

Computação II – Orientação a Objetos

Fabio Mascarenhas - 2014.1

<http://www.dcc.ufrj.br/~fabiom/java>

Tratamento de Erros

- Até agora não nos preocupamos com erros em nosso programa, apenas assumimos que tudo sempre dá certo
 - Todas as entradas para nossos construtores e métodos estão corretas
 - Toda operação é bem sucedida
- Na prática, erros acontecem, e operações falham
- Em C, falhas eram indicadas *em banda*: uma função retornava um ou mais valores especiais para indicar um erro, ao invés de seu retorno normal

Exceções

- Em Java, o tratamento de erros pode ser *fora de banda*: um erro é sinalizado e tratado por um mecanismo diferente da chamada e retorno de método
- Esse mecanismo são as *exceções*
- Uma *exceção* é um objeto que sinaliza que uma falha aconteceu
- Uma *exceção* é *lançada* quando ocorre uma falha, o lançamento interrompe a execução do programa, e transfere o controle para um *tratador de exceções*, um trecho de código instalado por algum dos métodos que está na pilha de chamadas

Exceções - Definindo

- Exceções são objetos como qualquer outro, mas sempre são instâncias de `RuntimeException`, ou uma *subclasse* de `RuntimeException`
 - Uma classe que estende `RuntimeException`, ou uma classe que estende uma classe que estende `RuntimeException`, etc.
- Geralmente, a classe da exceção deve indicar a *categoria* da falha que aconteceu
- `ArithmeticException`, `ArrayStoreException`, `BufferOverflowException`, `CannotRedoException`, `CannotUndoException`, `ClassCastException`, `EmptyStackException`, `IndexOutOfBoundsException`, `NullPointerException`, `SecurityException`, e muitas outras

Exceções - Lançando

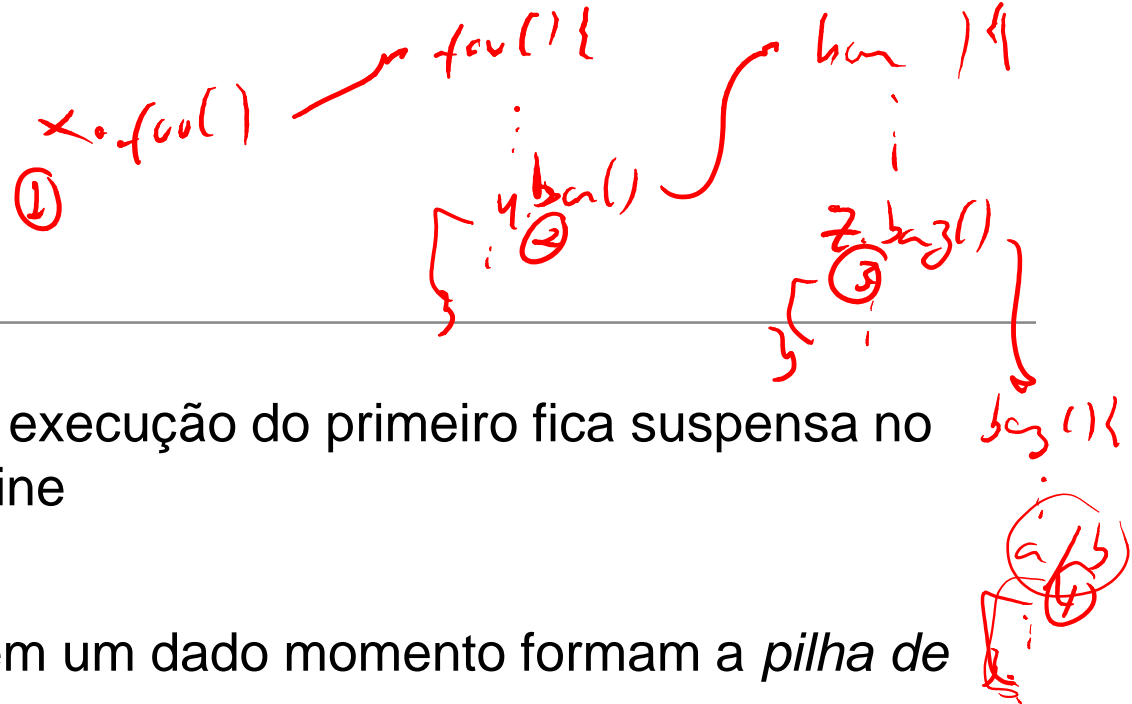
- Lançamos uma exceção (lembrando: uma *instância* de uma classe de exceção) usando a cláusula `throw`, passando para ela a exceção:

```
public int proximo() {
    if(acabou) {
        throw new IteradorVazio();
    }
    int r = proximo;
    avanca();
    return r;
}
```

- O mais comum é instanciar a exceção na própria cláusula `throw`, mas isso não é necessário

```
IteradorVazio iv = new IteradorVazio();
throw iv;
```

Pilha de Execução



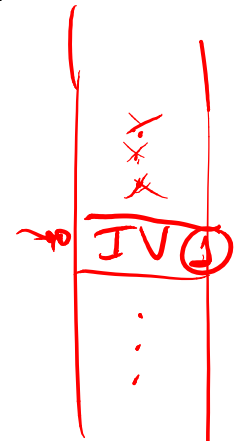
- Quando um método chama outro, a execução do primeiro fica suspensa no ponto da chamada até que ela termine
- Os métodos que estão suspensos em um dado momento formam a *pilha de execução*
- O lançamento de uma exceção interrompe o fluxo normal de execução, e faz a execução ir abandonando os métodos que estão na pilha até chegar a um que possua um *tratador* para aquela exceção
- Se não houver nenhum tratador, o programa inteiro é abortado!



Exceções - Capturando

- Instalamos um tratador de exceções usando um bloco try/catch:

```
try {  
    System.out.println(it.proximo());  
} catch(IteradorVazio iv) {  
    System.err.println("acabou!");  
}
```



- Qualquer exceção que aconteça durante a execução do bloco try que não tenha sido tratada e que seja subclasse da classe dada na cláusula catch é capturada, e então o bloco catch é executado
- Se não ocorrer nenhuma outra exceção no bloco catch, a execução prossegue normalmente com o que venha depois do try/catch

Exceções - Sutilezas

- O tratador de exceções mais *recente* na pilha de execução tem preferência, então não é garantido que o bloco `catch` seja executado caso uma exceção ocorra, já que ela pode ser tratada antes
- A declaração que segue a cláusula `catch` diz quais exceções queremos tratar: uma instância da classe dada, ou de uma classe que estende ela, ou de uma classe que estende uma classe que estende ela, etc.
- Se o tipo da exceção não bate, é como se o tratador não existisse, e continuamos a busca na pilha de execução
- Se uma exceção ocorre dentro do bloco `catch`, ela não pode ser tratada pelo próprio bloco, a execução dele é interrompida, e outra busca começa

Múltiplos catch

- Um bloco try/catch pode ter múltiplos blocos catch, cada um capturando um tipo diferente de exceção
- Cada catch introduz um tratador para o seu tipo:

```
try {  
    System.out.println(it.proximo());  
} catch(IteradorVazio iv) {  
    System.err.println("acabou!");  
} catch(RuntimeException re) {  
    System.err.println("outro problema: " + re);  
}
```

Exceções checadas

- As exceções derivadas de RuntimeException são chamadas de *não-checadas*
- Um método sempre pode ter como resultado exceções não-checadas, e o compilador não tem como informar isso
- A classe Exception e suas subclasses dão uma segunda categoria de exceções, chamadas de checadas
- Se a chamada a um método pode resultar no lançamento de uma exceção checada, isso fica explícito na sua assinatura

IO Exception

A cláusula throws

- Se um método lança exceções checadas, direta ou indiretamente, ele deve ter uma cláusula throws na sua assinatura, seguindo a lista de parâmetros
- Essa cláusula lista as exceções checadas que ele pode lançar; listar uma classe inclui todas as suas subclasses
- Mesmo um método que não tem uma cláusula throw pode precisar de uma cláusula throws, basta que ele chame um método que possa lançar exceções checadas (tenha uma cláusula throws na assinatura)
- A cláusula throws é uma indicação para o programador de quais falhas ele pode esperar daquele método

Exemplo de throws

- Se transformamos `IteradorVazio` em uma exceção checada, ela precisa fazer parte da assinatura de `próximo`:

```
public int proximo() throws IteradorVazio {  
    if(acabou) {  
        throw new IteradorVazio();  
    }  
    int r = proximo;  
    avanca();  
    return r;  
}
```

- Exceções checadas são parte da assinatura do método, então essa mudança tem que ir para a interface `Iterador` também, e todas as classes que implementam `Iterador` precisam dela em `próximo`, *mesmo que não lancem a exceção `IteradorVazio`*

Exceções Checadas sem throws

- Se um método chama outro que pode lançar uma exceção checada, mas a trata, então ele não precisa declará-la em uma cláusula throws
- Assumindo que:
 - O bloco catch não relance a exceção, ou chame outro método que possa lança-la
 - A exceção não é parte da assinatura do método, caso ele tenha sido herdado de uma interface ou outra classe

throws Exception

- Como toda classe declarada na cláusula throws inclui todas as suas subclasses, uma declaração throws Exception é um modo de declarar que aquele método pode lançar *qualquer exceção checada*
- Isso evita declarar as exceções uma a uma na cláusula throws, mas passa para qualquer chamador do método a obrigação de tratar Exception, ou listá-la também na cláusula throws
- O throws Exception pode acabar “contaminando” todo código do programa!
[Use com cuidado.](#)

Código de Finalização

- Algumas vezes, temos código que precisa ser executado mesmo que uma exceção aconteça, independente de qual seja a exceção
- Uma maneira de fazer isso seria duplicando esse código, e usando um tratador para Exception:

```
catch  
try {  
    try {  
        // código normal vsc  
    } catch(...) {  
        // outros tratadores de exceção...  
    }  
    // finalização fcdh  
} catch(Exception e) {  
    // finalização fcdh  
    throw e;  
}
```

- Existe um jeito melhor: a cláusula finally

Usando finally

- Podemos terminar uma cláusula try/catch com um bloco finally:

```
che  
try {  
    // código normal use  
} catch(...) {  
    // outros tratadores de exceção...  
} finally {  
    // finalização feche  
}
```

- O bloco finally sempre é executado; se não ocorrer uma exceção, ele é executado após o bloco try, se ocorrer alguma exceção, ele é executado antes de abandonar o método atual