Programação Orientada a Objetos em Java

Tratamento de Exceções

Prof. Dr. Francisco Isidro Massetto
isidro@professorisidro.com.br

Exeções

- Exceção
  - Evento que ocorre durante a execução de uma instrução (declaração, atribuição, método) na aplicação do usuário
- Possíveis de detecção e tratamento em tempo de execução
- Exemplos
  - Um valor não pode ser convertido de um tipo para outro
  - Uma leitura de teclado pode gerar exceções de tratamento
  - Uma operação aritmética pode ser inválida

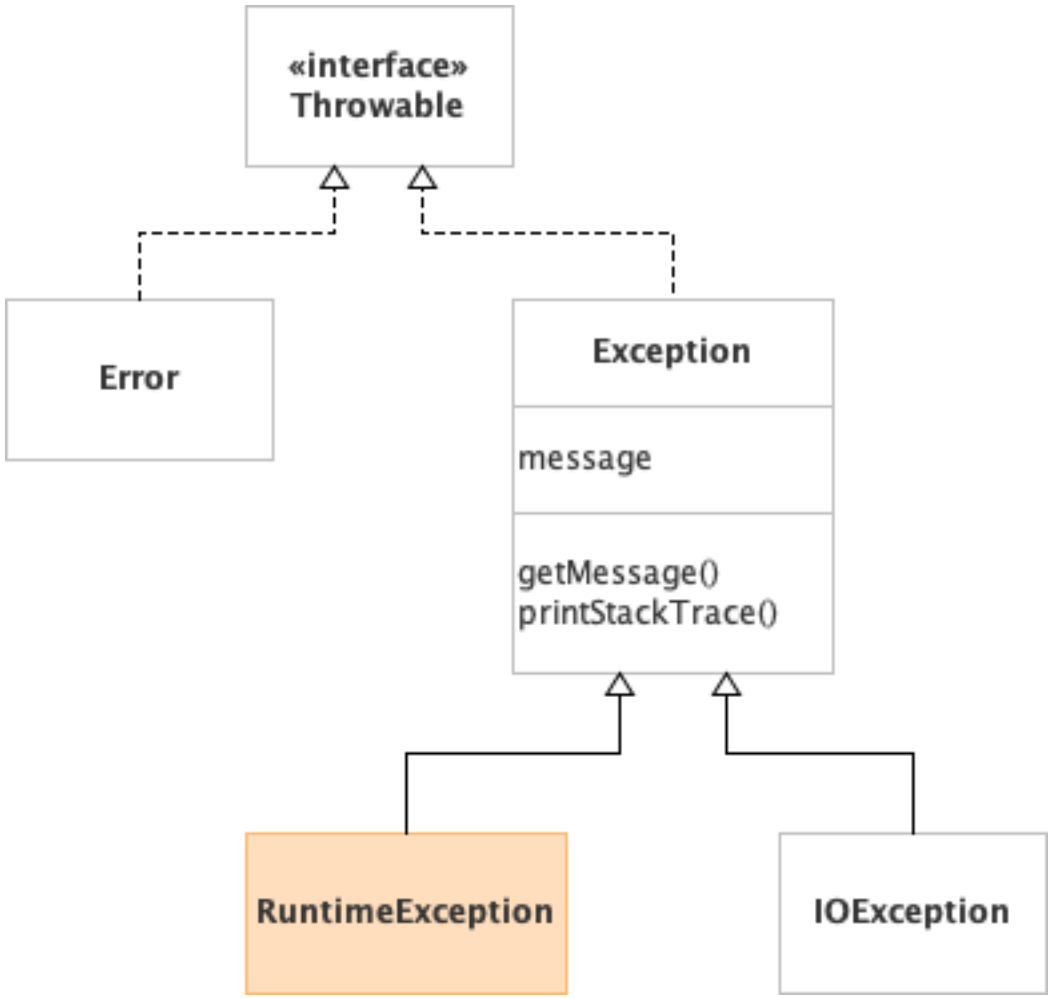
Erros

- Erro
  - Evento que ocorre na execução da máquina Virtua Java ou algum evento externo sério
  - Representam condições anormais de execução
- Exemplo
  - Não há mais memória disponível (A VM tentou utilizar o GC mas não obteve êxito)
  - Erro de execução da Virtual Machine
  - Erro de I/O – um arquivo foi corrompido durante sua leitura ou gravação devido a uma falha física em um HD

Mas por que tratar exceções?

- Mantém a mesma linha de execução sem precisar tratar casos de erros
- Forma mais elegante de manipulação de situações anormais
- Permite inclusive prever tipos de erros que não são detectáveis facilmente (ex.: entrada de dados com tipos diferentes)
- Engloba um mecanismo de transferência de execução não explícito

Hierarquia das Exceptions



Tipos de Exceções

- Não Verificadas
  - Exceções de tempo de execução.
  - Significa que os métodos podem ser declarados sem a cláusula de lançamento de exceção.
  - Entretanto o lançamento pode ocorrer no corpo do método
  - Exceções derivadas a partir da RuntimeException
- Verificadas
  - Quando obrigatoriamente o método deve ser declarado como lançador de uma exceção e sua manipulação deve ser feita dentro de um bloco de tratamento try/catch
  - Demais exceções

Tratando Exceções

- Basicamente deve-se conhecer se o objeto ou método utilizado lança exceções
- Deve-se, então, definir um bloco para a linha principal de execução
- E, no mínimo, um bloco de tratamento da exceção lançada
- Dependendo do nível de especialização da exceção lançada
- Opcionalmente pode-se definir um bloco final de execução em todos os casos (com execução normal ou com captura de exceções)

Bloco Try

- Indica que o bloco dentro dele será executado monitorando um eventual lançamento de exceções
- Obrigatoriamente deve haver um bloco de tratamento

```
try{
 // bloco de comandos
}
```

Bloco(s) Catch

- Contém o trecho de código responsável por tratar a exceção lançada no bloco try (pode haver vários blocos catch)

```
try{
 // bloco de comandos
}
catch(ClasseExcecao obj){
 // codigo de tratamento
}
```

Bloco(s) catch

- Podemos ter vários blocos catch?
  - Sim
- Qual o propósito?
  - Um trecho de código pode lançar diversos tipos de exceções e podemos tratar cada exceção de forma individual
- Cuidado!
  - Eles precisam ser declarados sempre na sequencia da Exceção mais específica para a mais genérica
  - Caso contrário seu código não irá compilar, pois a exceção mais específica nunca será tratada (a mais genérica já foi capturada)

Bloco Finally

- Bloco para execução final em casos onde há ou não exceções
- O bloco finally, se declarado, sempre é executado

```
try{...}
catch(Exception e){...}
finally{
}
```

Quando usar Finally?

- Geralmente operações que envolvam alocação de recursos (arquivos, sockets de rede) e esses recursos necessitam ser liberados para a VM ou outros usuários que necessitem usar
- Exemplo
  - Bloco para enviar uma mensagem pela rede via socket
  - Pode haver uma exceção durante o envio, por uma falha qualquer (perda, inconsistência, etc)
  - Mesmo com sucesso ou falha, o socket tem que ser finalizado para não deixar a aplicação com conexões em aberto

Exemplo com os 3 blocos

```
try{
   linha principal de execução
}
catch(Exception e){
   bloco de tratamento
}
finally{
   bloco final
}
```

Algumas Considerações

- O bloco **catch** deve vir na linha imediatamente posterior ao final do bloco **try**
- O bloco **finally** deve vir imediatamente após o bloco ou sequência de blocos **catch**
- Pode-se omitir um bloco do tipo **catch**, desde que a exceção seja do tipo **Não Verificada**

Alguns Métodos da classe Exception

- getMessage()
  - Mostra a mensagem associada à exceção – a mensagem que é passada ao construtor
- printStackTrace()
  - Exibe toda a pilha de erros
  - Métodos que chamam outros métodos de outras classes que também lançam ou tratam exceções

Lançando Exceções

- Cláusula **throws** e **throw**
  - Neste caso, o método deve ter em seu cabeçalho, a palavra reservada **throws** indicando que ele lança uma exceção
  - Serve inclusive para construtores
  - O corpo do método deve ter a instrução de lançamento da exceção
  - O operador **throw** é quem efetivamente faz o lançamento da Exceção

Exemplo

```
public double div(double n, double d) throws Exception {
    if (d == 0.0){
        throw new Exception("Divisão por zero!");
    } else{
        return (n/d);
    }
}
```

Mais considerações

- O complemento do cabeçalho do método pode mudar no lançamento das exceções
  - Quando lançamos exceções Verificadas, a cláusula **throws** é **obrigatória** no cabeçalho do método
  - Quando lançamos exceções Não Verificadas, a cláusula é **opcional**

Criando suas próprias Exceptions

- Toda exceção do usuário pode herdar características de qualquer classe a partir da classe Exception
- Caso queira gerar uma exceção Verificada • Herdar de qualquer classe Exception
- Caso queira gerar uma exceção Não-Verificada • Herdar de alguma subclasse a partir de RuntimeException
- Basicamente você declara o tipo da sua Exception e cria um construtor invocando o construtor da classe base

Exemplo

```
public class MyException extends RuntimeException{
    public MyException(String message){
        super(message);
    }
}
```

Qual a vantagem?

- Mesmo sendo uma classe que quase não tem funcionalidades, você tem agora um tipo específico para representar uma situação ou uma anormalidade da sua regra de negócios.
- Você amplia o conceito de classificação de erros, customizando suas próprias tratativas

try-with-resources

- Uma maneira elegante de instanciar objetos que fazem uso de métodos que lançam exceções e que, principalmente, implementam a interface Closeable
- Por que isso? Porque podemos já no cabeçalho da Cláusula try, instanciar o objeto que queremos e não precisamos explicitar o método close() ao final do seu uso.

Exemplo

```
try(FileReader fr = new FileReader("texto.txt")){
    String linha=null;
    BufferedReader br = new BufferedReader(fr);
    while ((linha = br.readLine()) != null){
        System.out.println(linha);
    }
}
catch(IOException ex){
    ex.printStackTrace();
}
```

Multicatch

- Possibilidade de simplificar a sintaxe quando diferentes exceções capturadas podem ter o mesmo tratamento
- Ao invés de múltiplos blocos **catch** repetindo código, podemos utilizar o operador **or-bitwise** para realizar esta declaração

Exemplo

```
Scanner scn = new Scanner(System.in);
try {
    int n = Integer.parseInt(scn.nextLine());
    if (99 % n == 0)
        System.out.println(n + " é divisor de 99");
}
catch (NumberFormatException | ArithmeticException ex){
    System.out.println("Exceção encontrada: " + ex);
}
```