Programação Orientada a Objetos

Aula 10 - Coleções em Java

Apresentação

Em um contexto prático de programação, utilizando conceitos de orientação a objeto, precisamos constantemente manipular muitas informações e muitos objetos ao mesmo tempo. Para isso, é necessária uma estrutura que permita armazená-los e recuperá-los sempre que desejarmos. A linguagem Java oferece várias dessas estruturas de dados em um conjunto de classes chamadas de coleções. Tais coleções ajudam a armazenar e recuperar nossos objetos dentro de um sistema. É isso que você verá nesta aula.

Vídeo 01 - Apresentação

Objetivos

Ao final desta aula, você será capaz de:

- Compreender como armazenar objetos em estruturas de dados conhecidas como arrays.
- Conhecer e aprender como usar de forma introdutória um conjunto de classes da biblioteca padrão de Java relacionada à manipulação de coleções de objetos.
- Aprender a utilizar o “for-each” para iterar sobre coleção.
- Escrever códigos que utilizem a versão “genérica”das coleções.

Armazenando Objetos

Vamos organizar os objetos semelhantes!

Existem várias situações durante a construção de sistemas e programas em que precisamos armazenar um número expressivo de objetos. Nesses casos, a criação e manuseio de uma variável para cada um dos objetos torna-se uma tarefa impraticável. Esse fato ocorre, por exemplo, quando precisamos armazenar uma lista de alunos de uma turma, ou mesmo da escola inteira, e em um dado momento precisamos recuperar apenas um dos objetos aluno dessa lista. Outro exemplo seria quando queremos encontrar um número de telefone e recorremos à lista telefônica, ela é nosso repositório de dados.

Imagine termos que criar 100 (cem) variáveis para 100 nomes de alunos! É possível criar um tipo especial de variável para armazenar um número específico de objetos ou valores primitivos. Esse tipo de variável é denominada de **array** (também conhecida como vetor) e será melhor discutida na sequência deste curso.

Arrays

Considere a seguir um exemplo que motiva o uso de arrays. Constantemente, precisamos tomar nota de um valor que muda ao longo do tempo, e precisamos medir sua média, valores máximo e mínimo. Por exemplo, suponha que você é um treinador de um atleta velocista de 100 metros rasos, como parte do treinamento, você irá fazer 5 medições de tiros de 100 metros.

Para cada tentativa do atleta, você vai medir o tempo e colocar o resultado em sua planilha de rendimento.

Precisamos armazenar 5 valores numéricos reais para representar a marca de tempo obtida. Porém, não podemos criar 5 variáveis, e sim apenas uma para comportar esses valores. Veja tal código na Listagem 1.

```
1  public class TreinadorMain{
2     public static void main(String[] args){
3        String nomeAtleta = "Bolt";
4        double[] marca= new double[5];
5
6        marca[0] = 10.02;
7        marca[1] = 10.0;
8        marca[2] = 9.56;
9        marca[3] = 9.57;
10       marca[4] = 9.56;
11    }
12 }
```

Listagem 1 - Declaração de um array

Conforme pode ser observado, um novo operador entrou em cena: o par de colchetes “[“ e “]”. Eles servem para indicar uma dimensão, ou seja, uma posição onde iremos colocar um valor inteiro que indica a quantidade de elementos que precisamos armazenar na variável em questão (marca). Por conta de tais colchetes, a variável marca não mais é capaz de armazenar um único valor do tipo double, mas sim irá armazenar um conjunto de valores do tipo double. Dessa forma, caracteriza-se a declaração de uma variável/atributo do tipo array (também chamada de vetor) em Java.

Observe que o atributo marca representa um array, mas é necessário o operador *new* para alocar espaço para armazenar o conjunto de valores. Assim, a palavra *new* é usada com o objetivo de indicar quantos valores do tipo Double serão necessários para armazenar tais valores. A quantidade de valores estipulada para armazenar no array, cinco (5) para o exemplo do atributo marca, é um valor que permanece fixo, após a chamada com new. Isso significa que no nosso exemplo o atributo marca será capaz de armazenar 5 elementos.

O código logo após a declaração indica uma atribuição de valor para armazenar uma determinada marca em cada posição do array. Veja que colocamos um número para indicar que posição estamos acessando. Esse valor sempre começa com 0 (zero) e vai até o comprimento do vetor menos uma unidade. No nosso caso, o array marca vai de 0 (zero) a 4 (quatro), compreendendo 5 (cinco) posições conforme foi declarado.

```
System.out.println("Tempo 1:"" + marca[0]);
System.out.println("Tempo 1:"" + marca[1]);
System.out.println("Tempo 1:"" + marca[2]);
System.out.println("Tempo 1:"" + marca[3]);
System.out.println("Tempo 1:"" + marca[4]);
```

Listagem 2 - Impressão dos valores do Array

Para exibirmos os valores, o programa da Listagem 1 pode ser incrementado pelas linhas de código da Listagem 2. Não indicamos a primeira tomada de tempo como sendo o tempo 0 (zero), pois naturalmente costumamos realizar contagens a partir do número 1 (um).

Há outra forma de definir os valores de um array, no momento de sua declaração podemos indicar seus valores, sendo que já precisamos conhecê-los de antemão. Veja a Listagem 3. Observe que não precisarmos indicar o comprimento do vetor, pois ele sabe de antemão pela quantidade de elementos definidos na sua inicialização.

```
double[] marca = {10.02, 10.0, 9.56, 9.57, 9.56};
```

Listagem 3 - Outra forma de definir os valores de um Array

Tanto na definição do array quanto no acesso para leitura ou escrita de suas posições, é possível usar um comando de repetição para acessá-lo diretamente. Na verdade, na programação diária, isso é o mais comum.

Vejamos como ficaria a exibição dos valores acima com um comando FOR na Listagem 4.

```
marca[0] = 10.02;
marca[1] = 10.0;
marca[2] = 9.56;
marca[3] = 9.57;
marca[4] = 9.56;

for(int i = 0;i<5;i++){
   System.out.println ("Marca"+(i+1)+": " +marca[i]);
}
```

Listagem 4 - Impressão dos valores de um array usando comando FOR

Atividade 01

1. Para praticar o conceito de array, crie um programa que declara uma lista (array) capaz de armazenar 10 (dez) nomes de ferramentas para uso de um mecânico. Inicialize cada uma das ferramentas armazenadas no array, em seguida, imprima cada um deles.

Arrays Bidimensionais

Primeiramente, vamos definir o que é uma dimensão de um array. **A dimensão, ou quantidade de dimensões, é o conjunto de valores que precisamos definir para localizar uma informação.** Por exemplo, uma lista de alunos de 0 a 100 pode ser organizada em um array de uma dimensão, pois para localizar um aluno nessa lista basta indicar um valor da sequência.

Já para localizar uma peça em um tabuleiro de xadrez precisamos de duas coordenadas, linha e coluna. Assim, para representarmos um tabuleiro de xadrez com arrays, são necessárias duas dimensões. A Listagem 5 mostra esse exemplo codificado em Java. Suponha que iremos representar um tabuleiro de xadrez com linhas e colunas de 0 (zero) a 7 (sete), compreendendo 8 valores de cada. Vamos também supor que cada valor representa uma String com o nome da peça que ocupa a casa. Observe a utilização de dois valores distintos para localizar uma casa no tabuleiro, e para cada valor um par de colchetes, um par para cada dimensão do array.

```
1 String[][] tabuleiro = new String[8][8];
2
3 tabuleiro[0][0] = "Torre branca";
4 tabuleiro[0][1] = "Cavalo branco";
5 tabuleiro[1][0] = "Peão branco";
6 tabuleiro[0][3] = "Rainha branca";
```

Listagem 5 - Exemplo de Array bidimensional

É possível definir *n* dimensões, porém, na prática, não é comum nem recomendável trabalhar com tantas dimensões. Na prática, é extremamente comum trabalharmos com apenas 1 (uma), e algumas vezes com 2 (duas) e, quase nunca

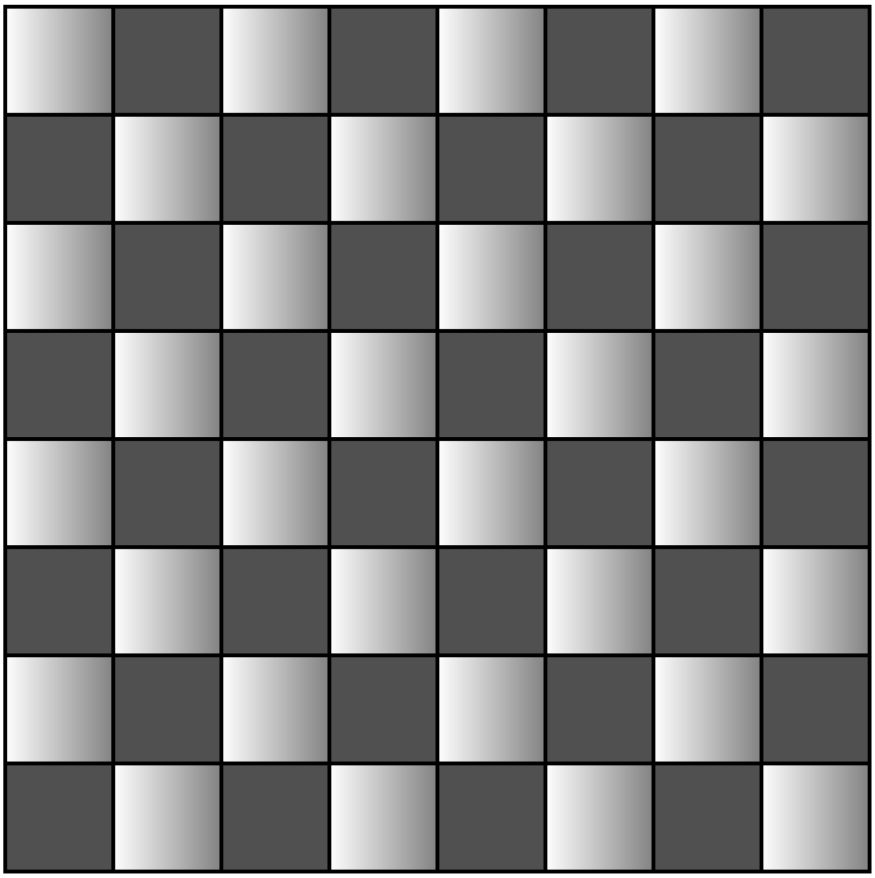
com 3 (três) dimensões, pois há outros recursos na programação orientada a objetos que desencorajam e oferecem alternativas melhores a essa prática.

Atividade 02

1. Observe o tabuleiro da Figura 1 abaixo e crie um programa em Java que o represente através de um array de duas dimensões, em que cada casa irá conter a sua cor BRANCA ou PRETA, como sendo valores do tipo String. Imprima cada uma das casas do array, após iniciá-los.

   Dica: não tente definir uma a uma, pois serão 64 linhas de códigos. Ao invés disso, crie estruturas com comandos de repetição para preenchê-lo e imprimi-lo.

Figura 01 - Tabuleiro



Arrays como Objetos

Vimos que arrays são estruturas para armazenar objetos, tais como Strings. Mas, é preciso dizer que arrays são também objetos, por isso é usado o comando *new()* para alocar espaço de armazenamento para ele. Graças a sua capacidade de se comportar como um objeto, os arrays possuem métodos e um atributo muito útil, o length, que indica seu comprimento.

É necessário conhecê-lo para não passarmos do limite e para realizar operações de manutenção em seus dados. Pois bem, vamos alterar o exemplo anterior (Listagem 1) que exibe os valores do array marca para utilizar seu atributo length. Tal programa modificado é apresentado na Listagem 6.

```
marca[0] = 10.02;
marca[1] = 10.0;
marca[2] = 9.56;
marca[3] = 9.57;
marca[4] = 9.56;

for(int i = 0;i<marca.lenght;i++){
   System.out.println("Marca"+(i+1)+": " +marca[i]);
}
```

Listagem 6 - Exemplo de uso do atributo lenght de um array

Coleções Java

A linguagem Java possui um conjunto de classes que servem para armazenar na memória vários objetos. Tais classes não possuem o inconveniente de termos que saber de antemão a quantidade exata de elementos que iremos armazenar, como no caso de arrays. E em alguns casos, nem mesmo o tipo.

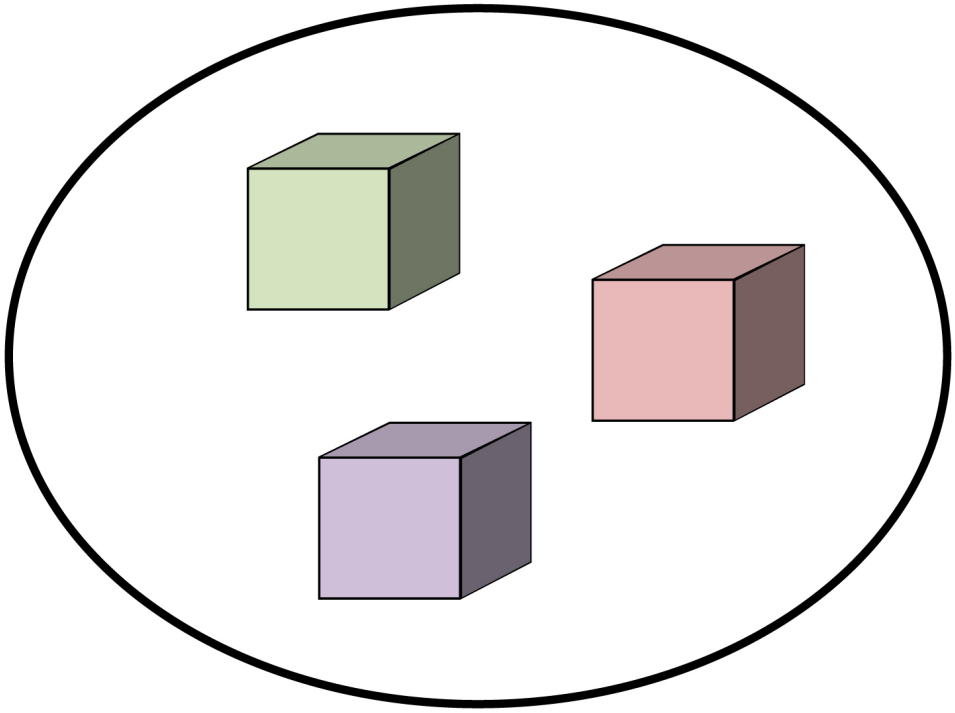
Existem tipos principais de Coleções em Java, cada uma com um propósito deferente, como veremos a seguir.

- **List:** Lista de coisas.

- **Set:** Conjunto de Coisas (Não permite repetição de elementos).
- **Map:** Coisas com um identificador único.

1. Set – representa a mesma ideia de conjuntos da matemática, ou seja, um grupo de objetos sem ordem definida, porém, únicos. Como mostra a Figura 2.

Figura 02 - Conjunto de objetos



Nunca poderemos prever a ordem com a qual serão apresentados os seus elementos. Essa situação pode não ser um incômodo diante do problema que tivermos. Os sets possuem uma característica importante em relação às buscas de objetos em seu interior, pois não precisam percorrer todos eles.

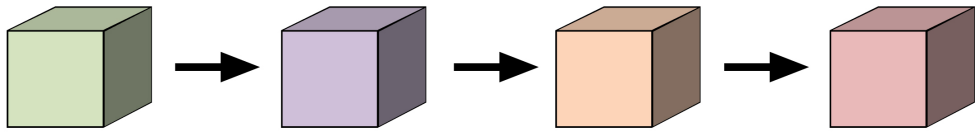
O principal representante dos Set é a classe HashSet, vejamos um exemplo da sua utilização, como mostra a Listagem 7:

```
1 public class TesteHashSet{
2    public static void main(String[] args){
3       HashSet itens = new HashSet();
4       itens.add("Chocolate");
5       itens.add("Bala");
6       itens.add("Brigadeiro");
7    }
8 }
```

Listagem 7 - Exemplo de um Set

2. List – como o próprio nome sugere, representa uma lista de objetos, sendo que nela os objetos podem se repetir. Veja a figura.

Figura 03 - Lista de objetos



Nas listas (List) definidas em Java, os objetos armazenados mantêm a ordem com que foram adicionados. Uma classe do tipo List bastante utilizada é a ArrayList, a qual representa uma alternativa aos arrays convencionais vistos anteriormente. Vejamos um exemplo da utilização de um ArrayList na Listagem 8:

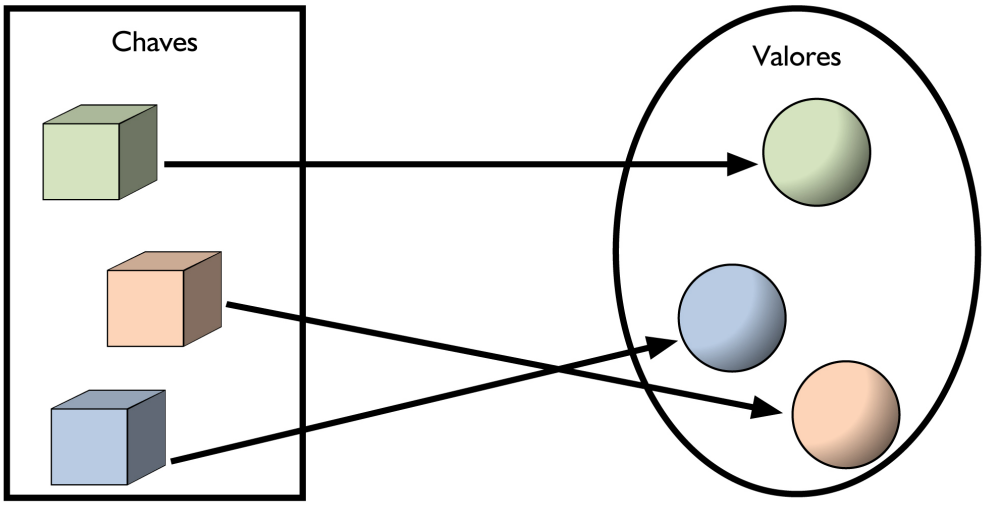
```
1 ArrayList nomes = new ArrayList();
2 nomes.add("Maria");
3 nomes.add("João");
```

Listagem 8 - Exemplo de ArrayList

3. Map – Mapas são estruturas que relacionam um objeto a outro, por exemplo, um número de CEP ao nome de uma rua.

Podemos imaginar dois conjuntos, um de campos-chave e outro de objetos-valor que queremos armazenar. Observe a Figura 4.

Figura 04 - Dois conjuntos de objetos



Observe que para encontrarmos nossos objetos precisamos localizá-los através de suas chaves. Vejamos uma aplicação desses conceitos na Listagem 9.

```
1 public class TesteHashSet{
2    public static void main(String[] args){
3       HashMap livros = new HashMap();
4
5       livros.put(1, "Volta ao mundo em 80 dias");
6       livros.put(2, "Alice no país das maravilhas");
7       livros.put(5, "Caninos Brancos ");
8    }
9 }
```

Listagem 9 - HashMap na prática

Todas as classes do tipo Set e List descendem (implementam) a interface Collection. Veremos mais detalhes sobre interfaces em aulas futuras. Por hora, podemos entender interfaces como contratos que definem um conjunto de métodos que devem ser implementados pelas classes. No caso da interface Collection, ela define métodos para adicionar, remover, verificar a presença de um dado objeto. Tais métodos devem necessariamente existir em todas as List e Sets definidos para a linguagem Java. Segue abaixo uma lista básica desses métodos.

Vídeo 02 - Coleções

boolean add(Object)	Adiciona um elemento na coleção. Como algumas coleções não suportam elementos duplicados (exemplo: Sets), esses métodos retornam verdadeiro (true) ou falso (false) para indicar se a adição foi bem sucedida.
boolean remove(Object)	Remove determinado elemento da coleção. Se ele não fizer parte da coleção, retorna falso (false).
int size()	Retorna a quantidade de elementos presentes na coleção.
boolean contains(Object)	Procura por um determinado objeto na coleção. Vale salientar: a comparação é feita pelo método equals().

Lista Básica de Métodos da Interface Collection

Anote a Dica!

As coleções Java mantêm um mecanismo interno de ordenação e recuperação de dados e para isso fazem uso de tabelas hash. Tais tabelas são utilizadas para que a pesquisa de um objeto seja feita de maneira rápida. Mas, como funciona? Cada objeto é “classificado” pelo seu hashCode, método de java.lang.Object que retorna um int, e com isso podemos agrupar os objetos por esse valor. Quando é realizada uma busca, só é percorrido o grupo de objetos com o mesmo hashCode.

Atividade 03

1. Crie um programa que registra uma lista de compras semanais com itens como: arroz, feijão, carne, pão etc. Utilize os três tipos de coleções vistos até aqui: HashSet, ArrayList e HashMap.

Iterando sobre Coleções

Agora que vimos algumas das coleções mais utilizadas no mundo Java, vamos então aprender como percorrer essas coleções. Para tal tarefa, existem várias opções, no entanto, nesta seção iremos falar especificamente do **for each**. Antes das apresentações oficiais, vamos dar uma revisada no velho e bom **for** tradicional, apenas para relembrar.

Loop for each

Também conhecido no mundo Java como **“enhancedloop”** e **for-in**, porém, todos se referem ao mesmo construtor Java. O for-each loop veio com o Java 6 como um loop especializado que simplifica a iteração sobre *Arrays* e *Coleções*.

Bom, já que o nosso novo loop veio para facilitar a nossa vida, quando iteramos sobre as coleções, então, vamos dar uma olhada mais de perto no seu funcionamento.

O que ele tem de bom?

Primeiramente, ao invés de ser dividido em **TRÊS** partes, temos apenas **DUAS**. Opa! Isso já está ficando bom! Diminuiu a complexidade da sintaxe. Então, vamos iterar sobre um **Array** utilizando o **for básico**(*antigo*) e então depois veremos como fica com o nosso novo amigo.

```
1 int [ ]a = {1,2,3,4}
2 for (int x = 0 ; x <a.length ; x++)
3       System.out.print(a[x]);
4 for (int n : a)
5       System.out.print(n);
```

Noossaaa! Realmente ficou muito mais simples!! Mas não entendi nada! Calma, veremos todo o seu comportamento a seguir.

Analisando o enchanced for de uma maneira mais formal:

```
1 for ( declaração : expressão ) {}
```

As duas partes são:

- **Declaração:** no bloco de declaração o tipo da variável pode ser *compatível* com o tipo dos *elementos da coleção* que você está acessando. Essa variável será a que utilizaremos no corpo do loop e o seu valor será o mesmo que o elemento corrente da coleção.

- **Expressão:** deve ser o Array ou coleção que você deseja iterar sobre. Pode ser qualquer tipo de coleção ou método cujo retorno seja uma coleção ou Array. Os tipos dos elementos dentro da coleção podem ser os seguintes: primitivos, objetos ou até mesmo outra coleção.

Vamos dar uma olhada em algumas restrições do enchanced loop:

```
long x2;
Long [] la = {7L, 8L, 9L};
Animal [] animals = { new Dog(), new Cat()}

for (Long x2 : la); // x2 já está declarado
for ( Dog d : animals); // você pode receber um Cat
```

Vídeo 03 - for...each e generics

Generics e Coleções

Chegamos ao momento mais interessante do nosso conteúdo, vamos aprender agora como a utilização de **Generics** irá facilitar a nossa vida na manipulação de coleções em Java, fazendo com que possamos criar códigos mais robustos e legíveis.

Sabemos que Arrays em Java sempre foi **type safe**, Opa! O que danado é isso? Calma, type-safe é apenas uma expressão muito utilizada no mundo Java para dizer que um determinado elemento é fortemente tipado. Voltando ao nosso Array, significa que se o declararmos como sendo do tipo String (String []), não poderemos fazer atribuições de outros elementos, tais como Integer, Dog, ou qualquer outra coisa diferente de String.

Devemos lembrar que antes do Java 5 não havia sintaxe para declarar uma coleção como type-safe. Para fazer um ArrayList de String, tínhamos que fazer a seguinte declaração:

```
ArrayList myList = new ArrayList();
```

Ou utilizarmos a forma polimórfica:

```
List myList = new ArrayList();
```

Não havia sintaxe que nos permitisse especificar que **myList** só deveria aceitar String e nada mais além de String. Sendo assim, como não tinha essa restrição nativa da linguagem, o compilador também não tinha como forçá-lo a colocar apenas os elementos do tipo especificado dentro da lista. Vamos dar uma olhada na prática para entender melhor a respeito do problema que tínhamos antes do Java 5.

```
1 List myList = new ArrayList(); // não podemos informar o tipo.
2 myList.add("João"); // neste ponto está OK, adicionamos uma String.
3 myList.add(new Dog()); // Ops! Agora colocamos um Dog.
4 myList.add(new Integer()); // ...E agora adicionamos um Integer
```

Note que tudo que fizemos acima foi totalmente legal, nenhum erro de compilação ou alerta de tipos indevidos, ou seja, uma coleção não genérica poderá conter todos os tipos de elementos, mas qual é o problema disso? O que isso significa?

Então, vamos lá... Isso significa que é de TOTAL *responsabilidade do programador* tomar todos os cuidados ao manipular a Lista, ou seja, não existe restrição nativa da sintaxe que force as coleções a só receberem elementos de um determinado tipo pré-estabelecido.

E você acha que acabou os seus problemas? Ainda não... tem mais, pois as coleções podem receber qualquer tipo de coisa, o método que recupera objetos da coleção só poderá ser de um único tipo! Alguém se arriscaria a dizer? Essa é fácil... Só poderia ser mesmo o famoso – **java.lang.Object**. E o que isso significa na prática? Então, vamos voltar ao nosso exemplo acima e recuperar o nosso elemento do tipo String que havíamos adicionado inicialmente:

```
1 String s = (String) myList.get(0); // fazendo cast.
```

Opa! Tenho que fazer um Cast para recuperar a minha String! É isso mesmo! Porém, ainda tem mais um pequeno probleminha! Não temos como garantirque o retorno será realmente uma String, sendo assim, o nosso Cast ainda pode falhar em tempo de execução! Tudo isso porque as coleções aceitam qualquer coisa como entrada e não temos como garantir o que vai ser retornado no final, apenas rezamos para que aquele programador extremamente experiente (Estagiário) tenha se preocupado em colocar todos os elementos na coleção como sendo String, e tudo irá funcionar corretamente!

Poxa, agora fiquei triste de verdade, eu estava tão acostumado com o compilador me dizendo tudo o que eu fiz de errado, e agora você me vem com essa de que ele não vai poder me ajudar nas coleções!

Calma, isso ficou para trás, você não prestou atenção quando eu disse que todo esse problema era antes do Java 5, agora temos uma ferramenta muito forte ao nosso favor, e que vai nos ajudar não apenas com as coleções, mas em quase tudo em Java podemos nos beneficiar do uso do **Generics**.

Ufa! Agora fiquei mais tranqüilo!

Então, vamos lá, chegou a hora das boas notícias! O Generics nos ajudará em dois pontos, na hora de colocarmos e na hora de retirarmos elementos das coleções, através da aplicação do tipo específico. Vamos dar uma olhada na prática de como isso acontece de verdade! Lembra do nosso exemplo anterior?

```
1 List<String> myList = new ArrayList<String>(); // Agora informamos o tipo
2 myList.add("João"); // neste ponto está OK, adicionamos uma String.
3 myList.add(new Dog()); // Ops! Agora recebemos um erro de compilação!!!
```

Muito Bom!! Era exatamente o que queríamos! Usar a sintaxe Generics significa que temos que colocar o tipo da coleção dentro dos elementos "<" e ">", como <String>. Então, agora tudo que você coloca dentro da coleção é garantido de ser uma String. E as boas notícias não param de chegar, com o Generics tudo o que recuperamos também é garantido de ser uma String. Vamos dar uma olhada mais de perto!

Antes do Generics, não tínhamos garantia do tipo de retorno:

```
1 String s = (String) myList.get(0); // fazendo cast.
```

Agora nós apenas fazemos:

```
1 String s = myList.get(0); // Sem Cast.
```

O compilador já sabe antecipadamente que myList contém apenas objetos do tipo String, os quais podem ser atribuídos a uma referência String, então, agora não é mais necessário fazer o Cast. Agora, a nossa vida ficou realmente mais simples e

podemos ir mais além, vamos dar uma olhada em como isso funcionaria com o novo **for-each** que aprendemos agora pouco.

```
for ( String s = myList ) {
        int x : s.length;
// note que não é necessário utilizar um Cast antes de chamar o método da string “s”! O compilado
}
```

Atividade 04

1. Crie um programa que manipule um Array de objetos do tipo Animal, adicione elementos a esse Array e depois recupere-os.
2. Crie um programa que manipule uma coleção de objetos do tipo Animal sem a utilização de Generics, adicione e recupere os elementos da coleção.
3. Evolua o exercício 2 para que a coleção utilize Generics. Adicione e recupere os elementos.

Leitura Complementar

Sobre definição de Arrays Java na Wikipédia:

<http://pt.wikipedia.org/wiki/Array>

Dois artigos que falam sobre Collections em Java:

<http://www.devmedia.com.br/articles/viewcomp.asp?comp=3162>

<http://javafree.uol.com.br/artigo/847654/Collections-Framework.html>

Resumo

Você estudou hoje que em sistemas grandes é bastante comum precisarmos de um grande conjunto de objetos. Você viu que a linguagem Java oferece dois mecanismos para armazenar um grande número de objetos: arrays e coleções. Agora, você já sabe que os arrays permitem armazenar um conjunto de objetos com tamanho fixo definido na sua inicialização. Já as coleções podem ser usadas para armazenar um conjunto não-finito de elementos, e são implementadas em Java como um conjunto de interfaces, que oferecem um conjunto de métodos padronizados para armazenamento e recuperação. Vimos também o uso do novo for-each, que facilita a manipulação das coleções, e aprendemos como a utilização do Generics nos ajuda e facilita, evitando a utilização de Cast e garantindo os tipos dentro das coleções.

Autoavaliação

1. Elabore um programa que exiba um calendário do dia 1 ao dia 31, utilizando um array de duas dimensões.
2. Qual a principal diferença entre um array convencional e a classe ArrayList que você destacaria?

3. Que tipo de coleção se assemelha a um conjunto matemático? Por quê? Explique.

4. Qual o principal motivação para a utilização de Generics?

Referências

THE JAVA tutorials. **Arrays**. Disponível em: <http://java.sun.com/docs/books/tutorial/java/nutsandbolts/arrays.html>. Acesso em: 15 maio 2010a.

_____.**Lesson**: interfaces. Disponível em: <http://java.sun.com/docs/books/tutorial/collections/interfaces/index.html>. Acesso em: 15 maio 2010b.

_____.**Lesson**: implementations. Disponível em: <http://java.sun.com/docs/books/tutorial/collections/implementations/index.html>. Acesso em: 15 maio 2010c.