CES/JF Centro de Ensino Superior de Juiz de Fora

MANTENEDORA

SVD - ESDEVA

Laboratório de Programação de Persistência de Objetos

Erros, Falhas e Exceções

Christien Lana Rachid

2019/1

Erros, Falhas e Exceções

Erro? Não. É uma *feature* do meu programa...

- Bons programas devem ser robustos.
- Programas robustos previnem erros e se recuperam de eventuais falhas do sistema e de seus usuários.

Erros e Exceções

São condições anormais que nem sempre são previstas pelo programador e que podem afetar o fluxo lógico da aplicação.

Conforme a natureza da exceção ou de como ela é tratada, a exceção pode até finalizar prematuramente a aplicação.

Erros são situações normalmente irrecuperáveis (ex: Falta de memória).

Exceções são normalmente recuperáveis (ex: Acesso a uma posição inválida de um vetor).

Exemplo

```
public class Application {
  public static void main(String args[]) {
    int a = 10;
    int b = 0;
    int resultado = a / b;
    System.out.println(resultado);
  }
}
```

Resultado

```
Exception in thread "main" java.lang.ArithmeticException: / by zero at Application.main(Application.java:5
```

Ou seja, ocorreu um erro de divisão indevida por zero, que não é definida para números Reais.

Exceções

Java oferece duas formas de tratamento: Uma voltada ao tratamento, e outra voltada a proteção de recursos.

try / catch – Tratamento de Exceções

try / finally – Proteção de Recursos

Ambas as instruções podem ser usadas de forma conjunta, ou seja, aninhadas.

Tratamento de Exceções

```
try {
    // Código potencialmente perigoso
} catch (Uma exceção qualquer) {
    // Tratamento: Código alternativo
}
```

Exemplo anterior modificado

```
public class Application {
  public static void main(String args[]) {
    int a = 10;
    int b = 0;
    int resultado = 0;
    try {
          resultado = a / b;
    } catch (ArithmeticException ae) {
          // Código de tratamento ...
    }
    System.out.println(resultado);
  }
}
```

Proteção de Recursos

```
try {
    // Código qualquer ...
} finally {
    // Código que deve ser executado
   // sob qualquer circunstância
}
```

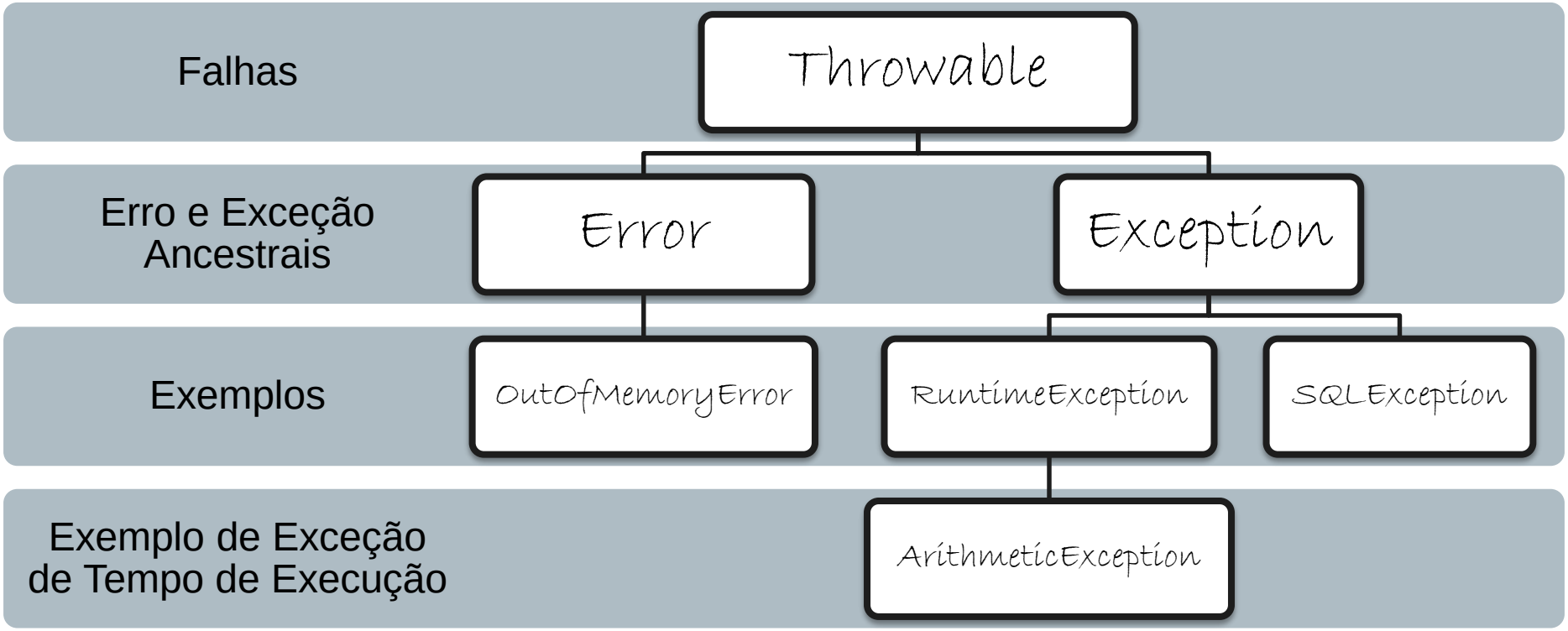
Exemplo

```
public class Application {
  public static void main(String args[]) {
    int a = 10;
    int b = 0;
    try {
        System.out.println(a / b);
    } finally {
        // Aqui vai ser sempre executado.
    }
  }
}
```

Exceções Comuns

ArithmeticException	• Divisão por zero.
NullPointerException	• Referência inválida a um objeto nulo.
IOException	• Dispositivo não preparado/erros de leitura.
IndexArrayOutOf BoundsException	• Acesso a posiçÃo inválida no array
ClassCastException	• Conversão para um tipo (classes) inválida.

Hierararquia de Erros e Exceções



Checked Exception

Serve para marcar um método a ser tratado explicitamente pelo programador.

Isso garante que o chamador do método não se “esqueça” de tratar a exceção, garantindo a qualidade da aplicação.

Diversos bugs de software são causados por exceções não tratadas pelo programa.

Ajuda o programador a não ter que tratar todas as exceções dentro do método.

Checked Exception: Exemplo

```
public void someMethod(int x) throws java.lang.Exception {
    System.out.println(x);
}
```

- Caso seja invocado o método ‘someMethod’ o chamador deverá colocar a chamada dentro de um bloco ‘try / catch’.
- Exceções usuárias derivadas de Exception são definidas como exceções checadas.

Levantamento de exceções

Técnica utilizada para enviar a exceção para o método chamador a fim de ser tratado.

Serve também para interromper o fluxo normal da aplicação baseado no conceito de robustez (programação por contrato).

Propagação de Exceções

```
public void someMethod(int x) throws RuntimeException {
  if (x < 10) {
    throw new RuntimeException(
    "Valor menor que 10!");
  }
}
```

Criação de novas Exceções

Java possibilita que sejam criadas novas exceções.

Basta criar uma sub-classe de RuntimeException ou Exception (situações mais excepcionais).

Exceções personalizadas tornam a aplicação mais legíveis e OO (modelos de domínio).

Criação de novas Exceções

```
public class SaldoInsuficienteException
        extends RuntimeException {

}

public class ContaCorrente  {

   public void debitar(double valor) {

      if (valor > saldoConta) {

         throw new SaldoInsuficienteException();

      }

   }
}
```

Conclusões

- Java possui suporte a programação por contrato/robustez através de exceções.
- Try/Catch trata falhas/exceções.
- Try/Finally é usado para recuperar um recurso à uma condição anterior.
- Exceções podem ser marcadas como checadas (checked).
- Novas exceções podem ser criadas para aumentar a legibilidade de uma aplicação.

Enumerações

Uma enumeração é um tipo definido pelo usuário para modelar domínios fechados.

Meses do ano, dias da semana, planetas do sistema solar são exemplos de domínios fechados.

Enumerações: Anti-padrão

A solução padrão para representar uma enumeração é:

- `public static final int INVERNO = 0;`
- `public static final int PRIMAVERA = 1;`
- `public static final int VERAO = 2;`
- `public static final int OUTONO = 3;`

Enumerações: Anti-padrão

Não possui tipagem segura.

Como a variável estação é simplesmente um valor inteiro *(int), podemos ter qualquer valor usado para representar.*

- `setEstação(5); //estação não existente`
- `setEstação(INVERNO + VERAO);`
- `// representaria o clima em Minas Gerais?`

Não possui um namespace.

- Pode-se misturar com outras constantes existentes na classe.

Pode haver colisões de nomes com constantes herdadas.

Requer recompilação do código se novas constantes forem adicionadas.

Enumerações com *enums*

Typesafe enums:

- Antes

```
public static final int INVERNO  = 0;
public static final int PRIMAVERA = 1;
public static final int VERAO     = 2;
public static final int OUTONO    = 3;
    ...
void setEstacao(int estacao) { ... }
```

- Depois

```
enum Estacao {INVERNO, PRIMAVERA, VERAO, OUTONO};
...
void setEstacao(Estacao estacao) { ... }
```

Conclusões

- Tipos enumerados permitem criar domínios fechados.
- *Enum* é a primitiva Java para definir um tipo enumerado.
- Maior segurança no código, menos erros e manutenção facilitada.