

ORIENTAÇÃO A OBJETOS

AULA 6

Estrutura Homogênea
Dinâmica, Pacotes e
Generics

Vandor Roberto Vilardi Rissoli



APRESENTAÇÃO

- Estrutura Homogênea Dinâmica
 - *ArrayList*
- Pacotes
- Generics
- Referências



Estrutura Homogênea Dinâmica

ArrayList

Um ArrayList combina características de Array e List, possibilitando acesso aleatório e eficiente através de seu índice (como Array) e a inserção de novos elementos sem a limitação de quantidade máxima, como no Vector.

- Disponível no pacote **java.util**
- Implementação de uma **lista** que usa internamente um array de objetos
- Um novo elemento é inserido como último elemento
- O acesso a um **objeto** já existente é realizado pelo método **get()** que tem como parâmetro um valor inteiro que especifica o índice do elemento no ArrayList
- Na inserção onde o array interno não é suficiente, um novo array é alocado com aumento de metade do tamanho original, sendo todo seu conteúdo copiado para este novo array maior (1,5 vezes o Array original)

Estrutura Homogênea Dinâmica

Métodos Importantes no Uso da ArrayList

- **ArrayList**: cria objeto ArrayList
- **add(object)**: adiciona objeto indicado como último elemento desta estrutura de dados flexível
- **Object get(int)**: retorna o objeto armazenado na posição indicada, devendo ser convertido (*cast*) para objeto correto a ser manipulado pelo programa
- **remove(int)**: remove o objeto da posição indicada
- **remove(object)**: remove o objeto indicado



Estrutura Homogênea Dinâmica

Vector X ArrayList

A principal diferença entre estas duas estruturas de dados compostas dinâmicas em Java está na inserção de novos dados. As duas estruturas são dinâmicas e aumentam conforme a necessidade de armazenamento e quantidade de recurso disponível no computador.

Vector

- tamanho variável
- armazena objetos
- adiciona novos elementos na posição indicada
- sincronizado

ArrayList

- tamanho variável
- armazena objetos
- adiciona novos elementos na última posição (operação mais adequada e eficiente)

PACOTES

O desenvolvimento de uma aplicação Java pode ser melhor organizada em sua estrutura lógica e de armazenamento de seus recursos, contribuindo com a segurança e a manutenção no código da aplicação.

- Respeitando o padrão de desenvolvimento empregado são criados pacotes que guardam as classes e demais recursos elaborados por uma aplicação;
- Esta criação usa nomes significativos para aplicação;
- O uso dos recursos de um pacote são coerentes aos seus respectivos qualificadores de acesso, geralmente, sendo necessária a importação de pacotes diferentes ao que está usando e se deseja aproveitar um recurso disponível em outro pacote;
- Cada pacote cria uma estrutura de diretórios (pastas) no projeto que se está desenvolvendo.

Estrutura Homogênea Dinâmica

```
/** Síntese
 *   Objetivo: cadastrar um grupo de pessoas
 *   Entrada: nome e idade de cada pessoa
 *   Saída: relação de todas as pessoas cadastradas
 */
package principal;          // pacote chamado principal
import java.util.ArrayList;
import java.util.Scanner;
import servicos.*; // importa classe Servicos e Visao
public class Principal {
    public static void main(String[] args) {
        // Declarações
        ArrayList<Pessoa> pessoas = new ArrayList();
        // Instruções
        do {
            pessoas.add(Servicos.lePessoa(
                Visao.lerString("Informe o nome da pessoa: "),
                Visao.lerInteiro("Informe a idade: ", 1, 130)));
            Servicos.limpaTela(5);
        } while (Visao.lerContinua(
            "Deseja fazer novo cadastro (S=Sim e N=Não)?"));
        Visao.mostraPessoa(pessoas);
    }
}
```



Estrutura Homogênea Dinâmica

```
/** Síntese
 * Conteúdo:
 *   - isValidaString(String), lePessoa(String, int)
 *   - isValidaContinua(char), limpaTela(int)
 *   - isValidaInteiro(int, int, int)
 */
package servicos; // pacote chamado servicos
import dados.Pessoa; // importa classe Pessoa
public class Servicos {
    public static boolean isValidaString(String str) {
        return (!str.isEmpty());
    }
    public static boolean isValidaInteiro(int minimo,
                                         int maximo, int inteiro) {
        return (((inteiro < minimo) || (inteiro > maximo)) ?
                false : true);
    }
    public static boolean isValidaContinua(char continua) {
        return (((continua != 's') && (continua != 'n')) ?
                false : true);
    }
    public static Pessoa lePessoa(String nome, int idade) {
        Pessoa pes = new Pessoa(nome, idade);
        return pes;
    }
}
```

Estrutura Homogênea Dinâmica

```
// continuação...
public static void limpaTela(int linhas) {
    for(int aux = 0; aux < linhas; aux++)
        System.out.println();
}

}

/** Síntese
 * Conteúdo: Pessoa - nome, idade
 *   - getNome(), getIdade()
 *   - setNome(String), setIdade(int)
 */
package dados;
public class Pessoa {
    private String nome;
    private int idade;
    public Pessoa() { // construtor com valores padrões
    }

    public Pessoa(String nomeParametro,
                  int idadeParametro) {
        this.setNome(nomeParametro);
        this.setIdade(idadeParametro);
    }
}
```



Estrutura Homogênea Dinâmica

```
// continuação...
// Métodos assessores (get's e set's)
public String getNome() {
    return nome;
}
public void setNome(String nome) {
    this.nome = nome;
}
public int getIdade() {
    return idade;
}
public void setIdade(int idade) {
    this.idade = idade;
}
}
//
/** Síntese
 * Conteúdo:
 *     - leString(), leInteiro(), leChar()
 */
package servicos;
import java.util.Scanner;
public class MeuScanner {
```

Estrutura Homogênea Dinâmica

```
// continuação...
public static String leString() {
    Scanner ler = new Scanner(System.in);
    String string = ler.nextLine();      // nome completo
    return string;
}
public static int leInteiro() {
    Scanner ler = new Scanner(System.in);
    int inteiro = ler.nextInt();
    return inteiro;
}
public static char leChar() {
    Scanner ler = new Scanner(System.in);
    char caracter = ler.next().toLowerCase().charAt(0);
    return caracter;
}
/** Síntese
 * Conteúdo:
 * - lerString(String), lerInteiro(String,int,int)
 * - lerContinua(String), mostraPessoa(ArrayList)
 */
package servicos;
import java.util.ArrayList;
import java.util.InputMismatchException;
import dados.Pessoa;
```

Estrutura Homogênea Dinâmica

```
// continuação...
public class Visao {
    public static String lerString(String mensagem) {
        String valorLido;
        System.out.println(mensagem);
        do { valorLido = MeuScanner.leString();
            if(!Servicos.isValidaString(valorLido))
                System.out.print(
                    "Valor inválido, informe novamente: ");
            } while (!Servicos.isValidaString(valorLido));
        return valorLido;
    }
    public static boolean lerContinua(String mensagem) {
        char valorLido;
        System.out.println(mensagem);
        do { valorLido = MeuScanner.leChar();
            if(!Servicos.isValidaContinua(valorLido))
                System.out.print("Valor inválido, "+
                    "informe novamente: ");
            } while(!Servicos.isValidaContinua(valorLido));
        if(valorLido == 's')
            return true;
        else
            return false;
    }
}
```

Estrutura Homogênea Dinâmica

Estrutura Homogênea Dinâmica

```
// continuação...
public static void mostraPessoa(ArrayList<Pessoa> pessoas) {
    Pessoa pes;
    Servicos.limpaTela(15);
    System.out.println("NOME\t\tIDADE");
    System.out.println("====\t\t=====");
    for(int aux = 0;aux < pessoas.size(); aux++) {
        pes = (Pessoa) pessoas.get(aux);
        System.out.println(pes.getNome() +"\t\t"+
                           pes.getIdade());
    }
}
```

— // —

A nova classe, **Visao**, corresponde aos novos aspectos lógicos deste programa que interage com seu usuário, por meio de sua interface mostrada para “visão” orientadora do usuário.

Generics

Na versão Java 1.5 (ou Java 5) foi realizada certa adequação para evolução da linguagem no uso de **Generics** ou tipo parametrizado.

Por meio desta implementação se almeja diminuir os problemas constantes com conversões errôneas em Java, pois esta linguagem realiza comumente várias conversões (*cast*).

O uso de Generics permite que uma única classe trabalhe com uma grande variedade de tipos, eliminando, de forma natural, a necessidade de conversões constantes.

O Generics na classe `ArrayList` foi elaborado para trabalhar nativamente com qualquer tipo de classe, preservando ainda os benefícios da checagem de tipos.

Generics

```
String str1 = (String) ArrayList1.get(0);
```

evoluiu para

```
String str1 = ArrayList1.get(0);
```

sem necessidade de conversão (*cast*)

Apesar de não necessitar mais do *cast*, esta estrutura de dados pode receber qualquer objeto, independente de ser String ou não.

O método `get()` a recuperaria, mas um erro de equiparação de tipos seria apresentado em tempo de compilação, pois tal objeto não seria uma String para ser armazenada em `str1`.



Generics

Suponha a criação de um `ArrayList` para guardar **Cães**, onde equivocadamente alguém inseriu um **Gato** no meio de todos os Cães que já estavam lá.

Sem Generics isso poderia acontecer sem que o compilador nos comunique o problema antes dele ser executado. No entanto, o Generics permite a checagem de tipo em tempo de compilação, impedindo os possíveis transtornos ao usuário final que se deparará com este problema identificado em tempo de execução (exceção gerada **ClassCastException**).

A sintaxe geral para definição do Generics envolve o tipo utilizado de parâmetro entre "<" e ">" junto ao nome da classe.

```
ArrayList<String> str // por exemplo
```

Generics

Com isso é possível rever a solução do exemplo anterior (lista de nomes) e parametrizar o ArrayList para receber somente String.

```
package strings;
import java.util.*;
public class Strings {
    public static void main(String[] args) {
        String nome1 = new String("Ana Maria Braga");
        String nome2 = new String("Lula da Silva");
        String nome3 = new String("Carlos Drumont");
        // Conjunto de objetos que só guarda String
        ArrayList<String> nomes = new ArrayList<String>();
        nomes.add(nome1);
        nomes.add(nome2);
        nomes.add(nome3);
        for(int aux = 0; aux < nomes.size(); aux++)
            System.out.println(nomes.get(aux));
    }
}
```

Generics

Uma classe ou método **paramétrico** pode ser invocado com tipos diferentes, sendo possível definir uma variável dentro de uma classe ou parâmetro de um método como um tipo Generics. Somente quando estes forem ser utilizados é que este tipo paramétrico será definido por seu usuário.

Características Importantes:

- Flexibiliza a codificação, permitindo a criação de soluções mais genéricas
- Reduz bastante o programa (linhas de código)
- Facilita o processo de manutenção

→ Toda a API padrão da linguagem (todas as classes que implementam **coleções** por exemplo) foi refeita para tirar proveito destas facilidades possíveis com **Generics**.

Generics

Observe abaixo um exemplo básico de uso de tipos paramétricos ou genéricos (Generics).

```
package strings;
/** Síntese
 *   Objetivo: mostrar objetos guardados na lista
 *   Entrada: nenhuma (só atribuições)
 *   Saída: apresentar os elementos armazenados */
import java.util.ArrayList;
public class ListaFutebol {
    public static void main(String[] args) {
        ArrayList<String> dados = new ArrayList<String>();
        dados.add(new String("Flamengo"));
        dados.add(new String("Vasco"));
        dados.add(new Times("Botafogo")); // checa o tipo
        String nome = null;
        int aux = 0;
        while(aux < dados.size()) {
            nome = dados.get(aux);
            System.out.println(nome);
            aux++;
        }
    }
}
```

Exercício de Fixação

- 1) Elabore um programa que permita o cadastramento dos nomes e quantas vezes os times de futebol nacionais já foram campeões do Campeonato Brasileiro. A quantidade de cadastro não é conhecida, mas o usuário poderá cadastrar quantos times ele desejar neste programa orientado a objeto. Empregue em sua solução todos os conteúdos estudados em POO (Programação Orientada a Objeto) até o momento e utilize a parametrização para armazenar dados referenciáveis para diminuir a quantidade de conversões (*casting*). Sua solução deverá estar no pacote **campeonato**.



Referência de Criação e Apoio ao Estudo

Material para Consulta e Apoio ao Conteúdo

- HORSTMANN, C. S., CORNELL, G., Core Java2 , volume 1, Makron Books, 2001.
 - Capítulo 5
- FURGERI, S., Java 2: Ensino Didático: Desenvolvendo e Implementando Aplicações, São Paulo: Érica, 2002.
 - Capítulo 7
- Universidade de Brasília (UnB Gama)
 - <http://cae.ucb.br/conteudo/unbfga>
(escolha a disciplina **Orientação a Objetos** no menu superior)

