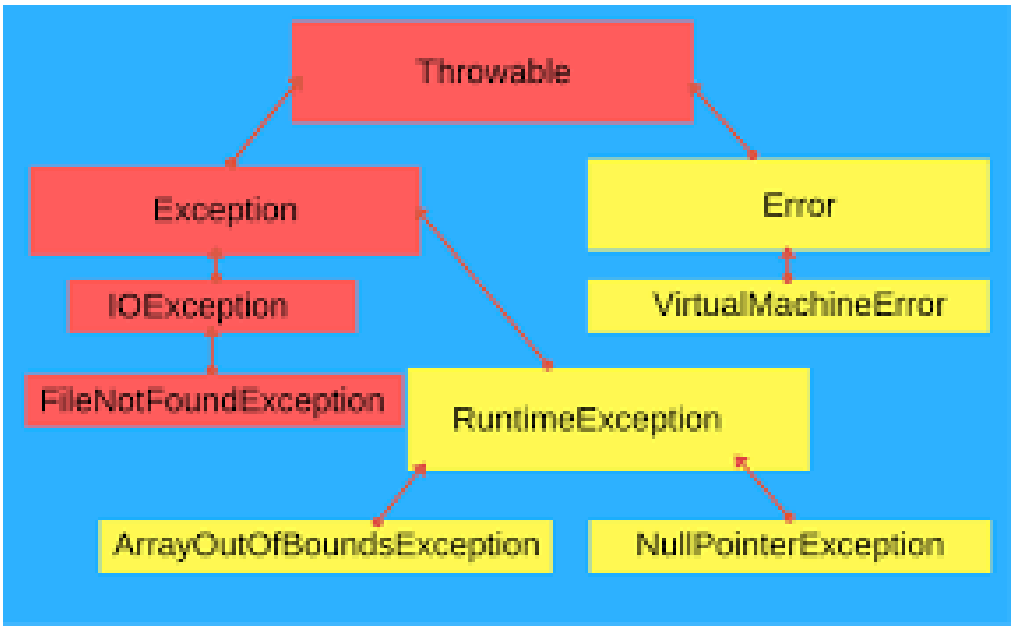
Dia 20 - Tratamento de exceções



CONTEUDO

Introdução

Você pode imaginar tentar escrever código usando uma linguagem que não forneceu uma maneira de

executar declarações condicionalmente? O controle de fluxo é uma parte fundamental de quase todas as

linguagem de programação, e Java oferece várias maneiras de fazer isso. Alguns, como declarações if

e loops for, são comuns à maioria dos idiomas. Mas Java também oferece alguns controles de fluxo

recursos que você pode não ter usado antes - exceções e afirmações.

A instrução if e a instrução switch são tipos de condicional / decisão

controles que permitem que seu programa se comporte de maneira diferente em uma "bifurcação",

dependendo do resultado de um teste lógico. Java também oferece três diferentes

constrói - for, while e do - para que você possa executar o mesmo código sobre e

de novo, dependendo de alguma condição ser verdadeira. As exceções dão a você um ambiente limpo,

maneira simples de organizar o código que lida com problemas que podem surgir durante a execução.

Dia 20 - Tratamento de exceções

Finalmente, o mecanismo de asserção, adicionado à linguagem com a versão 1.4, oferece

uma maneira de fazer testes e verificações de depuração nas condições que você espera que sejam eliminadas

durante o desenvolvimento, quando você não precisa ou deseja necessariamente a sobrecarga do tempo de execução

associado ao tratamento de exceções.

Com essas ferramentas, você pode construir um programa robusto que pode lidar com qualquer

situação com graça. Espere ver uma grande variedade de questões no exame que

incluir controle de fluxo como parte do código da pergunta, mesmo em perguntas que não são

testando seu conhecimento de controle de fluxo.

Vamos ver tudo isso no código:

```
1.  public class Excecao {

2.  public static void main(String[] args) {

3.  try{
4.  int[] vetor = new int[4];

5.  System.out.println("Antes da exception");

6.  vetor[4] = 1;

7.  System.out.println("Esse texto não será impresso");
8.  } catch(ArrayIndexOutOfBoundsException exception){
9.  System.out.println("Exceção ao acessar um índide do vetor que não
   existe");
10. }
11. System.out.println("Esse texto será impresso após a exception");
12. }
13. }
```

Neste código vamos declara um array e chamar uma Index que não existe por exemplo chamar um vetor de 4

Int[] vetor = new int[4];

E chamar uma variável vetor[4] = 1; a Index 4 não existe porque o vetor se conta com [0,1,2,3] e declaro a Index 4.

Retornando um dos erros mais comum no Java o `ArrayIndexOutOfBoundsException, para evitar esse erro e caso tu queira continuar o código mesmo com esse erro vamos trabalhar com exceções que o próprio nome fala, ele abre uma exceção numa determinada linha e continua o código como se "nada tivesse acontecido".`

Dia 20 - Tratamento de exceções

Vídeo

https://www.youtube.com/watch?v=ld2C4GcAtsg&ab_channel=LoianeGroner

Exercício

Dado o seguinte código: Responda o exercicio 1, 2 e 3

```
1.  public class Excecao {

2.  public static void main(String[] args) {

3.  try{
4.  int[] vetor = new int[4];

5.  System.out.println("Antes da exception");

6.  vetor[4] = 1;

7.  System.out.println("Esse texto não será impresso");
8.  } catch(ArrayIndexOutOfBoundsException exception){
9.  System.out.println("Exceção ao acessar um índide do vetor que não
    existe");
10. }
11. System.out.println("Esse texto será impresso após a exception");
12. }
13. }
```

1) O que irá retornar no código:

a)

Antes do exception

```
Esse texto será impresso após a exception
```

b)

```
Exceção ao acessar um índide do vetor que não existe
```

c)

Antes do exception

```
Exceção ao acessar um índide do vetor que não existe

Esse texto será impresso após a exception
```

d)Erro de compilação

Dia 20 - Tratamento de exceções

2) Para que serve os tratamentos de erro do Java:

   a) Serve para abrir uma exceção no código lendo o código no Try{} e no Catch retornando uma ação

   b) Serve para abrir uma exceção no código lendo o código no Catch{} e no Try retornando uma ação

   c) Serve para retornar um erro no código lendo o código no Try{} e no Catch retornando uma ação

   d) Serve para abrir uma exceção através do Try{} Final{};

3) Se ocorrer alguma exceção no Try o que ira acontecer?

   a) Ocorre que abrir uma exceção através do Try{} Final{};

   b) Recebe uma exceção e retorna o erro no compilador

   c) Ira ignorar a exceção somente com o Try

   d) Retorna uma ação no catch de acordo com o erro

Prova

---

1) Dado:

```
class Plane {
  String estática s = "-";
  public static void main (String [] args) {
  novo Plano (). s1 ();
```

Respostas do autoteste 411

```
  System.out.println (s);
  }
  void s1 () {
  tente {s2 (); }
  catch (exceção e) {s + = "c"; }
  }
  void s2 () lança Exception {
  s3 (); s + = "2";
  s3 (); s + = "2b";
```

}

void s3 () lança Exception {

lance new Exception ();

}}

Qual é o resultado?

A. -

B. -c

C. -c2

D. -2c

E. -c22b

F. -2c2b

G. -2c2bc

H. A compilação falha

2) Dado:

tente {int x = Integer.parseInt ("dois"); }

Qual poderia ser usado para criar um bloco catch apropriado? (Escolha todas as opções aplicáveis.)

A. ClassCastException

B. IllegalStateException

C. NumberFormatException

E. ExceptionInInitializerError

3)Dado:

class Emu {

String estática s = "-";

public static void main (String [] args) {

tentar {

lance new Exception ();

} catch (exceção e) {

Dia 20 - Tratamento de exceções

```
  tentar {
  tente {lançar uma nova exceção ();
  } catch (Exceção ex) {s + = "ic"; }
  lance new Exception (); }
  catch (exceção x) {s + = "mc"; }
  finalmente {s + = "mf"; }
  } finalmente {s + = "de"; }
  System.out.println (s);
}}
```

Qual é o resultado?

A. -ic de

B. -mf de

C. -mc mf

D. -ic mf de

E. -ic mc mf de