



# Exceções Direcionadas



## Exceções personalizadas

Falamos sobre como lidar com exceções quando elas são produzidas chamando métodos nas APIs Java. Java também permite criar e usar exceções personalizadas - classes de sua própria exceção, conforme a necessidade do aplicativo, que serão usadas para representar erros. Normalmente, você cria uma exceção personalizada para representar algum tipo de erro no seu aplicativo - para dar um significado novo e distinto a um ou mais problemas que podem ocorrer no seu código. Você pode fazer isso para mostrar semelhanças entre os erros existentes em vários locais do código, para diferenciar um ou mais erros de problemas semelhantes que podem ocorrer à medida que o código é executado ou para dar um significado especial a um grupo de erros em seu aplicativo.

É bastante fácil criar e usar uma exceção personalizada. Há três etapas básicas que você precisa seguir. Explicaremos isso com um exemplo de saldo da conta bancária. Aqui queremos ter a funcionalidade de depósito flexível, ou seja, quando o saldo da conta ultrapassar os 20k, um novo depósito será criado.

## Defina a classe de exceção

Você geralmente representa uma exceção personalizada ao definir uma nova classe. Em muitos casos, tudo que você precisa fazer é criar uma subclasse de uma classe de exceção existente:





```
public class SaldoContaException extends Exception{

    private float saldoConta;

    public SaldoContaException(float f){

        super();

        this.saldoConta=f;

    }

    public SaldoContaException(Stringmsg){

        super(message);

    }

    public float getSaldoConta(){

        return saldoConta;

    }

}
```

No mínimo, você precisa subclassificar `Throwable` ou uma de suas subclasses. Frequentemente, você também define um ou mais construtores para armazenar informações como uma mensagem de erro no objeto, conforme mostrado nas linhas 6-12. Nossa classe de exceção `SaldoContaException` possui dois construtores. Um com argumento `String` e o segundo construtor está tendo argumento `float`. Ao subclassificar qualquer exceção, você herda automaticamente alguns recursos padrão da classe `Throwable`, como:



## Declare que seu método de produção de exceção lança sua exceção personalizada

Usando a palavra-chave `throws`, podemos declarar o método que pode estar produzindo exceções. Para usar uma exceção personalizada, você deve mostrar as classes que chamam seu código que precisam planejar para esse novo tipo de exceção. Você faz isso declarando que um ou mais dos seus métodos lança a exceção. Abaixo está o código para gerenciamento de saldo da conta. A classe `GerenciaConta` possui o método `main ()` e dois métodos utilitários `addAmount ()` e `criaDepositoFixo ()`. Aqui assumimos o Balanço atual como Rs. 15000. Se o saldo da conta ultrapassar Rs. 20000, o valor acima de 20.000 será repassado para fazer um depósito fixo.

```
import java.util.Scanner;
```

```
public class GerenciaConta{
```

```
    private float saldoAtual=15000f;
```

```
    public static void main(String[] args){
```

```
        Scanner inputDevice=new Scanner(System.in);
```

```
        System.out.print("Por favor, insira um valor acrescentado no  
seu saldo: ");
```

```
        float novoValor=inputDevice.nextFloat();
```

```
        try{
```

```
float valorTotal=new
GerenciaConta().AdicionaValor(novoValor);

System.out.println("Valor total da conta =
"+valorTotal);

}catch(SaldoContaException a){

floatfdAmount=a.getSaldoConta()-20000;

System.out.println("Agora, o saldo da sua conta é
superior a 20.000, criando um FD de quantia: "+fdAmount);

new GerenciaConta().criaDepositoFixo(fdAmount);

System.out.println("Valor da conta = "+20000);

}

}

public float AdicionaValor(float montante)throws
SaldoContaException{

float total =saldoAtual+montante;

if(total>20000){

throw new SaldoContaException(total);

}

return total;

}
```

```
public void criaDepositoFixo(float valorFixo){  
  
    //Implimentationof FD creation  
  
}  
  
}
```



```
class SaldoContaException extends Exception{  
  
    private float saldoConta;  
  
    public SaldoContaException(float f){  
  
        super();  
  
        this.saldoConta=f;  
  
    }  
  
    public SaldoContaException(Stringmsg){  
  
        super(message);  
  
    }  
  
    public float getSaldoConta(){  
  
        return saldoConta;  
  
    }  
  
}
```

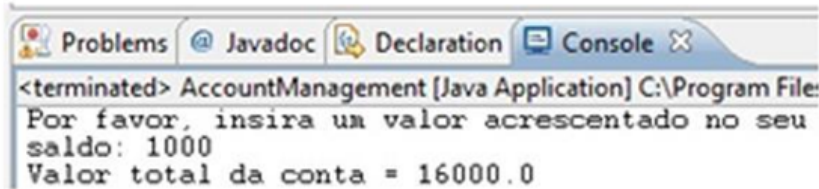


## **Encontre o (s) ponto (s) de falha no seu método, crie a exceção e envie-a usando a palavra-chave “throw”**

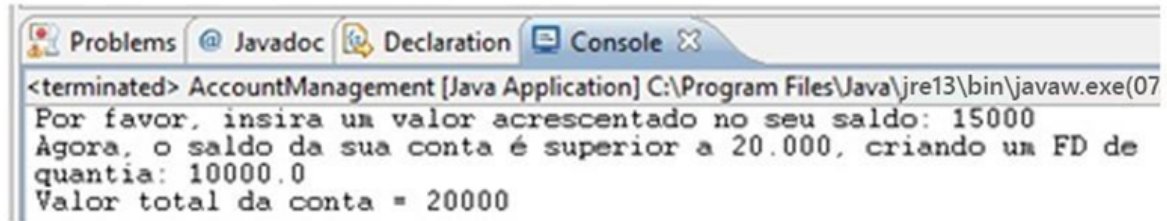
A terceira e última etapa é realmente criar o objeto e propagá-lo através do sistema. Para fazer isso, você precisa saber onde seu código precisa de processamento especial no método. Aqui, `ThrowableInstance` deve ser um objeto do tipo `Throwable` ou uma subclasse de `Throwable`. Há duas maneiras de obter um objeto `Throwable`: usando um parâmetro em uma cláusula `catch` ou criando um com o novo operador. O fluxo de execução para imediatamente após a instrução `throw`; quaisquer instruções subsequentes não são executadas. O bloco `try` mais próximo é inspecionado para verificar se há uma instrução `catch` que corresponde ao tipo de exceção. Se encontrar uma correspondência, o controle é transferido para essa instrução. Caso contrário, a próxima instrução `try` anexada será inspecionada e assim por diante.

- Mensagem de erro
- Rastreio de pilha
- Quebra de exceção

Saída do programa acima com base em vários parâmetros de entrada como abaixo,



```
<terminated> AccountManagement [Java Application] C:\Program File:
Por favor, insira um valor acrescentado no seu
saldo: 1000
Valor total da conta = 16000.0
```



```
<terminated> AccountManagement [Java Application] C:\Program Files\Java\jre13\bin\javaw.exe(07
Por favor, insira um valor acrescentado no seu saldo: 15000
Agora, o saldo da sua conta é superior a 20.000, criando um FD de
quantia: 10000.0
Valor total da conta = 20000
```

## Atividade extra

Vídeo: “Exception personalizadas”

Link: <https://www.youtube.com/watch?v=6e-iKkiH0Lg>

## Referências Bibliográficas

BARNES, D. J. KOLLING, M. **Programação orientada a objetos com java: uma introdução prática usando o bluej**. 4.ed. Pearson: 2009.

FELIX, R. (Org.). **Programação orientada a objetos**. Pearson: 2017.

MEDEIROS, L. F. de. **Banco de dados: princípios e prática**. Intersaberes: 2013;

ORACLE. Java Documentation, 2021. **Documentação oficial da plataforma Java.** Disponível em: <<https://docs.oracle.com/en/java/>>.



**Ir para exercício**