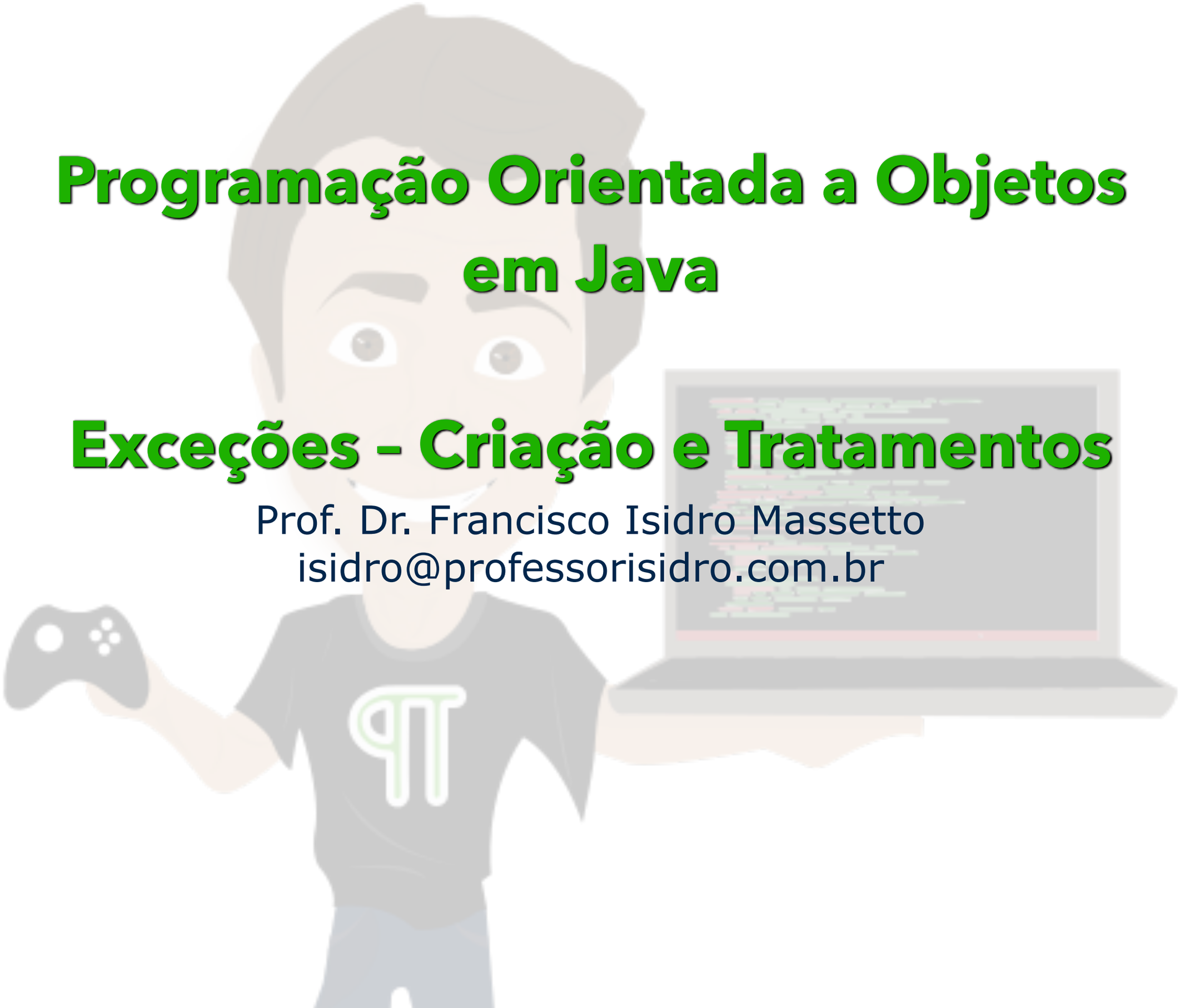
Programação Orientada a Objetos em Java

Exceções - Criação e Tratamentos

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



Exceções e Erros

- 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

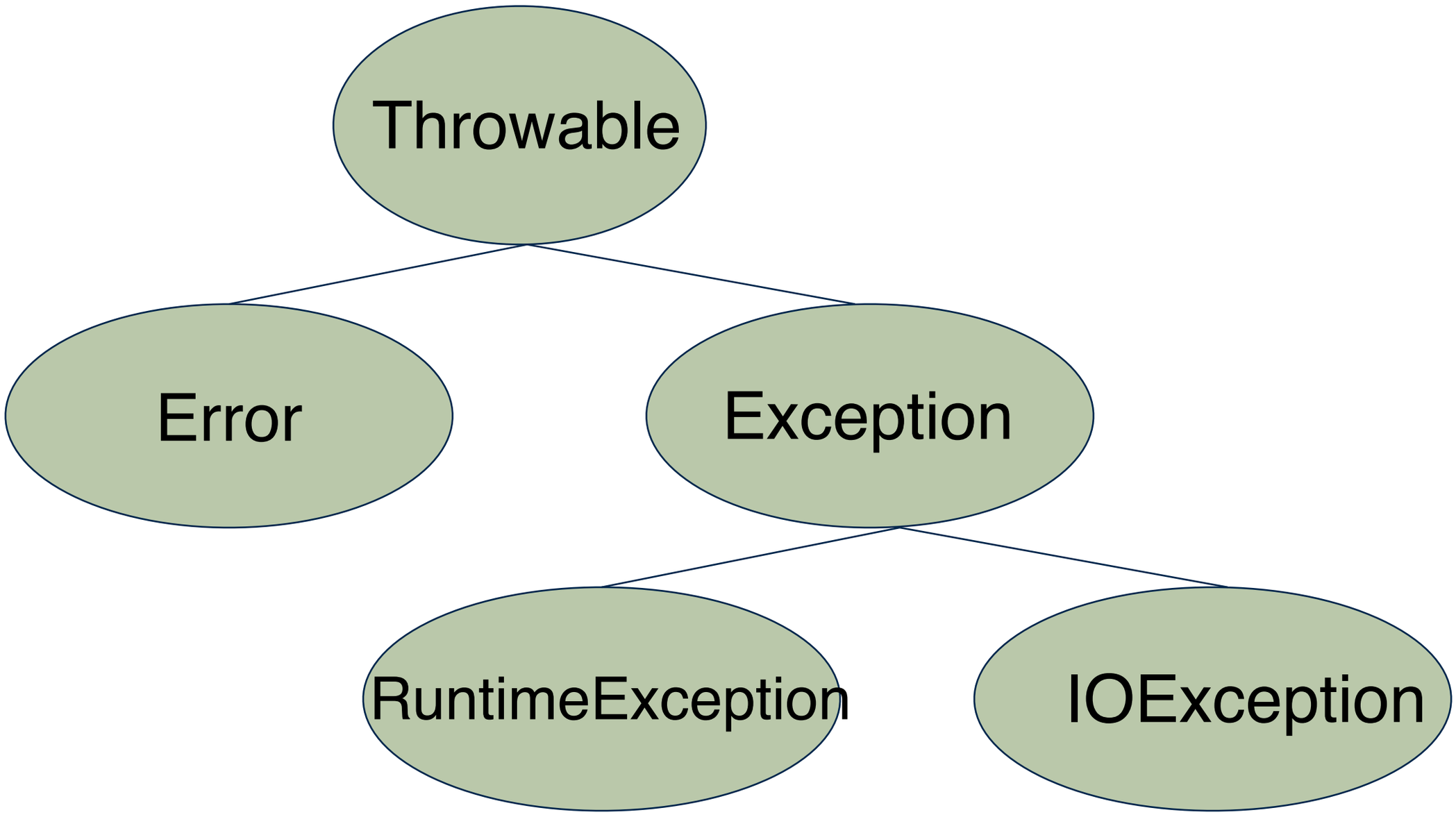
Exceções e Erros

- Erro
  - Evento que ocorre na execução da máquina Virtual 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

Por que usar 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 de classes



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

```
try{
    bloco de execução
}
```

- Obrigatoriamente deve haver um bloco de tratamento

Bloco Catch

```
try{
      ...
}
catch (TipoDaExceção objExcecao){
    bloco de tratamento
}
```

- Contém o trecho de código responsável por tratar a exceção lançada no bloco try

Bloco Finally

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

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

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

```
try{
     linha principal de execução
}
catch (TipoDaExceção objExcecao){
     bloco de tratamento
}
finally{
     bloco final
}
```

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

- 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

Exemplo

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

Lançando várias exceções

- Um método pode lançar várias exceções
  - Dependendo do comportamento do método

- A captura e tratamento das exceções lançadas são tratadas de forma genérica ou específica pela aplicação
  - Pelo menos uma exceção deve ser tratada

Criando as próprias exceções

- 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