6ª Aula Prática de PLC: Conceitos de Orientação a Objetos em Java

Roteiro

- Mecanismo de Exceções e Controle de Erros
- Captura e Tratamento de Exceções
- Hierarquia de Exceções
- Criação de Exceções

Mecanismo de Exceções e Controle de Erros

A robustez é a capacidade do sistema funcionar mesmo em condições adversas.

Como boa parte dos sistemas atuais são implementados por seres humanos, esses sistemas são, portanto, passíveis de falha. Mesmo procedimentos de testes rigorosos conseguem, quando muito, mitigar, mas não eliminar completamente a probabilidade de que existam falhas no código final.

Mecanismo de Exceções e Controle de Erros

Exemplo de um sistema com falha de robustez:

```
public class Conta {
	private int numero, agencia;
	private double saldo, limite;
	private String nomeDono;
	//...
	public void sacar(double valor) {
		this.saldo = this.saldo - valor;
	}
}
```

Como evitar saques fora do o limite?

Mecanismo de Exceções e Controle de Erros

Uma solução seria adicionar uma cláusula condicional.

```
public void sacar(double valor) {
        if (this.saldo + this.limite >= valor){
                this.saldo = this.saldo - valor;
        }
}
```

Entretanto como o usuário saberia se a operação foi realizada ou não?

Mecanismo de Exceções e Controle de Erros

O sistema também poderia mostrar uma mensagem de erro informando ao usuário quando a operação não foi realizada.

```
public void sacar(double valor) {
        if (this.saldo + this.limite >= valor){
                this.saldo = this.saldo - valor;
        } else {
                System.out.println(“Saque excede o limite.”);
        }
}
```

Quem chamou o método sacar não saberá quando o saque não ocorreu.

Mecanismo de Exceções e Controle de Erros

Uma opção seria retornar um valor booleano indicando se o saque ocorreu ou não.

```
public boolean sacar(double valor) {
        boolean saque = false;
        if (this.saldo + this.limite >= valor){
                this.saldo = this.saldo - valor;
                saque = true;
        } else {
                System.out.println(“Saque excede o limite.”);
        }
        return saque;
}
```

E se o método já possuísse algum retorno?

Mecanismo de Exceções e Controle de Erros

Quem invocou o método sacar teria que testar o retorno do método, mas não é obrigado a fazer isso. Esquecer de testar o retorno desse método poderia ocasionar consequências drásticas como criar máquinas de autoatendimento que liberassem a quantia desejada de dinheiro.

Mesmo invocando o método e tratando o retorno de maneira correta, o que deveria ser feito para sinalizar quando o usuário passou um valor negativo?

Mecanismo de Exceções e Controle de Erros

No exemplo anterior, uma solução seria alterar o retorno de boolean para int e retornar o código do erro que ocorreu. Mas isso é considerado uma má prática de programação (Números Mágicos).

Por esses e outros motivos, é recomendável utilizar um mecanismo diferente em Java para tratar essas exceções.

Mecanismo de Exceções e Controle de Erros

Uma exceção representa uma situação que normalmente não ocorre e representa algo de estranho ou inesperado no sistema.

Java facilita o gerenciamento de tais erros, desviando o controle para blocos especiais que são executados quando os erros ocorrem.

Algumas exceções:
- Erros aritméticos (ArithmeticException);
- Estouro de limite de array (ArrayIndexOutOfBoundsException);
- Arquivo não encontrado (FileNotFoundException);
- Instância de objetos nulos (NullPointerException);
- Erros nos dispositivos/arquivos de entrada e saída (IOException).

Captura e Tratamento de Exceções

Uma exceção em Java é uma instância da classe Throwable ou de uma de suas extensões. Um objeto assim pode ser instanciado por um programa em execução através do comando throw no código.

Quando uma exceção é disparada, ela pode ser capturada por uma cláusula catch de um comando try. Se ela não for capturada, então ela vai ser novamente disparada pelo método que chamou o método atual, a menos que seja disparada no método main, o que causa o término do programa.

Comandos que podem disparar exceções devem ser colocados dentro de um comando try, ou estas devem ser declaradas no cabeçalho do método através do comando throws.

Captura e Tratamento de Exceções

Exemplo de um método que lança uma exceção:

```
public double raizQuadrada(double x) throws IllegalArgumentException {
        if (x<0) throw (new IllegalArgumentException(  ));
        return Math.sqrt(x);
}
```

Exemplo da execução do método raizQuadrada:

```
try {
        raizQuadrada(-25.0);
} catch (IllegalArgumentException e){
        e.printStackTrace( );
}
```

Captura e Tratamento de Exceções

A execução do bloco try termina assim que uma exceção é lançada. Logo, o primeiro catch de um supertipo da exceção é executado (se existir) e o fluxo de controle passa para o código seguinte ao último catch. Caso contrário, ela será novamente disparada para o método que a chamou.

Após os catchs, pode vir um bloco finally que sempre será executado, independente se alguma exceção foi lançada, capturada ou relançada.

Captura e Tratamento de Exceções

Exemplo do uso do bloco finally:

```
public void alterarFrase( ) {
        String frase = null;
        String novaFrase = null;
        try {
                novaFrase = frase.toUpperCase( );
        } catch (NullPointerException e) {
                System.out.println("A frase inicial esta nula!");
                frase = "Frase vazia";
        } finally {
                novaFrase = frase.toUpperCase( );
        }
        System.out.println("Frase antiga: " + frase);
        System.out.println("Frase nova: " + novaFrase);
}
```

Exercícios Resolvidos

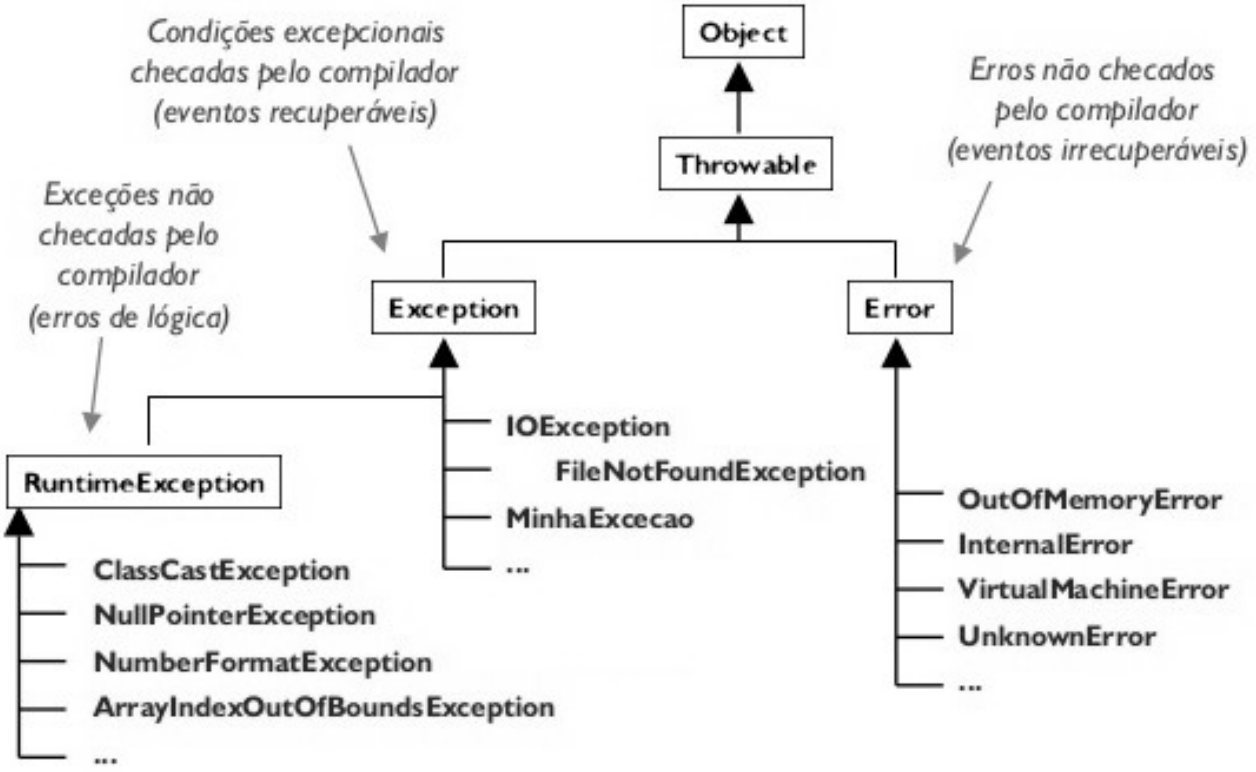
1. Sejam os trechos de código abaixo, onde method1 e method2 representam métodos genéricos e ExceptionA e ExceptionB duas exceções diferentes:

```
try {
        method1( );
} catch (ExceptionA e) {
        e.printStackTrace( );
} catch (ExceptionB e) {
        e.printStackTrace( );
}
method2( );
```

```
try {
        method1( );
} catch (ExceptionA e) {
        e.printStackTrace( );
} catch (ExceptionB e) {
        e.printStackTrace( );
} finally {
        method2( );
}
```

Compare-os e indique em que situações eles funcionam da mesma forma e em que situações eles teriam funcionamentos diferentes.

Hierarquia de Exceções



Criação de Exceções

Um programa pode ter um problema que não esteja descrito adequadamente em nenhuma das classes de exceções. Nesse caso, talvez seja mais apropriado criar um novo tipo de exceção. Uma nova exceção pode ser criada como um subtipo da classe Exception ou de suas derivadas. Por exemplo:

```
public class ValorInvalidoException extends Exception {
        public ValorInvalidoException(double valor) {
                super(“Valor invalido: ” + valor);
        }
}
```

Exercícios Resolvidos

2. Crie um método obterCPF que leia do teclado uma String digitada pelo usuário representando um número de inscrição no Cadastro de Pessoas Físicas (CPF) e retorne o número passado pelo teclado caso ele seja valido.

Crie duas classes de exceções: DigitosVerificadoresInvalidosException (indica que os dígitos verificadores do CPF estão incorretos) e FormatoCPFInvalidoException (indica que o formato do CPF não está correto, ou seja, está diferente de xxx.xxx.xxx-xx). Inclua, também, na sua assinatura de obterCPF essas exceções que devem ser lançadas quando necessário.

1. O código abaixo lança uma exceção (propositalmente) e interrompe sua execução. Utilizando o tratamento de exceções, corrija a classe com o objetivo de não parar sua execução.

```
public class TesteException {

    public static void main(String[] args) {
        System.out.println("inicio do main");
        metodo1();
        System.out.println("fim do main");
    }

    static void metodo1() {
        System.out.println("inicio do metodo1");
        metodo2();
        System.out.println("fim do metodo1");
    }

    static void metodo2() {
        System.out.println("inicio do metodo2");
        int[] array = new int[10];
        for (int i = 0; i <= 15; i++) {
            array[i] = i;
            System.out.println(i);
        }
        System.out.println("fim do metodo2");
    }
}
```

b) Nesta questão você deve identificar as partes problemáticas do código e reescrevê-lo utilizando tratamento de exceções. Ou seja, devem ser identificadas todas as exceções que podem ser levantadas e, para cada uma, deve ser dado o tratamento adequado que, nesse exercício, significa alertar o usuário quanto ao problema. Entretanto, nesse programa a leitura dos valores deve ser feita, mesmo que para isso o usuário tenha que tentar informar várias vezes os valores na mesma execução do programa.

```
public class Questao2 {

    public static void main(String[] args) {
        Scanner teclado = new Scanner(System.in);
        System.out.println("Eu sei dividir!");
        System.out.print("Informe o primeiro valor: ");
        int x = teclado.nextInt();
        System.out.print("Informe o segundo valor: ");
        int y = teclado.nextInt();
        double r = (x / y);
        System.out.println("O resultado da soma é " + r);
    }
}
```

c) Suponha que o método "saca" da classe Conta vai ser rescrito de forma a lançar uma exceção criada por você, cuja classe é ContaExcecao (extends Exception). A exceção é lançada sempre que o saldo da conta for inferior ao valor sacado. Implemente a classe ContaExcecao. Implemente o método saca que lança a exceção. E rescreva o código da caixa com o devido tratamento da exceção.

```
Conta minhaConta = new  Conta();
minhaConta.deposita(100 );
minhaConta.setLimite(100 );
minhaConta.saca(1000 );
```

d) Retomando o exercício anterior, suponha que quando lançada a exceção ContaExcecao, através do objeto exceção instanciado, seja possível recuperar o saldo da pessoa. Como você implementaria isso? Mostre tudo que deve ser modificado/acrescentado no exercício para que isto funcione.

2. A conjectura de Collatz ou conjectura do (3n + 1) pressupõe que para um número positivo inteiro qualquer n sempre pode-se chegar no número 1 seguindo dois critérios: se n for um número par, deve-se dividi-lo por 2, se n for um número ímpar, deve-se multiplicá-lo por 3 e, em seguida, somar o resultado ao número 1 para obter (3n + 1). O processo deve ser repetido por tempo indeterminado, até que, eventualmente, se chegue ao número 1.

Escolhendo-se o número 22 como número inicial, por exemplo, a seguinte sequência de 16 números é gerada até chegar na condição de parada do algoritmo: 22 -> 11 -> 34 -> 17 -> 52 -> 26 -> 13 -> 40 -> 20 -> 10 -> 5 -> 16 -> 8 -> 4 -> 2 -> 1.
Dado um intervalo numérico, qual seria a maior sequência gerada por um número pertencente ao mesmo?

a) Crie uma classe abstrata ou uma interface chamada Collatz (o que você achar mais adequado) com os seguintes métodos e/ou assinaturas:

public int calcularTamanhoSequencia(int numeroCollatz);
public int tamanhoMaiorSequencia(int numeroCollatzInicial, int numeroCollatzFinal);

No caso de uma classe abstrata, os métodos podem ser definidos como abstratos também. Justifique sua escolha.

b) Crie as classes CollatzRecursivo, CollatzIterativo e CollatzOtimizado de forma que todas essas classes herdem da classe abstrata ou implementem a interface criada. Cada uma dessas classes calculará o tamanho de uma sequência de uma forma diferente. A primeira calculará de forma recursiva, chamando o próprio método calcularTamanhoSequencia sucessivas vezes, a segunda calculará de forma iterativa (chamando o método uma única vez) e atualizando o seu estado em uma estrutura de repetição (laço).

A terceira calculará de forma recursiva utilizando programação dinâmica, ou seja, o tamanho de uma determinada sequência só deverá ser calculado uma única vez. Salve em um vetor (ou em uma estrutura de dados de sua preferência) de tamanho 100000000 as soluções encontradas por esta última classe, de forma, que sempre possam ser reaproveitadas.

O método tamanhoMaiorSequencia de cada classe deve, obrigatoriamente, chamar o respectivo método calcularTamanhoSequencia para cada número do intervalo e retornar apenas o tamanho da maior sequência do intervalo. O número que a gerou não precisa ser identificado.

• Os exercícios práticos devem ser realizados individualmente e enviados por e-mail com o assunto "[IF686EC] EXERCÍCIOS PRÁTICOS 06" para monitoria-if686-ec-l@cin.ufpe.br até as 23:59 da hoje (13.06.2017).

• As resoluções dos exercícios devem estar em arquivos compactados diferentes, um arquivo por exercício, com os nomes no formato "Q[número do exercício].zip". Onde cada arquivo desse poderá conter vários classes de Java no formato "[nome da classe].java".