EAD

EDUCAÇÃO A DISTÂNCIA

UNIDADE IV

Linguagem de Programação Orientada a Objetos

Prof. Me. Ricardo Veras

Exceções

- Uma exceção é um objeto gerado em execução para indicar a ocorrência de algum tipo de condição excepcional durante a execução de um método (comum ou construtor).

- Uma vez detectada essa condição anormal, é necessário ativar mecanismos para corrigi-la e permitir o prosseguimento da execução.
- Esse tipo de procedimento é chamado de tratamento (ou manipulação) de exceções, o que, no código, faz-se por meio da utilização de uma estrutura própria, um bloco de Tratamento de Exceções (*Exception Handler*).

    - O programador deverá prever os pontos do programa suscetíveis a algum tipo de exceção e definir, antecipadamente, blocos de tratamento de exceções para cada um deles.

Exceções – captura e tratamento

- O tratamento de exceções consiste em duas etapas:
  - Ocorrência do problema.
  - Captura do tipo da exceção.

A hierarquia das exceções

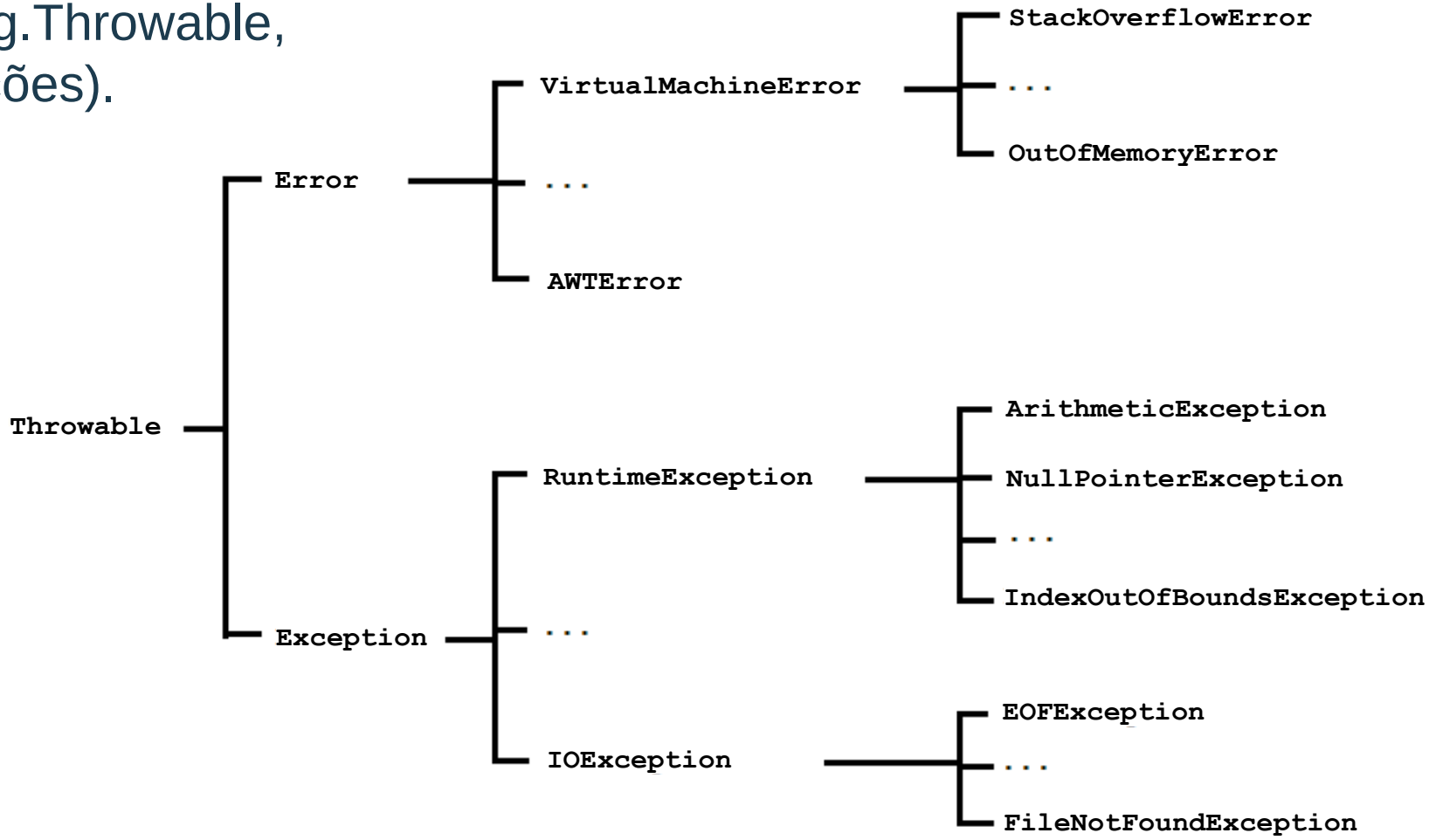
- Sabemos que, no Java, a base de construção de sistemas é a criação de classes com seus atributos e métodos. Então, as exceções também serão objetos (em memória) que possuem atributos, métodos e construtores.

  - Assim, as exceções se utilizam dos conceitos de orientação a objetos e compartilham de suas possibilidades e facilidades, como herança, polimorfismo e encapsulamento.

Exceções – captura e tratamento

A Árvore Hierárquica das Exceções:

- (Estrutura da classe java.lang.Throwable, raiz (mãe) de todas as exceções).



Fonte: autoria própria.

Exceções – captura e tratamento

- Algumas exceções obrigam seu tratamento sempre, durante a codificação, enquanto outras não obrigam necessariamente seu tratamento.
- Todas as exceções derivadas de java.lang.Exception – exceto as derivadas de java.lang.RuntimeException – precisam, obrigatoriamente, ser tratadas, já que são condições esperadas que podem acontecer ao longo da execução de um método.

Exemplo:

- Conexão com um banco de dados, utiliza-se a exceção SQLException.

Exceções – captura e tratamento

- Quando ocorre uma exceção, o sistema (Máquina Virtual do Java) está preparado para verificar se há um tratamento previsto (codificado no programa) para aquela exceção e o executa.
- No entanto, se não foi previsto esse tratamento e o programador não tratou aquela exceção específica, o sistema gerará uma falha e bloqueará a execução dos códigos subsequentes, provocando uma parada em todas as eventuais execuções da máquina virtual.
- Sendo assim, a importância de se capturar uma exceção e tratá-la é impedir que haja uma parada nas execuções da máquina virtual, impedindo com isso que haja um travamento no servidor de aplicações do Java.

Exemplos de servidores de aplicação Java:

- Apache Tomcat (servidor Http e de aplicações).
- IBM WebSphere AS.
- Red Hat Jboss Middleware (todo escrito em Java).
- GlassFish Server (da Oracle).
- ...entre outros.

Exceções – captura e tratamento

A estrutura geral de captura e tratamento de exceções (*try-catch-finally*) apresenta a seguinte sintaxe:

```
try {
   // códigos que podem gerar uma exceção (tenta executar)
} catch (Excecao01 e) {
   // bloco de tratamento realizado a partir da
   // identificação de uma exceção do tipo Excecao01
} catch (Excecao02 e) {
   // bloco de tratamento realizado a partir da
   // identificação de uma exceção do tipo Excecao02
...
} catch (ExcecaoN e) {
   // bloco de tratamento para exceção mais genérica
} finally {
   // bloco opcional mas que sempre será executado
}
```

Exceções – captura e tratamento

Sendo assim, uma estrutura de captura e tratamento de exceções (try-catch-finally) deve ter:

- apenas 1 (um) bloco try (no início da estrutura);
- 1 (um) ou mais blocos catch;

...e opcionalmente pode ter:

- apenas 1 (um) bloco finally (ao final da estrutura).

Observações:

- O bloco finally, quando existente, será sempre executado, tendo ocorrido uma exceção ou não.
    - De um modo geral, esse bloco (finally) inclui comandos de liberação de recursos que possam ter sido alocados no processamento do bloco try e que precisam ser liberados, tendo sua execução sido concluída com sucesso ou não.

Interatividade

O que significa “tratar uma exceção”?

a) É criar no programa uma estrutura “try-catch” envolvendo uma ou mais linhas de código, de modo que, à medida que uma exceção ocorra, o sistema tenha um comportamento específico.

b) É definir elementos de códigos que impeçam que as exceções ocorram a fim de não paralisar a execução da JVM.

c) É corrigir o erro na codificação após a verificação da sua ocorrência.

d) É retirar do código a linha de comando que causou a exceção.

e) É criar uma estrutura condicional que antecipa a ocorrência da exceção e desvia o programa para funções que corrigem a situação.

Resposta

O que significa “tratar uma exceção”?

a) É criar no programa uma estrutura “try-catch” envolvendo uma ou mais linhas de código, de modo que, à medida que uma exceção ocorra, o sistema tenha um comportamento específico.

b) É definir elementos de códigos que impeçam que as exceções ocorram a fim de não paralisar a execução da JVM.

c) É corrigir o erro na codificação após a verificação da sua ocorrência.

d) É retirar do código a linha de comando que causou a exceção.

e) É criar uma estrutura condicional que antecipa a ocorrência da exceção e desvia o programa para funções que corrigem a situação.

Exceções – captura e tratamento

Exemplo:

A linha de comando abaixo gerará uma exceção de cálculo:

```
int a = 3, b = 0;
int c = a / b;
```

A execução da segunda linha gerará a seguinte exceção:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
                    at TesteA.main(TesteA.java:9)
```

Assim, nesse caso, para se tratar essa exceção, basta envolver a linha que gerou a exceção por um bloco try-catch e colocar o nome da exceção como parâmetro de captura:

```
int a = 3, b = 0;
try {
   int c = a / b;
} catch (ArithmeticException e) {
   // lógica de tratamento
}
```

Exceções – captura e tratamento

Genericamente, uma exceção, quando ocorre, é gerada a seguinte mensagem na Console:

```
Exception in thread "main" java.lang.NomeDaExcecao: decricao geral
                            at Class.method01(Class.java:line)
                            at Class.method02(Class.java:line)
                            ...
                            at Class.main(Class.java:line)
```

- Sendo assim, é possível rastrear uma exceção, já que na mensagem aparece o nome do método e de sua classe, além do número da linha de comando que gerou aquela exceção.

Exceções – captura e tratamento

Para uma linha de comando que pode gerar uma exceção ao ser executada, não adianta envolvermos com o bloco try-catch, se capturarmos uma exceção que não for aquela resultante da execução daquele comando:

```
int[] x = new int[5];
try {
     int w = x[9];
} catch (NumberFormatException e) {
     System.out.println("Erro na execução");
}
```

- A execução do comando do bloco “try” (acima) resultará numa outra exceção que não é a “`NumberFormatException`”, mas sim a “`ArrayIndexOutOfBoundsException`”. Nesse caso, a máquina virtual sofrerá uma parada.

Exceções – captura e tratamento

É comum deixarmos uma captura de uma exceção “genérica”, como as do tipo “Exception” ou “Throwable”, garantindo que não importando o erro ocorrido no bloco try, caso esse ocorra, a máquina virtual desviará a execução para o bloco catch, impedindo o travamento de sua execução:

```
int[] x = new int[5];
try {
   int w = x[9];
} catch (Exception e) {
   System.out.println("Erro na execução");
}
```

- A execução do comando do bloco “try” (acima) resultará na impressão do texto “`Erro na execução`” na tela da console, de modo a não travar a continuidade das execuções subsequentes.

Exceções – captura e tratamento

Múltiplas exceções:

- Às vezes, num programa, temos a situação em que mais de uma exceção pode ocorrer, sendo que, para cada exceção, o sistema deve “reagir” de forma diferente. Nesse caso, sabemos que a estrutura de tratamento de exceções permite o tratamento de mais de uma exceção. Desta forma, a ordem de tratamento vai da exceção mais específica para a mais genérica, de forma que a última captura geralmente é sobre “Exception” ou sobre “Throwable”.

Exceções – captura e tratamento

Múltiplas exceções (exemplo) – quando formos acessar um banco de dados:

```
public void acessarBD() {
    try{
        //realizando o acesso a um Banco de Dados
    }catch(ClassNotFoundException ex){
        System.out.println(":: ERRO :: Driver JDBC não
        encontrado na aplicação!");
    }catch(SQLException ex){
        System.out.println(":: ERRO :: Problemas na conexão
        com a fonte de dados");
    }catch(Exception ex){
        System.out.println(":: ERRO :: Ocorreram outros
        problemas de acesso ao Banco de Dados...");
    }
    //... demais códigos
}
```

Interatividade

Numa estrutura de tratamento de exceções com múltiplas capturas, por que devemos deixar a captura de uma exceção mais genérica como sendo a última delas?

a) Para garantir a execução de mais de um tratamento sempre que ocorrer uma exceção.

b) Na verdade, a estrutura de tratamento permite apenas a captura de uma única exceção.

c) Uma exceção mais genérica supõe a ideia de superclasse, que, de acordo com o polimorfismo, é uma classe que pode se comportar como uma de suas filhas, se ela não for a última, impede a execução dos demais blocos de tratamento.

d) Para que, ao paralisar a execução da JVM, saibamos qual foi o evento responsável pela paralisação, já que seu nome é impresso na tela da console.

e) Na verdade, numa estrutura com múltiplas capturas de exceções, a captura da exceção mais genérica deve ser a primeira.

Resposta

Numa estrutura de tratamento de exceções com múltiplas capturas, por que devemos deixar a captura de uma exceção mais genérica como sendo a última delas?

a) Para garantir a execução de mais de um tratamento sempre que ocorrer uma exceção.

b) Na verdade, a estrutura de tratamento permite apenas a captura de uma única exceção.

c) Uma exceção mais genérica supõe a ideia de superclasse, que, de acordo com o polimorfismo, é uma classe que pode se comportar como uma de suas filhas, se ela não for a última, impede a execução dos demais blocos de tratamento.

d) Para que, ao paralisar a execução da JVM, saibamos qual foi o evento responsável pela paralisação, já que seu nome é impresso na tela da console.

e) Na verdade, numa estrutura com múltiplas capturas de exceções, a captura da exceção mais genérica deve ser a primeira.

Exceções – lançamento

- Quando queremos que, em determinadas circunstâncias, uma exceção seja lançada, utilizamos a palavra chave *throw*, que gera a exceção na JVM, bloqueando a execução daquele bloco de códigos a partir do lançamento.

```
public void depositar(double valor) {
   if (valor < 0) {
      String msg = "Este valor não pode ser negativo!!";
      throw new IllegalArgumentException(msg);
   } else {
      saldo += valor;
   }
}
```

  - Nesse caso, se a situação não for tratada como exceção, ela trava a JVM, parando todas as execuções em andamento naquela JVM.

```
objeto.depositar(-200);  //não tratado
```

Exceções – lançamento

Continuando o exemplo anterior:

- Mas se a exceção for tratada, não ocorrerá o travamento da JVM, mas sim a continuidade da execução do sistema (de todos os sistemas existentes em execução).

```
try {
   objeto.depositar(-200);  // tratado
} catch (Exception e) {
   //... Lógica de tratamento
}
```

Mas como o desenvolvedor iria saber que a exceção tinha que ser tratada?

Exceções – lançamento

- Para obrigarmos o tratamento da exceção, utilizamos a declaração *throws* na linha de declaração do método, arremessando uma exceção que obriga o tratamento (desde que não pertençam a RuntimeException, como, por exemplo, a Exception).

```
public void sacar(double valor) throws Exception {
    ...
}
```

Assim, a linha de código que chamar esse método "sacar" será obrigada a ser tratada como exceção.

```
public void metodoQualquer(double valor) {
   try {
      objeto.sacar(valor);
   } catch (Exception e) {
      // ...lógica de tratamento
   }
}
```

Exceções – lançamento

Resumo sobre lançamento de exceções:

- *Throw* – palavra-chave utilizada dentro do código de um método, com a finalidade de interromper a execução por meio de arremesso de uma exceção.

```
...
throw new TipoDeException(...);
...
```

- *Throws* – palavra-chave utilizada na declaração de um método e que vai arremessar aquele tipo de exceção.

```
public void metodo(parametro) throws TipoDeException {
   ...
}
```

Exceções – criação

- Além das exceções disponibilizadas na Biblioteca do Java (em java.lang.Exception e java.lang.Error), é possível criar novas exceções de acordo com a necessidade do programador.

- Exceções personalizadas podem ser úteis para associar a determinadas falhas de execução, ou regras de negócio da empresa que exigem determinadas situações que definem como ela deve ser utilizada.
    - Sendo assim, criamos exceções para expressar erros específicos relativos às regras de negócios da empresa. Nesse caso deve-se criar uma classe que representa essa regra (com a palavra Exception ao final – por convenção) e fazemos com que ela herde a Classe Exception. Essa nova classe terá métodos construtores que devem chamar a classe superior com a mensagem a ser mostrada (específica da regra de negócio).

Exceções – criação

Exemplo de uma classe de exceção que pode ser criada

```
public class SaldoInsuficienteException extends Exception {
   public SaldoInsuficienteException() {
      super("Falta de Saldo para esta Operação!!");
   }
   public SaldoInsuficienteException(String msg) {
      super(msg);
   }
}
```

Exceções – criação

A implementação para a exceção criada (vista no *slide* anterior) poderia ser:

```
public class Conta {
    private double saldo;
    public void sacar (double valor) throws SaldoInsuficienteException {
        if (saldo < valor) {
            throw new SaldoInsuficienteException (“Este valor não pode ser sacado”);
                                } else {
                                    // ...sacando o valor
                                    saldo = saldo – valor;
                                }
                            }
```

Exceções – criação

Desta forma, sempre que o método “sacar” da classe “Conta” for utilizado, ele deve estar envolto por uma estrutura “try – catch”:

```
public class Qualquer {
    public void metodoQualquer () {
        Conta c1 = new Conta();
        c1.sacar(500); //ERRO DE COMPILAÇÂO
        try {
            c1.sacar(500);
        } catch (SaldoInsuficienteException e) {
            e.printStackTrace();
        }
    }
}
```

Exceções – criação

Desta forma, sempre que o método “sacar” da classe “Conta” for utilizado, ele deve estar envolto por uma estrutura “try – catch”:

```
public class Qualquer {
    public void metodoQualquer () {
        Conta c1 = new Conta();
        //c1.sacar(500); //ERRO DE COMPILAÇÂO
        try {
            c1.sacar(500);
        } catch (SaldoInsuficienteException e) {
            e.printStackTrace();
        }
    }
}
```

Interatividade

Qual das opções abaixo representa a palavra reservada que lança uma exceção, de modo que em sua ocorrência gere uma interrupção na execução de um conjunto de linhas de código?

a) Try.
b) Throw.
c) Catch.
d) Throws.
e) Exception.

Resposta

Qual das opções abaixo representa a palavra reservada que lança uma exceção, de modo que em sua ocorrência gere uma interrupção na execução de um conjunto de linhas de código?

a) Try.
b) Throw.
c) Catch.
d) Throws.
e) Exception.

Threads

- Imagine um programa que gera um relatório muito grande em PDF. Como esse é um processo demorado, para dar alguma satisfação para o usuário, é interessante mostrar a ele uma barra de progresso. Isso caracteriza um processamento paralelo.
- Quando usamos o computador, fazemos várias coisas ao mesmo tempo, ou paralelamente (p. ex.: navegar na internet e ouvir música).
- Para vários programas distintos, normalmente o próprio sistema operacional gerencia isso rodando vários processos em paralelo.
- Quando se tem um programa só (um processo só) e vários processamentos, normalmente utilizamos Threads.
    - Assim, Threads permitem múltiplas atividades dentro de um único processo.

Threads

Existem várias razões para se utilizar threads:

- Maior desempenho em ambientes multiprocessados;
- Responsividade em interfaces gráficas;
- Simplificação na modelagem de algumas aplicações.

- Um bom exemplo de utilização de Threads são os jogos que estamos acostumados a jogar, em que outros personagens atuam em conjunto com nossos personagens, como o “pac-man”, em que os “fantasminhas” perseguem o personagem principal controlado pelo jogador.

Threads

Criando Threads:

Há duas formas de criarmos uma Thread em Java:

- Usando herança: criamos uma classe que estende a classe Thread;
- Usando implementação de interface: criamos uma classe que implementa a interface Runnable.

- Por boas práticas, geralmente implementamos a interface Runnable ao invés de estender a classe Thread.

Threads

Usando herança

- Usando herança, a classe criada deve herdar a Classe Thread e, para que funcione, deve sobrescrever o método “public void run()”.

Exemplo:

```
public class Exemplo1 extends Thread {
   public void run() {
      // executa alguma lógica, por exemplo:
      for(int i=0; i<500; i++)
      System.out.println("Usando Herança");
   }
}
```

Threads

Usando a interface Runnable

- Implementando a interface Runnable (que é a forma mais indicada), somos obrigados a implementar o método: “public void run()”.

Exemplo:

```
public class Exemplo2 implements Runnable {
   public void run() {
      // executa alguma lógica, por exemplo:
      for(int i=0; i<500; i++)
      System.out.println("Usando Runnable");
   }
}
```

Threads

- Para acionarmos uma Thread, instanciamos a classe do tipo Thread e acionamos o seu método “start”.
- Assim, quando damos o start, estamos iniciando o processamento paralelo (o Thread) e liberando o programa para executar qualquer outra thread.
- Desta forma, ao darmos um start em uma Thread, estaremos rodando o método run criado na classe que está sendo rodada pela Thread.

Exemplo:

```
Exemplo1 ex1 = new Exemplo1(); // que herda Thread
ex1.start();

Exemplo2 ex2 = new Exemplo2(); // que implementa Runnable
Thread t2 = new Thread(ex2);
// ou Thread t2 = new Thread(ex2, "Nome da Thread");
t2.start();
```

Threads

Observação:

- O programador não tem nenhum controle sobre o escalonador do processador.
- Isso significa que, sendo os processos executados paralelamente, não há como saber qual a ordem exata de execução desses processos.

Interatividade

Qual a forma mais utilizada na geração de classes que serão controladas por Threads?

a) Gerar uma classe que herda a classe Exception.
b) Gerar uma classe que herda a classe Throwable.
c) Criar uma classe com o método start(...).
d) Gerar uma classe que implementa a interface Throwable.
e) Gerar uma classe que implementa a interface Runnable.

Resposta

Qual a forma mais utilizada na geração de classes que serão controladas por Threads?

a) Gerar uma classe que herda a classe Exception.
b) Gerar uma classe que herda a classe Throwable.
c) Criar uma classe com o método start(...).
d) Gerar uma classe que implementa a interface Throwable.
e) Gerar uma classe que implementa a interface Runnable.

ATÉ A PRÓXIMA!