Universidade de Brasília Faculdade UnB Gama

Orientação a Objetos

Aula 7 - Vector e Exceções

Daniel Porto

daniel.porto@unb.br

APRESENTAÇÃO

Superclasse Object

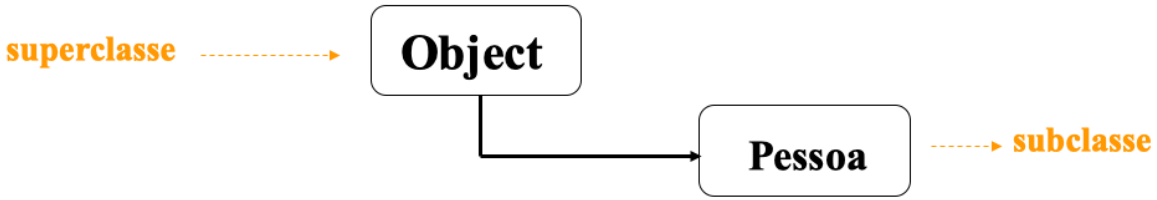
Estrutura Homogênea Dinâmica

- Vector

Tratamento de Exceções

SUPERCLASSE OBJECT

Toda classe criada em Java é subclasse da classe Object, não sendo necessário incluir nenhuma instrução específica para que isso se estabeleça.



Object é a classe referida quando nenhuma superclasse (classe pai) é indicada.

Pode-se usar uma variável do tipo Object para referenciar objetos de qualquer tipo (outros objetos).

O tipo Object só é útil como contêiner genérico de valores arbitrários, sendo necessário conhecer melhor o tipo original da classe para trabalhar sobre seus valores.

ESTRUTURA HOMOGÊNEA DINÂMICA

A criação de estruturas de dados compostas dinâmicas, para o armazenamento de objetos em memória, é possível em Java, por meio da criação de objetos da classe **Vector** ou da classe **ArrayList** (estudada a frente).

Vector

Disponível em **java.util.Vector**.

Consiste em um array de Object com dimensão dinâmica, conforme a necessidade de armazenamento.

Não guarda nenhum tipo de dado primitivo (embutidas).

Possui um número menor ou igual a sua capacidade de armazenamento.

Sua expansão pode acontecer por meio de um valor informado em sua construção ou pelo valor padrão, que corresponde ao dobro de seu tamanho atual.

ESTRUTURA HOMOGÊNEA DINÂMICA

Vector

- tamanho variável
- armazena somente objetos (tipo Object)
- Alteração sobre o mesmo objeto

Array

- tamanho fixo
- armazena valores de um mesmo tipo de dado
- Alteração cria um novo objeto em memória

ESTRUTURA HOMOGÊNEA DINÂMICA

Métodos Importantes no Uso da Vector

- **vector**: cria um vector vazio (sem objetos)
- **vector(int)**: cria um vector vazio com tamanho inicial
- **vector(int, int)**: cria vector vazio com tamanho inicial e valor de incremento indicado
- **size**: obtém o número de elementos do vector
- **setSize(int)**: configura tamanho do objeto vector, sendo maior que o atual adiciona novos elementos nulos e menor descarta os itens além do tamanho informado
- **isEmpty**: verifica se vector esta vazio
- **contains(object)**: verifica se object é componente de vector
- **indexOf(object)**: retorna o índice da primeira ocorrência de object no vector ou –1 se ele não estiver no vector
- **element(int)**: obtém o componente do índice parametrizado
- **toString**: retorna objeto String representando o vector

ESTRUTURA HOMOGÊNEA DINÂMICA

Métodos Importantes no Uso da Vector

- **setElementAt(object, int)**: copia object sobre o índice indicado
- **removeElement(int)**: remove o elemento de índice indicado, sendo vector reorganizado e diminuído em 1 posição
- **removeAllElements**: remove todos os elementos de vector e reduz seu tamanho para zero
- **insertElementAt(object, int)**: inseri um novo componente na posição indicada e reorganiza o vector em mais uma posição
- **addElement(object)**: vector é acrescido de uma nova posição que recebe o objeto indicado

ESTRUTURA HOMOGÊNEA DINÂMICA

```java
/** Síntese
 *    Objetivo: guardar e mostrar dados de pessoas
 *    Entrada:  nomes e idades
 *    Saída:    listagem das pessoas cadastradas
*/
import java.util.Vector;
public class CadastraPessoa {
    public static void main(String[] args) {
        Vector variasPessoas = new Vector();
        String nome;
        int idade, opcao;
        do {Pessoa pessoa = new Pessoa();
            System.out.println("Informe o nome: ");
            nome = Servicos.lerString();
            pessoa.setNome(nome);
            System.out.println("Digite a idade em anos: ");
            idade = Servicos.lerInt(1,130);
            pessoa.setIdade(idade);
            variasPessoas.add(pessoa);
            System.out.println("Novo cadastro (0-NÃO / 1-SIM)?");
            opcao = Servicos.lerInt(0, 1);
        } while (opcao == 1);
        Servicos.limparTela(20);
        Servicos.mostraTabela(variasPessoas);
    }
}
```

ESTRUTURA HOMOGÊNEA DINÂMICA

```java
/** Síntese
 *    Atributos: nome, idade
 *    Métodos:  getIdade(), setIdade(String), getNome(),
 *                setNome(String)
 */
public class Pessoa {
    private String nome;
    private int idade;

    public int getIdade() {
        return idade;
    }
    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
    public void setIdade(int idade) {
        this.idade = idade;
    }
}
```

ESTRUTURA HOMOGÊNEA DINÂMICA

```java
/** Síntese
 *    Atributos:
 *    Métodos: lerString(), lerInt(int, int),
 *             validaString(String, Scanner),
 *             validaInteiro(int, int, int)
 *             limparTela(int), mostraTabela(Vector)
 */
import java.util.Scanner;
import java.util.Vector;
public class Servicos {
    // Métodos de Leitura
    public static String lerString(){
        Scanner ler = new Scanner(System.in);
        return validaString(ler.nextLine(),ler);
    }
    public static int lerInt(int min, int max){
        Scanner ler = new Scanner(System.in);
        return validaInteiro(ler.nextInt(),min,max);
    }
    // Métodos de validação
    public static String validaString(String str, Scanner ler){
        while(str.isEmpty()) {
            System.out.println("Inválido, informe novamente.");
            str = ler.nextLine();
        }
        return str;
    }
```

ESTRUTURA HOMOGÊNEA DINÂMICA

```java
    // continuação do exemplo anterior

    public static int validaInteiro(int inteiro, int min, int max){
        while((inteiro < min) ||(inteiro > max)) {
            System.out.println("Inválido, informe novamente.");
            inteiro = lerInt(min,max);
        }
        return inteiro;
    }

    // Métodos de apresentação de dados
    public static void limparTela(int linhas) {
        for(int aux=1;aux<=linhas;aux++)
            System.out.println();
    }
    public static void mostraTabela(Vector pessoas) {
        System.out.println("NOME\t\tIDADE");
        for(int aux=0;aux < pessoas.size(); aux++) {
            Pessoa pes = (Pessoa) pessoas.get(aux);
            System.out.println(pes.getNome() + "\t" + pes.getIdade());
        }
    }
}
```

EXERCÍCIOS DE FIXAÇÃO

1) Elabore um programa que permita ao usuário cadastrar quantos nomes ele quiser e quando terminar de cadastrar apresente todos os nomes cadastrados na ordem que o usuário escolher:

OPÇÕES DE APRESENTAÇÃO

1- Sequência de inserção;

2- Sequência inversa de inserção;

0- Encerra sem mostrar os nomes cadastrados.

Esta solução deverá possuir um método específico para cada tipo de apresentação (inserção ou inversa).

EXERCÍCIOS DE FIXAÇÃO

2) Desenvolva um programa que armazene o nome completo de um aluno, sua matrícula na instituição, que nunca pode se repetir (valor chave sempre maior que a constante MAX com valor 1000 e nunca podendo ser igual a outra matrícula já existente), além de sua média final (igual ou maior que zero e até 10 pontos).

Faça um programa respeitando TODAS as regras e definições para POO e cadastre todos os alunos desejados pelo usuário. Quando este usuário não quiser mais cadastrar alunos apresente um relatório do tipo de uma tabela (tabelar) contendo somente uma linha de cabeçalho seguida de uma linha para cada registro de aluno com todos registro efetuados neste programa.

TRATAMENTO DE EXCEÇÕES

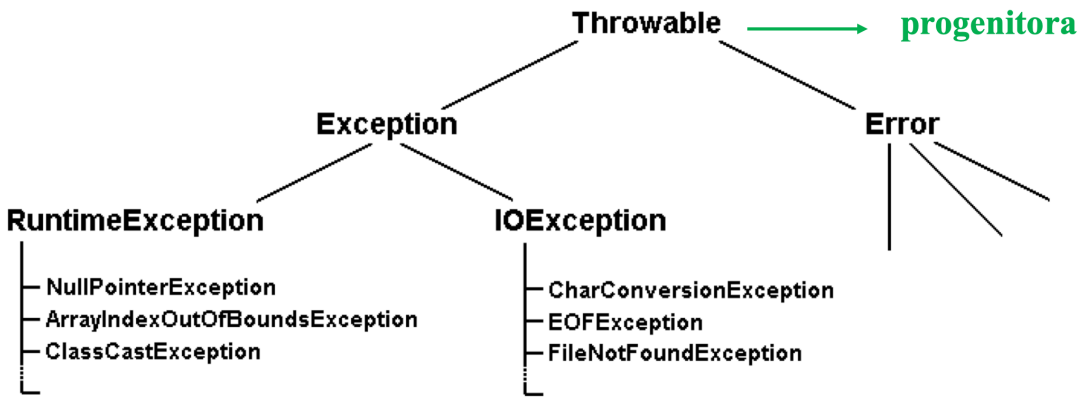
A ocorrência de uma situação inesperada no fluxo normal de processamento pode acontecer durante a execução de um programa, geralmente deixando seus usuários descontentes e até prejudicando as atividades realizadas até aquele momento inoportuno.

Estas ocorrências são tratadas de maneira especial em Java, sendo denominadas de **tratamento de exceção**.

As exceções se referem aos erros que podem ser gerados durante a execução de um programa, sendo aplicada a esta situação um conjunto de recursos Java que permitirão a **manipulação e o tratamento da maioria** dessas possíveis situações.

TRATAMENTO DE EXCEÇÕES

Estas possíveis exceções em Java possuem uma sintaxe própria para serem identificadas e tratadas, além de fazerem parte de uma hierarquia especial de heranças.



O tratamento das exceções se concentra, quase que totalmente, na Exception, pois praticamente a totalidade da Error esta fora de seu controle, sendo interessante na ocorrência desta última o encerramento de seu processo.

TRATAMENTO DE EXCEÇÕES

Dois tipos de tratamento de exceção podem ser implementados em um programa Java, sendo estes classificados em:

- **Verificados**: exigem um tratamento de exceção no desenvolvimento do programa
  - Provoca **erro de compilação** quando não é implementado em seu código (método)
  - Exemplo: abertura de um arquivo de dados
- **Não Verificados**: ou estão fora do controle do programa que está sendo elaborado, ou seu programa não deveria permitir que os mesmos acontecessem
  - **Não** provoca erro de compilação se não for implementado, pois é considerado de responsabilidade de seu programador
  - Exemplo: divisão por zero

TRATAMENTO DE EXCEÇÕES

Instrução try... catch

Uma das formas de tratar as exceções é o uso da instrução try...catch, que possibilita a averiguação de uma ou várias exceções em um conjunto de instruções.

Suas principais características são:

- Abertura e fechamento de bloco de instrução é obrigatória para cada sub-bloco que compõem esta instrução ({ })
- A instrução finally corresponde a um outro bloco de instrução **opcional** na instrução try...catch, similar ao else na instrução if
- Para cada try pode haver um ou mais catch, mas somente um finally ou ambos (catch e finally) nesta instrução try
- Cada bloco catch define o tratamento de uma **única exceção**, recebendo como parâmetro sua identificação
- Esta exceção deve pertencer a classe Throwable ou uma de suas subclasses

TRATAMENTO DE EXCEÇÕES

```java
/** Síntese
 *    Objetivo: analisar a paridade de um número
 *    Entrada:  número inteiro
 *    Saída:    paridade do número
*/
import java.util.Scanner;
public class Paridade {
    public static void main(String[] args) {
        int numero;
        Scanner ler = new Scanner(System.in);
        System.out.println("Digite um número inteiro.");
        numero = ler.nextInt();
        System.out.print("\n\n\nO número "+ numero +" é\t");
        if((numero % 2) == 0)
            System.out.print("PAR");
        else
            System.out.println("IMPAR");
    }
}
```

TRATAMENTO DE EXCEÇÕES

Após observar o funcionamento correto do programa anterior, verifique o que a desatenção de um usuário pode ocasionar no uso do mesmo.

Imagine que o valor informado pelo usuário foi um valor real, ou ainda ele digitou por engano uma letra ou um outro caracter especial. Este programa não previa esta possibilidade e por isso será encerrado sem uma orientação do motivo disso acontecer para seu usuário.

Para se evitar esta situação tente elaborar no mesmo um tratamento de exceção que impeça este término de programa mais indelicado e oriente seu usuário sobre o que aconteceu **usando o tratamento de exceção**.

Muitas são as exceções disponíveis em Java, sendo por isso interessante testar a ocorrência das mesmas e implementar seu tratamento, fazendo uma **cópia de sua identificação**.

TRATAMENTO DE EXCEÇÕES

```java
/** Síntese
 *    Objetivo: analisar a paridade de um número
 *    Entrada:  número inteiro
 *    Saída:    paridade do número
*/
import java.util.Scanner;
import java.util.InputMismatchException;
public class Paridade2 {   // novo código com exceção
    public static void main(String[] args) {
        int numero;
        Scanner ler = new Scanner(System.in);
        System.out.println("Digite um número inteiro.");
        try {
            numero = ler.nextInt();
            System.out.print("\n\nO número "+ numero +" é\t");
            if((numero % 2) == 0)
                System.out.print("PAR");
            else
                System.out.println("IMPAR");
        } catch (InputMismatchException excecao) {
            System.out.print("Valor incorreto. Paridade ");
            System.out.println("não pode ser verificada.");
        }
    }
}
```

TRATAMENTO DE EXCEÇÕES

Usando finally

O bloco opcional do try...catch pode se tornar obrigatório caso o bloco try não possua nenhum catch definido, sendo possível então as seguintes variações no uso desta instrução.

```
        :
try {
        // bloco de instruções
} catch (exceção) {
        // bloco de tratamento
}
        :
```

```
        :
try {
        // bloco de instruções
} finally (exceção) {
        // bloco de tratamento
}
        :
```

Além destas variações mais simples, também é possível o tratamento de exceções múltiplas em um mesmo try, bem como o uso do finally com estes.

TRATAMENTO DE EXCEÇÕES

```
        :
try {
        // bloco de instruções
} catch (exceção1) {
        // bloco de tratamento
} catch (exceção2) {
        // bloco de tratamento
}
        :
```

```
        :
try {
        // bloco de instruções
} catch (exceção1) {
        // bloco de tratamento
} catch (exceção2) {
        // bloco de tratamento
} finally {
        // bloco de tratamento
}
        :
```

finally é sempre executado, independente de ter ocorrido ou não uma exceção.

A ocorrência de alguma exceção executará sempre sua catch e em seguida o bloco finally, se houver sua definição.

O finally permite a liberação de recursos para continuidade do processamento, após a ocorrência de alguma exceção.

Seu uso ainda permite o tratamento de algumas exceções, sendo a ocorrência de outras tratadas exatamente no finally.

TRATAMENTO DE EXCEÇÕES

Características Importantes

Evite usar exceções que não sejam para manipulação de erros, pois sua utilização interfere diretamente no desempenho do programa, tornando-o bem mais lento.

A execução do bloco try é interrompida na instrução que gerar uma exceção, não sendo suas outras instruções executadas neste bloco, pois esta execução é desviada ao bloco catch que corresponda a tal tipo de exceção.

Sobre a catch será procurada uma classe específica ou uma de suas subclasses:

- O catch específico tem maior precedência que um geral
- Só o primeiro catch identificado será executado num try

O bloco finally pode também não ser executado totalmente, se dentro dele ocorrer uma exceção, que deverá ser tratada como uma outra exceção qualquer.