Exceções

Os alunos devem modelar o seguinte sistema (~30min). Em seguida irão se juntar em grupos de 4 pessoas para discutir os modelos criados e sugerir um novo modelo (~15min).

Sistema Calculadora

- Como usuário, quero colocar operação e operandos numa linha de comando e receber o resultado (4 operações básicas entre doubles)
- Como usuário, quero sair do sistema ao digitar "fim"
- Como usuário, quero ser sinalizado quando uma operação for inválida (cenários: operação inexistente ou divisão por zero)
- Como administrador, quero o histórico das operações feitas pelos usuários em um arquivo, o log não pode falhar (uso de uma classe auxiliar Logger que já existe no sistema. Método: info(String msg))
- Como arquiteto, quero que cada operação seja representada por uma classe.

Exemplo de Modelo Proposto

```
Main -> Facade
         \----> Operações ----> Operação ----> Multiplicação
         \----> Logger            |----------> Divisão
                                  \----------> Soma
```

Exemplo de Implementação

Main

```
Scanner sc = new Scanner(System.in);
String op = sc.next();
if (op.toLowerCase() == "fim" && op != null) {
    System.exit(0);
}
double a =sc.nextDouble();
double b =sc.nextDouble();
System.out.println(facade.execute(op, a, b));
```

Facade

```
String execute(op, a, b) {
   logger.log(op, a, b);
   return operacoes.execute(op, a, b);
}
```

Operacoes

```
String execute(op, a, b) {
	If (op.length() != 1 && "+-/*".indexOf(op) == -1) {
		return "Error";
	}
	return new String(realExecute(op, a, b));
}

Double realExecute(op, a, b) {
	switch (op) {
		case "+":
			return this.plus.execute(a, b);
		case "-":
			return this.minus.execute(a, b);
		case "*":
			return this.times.execute(a, b);
		case "/":
			return this.divide.execute(a, b);
		default:
			return null;
	}
}
```

Divisao

```
Double execute(a, b) {
	return a / b;
}
```

Comentários

- Divisão por zero, como tratar?
  - Podemos tratar no Main.
    - Mas nesse caso a lógica de exceção fica distante do responsável pela lógica de negócio (Divisao) -> :(
  - Podemos fazer com que Divisao retorne null quando segundo operando for zero
    - Se mais operações precisarem retornar erro, não há como diferenciar a natureza do erro
    - Talvez seja necessário tratar valor nulo nos elementos que fazem uso da chamada (exemplo: se a classe Operacoes fizer uma transformação do resultado, precisa tratar o caso nulo)
  - Solução:
    - Criar uma exceção que herda RuntimeException
    - Lançar essa exceção

        - Capturar (catch) a exceção no lugar adequado
- Operação inexistente, como tratar?
    - Operacoes pode lançar erro OU
    - Pode ter um método de validação da operação
        - É melhor usar um método de validação se o fluxo for possível e comum, mesmo que seja um fluxo de "erro".
- Logger.info throws IOException
    - E se o método lançar uma exceção, i.e. na situação de checked exceptions?
        - Vc pode propagar o checkedException
            - Pode poluir todos os métodos da cadeia de elementos
            - Mas força que seja tratada

<u>Teoria</u>

Exceção

- Encapsula as situações de erros de um sistema
- Permite tratar o fluxo de exceção
- Exceções podem ser aninhadas (chained exceptions), você pode capturar uma exceção, jogá-la em outra exceção de seu controle. No sistema calculadora, uma IOException poderia ser encapsulada numa CalculadoraException.
- Existem diferentes tipos de exceção. Elas herdam de Throwable (interface) e podem ser:

    Unchecked Exceptions (RuntimeException)
    - Herdam de RuntimeException
    - Checadas apenas em tempo de execução
    - Situações em que o programador cometeu um erro ou você tem controle do fluxo de execução

    Checked Exceptions (Exception)
    - Herdam de Exception
    - Obrigam o tratamento
    - Situações em que você não tem controle do erro que pode acontecer (exemplo: manipulação de arquivos ou chamadas de rede) ou quer forçar o tratamento de exceções (em geral, nas suas classes de borda, quando implementar uma biblioteca ou API que os outros usarão)

    Error
    - Situação completamente anômala em quem a própria JVM não se comporta como esperado (exemplo mais comum: falta de memória).

Callstack

- Quando se faz uma chamada, esta "chamada" (call) é "empilhada" (stack) de forma que o sistema sabe, ao retornar de uma chamada (desempilhar da stack), a chamada que deve continuar executando

Stacktrace

- É a descrição das chamadas que estão na pilha de execução
- É comum visualizar a stacktrace no erro: permite entender o que aconteceu e que chamadas foram executadas até chegar ao erro.