Desenvolvimento de Sistemas Orientados a Objetos I

Tratamento de Exceções

Jean Carlo Rossa Hauck, Dr.

jean.hauck@ufsc.br

http://www.inf.ufsc.br/~jeanhauck

Conteúdo Programático

- Conceitos e mecanismos da programação orientada a objetos
- Técnicas de uso comum em sistemas orientados a objetos
  - Tratamento de exceções

Tratamento de Exceções

Exceção é um evento que ocorre durante a **execução de um programa** e que **interrompe o fluxo normal** de instruções, por exemplo:

- Divisão por zero
- Operação matemática inválida
- Tentativa de acesso a uma posição inválida em um vetor
- Tentativa de acesso a um objeto que não foi criado
- etc.

Tratamento de Exceções

Quando uma exceção ocorre dentro de um método, é criado um objeto que é entregue para o ambiente de execução

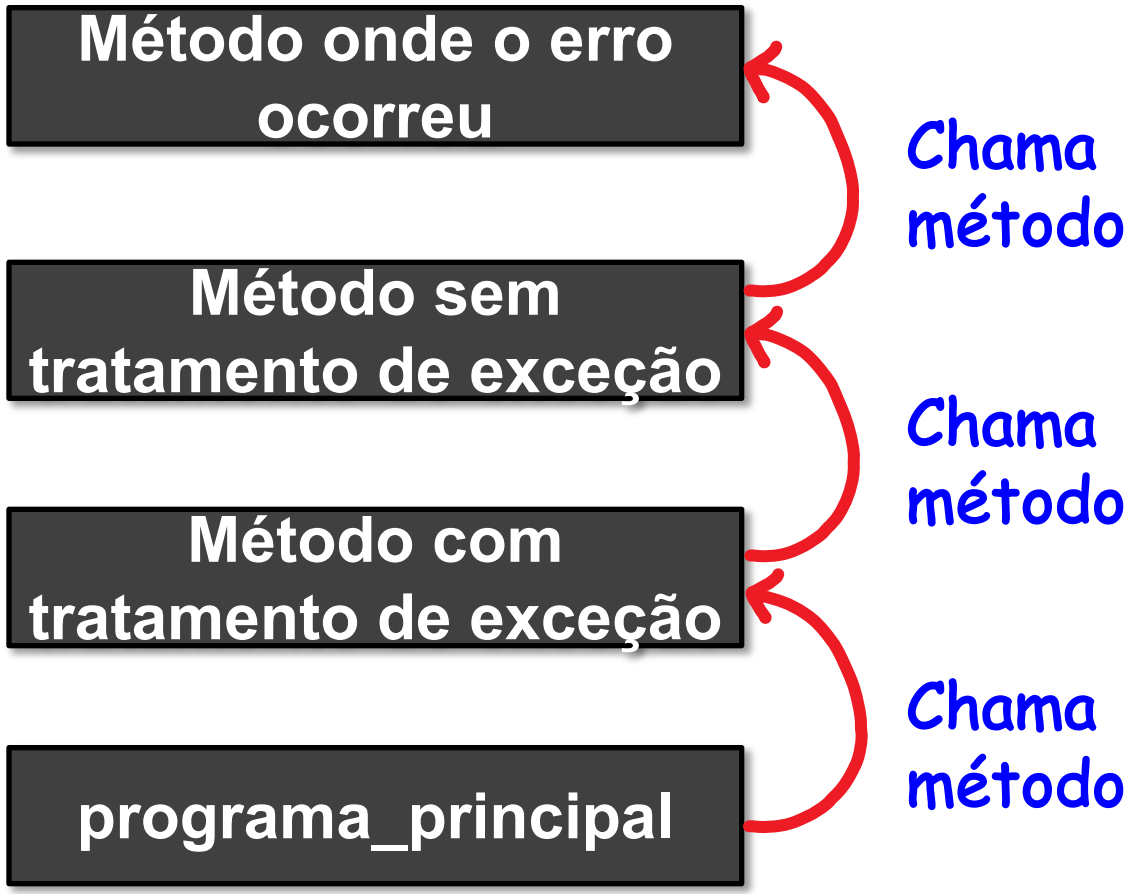
- Este comportamento é denominado levantar ("**raise**") uma exceção
- O objeto criado é chamado de **objeto de exceção** e contém informações sobre a exceção, incluindo seu tipo e o estado do programa quando a exceção ocorreu

https://docs.python.org/3/library/exceptions.html

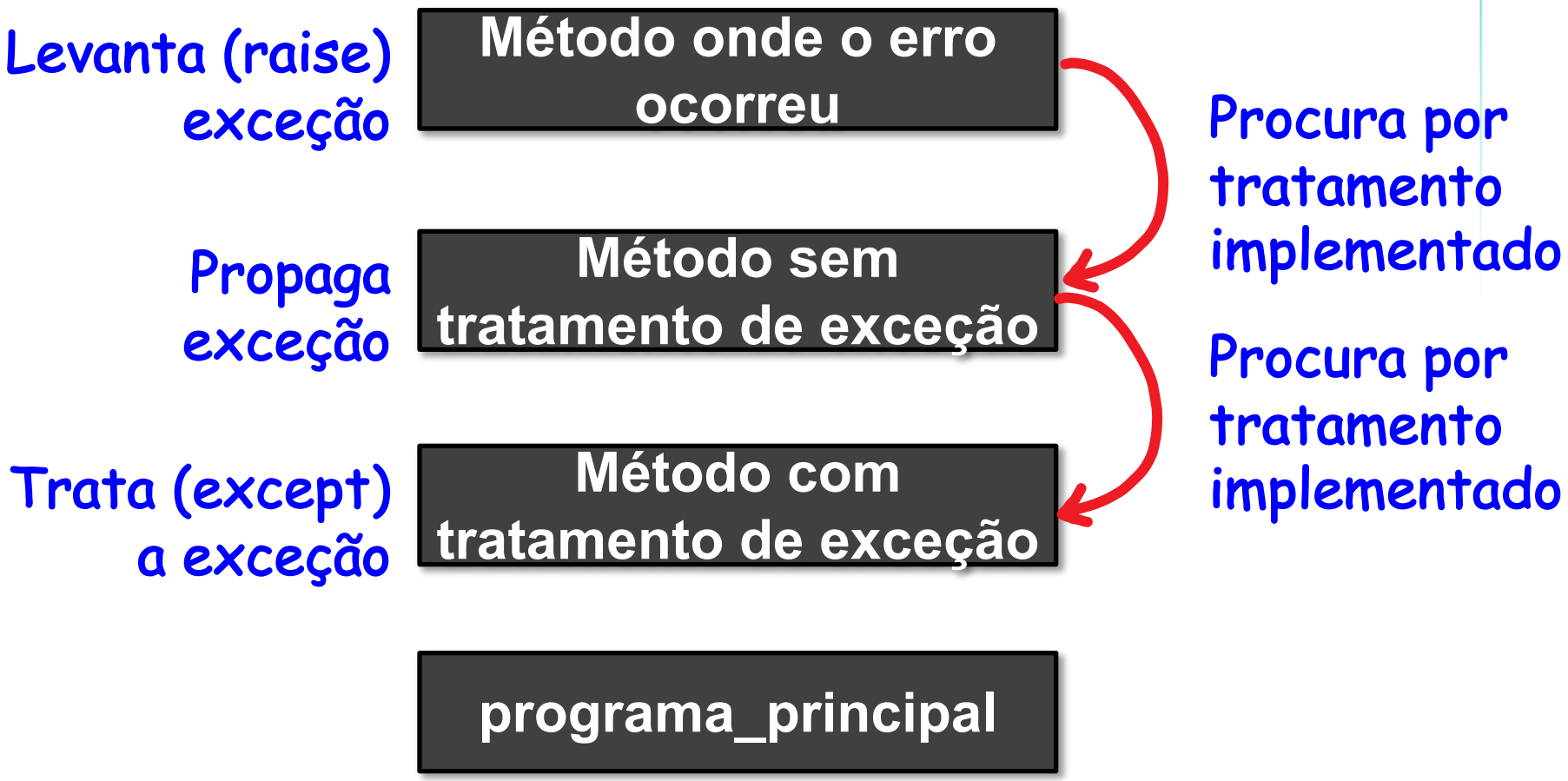
Tratamento de Exceções

- Depois que um método levanta (*raise*) uma exceção, o ambiente de execução tenta encontrar algum tratamento para a exceção
- A sequência de possíveis tratamentos para manipular a exceção é a pilha dos métodos que foram chamados para chegar até o método onde o erro ocorreu
- Antes de uma cláusula de exceção ser executada, os detalhes sobre a exceção são armazenados no módulo *sys* e podem ser acessados via *sys.exc_info()*

Tratamento de Exceções



Tratamento de Exceções

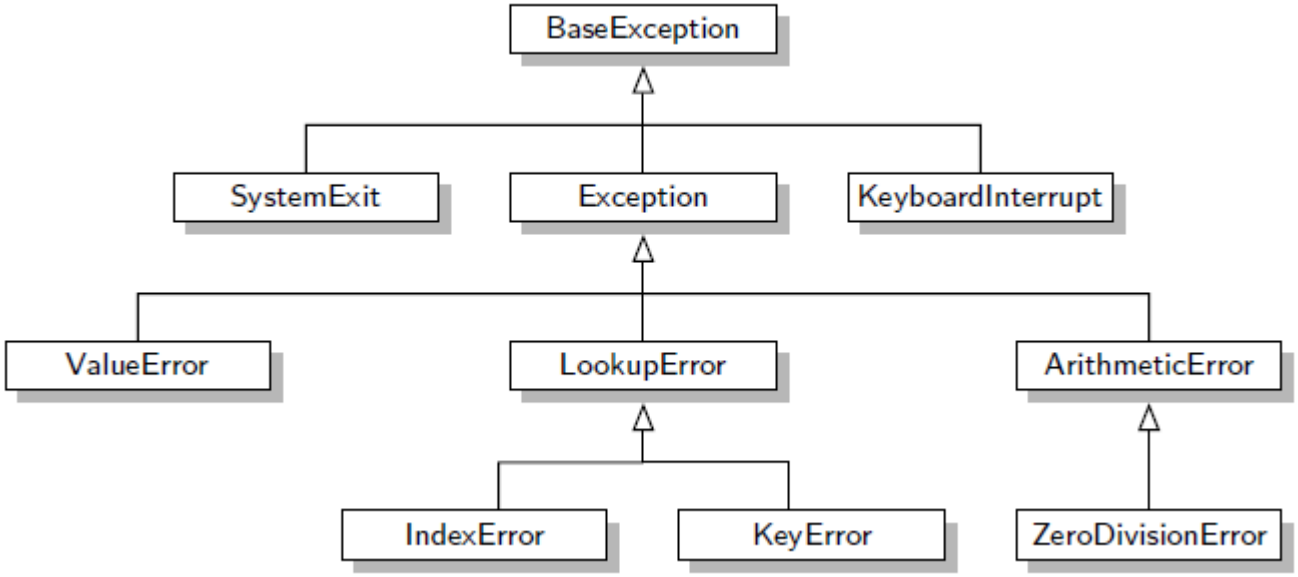


try ... except

```
try:
    1/0
except Exception:
    print("Erro!")
else:
    print("Nao deu erro!")
finally:
    print("Sempre executa!")
```

1. O código controlado pela cláusula ***try*** é executado
2. Se **ocorrer uma exceção**, o controle é desviado para a cláusula ***except***
3. Se a exceção ocorrida **estiver sendo tratada** pela cláusula ***except***, o código de tratamento da exceção é executado.
4. Se **não ocorrer uma exceção**, o código da clausula ***else*** é executado logo após o código do *try* e o tratamento da exceção **não é executado**
5. **Ocorrendo ou não exceção**, o código dentro da cláusula ***finally*** é **sempre executado**

Tipos de Exceções em Python

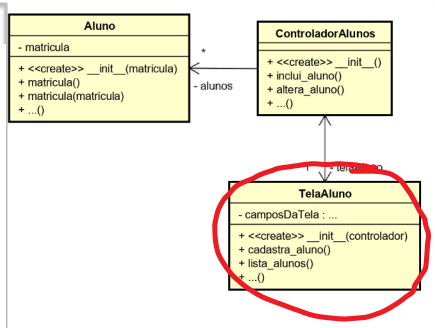


https://docs.python.org/3/library/exceptions.html

[http://www.devinline.com/]

Continuando com o Código da Tela

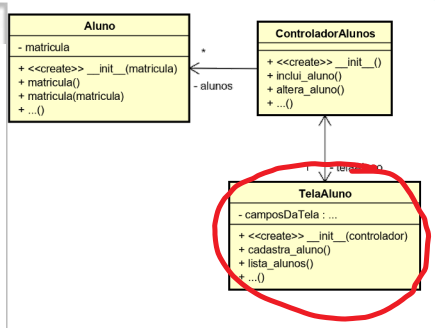
```
class TelaAluno:

    def __init__(self, controlador):
        self.__controlador = controlador

    def le_num_inteiro(self, mensagem: str = "", inteiros_validos: [] = None):
        while True:
            valor_lido = input(mensagem)
            try:
                inteiro = int(valor_lido)
                if inteiros_validos and inteiro not in inteiros_validos:
                    raise ValueError
                return inteiro
            except ValueError:
                print("Valor incorreto: Digite um valor num
                if inteiros_validos:
                    print("Valores validos: ", inteiros_valid

    def mostra_tela_opcoes(self):
        print("--------- CADASTRO ALUNOS ---------")
        print("1 - Incluir")
        print("2 - Alterar")
        print("3 - Excluir")
        print("4 - Listar")
        print("0 - Voltar")
        opcao = self.le_num_inteiro("Escolha a opcao: ", [1, 2, 3, 4, 0])
        return opcao
```



Também é possível disparar uma Exceção intencionalmente

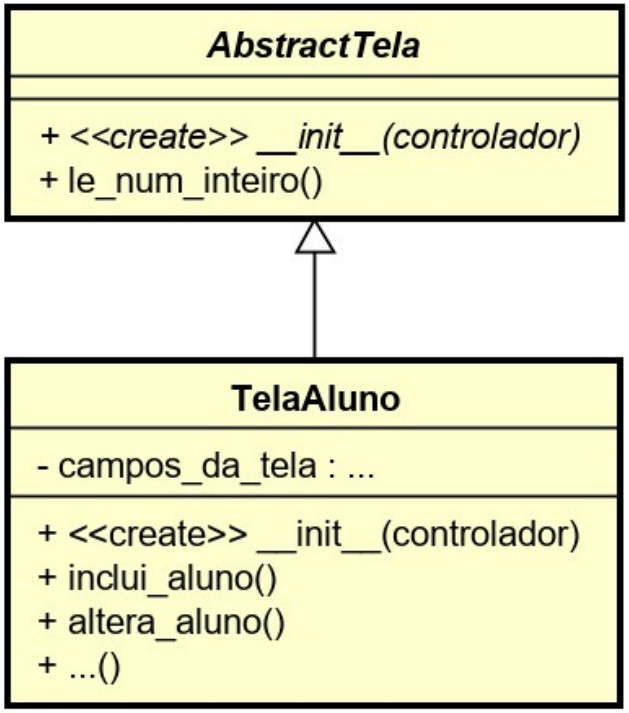
Continuando com o Código da Tela

```
class TelaAluno:

    def __init__(self, controlador):
        self.__controlador = controlador

    def le_num_inteiro(self, mensagem: str = "", inteiros_validos: [] = None):
        while True:
            valor_lido = input(mensagem)
            try:
                inteiro = int(valor_lido)
                if inteiros_validos and inteiro not in inteiros_validos:
                    raise ValueError
                return inteiro
            except ValueError:
                print("Valor incorreto: Digite um valor numerico inteiro valido")
                if inteiros_validos:
                    print("Valores validos: ", inteiros_validos)

    def mostra_tela_opcoes(self):
        print("--------- CADASTRO ALUNOS ---------")
        print("1 - Incluir")
        print("2 - Alterar")
        print("3 - Excluir")
        print("4 - Listar")
        print("0 - Voltar")
        opcao = self.le_num_inteiro("Escolha a opcao: ", [1, 2, 3, 4, 0])
        return opcao
```



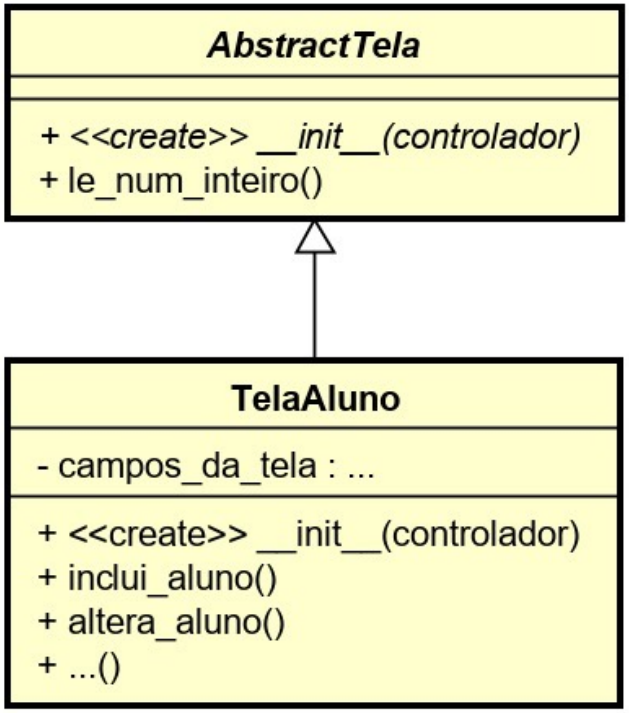
Este código será utilizado por outras telas?

Herança utilizada para Telas



Que tal então uma hierarquia de telas?

Herança utilizada para Telas

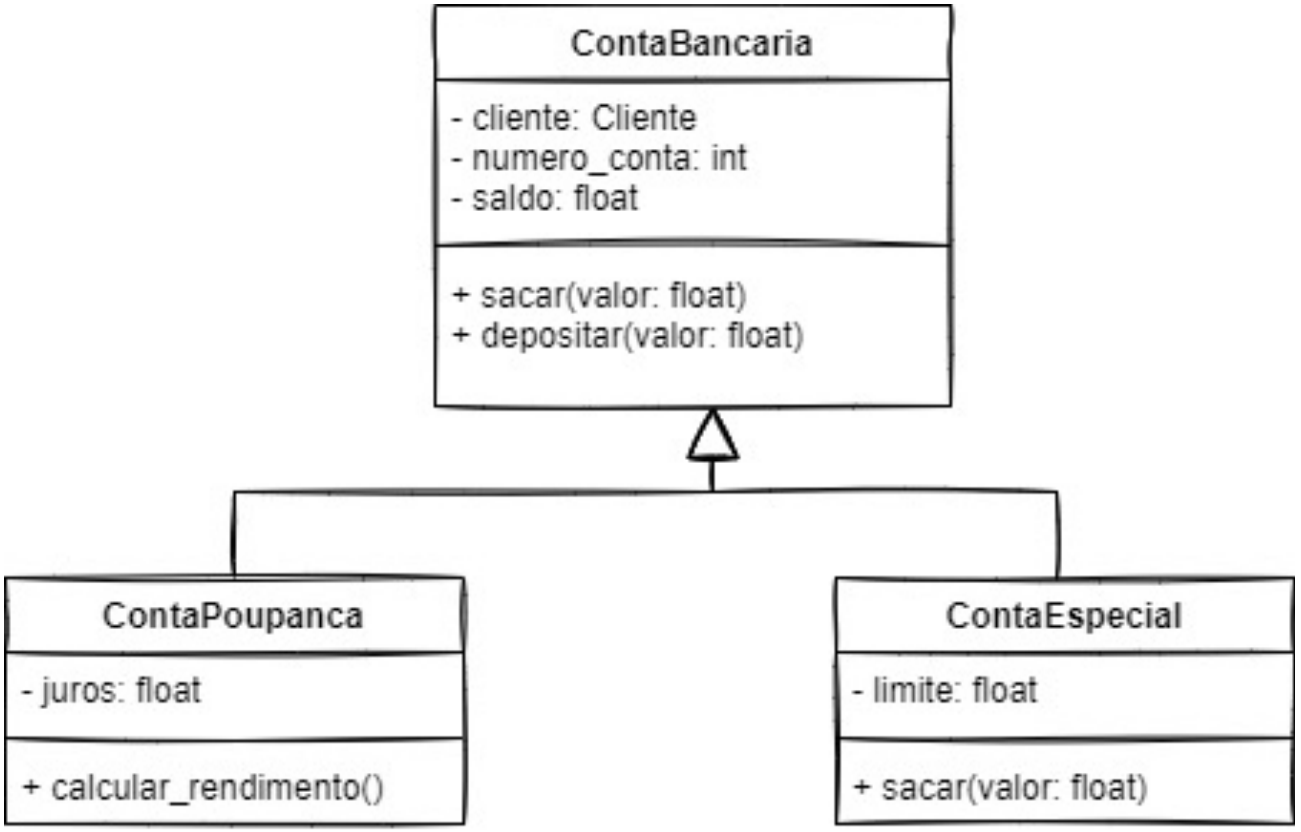


Que tal então uma hierarquia de telas?

Pode ser interessante também nos Controladores que tenham características comuns entre eles!

Criando suas próprias Classes de Exceções

Por que criar nossas próprias exceções?



* Métodos getters e setters ocultos para simplificar a visualização

Criando suas próprias Classes de Exceções

- O que aconteceria ao tentar chamar o método sacar com um valor fora do limite?
  - O sistema não deveria realizar o saque, mas quem chamou o método sacar não saberá que isso aconteceu.
- Como avisar aquele que chamou o método de que ele não conseguiu fazer aquilo que deveria?
  - Métodos devem seguir um contrato. Se, ao tentar sacar, o método não consegue fazer o que deveria, ele precisa, ao menos, avisar ao usuário que o saque não foi feito.

Criando suas próprias Classes de Exceções

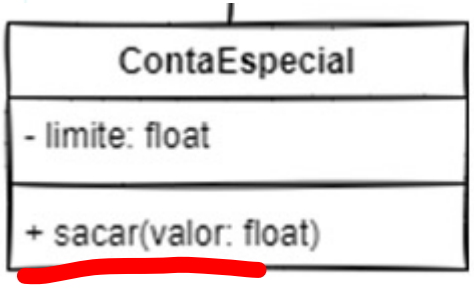
- Analise o código abaixo:

```
cliente = Cliente("Jean")
conta_especial = ContaEspecial(cliente, 3345, 1000.00, 100.00)
conta_especial.sacar(5000.00)
print(conta_especial.saldo)
```

  - O saldo é:
    - -3.900?
    - 0.0?
    - 1.000?

Criando suas próprias Classes de Exceções

A solução mais simples é a de marcar o retorno de um método como boolean e retornar true, se tudo ocorreu da maneira planejada, ou false, caso contrário:




```
class ContaEspecial(ContaBancaria):
    ...
def sacar(self, valor: float):
    if valor <= self.saldo:
        self.saldo = self.saldo - valor
        return True
    elif valor <= self.saldo + self.limite:
        self.limite = abs(self.saldo - valor)
        self.saldo = self.saldo - valor
        return True
    else:
        return False
```

Criando suas próprias Classes de Exceções

A solução mais simples é a de marcar o retorno de um método como boolean e retornar true, se tudo ocorreu da maneira planejada, ou false, caso contrário:

```
class ContaEspecial(ContaBancaria):
    ...
def sacar(self, valor: float):
    if valor <= self.saldo:
        self.saldo = self.saldo - valor
        return True
    elif valor <= self.saldo + self.limite:
        self.limite = abs(self.saldo - valor)
        self.saldo = self.saldo - valor
        return True
    else:
        return False
```

Conseguiu

Não Conseguiu

Criando suas próprias Classes de Exceções

- Como ficaria a chamada deste método?

```
cliente = Cliente("Jean")
conta_especial = ContaEspecial(cliente, 3345, 1000.00, 100.00)
if not conta_especial.sacar(5000.00):
    print("Saque não realizado!")
```

- Repare que tivemos de lembrar de testar o retorno do método!!!
  - Mas, não somos obrigados a fazer isso!!! Esquecer de testar o retorno desse método teria consequências drásticas: a máquina de autoatendimento poderia informar a liberação da quantia desejada de dinheiro, mesmo que o sistema não tivesse conseguido efetuar o método *sacar* com sucesso...

```
cliente = Cliente("Jean")
conta_especial = ContaEspecial(cliente, 3345, 1000.00, 100.00)
conta_especial.sacar(5000.00)
caixa_eletronico.emite_comprovante()
```

Criando suas próprias Classes de Exceções

- Mesmo invocando o método e tratando o retorno de maneira correta, o que faríamos se fosse necessário sinalizar quando o usuário passou um valor negativo como valor para saque?

- **Uma solução seria retornar o código do erro que ocorreu.**
- **Isso não é considerado uma boa prática (conhecida também como: "*magic numbers*").**
  - **O valor devolvido é "mágico" e precisa de documentação, além de não obrigar o programador a tratar esse retorno e, no caso de esquecer isso, seu programa continuará rodando em num estado inconsistente.**

Criando suas próprias Classes de Exceções

Por esses e outros motivos, podemos tratar também esses casos como **exceções** ...

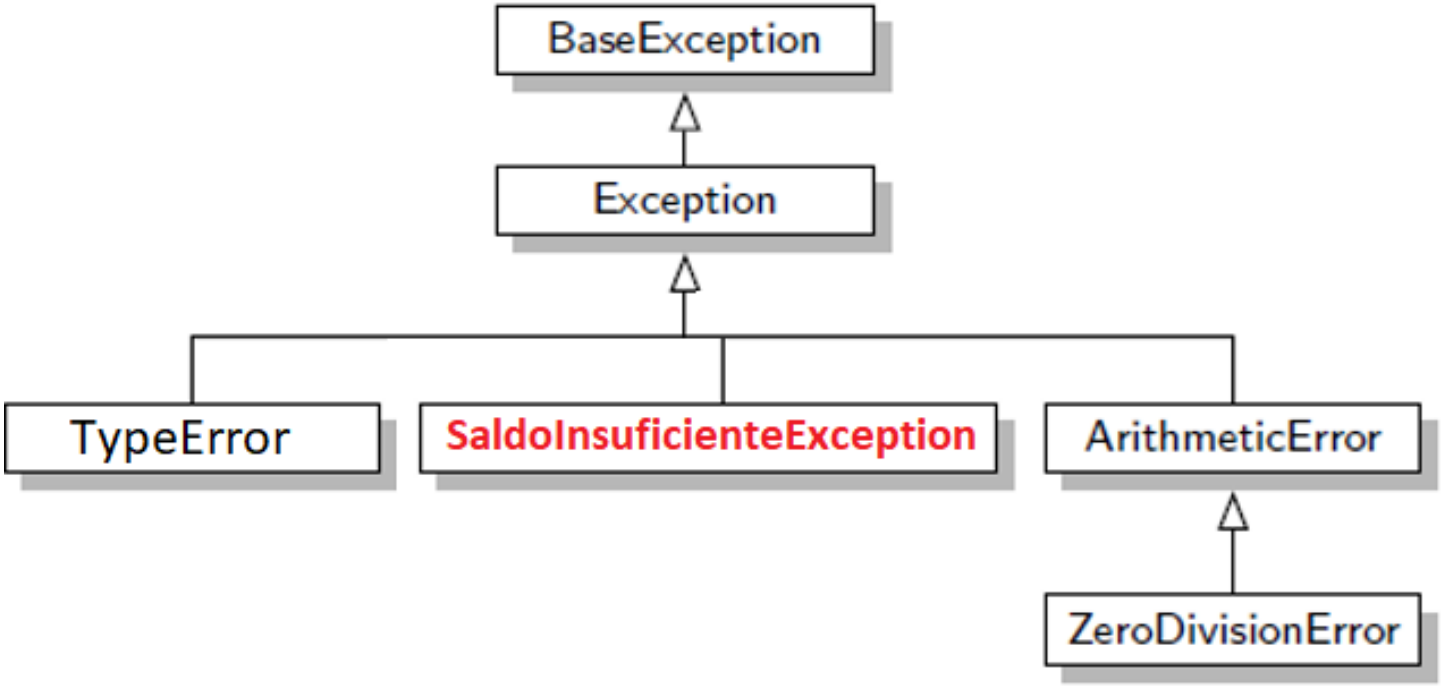
> Os casos onde acontece algo que, normalmente, não deveria acontecer.

O exemplo do saque inválido ou do cpf inválido de um cliente é uma **exceção à regra**.

Criando suas próprias Classes de Exceções

- Uma alternativa seria criar suas próprias classes de exceções:
  - ✓ Verifique se o Python já não oferece uma exceção que atenda às suas necessidades
  - ✓ Verifique se a nova exceção será realmente útil
  - ✓ Crie uma nova classe herdando de *Exception*
  - ✓ Utilize a palavra Exception também ao final do nome da classe criada (boa prática de legibilidade)

Criando suas próprias Classes de Exceções



Criando suas próprias Classes de Exceções

```python
class SaldoInsuficienteException(Exception):
    def __init__(self, valor: float):
        self.mensagem = f"Impossível sacar R$ {valor:.2f}"
        super().__init__(self.mensagem)
```

Criando suas próprias Classes de Exceções

```
def sacar(self, valor):
    if valor <= self.saldo:
        self.saldo = self.saldo - valor
    elif valor <= self.saldo + self.limite:
        self.limite = abs(self.saldo - valor)
        self.saldo = self.saldo - valor
    else:
        raise SaldoInsuficienteException(valor)
```

Criando suas próprias Classes de Exceções

```
conta = ContaEspecial(cliente, 123, 1000.00, 100.00)
try:
    conta.sacar(2000.00)
except SaldoInsuficienteException as e:
    print(e)
```

Agora implemente no Trabalho ...

Agradecimento

Agradecimento ao prof. Marcello Thiry pelo material cedido.

marcello.thiry@gmail.com

Licença