Semana 6
Tratamento de Exceções e Eventos

- Prof. Cassiano Moralles

Tratamento de Exceções

- As questões de projeto para tratamento de exceções em linguagens de programação podem ser resumidas em vários aspectos cruciais:

detectar erros em tempo de compilação ou em tempo de execução. A escolha afeta a complexidade do compilador e a performance do programa.

2. **Propagação de Exceções:** Como as exceções são propagadas através das funções ou métodos? A propagação pode ser automática, subindo a cadeia de chamadas até encontrar um bloco que trate a exceção, ou pode requerer que cada função ou método lide explicitamente com a exceção.

3. **Tratamento de Exceções:** Como as exceções são tratadas? As linguagens precisam definir a

Tratamento de Exceções

- As questões de projeto para tratamento de exceções em linguagens de programação podem ser resumidas em vários aspectos cruciais:

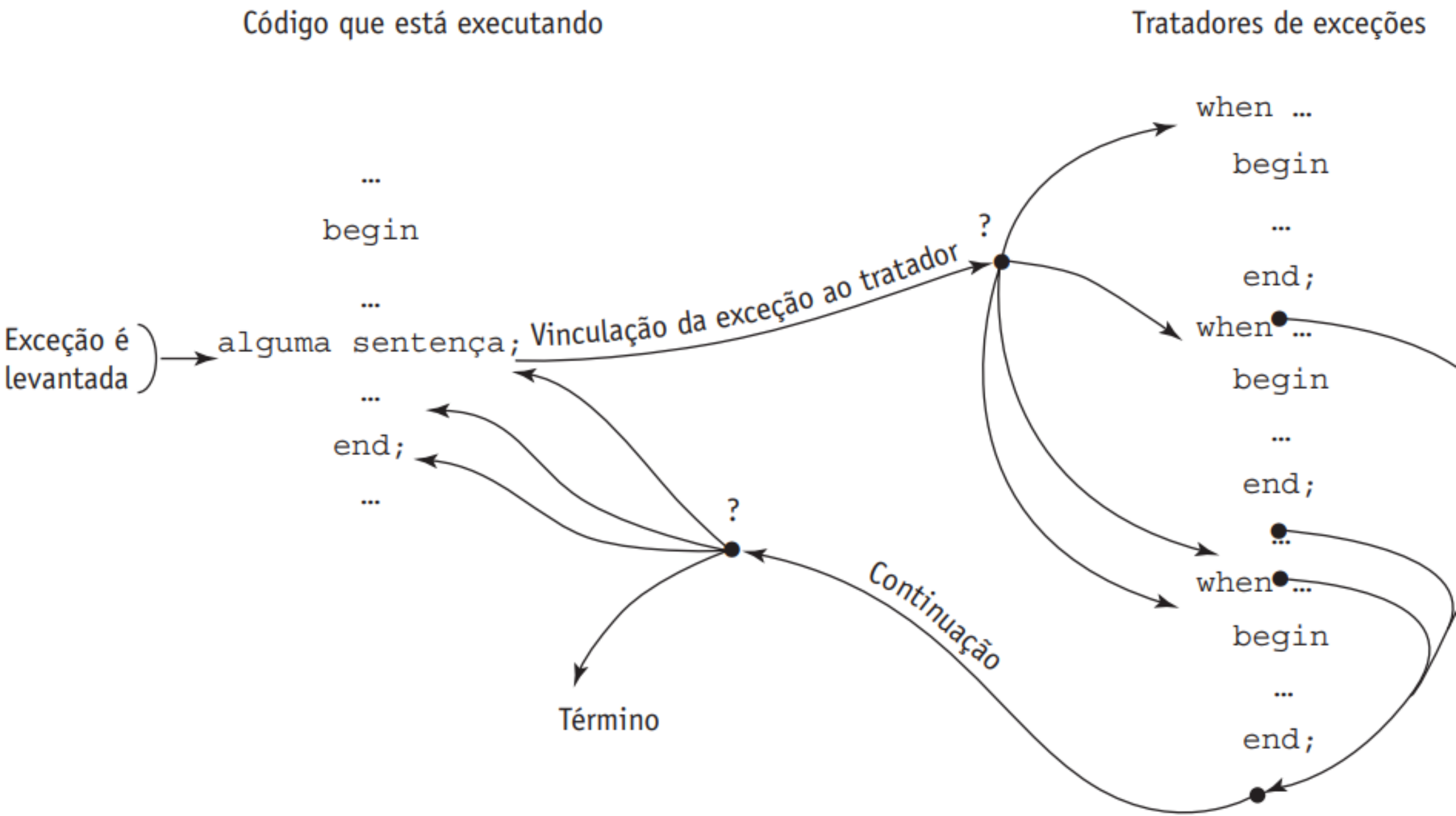
4. **Tipo de Exceções:** Que tipos de exceções são suportadas? As exceções podem ser categorizadas em exceções padrão, definidas pelo sistema, ou exceções personalizadas, definidas pelo usuário.

5. **Desempenho e Overhead:** Qual é o impacto do tratamento de exceções na performance do programa? Implementar um sistema de exceções pode introduzir overhead adicional, tanto em termos de processamento quanto de memória.

6. **Recursos e Limpeza:** Como os recursos são gerenciados quando uma exceção ocorre? Isso inclui garantir que recursos como arquivos e conexões de rede sejam corretamente liberados. Muitas linguagens oferecem mecanismos como blocos `` `finally` `` ou o padrão RAII (Resource Acquisition Is Initialization) em C++.

7. **Mecanismos de Reinicialização:** Existe a capacidade de reiniciar a operação após uma exceção? Algumas linguagens permitem tentar uma operação novamente ou realizar ações de correção.

8. **Integração com Bibliotecas e Frameworks:** Como o sistema de tratamento de exceções se integra com bibliotecas e frameworks existentes? É importante garantir que as exceções sejam compatíveis e coerentes em todo o ecossistema da linguagem.


Código que está executando
Tratadores de exceções
...
begin
...
alguma sentença;
...
end;
...
Exceção é levantada
Vinculação da exceção ao tratador
?
when ...
begin
...
end;
when ...
begin
...
end;
when ...
begin
...
end;
?
Continuação
Término

Exemplo Completo de Tratamento de Exceções em Python

Aqui está um exemplo completo que demonstra várias práticas de tratamento de exceções em Python, incluindo o uso de `try`, `except`, `else` e `finally`.

python Copiar código

```python
def dividir(a, b):
    try:
        # Tenta realizar a divisão
        resultado = a / b
    except ZeroDivisionError as e:
        # Captura e trata a exceção de divisão por zero
        print(f"Erro: Divisão por zero não é permitida. Detalhes: {e}")
        resultado = None
    except TypeError as e:
        # Captura e trata a exceção de tipo incorreto
        print(f"Erro: Tipos inválidos para divisão. Detalhes: {e}")
        resultado = None
    else:
        # Executa se não houver exceções
        print(f"Divisão bem-sucedida: {a} / {b} = {resultado}")
    finally:
        # Executa sempre, independente de exceções
        print("Execução do bloco finally")
        return resultado

# Testando a função com diferentes entradas
print("Teste 1: Divisão normal")
resultado1 = dividir(10, 2)
print(f"Resultado 1: {resultado1}\n")

print("Teste 2: Divisão por zero")
resultado2 = dividir(10, 0)
print(f"Resultado 2: {resultado2}\n")

print("Teste 3: Tipos inválidos")
resultado3 = dividir(10, "dois")
print(f"Resultado 3: {resultado3}\n")
```

Explicação do Código

1. **Definição da Função `dividir`**: A função `dividir` recebe dois argumentos `a` e `b` e tenta realizar a divisão `a / b`.
2. **Bloco `try`**: Contém o código que pode potencialmente gerar uma exceção, neste caso, a operação de divisão.
3. **Blocos `except`**:
   - `except ZeroDivisionError`: Captura e trata a exceção específica de divisão por zero.
   - `except TypeError`: Captura e trata a exceção de tipo incorreto, como tentar dividir um número por uma string.
4. **Bloco `else`**: Executa se nenhum erro ocorrer no bloco `try`. Aqui, imprime o resultado da divisão.
5. **Bloco `finally`**: Executa sempre, independentemente de ter ocorrido uma exceção ou não. Geralmente usado para limpar recursos ou realizar ações que devem ocorrer de qualquer maneira.
6. **Testes da Função**:
   - **Teste 1**: Realiza uma divisão normal.
   - **Teste 2**: Tenta dividir por zero.
   - **Teste 3**: Tenta dividir por um tipo inválido.