UNINASSAU
Campus Graças

Estrutura de dados

PROFESSOR:

Adilson da Silva

CURSO (2024.1)

UNINASSAU UNAMA UNG UNINORTE UNESC UNIFAEL UNI7
Grupo Ser Educacional

MA UNG UNINORTE UNESC UNIFAEL UNI7
Grupo Ser Educacional

UNINASSAU
Campus Graças

ArrayList

ArrayList

UNINASSAU
Campus Graças

- As ***Collections*** são estruturas de dados pré-definidas, fornecidas pela API do Java, que são empregadas para armazenar grupos de objetos relacionados na memória durante a execução do programa.
- Com estas classes é possível organizar, armazenar e ler dados sem a necessidade de saber exatamente como os dados são armazenados (encapsulamento).

ArrayList

UNINASSAU
Campus Graças

- Um **array convencional** não tem seu tamanho modificado automaticamente durante a execução do programa, por exemplo para armazenar novos elementos, o que pode ser um problema, principalmente se surge a necessidade de armazenar mais elementos após um array estar completo.

ArrayList

UNINASSAU
Campus Graças

- O framework **Collections** em Java fornece diversas interfaces e classes para manipulação de grupos de objetos, como por exemplo:

- Interface
  - List
  - Set
  - Queue
  - Deque
  - SortedSet

- Classes
  - ArrayList
  - LinkedList
  - Vector
  - HashSet
  - ProyityQueue
  - LinkedHashSet
  - TreeSet

Classe ArrayList

UNINASSAU
Campus Graças

- A classe de coleção (collections class) **ArrayList<T>**, pertencente ao pacote **java.util**, e que implementa a **interface List**, resolve o problema de modificação do tamanho de um array em tempo de execução, pois ela pode mudar seu tamanho de forma dinâmica – ou seja, permite criar um array dinâmico.
- A letra **T** indica um *placeholder*, que é substituído ao declarar um ArrayList pelo tipo de elementos que a estrutura irá armazenar.
- Suponha que desejemos criar um ArrayList de **strings**, de nome produtos. Para tal, executamos a seguinte declaração:

ArrayList<String> produtos;

Classe ArrayList

UNINASSAU
Campus Graças

- Note que somente tipos não-primitivos podem ser usados para declarar variáveis e criar objetos em classes genéricas – e o ArrayList é uma classe genérica. Ainda assim, em Java é possível usar tipos primitivos, desde que empacotados como se fossem objetos, em um processo denominado **boxing**.

- Desta forma, poderíamos criar um ArrayList de inteiros também, por exemplo:

```
ArrayList<Integer> numeros;
```

Características do ArrayList

UNINASSAU
Campus Graças

- O ArrayList em Java é uma classe que implementa a estrutura de dados de lista dinâmica, o que significa que ela pode crescer e diminuir dinamicamente de tamanho conforme necessário. Aqui estão algumas características importantes do ArrayList em Java:
- **Tamanho Dinâmico:** Uma das principais características do ArrayList é sua capacidade de crescer e diminuir dinamicamente de tamanho conforme os elementos são adicionados ou removidos. Isso significa que você não precisa especificar o tamanho do ArrayList antecipadamente.
- **Acesso Aleatório:** Os elementos em um ArrayList podem ser acessados rapidamente por meio de índices. Isso permite um acesso aleatório eficiente aos elementos, o que significa que você pode acessar qualquer elemento diretamente, sem precisar percorrer a lista sequencialmente.

Características do ArrayList

UNINASSAU
Campus Graças

- **Permite Duplicatas:** O ArrayList permite que você armazene elementos duplicados. Isso significa que você pode ter vários elementos na lista com o mesmo valor.
- **Implementa a Interface List:** O ArrayList implementa a interface List, o que significa que ele herda todos os métodos definidos na interface List. Isso inclui métodos para adicionar, remover, acessar elementos e muito mais.
- **Suporta Iteração:** O ArrayList suporta iteração por meio de loops for-each ou usando iteradores. Isso permite percorrer todos os elementos da lista de maneira eficiente.
- **Reserva de Capacidade:** O ArrayList tem a capacidade de reservar uma capacidade inicial para evitar realocações frequentes de memória à medida que os elementos são adicionados. Isso pode melhorar o desempenho em casos onde o tamanho máximo da lista é conhecido antecipadamente.

Características do ArrayList

UNINASSAU
Campus Graças

- **Realocação Automática de Memória:** Quando o ArrayList atinge sua capacidade máxima, ele realoca automaticamente memória para acomodar mais elementos. Isso é transparente para o usuário e permite que o ArrayList cresça dinamicamente conforme necessário.
- **Eficiência para Acesso e Modificação:** O acesso e a modificação de elementos em um ArrayList são geralmente muito eficientes, especialmente para operações de acesso aleatório e adição/remoção de elementos no final da lista. No entanto, adicionar ou remover elementos no meio da lista pode ser menos eficiente devido à necessidade de deslocar elementos subsequentes.

Métodos de um ArrayList

UNINASSAU
Campus Graças

- **Adição de Elementos:**
  - **add(E elemento):** Adiciona o elemento especificado ao final da lista.
  - **add(int índice, E elemento):** Insere o elemento especificado na posição especificada na lista.
- **Remoção de Elementos:**
  - **remove(int índice):** Remove o elemento na posição especificada na lista.
  - **remove(Object objeto):** Remove a primeira ocorrência do objeto especificado da lista.
- **Acesso a Elementos:**
  - **get(int índice):** Retorna o elemento na posição especificada na lista.
  - **set(int índice, E elemento):** Substitui o elemento na posição especificada na lista pelo elemento especificado.

Métodos de um ArrayList

UNINASSAU
Campus Graças

- **Verificação e Busca:**
  - contains(Object objeto): Retorna true se a lista contiver o objeto especificado.
  - isEmpty(): Retorna true se a lista estiver vazia.
  - indexOf(Object objeto): Retorna o índice da primeira ocorrência do objeto especificado na lista.
  - lastIndexOf(Object objeto): Retorna o índice da última ocorrência do objeto especificado na lista.
- **Informações sobre a Lista:**
  - size(): Retorna o número de elementos na lista.
  - clear(): Remove todos os elementos da lista.
- **Conversão para Arrays:**
  - toArray(): Retorna uma matriz contendo todos os elementos da lista na ordem em que são armazenados.

Uso de ArrayList

UNINASSAU
Campus Graças

- Neste exemplo criamos um ArrayList de String chamado de frutas, e adicionamos dois nomes de frutas a ele usando o método ***add***.
- Logo após, mostramos o conteúdo do ArrayList iterando pelos seus elementos com um laço for, e lendo cada item com o método ***get***.
- Também criamos um método especial para mostrar o conteúdo do ArrayList, de nome ***mostrar()***, que recebe como argumentos o nome do ArrayList e uma mensagem a ser exibida juntamente com os elementos retornados.
- Removemos um elemento usando o método ***remove***, e verificamos o seu tamanho com o método ***size***.
- Finalmente, verificamos se um item está contido no ArrayList usando o método ***contains***.
- Na próxima lição vamos estudar as listas ligadas, implementadas com a classe LinkedList.

Exemplo

UNINASSAU
Campus Graças

```
package exeplo0;
import java.util.ArrayList;

public class Exemplo0 {

    public static void main(String[] args) {
    // Criar um novo ArrayList de Strings com capacidade inicial de 10 elementos
    (padrão)
    ArrayList<String> frutas = new ArrayList<>();

    frutas.add("Maçã"); // Adicionar um item à lista
    frutas.add(1, "Uva"); // Inserir a fruta Uva na posição de índice 0
    frutas.add(0, "Laranja");
    System.out.println("Mostrar o conteúdo do ArrayList usando laço for
    convencional:");
```

Exemplo

UNINASSAU
Campus Graças

```
for (int i = 0; i < frutas.size(); i++)
    System.out.printf(" %s", frutas.get(i));
// Mostrar as frutas com o método mostrar(). Passamos a lista e um frase-cabeçalho
mostrar(frutas,"%nMostrando a lista de frutas com o método mostrar:");

frutas.add("Tangerina"); // Adicionar tangerina ao final da lista
frutas.add("Morango"); // Idem com Morango
mostrar(frutas, "Nova visualização da lista após adicionar frutas:");

frutas.remove("Uva"); // remover a fruta Uva"
mostrar(frutas, "Nova visualização da lista após remover frutas:");

frutas.remove(1); // Remover item na posição de índice 1
mostrar(frutas, "Remover o segundo elemento da lista (índice 1):");
```

Exemplo

UNINASSAU
Campus Graças

```
    // Verificar número de elementos na lista
    System.out.printf("Tamanho atual do ArrayList: %s itens%n", frutas.size());

    // Verificar se um item existe na lista
    System.out.printf("A fruta \"Tangerina\" %sestá na lista%n",
    frutas.contains("Tangerina") ? "": "não ");
   }
   // criar método para mostrar os elementos do ArrayList frutas
  public static void mostrar(ArrayList<String> frutas, String cabecalho) {
    System.out.printf(cabecalho); // mostrar cabeçalho

  for (String item : frutas)
     System.out.printf(" %s", item);
     System.out.println();
  }
}
```

UNINASSAU
Campus Graças

Exceções

O que são exceções?

UNINASSAU
Campus Graças

- Uma exceção é uma condição anormal que altera ou interrompe o fluxo de execução.
- Podem ser causadas por diversas condições:
  - Erros sérios de hardware;
  - Erros simples de programação;
  - Erros de divisão por zero;
  - Valores fora de faixa
  - Valores de variáveis
  - Erro na procura/abertura de um arquivo (entrada/saída)
  - Falha na memória

O que são exceções?

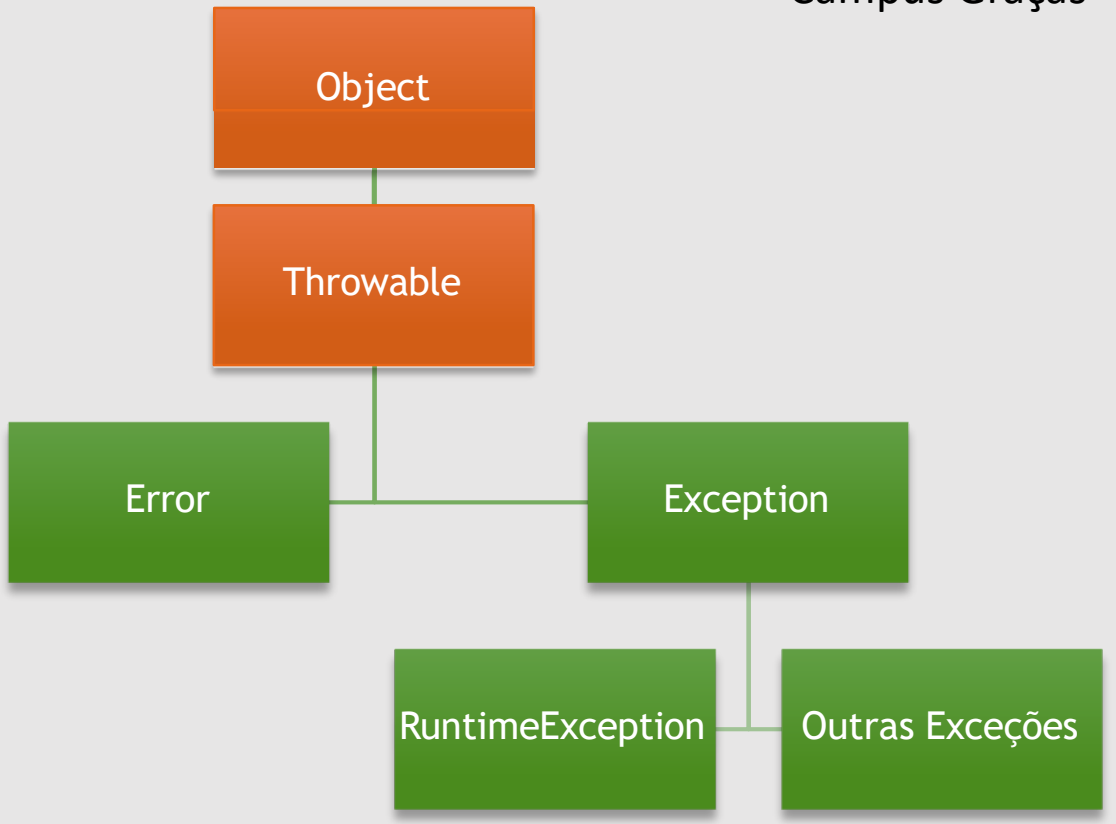
UNINASSAU
Campus Graças

- Se não for tratada, o programa pode parar
- O uso correto de exceções torna o programa mais robusto e confiável
  - Uso exagerado polui o código e torna o programa mais lento

Tipos de exceção

UNINASSAU
Campus Graças

- Exceções, como (quase) tudo em Java, são objetos;



Tipos de exceção

UNINASSAU
Campus Graças

- Exceções e erros se enquadram em **três categorias**:
  - Exceções verificadas;
  - Exceções não verificadas e;
  - Erros.

Exceções Verificadas

UNINASSAU
Campus Graças

- São **verificadas pelo compilador em tempo de compilação**;
- Os métodos que lançam uma exceção devem indicar isso na declaração do método através da cláusula **throws**.
- Todas as exceções verificadas devem ser capturadas explicitamente com
  - um bloco **try – catch**;
- Exceções do tipo **Exception** e todos os seus subtipos, exceto
  - **RunTimeException** e os seus subtipos.

Exceções Não Verificadas

UNINASSAU
Campus Graças

- **Não verificadas em tempo de compilação**;
- Ocorrem **durante a execução por causa de erro do programador** (divisão por zero, índice fora de faixa e uso de referências null);
- Não exigem tratamento;
- As exceções não verificadas incluem as exceções do tipo
  - **RunTimeException** e todos os seus subtipos.
- A regra "Se for um RunTimeException, a culpa é sua" funciona muito bem.

Erros

UNINASSAU
Campus Graças

- Os erros **são tipicamente irrecuperáveis e apresentam condições sérias**.
- Os erros não são verificados em tempo de compilação;

Palavras-chave

UNINASSAU
Campus Graças

- Em Java, o código para tratamento de erros é separado de forma limpa do código que gera erros.
  - Diz-se que **o código que gera a exceção "lança" ("throw") uma exceção**,
  - enquanto que **o código que trata a exceção "captura" ("catch") a exceção**.

Testando e capturando exceções

UNINASSAU
Campus Graças

- Um bloco que tenta (try) chamar um método (ou conjunto de classes) que
  - pode disparar uma exceção deve trata-la
    - Chamada normal de um método, mas que deve estar em um bloco try {...} catch
      - {...}
- Uma exceção é um objeto que deve ser capturado (catch)
  - É nesse bloco que a exceção deve ser tratada
- Um trecho de código pode ser executado sempre
  - bloco finally

Testando e capturando exceções

UNINASSAU
Campus Graças

```
try
{
   //Executa código que pode disparar exceção
}
catch(Exception ex)
{
   //Trata exceção. ex é uma referência para o
   objeto da classe Excpetion a ser tratado
}
finally
{
   //Esse bloco é sempre executado,
   independente de ocorrer exceção.
}
```

Tenta Executar

Captura exceção

Sempre executa

Exceções Frequentes

UNINASSAU
Campus Graças

- ArithmeticException
- ClassNotFoundException
- DataFormatException
- FileNotFoundException
- IndexOutOfBoundsException
- NullPointerException
- NumberFormatException
- SQLException

Testando e capturando exceções

UNINASSAU
Campus Graças

- Se a exceção for disparada, o bloco try não será mais executado a partir do ponto onde a exceção ocorreu
- O fluxo de execução muda para a captura catch
- Pode haver mais de um bloco catch. A exceção é capturada pelo bloco que tratar a exceção correspondente.
- O bloco finally é opcional.

Testando e capturando exceções

UNINASSAU
Campus Graças

```
int x = 0;
try{
    int y = 100 / x;
    System.out.println ("Resultado: " + y);
}

catch (ArithmeticException e){ System.out.println

    ("Operação inválida!");
    System.out.println("\n Detalhes do erro: "+ e.getMessage());
}
```

Testando e capturando múltiplas exceções

UNINASSAU
Campus Graças

```
Scanner sc = new Scanner(System.in);
try{
        int x = sc.nextInt(); int y = 100 / x;
        System.out.println ("Resultado: " + y);
}
catch (InputMismatchException e){
        System.out.println (“Formato inválido!");
        System.out.println("\n Detalhes do erro:“+ e.getMessage());
}
catch (ArithmeticException e){ System.out.println (“Operação
        inválida!");
        System.out.println("\n Detalhes do erro:“+ e.getMessage());
```

java.lang.Throwable

UNINASSAU
Campus Graças

- Ancestral de todas as classes que recebem tratamento do mecanismo de exceções;
- Principais métodos:
  - **void printStackTrace():** lista a seqüência de métodos chamados até o ponto onde a exceção foi lançada;
  - **String getMessage():** contém uma mensagem indicadora da exceção;
  - **O método toString():** retorna uma descrição breve da exceção.

java.lang.Error

- Representa um **problema grave**, de difícil (ou impossível) recuperação;
- Exemplos:
  - **OutOfMemoryError**
  - **StackOverflowError**
  - **VirtualMachineError**
  - etc.
- Geralmente causam o encerramento do programa;
- Não devem ser usadas pelos programadores.

java.lang.Exception

UNINASSAU
Campus Graças

- Exceções que podem ser lançadas pelos métodos da API Java ou pelo seu programa;
- Devem ser tratadas;
- Em geral, representam situações inesperadas, porém contornáveis;
- O programador tem contato com esta classe e suas subclasses.

java.lang.RuntimeException

UNINASSAU
Campus Graças

- Tipo especial de exceção;
- Não necessitam ser lançadas explicitamente pelo programa;
- Seu tratamento não é obrigatório; Ex.:
  - **NullPointerException**,
  - **IndexOutOfBoundsException**, etc.

Exemplo (NumberFormatException)

UNINASSAU
Campus Graças

```
public static void main(String[] args) {

    String var = JOptionPane.showInputDialog("Digite um número inteiro:");

    try {

        Integer i = new Integer(var);
        JOptionPane.showMessageDialog(null, "O número digitado foi: " + i);

    } catch (NumberFormatException nfe) {

        JOptionPane.showMessageDialog(null,
                "Não é possível atribuir esse valor ao número inteiro");

        System.out.println("Erro: A seguinte mensagem foi retornada:\n"+
                nfe.getMessage());
    }
}
```

Exemplo (IndexOutOfBoundException)

UNINASSAU
Campus Graças

```
public static void main(String[] args) {
    int [] values = {0, 10, 20, 30, 40, 50};

    String var = JOptionPane.showInputDialog("Digite um número: ");
    try {
       int i = Integer.parseInt(var);
       JOptionPane.showMessageDialog(null,
               "O valor na posição "+ i + " é: " + values[i]);


    } catch (IndexOutOfBoundsException iob) {

        JOptionPane.showMessageDialog(null,
                "Não é possível acessar esse índice no vetor");
        iob.printStackTrace();
    }
}
```

A palavra-chave throw

UNINASSAU
Campus Graças

- Para lançar uma exceção, use a palavra **throw**. Qualquer exceção verificada/não verificada, ou erro, pode ser lançado.

```
if (arquivo == null)
    throw new FileNotFoundException();
```

A palavra-chave throws

UNINASSAU
Campus Graças

- O lugar onde você anuncia qual método pode lançar uma exceção é o
  - cabeçalho do método;
- O cabeçalho muda para refletir as exceções verificadas que o método pode lançar.

```
public FileInputStream(String name) throws FileNotFoundException {
    //...
}
```

A cláusula finally

UNINASSAU
Campus Graças

- O código na cláusula finally executa se uma exceção foi ou não capturada.

```
int x = 0;
try{
        int y = 100 / x;
        System.out.println ("Resultado: " + y);
}
catch (ArithmeticException e){ System.out.println ("Operação
        inválida!");
        System.out.println("\n Detalhes do erro: "+ e.getMessage());
}
finally{ System.out.println ("Execução finalizada!");
        }
```

Obrigado

E- mail:adilson.silva@sereducacional.com

@prof.Adilson.silva