FICHA PRÁTICA 4

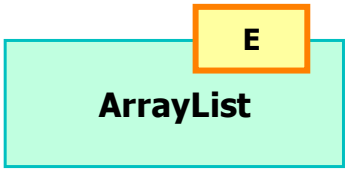
LABORATÓRIO DE COLECÇÕES I

ARRAYLIST<E>

SÍNTESE TEÓRICA

Em JAVA5, tal como em algumas outras linguagens de programação por objectos, certas estruturações de objectos (colecções) são *parametrizadas*, ou seja, aceitam um tipo parâmetro a ser posteriormente substituído por um tipo concreto (p.e., uma classe).

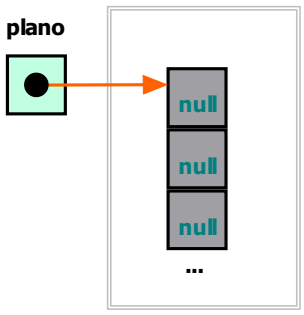
Uma colecção de JAVA5 de extraordinária utilidade na agregação de objectos e sua estruturação sob a forma de uma lista, é **ArrayList<E>**. Tendo por suporte um *array* de características dinâmicas, ou seja, capaz de aumentar ou diminuir de dimensão ao longo da execução de um programa, um ArrayList<E>, implementa listas de objectos, em que E representa a classe/tipo parâmetro de cada um dos objectos nele contidos. Um *arraylist* é pois, ao contrário de um array, um objecto e de dimensão dinâmica, possuindo um grande conjunto de métodos a si associados para realizar as operações mais usuais sobre listas, não podendo porém conter valores de tipos primitivos (apenas objectos).



A declaração de um arraylist concreto, consiste em definir-se qual o tipo concreto para o seu tipo parâmetro, numa declaração normal mas paramétrica, onde, em geral, usando um **construtor**, é criado imediatamente um *arraylist* inicialmente vazio (atenção ao **construtor** que também é parametrizado), com dimensão inicial ou não.

```
ArrayList<String> nomes = new ArrayList<String>(100);
ArrayList<Ponto2D> plano = new ArrayList<Ponto2D>();
```

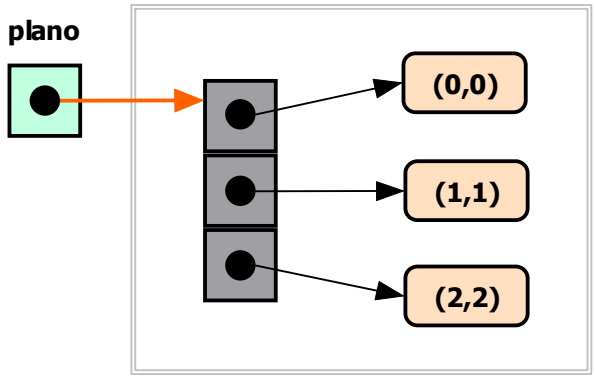
Em ambos os casos o estado inicial do arraylist deverá ser visto como sendo uma lista de apontadores a **null**, já que nenhum elemento foi ainda criado e inserido, cf.



ArrayList<Ponto2D>

Posteriormente, e à medida que cada um dos Ponto2D forem inseridos usando os diversos métodos que apresentaremos em seguida, cada posição do arraylist (iniciadas no índice 0),

referenciará uma instância de Ponto2D ou manter-se-á a **null**, tal como se ilustra na figura seguinte.



ArrayList<Ponto2D>

A API da classe **ArrayList<E>** é apresentada em seguida, encontrando-se as várias operações disponíveis agrupadas por funcionalidades para melhor compreensão. Todos os parâmetros representados como sendo do tipo *Collection* devem ser assumidos como podendo ser uma qualquer colecção de JAVA, ainda que neste momento estejamos limitados a trabalhar apenas com **ArrayList<E>**.

SINTAXE ESSENCIAL

Categoria de Métodos	API de ArrayList<E>

Construtores	`new ArrayList<E>()`
	`new ArrayList<E>(int dim)`
	`new ArrayList<E>(`*`Collection`*`)`

Inserção de elementos	`add(E o); add(int index, E o);` `addAll(`*`Collection`*`); addAll(int i, `*`Collection`*`);`
Remoção de elementos	`remove(Object o); remove(int index);` `removeAll(`*`Collection`*`); retainAll(`*`Collection`*`)`

Consulta e comparação de conteúdos	`E get(int index); int indexOf(Object o);`
	`int lastIndexOf(Object o);`
	`boolean contains(Object o); boolean isEmpty();`
	`boolean containsAll(`*`Collection`*`); int size();`

Criação de Iteradores	`Iterator<E> iterator();`
	`ListIterator<E> listIterator();`
	`ListIterator<E> listIterator(int index);`

Modificação	`set(int index, E elem); clear();`
Subgrupo	`List<E> sublist(int de, int ate);`
Conversão	`Object[] toArray();`
Outros	`boolean equals(Object o); boolean isEmpty();`

Quadro – API de `ArrayList<E>`

NOTA:

Os parâmetros designados como sendo de tipo ***Collection*** serão explicados mais tarde. De momento apenas será necessário que se compreenda que são colecções de JAVA. De momento apenas podemos usar como parâmetros dos métodos colecções que sejam ArrayList<T> e em que T seja um tipo compatível com o tipo E (por exemplo, sendo mesmo T = E). Tal significa quese estivermos a trabalhar, por exemplo, com um ArrayList<Ponto2D> apenas parâmetros do tipo ArrayList<Ponto2D> devem ser considerados em métodos como addAll() e outros que aceitem Collections como parâmetros. Mais tarde veremos porquê.

1.- Exemplos Simples

Consideremos um exemplo simples cujo objectivo é a apresentação da semântica dos principais métodos da API de **ArrayList<E>**, através da sua utilização.

Vamos criar duas listas de nomes, uma contendo os nomes de amigos e outra contendo os nomes de pessoas que conhecemos e são músicos. Serão, naturalmente dois **ArrayList<String>**. Vamos em seguida realizar a sua declaração completa, limitando o número de músicos a uns 50.

Declarações

```
ArrayList<String> amigos =  new ArrayList<String>();  // vazio
ArrayList<String> musicos = new ArrayList<String>(50);  // capacidade = 50
```

Inserções no fim usando add(E elem)

```
amigos.add("Jorge"); amigos.add("Ivo"); amigos.add("Rino");
// ERRO DE COMPILAÇÃO => amigos.add(new Ponto2D(0.0, 2.0));
musicos.add("Rino"); musicos.add("Zeta");
```

Número Actual de elementos de uma Lista

```
int numAmigos = amigos.size();
```

Consulta do elemento no índice i <= amigos.size() - 1

```
String amigo = amigos.get(10);
String nome = amigos. get(i);
```

Iteração sobre todos os elementos de um ArrayList<String>

```
// Imprimir os nomes de todos os amigos – Solução A: Ciclo for
for(int i = 0; i <= amigos.size()-1; i++)
   out.printf("Nome: %s%n", amigos.get(i);

// Imprimir os nomes de todos os amigos – Solução B: Criando um Iterador
// Depois de criado, o Iterator<String> é percorrido até it.hasNext() ser falso,
// sendo cada elemento devolvido usando it.next()
for(Iterador<String> it = amigos.iterator(); it.hasNext();)
   out.printf("Nome: %s%n", it.next());

// Imprimir os nomes de todos os amigos – Solução C: Usando for(each)
for(String nome : amigos)
   out.printf("Nome: %s%n", nome);
```

Procura de um elemento => usar Iterator<E> e while

```
String chave = Input.lerString();
encontrado = false; int index = 0;
Iterator<String> it = amigos.iterator();  // Iterator<String> criado
while(it.hasNext() && !encontrado) {      // iteração
   encontrado = it.next().equals(chave); index++;
}
if (encontrado)
  out.printf("O nome %s está na posição %2d%n", chave, index-1);
else
  out.println("O nome " + chave + " não existe !\n");
```

Procura de um elemento (versão simples)

```
String chave = Input.lerString();
int index = amigos.indexOf(chave);
if (index = -1)
   out.println("Não existe tal nome !");
else
   out.println("Está no índice: " + index);
```

ArrayList esparso (não contíguo): utilização correcta

A) Inicializar a **null** as posições que se pretendem aceder;

B) Usar **set(int index, E elem)** sendo **index <= size()-1**;

```
ArrayList<String> cantores = new ArrayList<String>(DIM);  // capac = DIM
for(int i = 0; i <= DIM-1; i++) cantores.add(null);
// inserção de nomes em qualquer índice até DIM-1 !!
cantores.set(10, "Rui"); cantores.set(2, "Ana");
```

Elementos comuns a dois arrays

```
// Determinar quais os amigos que também são músicos
//
// 1) Cria um arraylist e copia o arraylist amigos na criação
ArrayList<String> temp = new ArrayList<String>(amigos);
// ou, em alternativa equivalente usando temp1
ArrayList<String> temp1 = new ArrayList<String>();
for(String am : amigos) temp1.add(am);
// 2) Remove da lista temp todos os que não são musicos, ou seja,
// retém os que também estão na lista musicos.
temp.retainAll(musicos);
// imprime os nomes
out.println("\n--- AMIGOS MÚSICOS ---\n");
for(String name : temp) out.printf(" %s%n", name);
// o mesmo para a lista de amigas
temp = new ArrayList<String>(amigas); temp.retainAll(musicos);
out.println("\n--- AMIGAS MÚSICAS ---\n");
for(String n : temp) out.printf(" %s%n", n);

// Remover da lista de amigos e da lista de amigas os que são músicos;
//
// As listas amigos e amigas são modificadas caso alguns sejam músicos !!
// Outra hipótese seria usar, tal como acima, um arraylist temporário.
amigas.removeAll(musicos); amigos.removeAll(musicos);
// imprime os nomes de amigos e amigas restantes
out.println("\n--- AMIGOS NÃO MÚSICOS ---\n");
for(String n : amigos) out.printf(" %s%n", n);
out.println("\n--- AMIGAS NÃO MÚSICOS ---\n");
for(String n : amigas) out.printf(" %s%n", n);

// Criar um arraylist que seja a reunião das listas de amigos e amigas;
//
// Esta operação é semelhante a uma reunião matemática, mas não faz
// qualquer filtragem de elementos em duplicado (também não devem
// existir neste caso !!).
ArrayList<String> todos = new ArrayList<String>(amigos);
todos.addAll(amigas);
out.println("--- Todos : ---\n");
for(String nom : todos) out.printf("Nome: %s%n", nom);
```

Exercícios:

Ex 1: Compreensão do funcionamento de ArrayList<E>
Crie no BlueJ, uma pasta de projecto de nome TesteArrayList e importe para tal pasta a classe Ponto2D anteriormente desenvolvida. Em seguida, usando Tools/Use Library Class…, crie uma instância vazia de um ArrayList<String> e outra de um ArrayList<Ponto2D>. De seguida, com ambas as instâncias de arraylist criadas, invoque os métodos da API de ArrayList<E>, cf. add(), get(), addAll(), remove(), removeAll(), size(), indexOf(), contains(), retainAll(), etc., e conclua sobre a semântica de cada método usando INSPECT após a sua invocação.

Ex 2: Desenvolva uma classe Plano que represente um conjunto de pontos 2D de um plano cartesiano num ArrayList<Ponto2D> (a representação correcta seria um conjunto mas como ainda não estudamos como representar conjuntos em JAVA usaremos um arraylist).
Desenvolva, para além dos construtores e métodos usuais, métodos que implementem as seguintes funcionalidades:

- Determinar o número total de pontos de um plano;
- Adicionar um *novo* ponto ao plano e remover um ponto caso exista;
- Dado um `ArrayList` de `Pontos2D`, juntar tais pontos ao plano receptor;
- Determinar quantos pontos estão mais à direita ou mais acima (ou ambos) relativamente ao `Ponto2D` dado como parâmetro;
- Deslocar todos os pontos com coordenada em XX igual a cx, dada como parâmetro, de dx unidades, igualmente dadas como parâmetro (alterar os pontos portanto);
- Dado um plano como parâmetro, determinar quantos pontos são comuns aos dois planos;
- Criar a lista contendo os pontos comuns ao plano receptor e ao plano parâmetro;
- Criar uma lista contendo todos os pontos do plano com coordenada em XX inferior a um valor dado como parâmetro (atenção, pretende-se obter cópias dos originais);
- Criar um novo plano que contenha os pontos comuns entre o plano receptor e um plano dado como parâmetro;
- Não esquecer os métodos **`equals()`**, **`toString()`** e **`clone()`**.

Ex 3: Uma ficha de informação de um país, FichaPais, contém 3 atributos: nome do país, continente e população (real, em milhões). Crie uma classe ListaPaises que permita criar listas de FichaPais, por uma ordem qualquer, e implemente os seguintes métodos:

- Determinar o número total de países;
- Determinar o número de países de um continente dado;
- Dado o nome de um país, devolver a sua ficha completa, caso exista;
- Criar uma lista com os nomes dos países com uma população superior a um valor dado;
- Determinar a lista com os nomes dos continentes dos países com população superior a dado valor;
- Determinar o somatório das populações de dado continente;
- Dada uma lista de FichaDePais, para cada país que exista na lista de países alterar a sua população com o valor na ficha; caso não exista inserir a ficha na lista;
- Dada uma lista de nomes de países, remover as suas fichas;

Ex 4: Uma Stack (ou pilha) é uma estrutura linear do tipo LIFO ("last in first out"), ou seja, o último elemento a ser inserido é o primeiro a ser removido. Uma stack possui assim apenas um extremo para inserção e para remoção. Implemente uma Stack de nomes, com as usuais operações sobre stacks:

- `String top()`: que determina o elemento no topo da stack;
- `void push(String s)`: insere no topo;

- void pop(): remove o elemento do topo da stack, se esta não estiver vazia;
- boolean empty(): determina se a stack está vazia;
- int length(): determina o comprimento da stack;

Ex 5: Cada e-mail recebido numa dada conta de mail é guardado contendo o endereço de quem o enviou, a data de envio, a data de recepção, o assunto e o texto do mail (não se consideram anexos, etc.). Crie a classe Mail que represente cada um dos mails recebidos.
Em seguida crie uma classe designada MailList que permita guardar todos os actuais e-mails existentes numa dada conta, e implemente as seguintes operações sobre os mesmos:

- Determinar o total de mails guardados;
- Guardar um novo mail recebido;
- Determinar quantos mails têm por origem um dado endereço;
- Criar uma lista contendo os índices dos mails que no assunto contêm uma palavra dada como parâmetro (qualquer que seja a posição desta);
- O mesmo que a questão anterior, mas criando uma lista contendo tais mails;
- Eliminar todos os e-mails recebidos antes de uma data que é dada como parâmetro;
- Criar uma lista dos mails do dia;
- Dada uma lista de palavras, eliminar todos os mails que no seu assunto contenham uma qualquer destas (anti-spam);
- Eliminar todos os mails anteriores a uma data dada;