# Exceptions (exeções)

Agora que aprendemos sobre errors anteriormente, podemos ver sobre as exeções.

Vimos que alguns erros são perceptíveis apenas aos programadores, e outro podem já ser identificados pelo compilador, os erros que são capturados pelo compilador e lançados, todos são chamados de exeções, ou seja, exeção nesse caso são todos os eventos inesperados que ocorrem durante a execução de um programa, indicando que algo deu errado.

Analisamos anteriormente uma estrutura de um erro não capturado, nele, havia escrito “uncaught error”, parece ser meio redundante, o próprio erro não capturado informar que não foi capturado, mas é assim que ocorre.

**O Que É um try-catch?**

O try-catch é uma **estrutura de controle** em linguagens de programação, incluindo PHP, que permite lidar com **erros ou exceções** de forma controlada. Ele é usado para **tratar situações inesperadas** ou **erros que possam ocorrer durante a execução de um bloco de código** sem interromper o funcionamento do programa.

Esse modelo permite lidar com erros de uma forma estruturada e organizada. A lógica de exceções é baseada no conceito de lançar (throw) e capturar (catch) exceções.

**Estrutura Básica de um bloco try-catch**

* **Bloco try**: O bloco try envolve o código que pode causar um erro. Se o erro ocorrer, o código dentro do bloco try é interrompido, e o controle é passado para o bloco catch.
* **Bloco catch**: O bloco catch captura a exceção lançada no bloco try. Aqui, você pode lidar com o erro de maneira adequada, como exibir uma mensagem de forma amigável ou registrar o erro em um arquivo de log. Nós não podemos criar um catch sem um bloco try !!!
* **Bloco finally**: Este bloco é opcional e sempre será executado, independentemente de uma exceção ter sido lançada ou não. Ele é útil para liberar recursos ou garantir que certos códigos sejam executados, como fechar conexões de banco de dados.

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente com confiança baixa

Parece estranho né, como iremos por um código dentro de um bloco try-catch, se nem sabemos qual código que vai dar o erro ?

**A resposta é:** Não sabemos, o bloco try-catch serve para tratarmos erros muitos específicos do nosso sitema que já sabemos que pode acontecer, como exemplo, um nome de usuário incompleto ou uma senha errada etc.

**Por Que Usar try-catch Mesmo Sem Ter Certeza?**

* **Código Que Envolve Riscos**: Em muitas situações, você pode antecipar quais partes do seu código têm potencial para falhar, mesmo que não tenha certeza absoluta. Por exemplo:
  + **Conexões com o banco de dados**: Podem falhar se o servidor estiver fora do ar.
  + **Leitura/escrita de arquivos**: Pode falhar se o arquivo não existir ou se as permissões forem insuficientes.
  + **Operações com APIs externas**: Podem falhar devido a problemas de rede ou erros na API.
* **Prevenção**: O objetivo é se proteger de falhas inesperadas e garantir que o programa continue funcionando ou trate o erro de forma adequada.

**Como Saber Quais Partes Podem Gerar Erros?**

* **Experiência**: Com o tempo, você aprende quais operações são mais suscetíveis a falhas.
* **Documentação**: A documentação das funções e classes que você está usando geralmente indica quais exceções ou erros podem ocorrer.
* **Testes e Depuração**: Durante o desenvolvimento, você pode identificar áreas que são mais propensas a falhas.

**Texto

Descrição gerada automaticamenteExemplo 1: Conexão com o Banco de Dados**

**Explicação**: Aqui, você não sabe com certeza se a conexão vai falhar, mas é uma área onde erros são comuns, então você se previne.

**Texto, Linha do tempo

Descrição gerada automaticamenteExemplo 3: API Externa**

**Explicação**: A comunicação com APIs pode falhar por vários motivos, como falhas de rede.

**Quando Usar try-catch?**

* Quando você está realizando operações que podem falhar, como:
  + Conectar a um banco de dados.
  + Trabalhar com arquivos ou sistemas externos.
  + Fazer cálculos que podem gerar erros, como divisão por zero.
  + Trabalhar com APIs ou serviços externos que podem falhar.

**1. Objeto Exception**

No PHP, uma exceção é um evento que interrompe o fluxo normal de um programa quando ocorre um erro. Porém, temos nativamente dentro do PHP, uma classe que capta, reúne e organiza estruturalmente esses erros para que podemos extanciar essa classe e trabalhar-mos mais facilmente com os erros. Portanto, se Exception é uma classe, logo ela também terá definida suas propriedades e métodos nativos ou que podem ser usados também com ela.

**Propriedades da Classe Exception**

As principais propriedades da classe Exception incluem:

* **$message**: Uma mensagem de erro que descreve o que aconteceu. Você pode passar essa mensagem quando cria uma nova exceção.
* **$code**: Um código de erro opcional que você pode especificar. Ele é útil para identificar o tipo ou a natureza do erro.
* **$file**: O nome do arquivo onde a exceção foi lançada. Isso é definido automaticamente pelo PHP.
* **$line**: O número da linha onde a exceção foi lançada. Também é definido automaticamente pelo PHP.

**Métodos da Classe Exception**

A classe Exception também possui métodos que ajudam a obter detalhes sobre a exceção. Alguns dos métodos mais importantes são:

* **getMessage()**: Retorna a mensagem de erro.
* **getCode()**: Retorna o código de erro.
* **getFile()**: Retorna o nome do arquivo onde a exceção foi lançada.
* **getLine()**: Retorna o número da linha onde a exceção foi lançada.
* **getTrace()**: Retorna um array com a pilha de execução (call stack) no momento em que a exceção foi lançada.
* **getTraceAsString()**: Retorna a pilha de execução como uma string legível.
* Construct
* toString

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

**2. Palavra Reservada throw**

A palavra reservada throw no PHP é usada para "lançar" uma exceção, ou seja, indicar que ocorreu um erro. Isso interrompe o fluxo normal do programa e faz com que o PHP tente encontrar um bloco catch adequado para capturar e tratar a exceção.

**Por que é Necessário Usar throw com new Exception?**

* **new Exception**: Isso cria uma nova instância da classe Exception, mas não lança automaticamente o erro. Em outras palavras, o objeto de exceção é criado, mas o fluxo de execução do programa não é interrompido nem redirecionado.
* **throw**: A palavra throw é usada para lançar a exceção, ou seja, indicar que ocorreu um erro e que o fluxo normal do programa deve ser interrompido até que a exceção seja capturada por um bloco catch ou tratada de outra forma.
* O throw pode ser usado dentro de um bloco try para lançar uma exceção que será capturada pelo catch, ou fora de um bloco try para lançar uma exceção globalmente, o que pode resultar em um erro fatal se não houver um catch para capturá-la.

Texto

Descrição gerada automaticamente

**Uso de throw Sem try é Tratamento Global**

Texto

Descrição gerada automaticamenteSe você usar throw sem try e quiser evitar que o programa termine com um erro fatal, pode configurar um manipulador global de exceções com set\_exception\_handler():

Perceba que colocamos o Exception como parâmetro da função, isso indica que nós estamos esperando um $error ($e) do tipo Exception.

Exception é uma classe, então podemos ter instâncias de objeto dessa classe, inclusive usar como parâmetro de tipo objeto.

# Extendendo o métodos exceptions

Podemos também estender a classse Exception para outras classes, assim nós herdamos os parâmetros e métodos e podemos trabalhar com eles de forma personalizada para casa caso no sistema:

Texto

Descrição gerada automaticamente

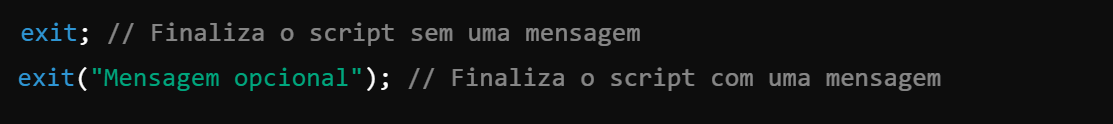
**Explicação do Exemplo**

1. **Extensão da Classe Exception**: class DatabaseException extends Exception cria uma nova classe DatabaseException que herda todas as funcionalidades da classe Exception.
2. **Método Personalizado**: errorMessage() é um método personalizado que retorna uma mensagem de erro específica.
3. **Lançando e Capturando**: A exceção DatabaseException é lançada com throw e capturada no bloco catch.
4. **Bloco catch:** Quando uma exceção é lançada, o PHP procura um bloco catch que tenha um tipo compatível com o objeto de exceção. Se throw new DatabaseException("Erro no banco de dados") for chamado, o PHP procura um catch que capture DatabaseException.
   1. **DatabaseException**: Especifica o tipo da exceção que o bloco catch deve capturar. Se uma exceção desse tipo for lançada, o PHP executará o código dentro do bloco catch.
   2. **$e**: É o nome da variável que armazena o objeto da exceção capturada. Com $e, você pode acessar as propriedades e métodos do objeto da exceção, como getMessage(), getCode(), etc.

# exit() & die()

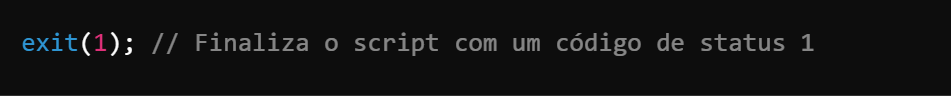
No PHP, as funções exit e die são usadas para finalizar a execução de um script imediatamente. As duas são equivalentes: die() é apenas um alias de exit(), o que significa que funcionam da mesma forma.

Texto

Descrição gerada automaticamente com confiança média

**Como Funcionam**

* Quando exit ou die é chamado, o PHP interrompe a execução do script naquele ponto. Se uma mensagem opcional for fornecida, ela será exibida na saída padrão (normalmente no navegador ou na linha de comando).
* A função exit() também aceita um código de status numérico, que pode ser usado para indicar o motivo pelo qual o script foi encerrado. Isso é especialmente útil em scripts de linha de comando.



Em scripts de linha de comando, um código de status diferente de 0 geralmente indica que o script terminou com um erro.

**Diferença Entre exit e die**

Como mencionado, exit e die são exatamente a mesma coisa. Você pode usar qualquer um deles, e a escolha geralmente se resume a uma questão de preferência pessoal ou de estilo do código.

## escopos & método exit e die

Texto

Descrição gerada automaticamenteQuando nós vamos tratar um erro, podemos acabar querendo criar muitas verificações e ficar com um código enorme dentro do try. O que pode também acabar dendo algum erro mais afrente e ser chato de resolver. Para evitar-mos essa mistureba, podemos subdividir essas verificações jogando elas em funções aparte, e chamar essas funções dentro de um bloco try-cath central.

Texto

Descrição gerada automaticamente

>>>>

Porém, perceba uma coisa, logo após chamarmos a função no bloco central, temos em seguida um echo com uma saída (“sucesso”).

Interface gráfica do usuário

Descrição gerada automaticamenteSe formos dar continuidade com esse código acima para testarmos nossa verificação, o resultado seria:

Texto

Descrição gerada automaticamenteOu seja, o “sucesso” apareceu mesmo com a exceção sendo lançada.... para resolver isso, podemos simplesmente colocar um exit() ou um die() lá dentro e no final na nossa função:

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Claro, iremos fazer apenas se desejarmos que a nossa aplicação pare.

Agora que entendemos isso, podemos entender melhor como utilizar o bloco “finally”.

# finally

Interface gráfica do usuário

Descrição gerada automaticamenteAo oposto do exit() e do die() temos o bloco finally, que irá fazer alguma coisa, independente se uma exceção for lançada ou não, nós sempre temos que colocar ele logo depois do último catch do nosso código.

Texto

Descrição gerada automaticamenteInterface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

# códigos de error

Texto

Descrição gerada automaticamentePerceba uma coisa, sempre que instanciâmos um objeto da classe Exception, podemos passar 2 parâmetros, uma mensagem e um código. Existe um terceiro parâmetro chamado de Throwable que veremos mais a frente.

* No exemplo do código que você forneceu, o 1 é um código de erro definido pelo desenvolvedor. Ele não tem um significado especial no PHP por si só, mas pode ser usado para diferenciar esse erro específico de outros.
* Você pode definir seus próprios códigos de erro para categorizar diferentes tipos de exceções. Por exemplo:

Texto

Descrição gerada automaticamente

Nós mesmos podemos criar códigos de erros personalizados para o nosso programa, eles não são nativos da linguagem, são padrões que nós podemos criar para o nosso sistema.

# throwable

**O Que é Throwable?**

Throwable é uma interface introduzida no PHP 7 que serve como a raiz de todas as exceções e erros no PHP. Isso significa que qualquer objeto que possa ser "lançado" (ou seja, usado com throw) deve implementar a interface Throwable.

* Antes do PHP 7, o sistema de tratamento de erros e exceções do PHP era mais limitado: as exceções derivavam apenas da classe Exception, e erros como "erros fatais" não podiam ser capturados diretamente com try-catch.
* Com a introdução de Throwable, o PHP unificou o tratamento de erros e exceções, permitindo capturar tanto exceções (Exception e suas subclasses) quanto erros (Error e suas subclasses) com a mesma abordagem.

**Hierarquia de Throwable**

A interface Throwable é implementada por duas classes principais:

1. **Exception**: A classe base para todas as exceções no PHP. As exceções são lançadas explicitamente com throw e capturadas com catch.
2. Texto

   Descrição gerada automaticamente**Error**: A classe base para todos os erros no PHP. Erros são eventos críticos que geralmente indicam problemas mais graves, como falhas de memória ou erros de tipo. Você também pode capturar Error com blocos try-catch.

**Principais Características de Throwable**

A interface Throwable define os métodos que todas as exceções e erros devem implementar. Aqui estão esses métodos:

1. **getMessage()**: Retorna a mensagem descritiva do erro ou exceção.
2. **getCode()**: Retorna o código de erro associado, que é opcional.
3. **getFile()**: Retorna o nome do arquivo onde o erro ou exceção ocorreu.
4. **getLine()**: Retorna o número da linha onde o erro ou exceção ocorreu.
5. **getTrace()**: Retorna um array com a pilha de execução (call stack) no momento em que o erro ou exceção ocorreu.
6. **getTraceAsString()**: Retorna a pilha de execução como uma string legível.
7. **getPrevious()**: Retorna uma exceção anterior, se a exceção atual for uma cadeia de exceções (útil para encadeamento de erros).
8. **\_\_toString()**: Retorna uma representação em string do erro ou exceção.

### **Como** Throwable **é Usado**

Você não pode instanciar diretamente Throwable porque é uma interface. Em vez disso, você trabalha com classes que a implementam, como Exception e Error. No entanto, você pode capturar qualquer coisa que implemente Throwable no seu bloco catch.

Assim você faz com que nenhum erro passe despercebido, tanto global quanto do bloco try-catch.

Texto

Descrição gerada automaticamente

# error também é classe

Vimos sobre error no capítulo anterior, mas agora, aqui podemos entender melhor sober ele.

Error é uma classe no PHP, assim como Exception!

A classe Error representa erros que são gerados pelo PHP quando ocorrem problemas graves, como erros de tipo, falhas de memória ou erros de sintaxe em tempo de execução. Antes do PHP 7, esses erros eram tratados de maneira diferente das exceções e muitas vezes causavam a interrupção do script sem a possibilidade de captura. Agora, com a classe Error, você pode capturar esses erros usando o sistema de tratamento de exceções.

**Subclasses da Classe Error**

A classe Error serve como a classe base para vários tipos de erros específicos. Aqui estão todas as principais subclasses de Error:

**TypeError**

* Lançada quando um tipo de argumento inválido é passado para uma função ou método.
* Exemplo: function add(int $a, int $b) { return $a + $b; } add("string", 5); lança TypeError.

**ParseError**

* Lançada quando ocorre um erro de sintaxe, como em um eval() com código inválido.
* Exemplo: eval("invalid code"); lança ParseError.

**UnhandledMatchError** (Introduzida no PHP 8.0)

* Lançada quando uma expressão match não consegue encontrar um caso correspondente.
* Exemplo: match($value) { 1 => 'one', 2 => 'two' }; lança UnhandledMatchError se $value não for 1 ou 2.

**ArithmeticError**

* Lançada para erros aritméticos.
* Subclasse: DivisionByZeroError é um caso específico de erro aritmético.

**DivisionByZeroError**

* Lançada especificamente quando uma operação de divisão por zero ocorre em operações de números inteiros.
* Exemplo: intdiv(1, 0); lança DivisionByZeroError.

**AssertionError**

* Lançada quando uma asserção falha, especialmente ao usar assert().
* Exemplo: assert(false); lança AssertionError.

**Propriedades da Classe Error**

A classe Error herda todas as propriedades da interface Throwable, que são as mesmas que a classe Exception possui. Essas propriedades fornecem informações importantes sobre o erro que ocorreu.

1. **Propriedades Comuns em Error**:
   * **$message**: Uma mensagem descritiva sobre o erro. Por padrão, é uma string vazia, mas pode ser definida no momento em que o erro é lançado.
   * **$code**: Um código de erro opcional. Por padrão, é 0, mas você pode definir um valor específico.
   * **$file**: O nome do arquivo onde o erro ocorreu. Isso é definido automaticamente pelo PHP.
   * **$line**: O número da linha onde o erro ocorreu. Também é definido automaticamente pelo PHP.

Texto

Descrição gerada automaticamente