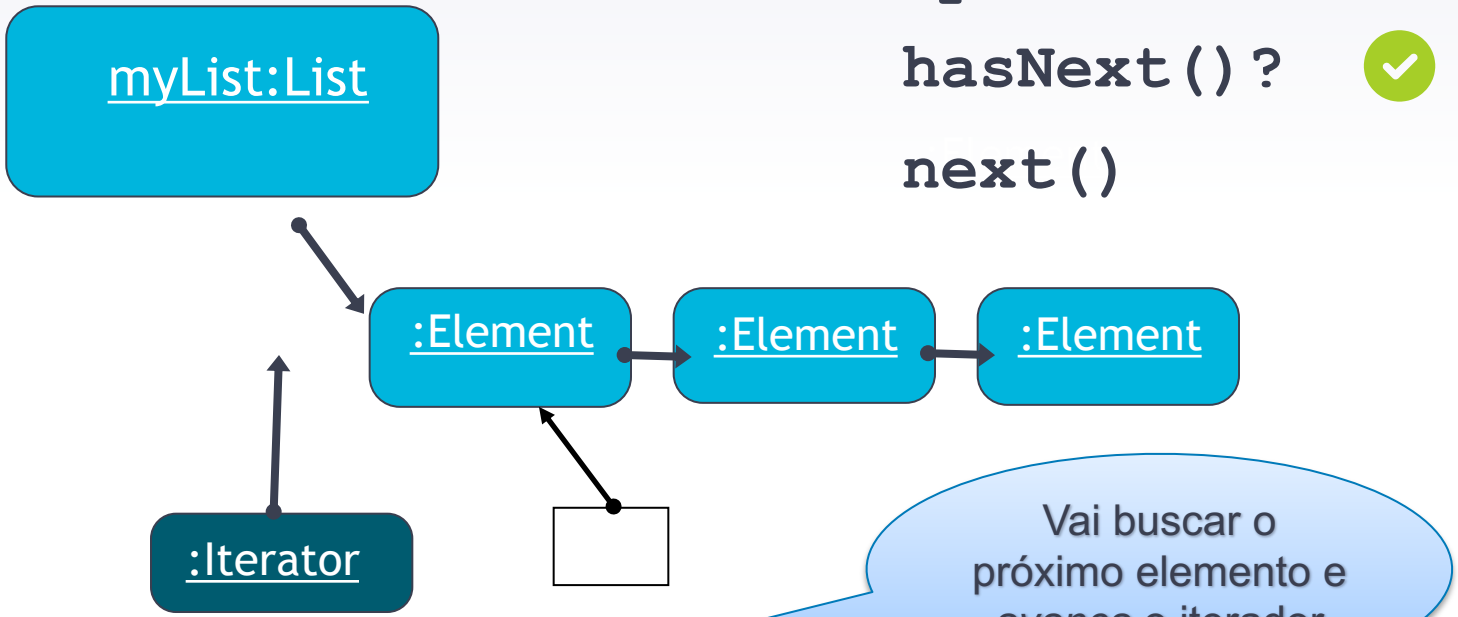
- Funcionamento do **iterador**

```
myList.iterator()
```



Percorrer coleções com Iteradores

- Funcionamento do **iterador**



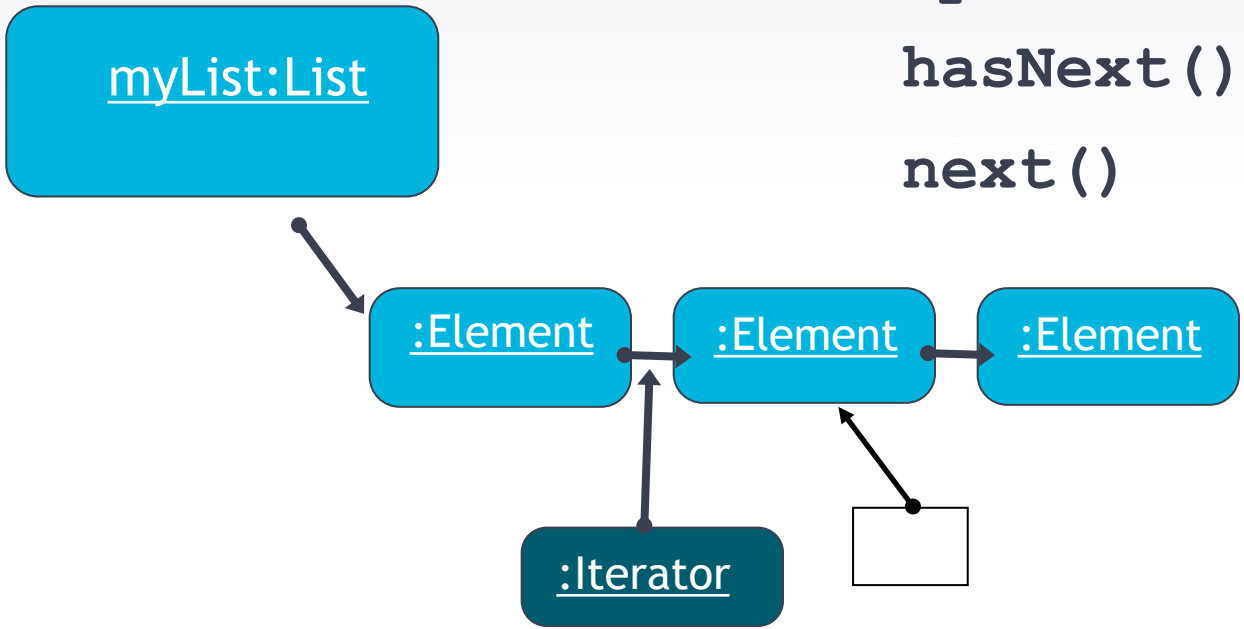

```
myList.iterator()
hasNext()?
next()
```

```
Element e = iterator.next();
```

Percorrer coleções com Iteradores

- Funcionamento do **iterador**

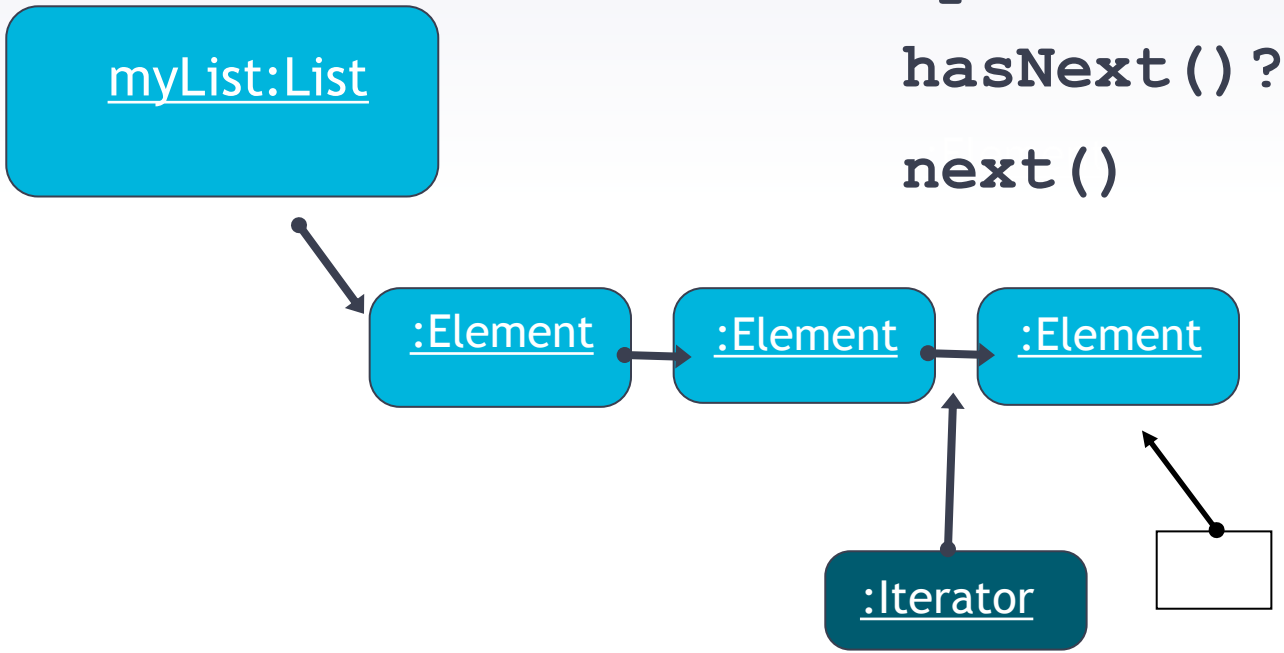
```
myList.iterator()
hasNext()?
next()
```




```
Element e = iterator.next();
```

Percorrer coleções com Iteradores

- Funcionamento do **iterador**

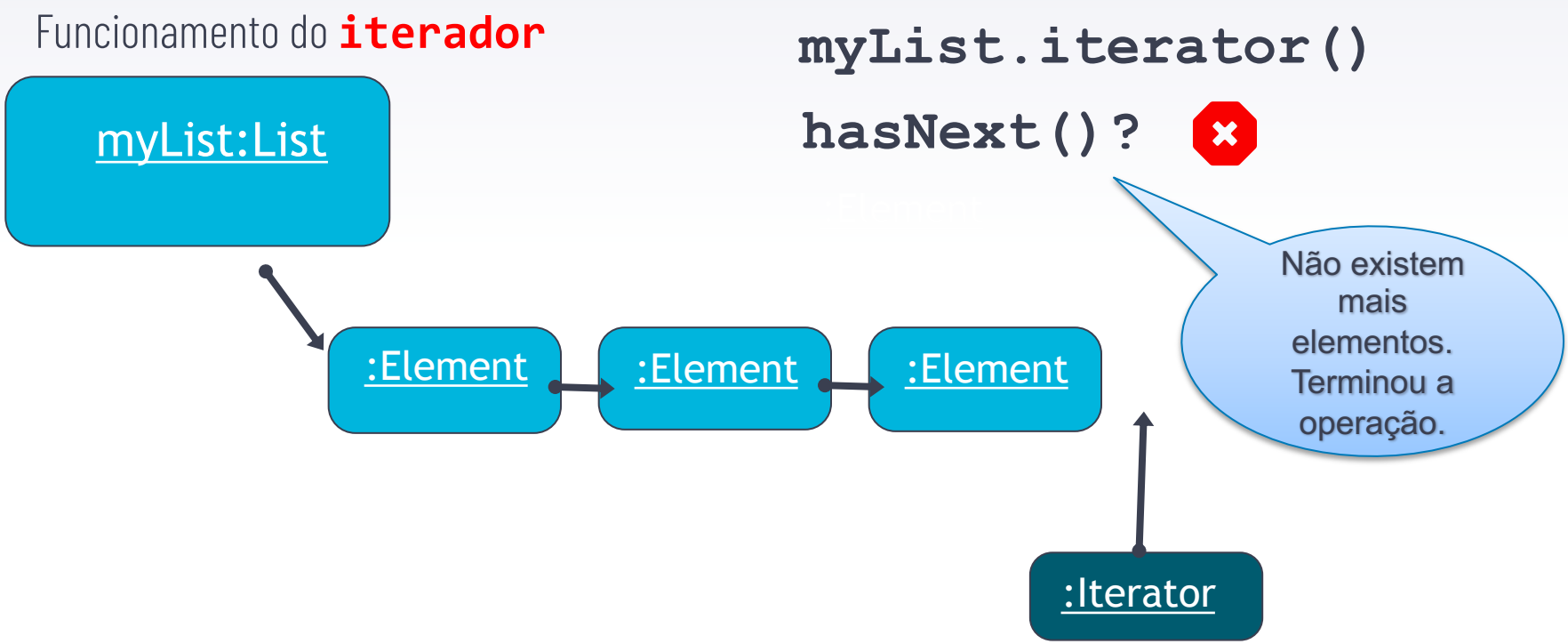



```
myList.iterator()
hasNext()?
next()
```

```
Element e = iterator.next();
```

Percorrer coleções com Iteradores

- Funcionamento do **iterador**



Implementação de `removeAssets(Asset[] assets)`

- Com um iterador

```
public void removeAssets(Asset[] assetsToRemove){
        Iterator<Asset> it = this.assets.iterator();
        while(it.hasNext()){
            Asset asset = it.next();
            for(int i = 0; i < assetsToRemove.length; i++){
              if (asset.equals(assetsToRemove[i])){
                it.remove();
              }
            }
        }
    }
```

Índices versus Iteradores

- Formas de percorrer (iterar) uma coleção
  - Ciclo **`for`**
    - Usado quando é útil usar os índices ao percorrer-se a coleção.
    - Usado em repetições que não envolvem coleções e que se repetem um determinado número de vezes
  - Ciclo **`for-each`**
    - Usado quando se pretende percorrer todos os elementos da coleção.
  - Ciclo **`while`**
    - Usado para percorrer os elementos da coleção quando se pretende interromper a meio.
    - Usado em repetições que não envolvem coleções e em que não se sabe determinar o número de vezes que se irá repetir.
  - Objeto **`iterator`**
    - Usado para percorrer os elementos da coleção quando se pretende interromper a meio.
    - Usado em coleções que não usam índices
    - Usado quando se pretendem remover elementos da coleção

Bibliografia

- Objects First with Java (6th Edition), David Barnes & Michael Kölling, Pearson Education Limited, 2016
  - Capítulo 4