MAPA - Tópicos em Computação 2

Aluno: Guionardo Furlan - RA 1650466-5

Solução 1 - Código com validação manual

programa\Data.java

```
package programa;

import java.time.LocalDate;

public class Data {

    protected int dia = 0;
    protected int mes = 0;
    protected int ano = 0;

    public Data(final int dia, final int mes, final int ano) throws
Exception {
        this.validarData(dia, mes, ano);
        this.dia = dia;
        this.mes = mes;
        this.ano = ano;
    }

    public void validarData(final int dia, final int mes, final int ano)
throws Exception {
        if (!(0 < dia && dia < 32)) {
            throw new Exception(String.format("Dia deve ser entre 1 e 31:
%d", dia));
        }
        if (!(0 < mes && mes < 13)) {
            throw new Exception(String.format("Mês deve ser entre 1 e 12:
%d", mes));
        }
        if (ano > LocalDate.now().getYear()) {
            throw new Exception(String.format("Ano deve ser anterior ao ano
atual: %d", ano));
        }

        boolean diaCerto = false;

        if (mes == 4 || mes == 6 || mes == 9 || mes == 11) {
            diaCerto = dia <= 30;
        } else if (mes == 1 || mes == 3 || mes == 5 || mes == 7 || mes == 8
|| mes == 10 || mes == 12) {
            diaCerto = dia <= 31;
        } else if (mes == 2) {
            diaCerto = (ano % 4 == 0) ? dia <= 29 : dia <= 28;
```

```
        }
        if (!diaCerto) {
            throw new Exception(String.format("Dia inválido: %d", dia));
        }
    }

    public float calculoIdade() {
        return this.dia == 0 ? 0
                : ((float) LocalDate.now().getYear() - this.ano)
                        + (float) (LocalDate.now().getMonthValue() -
this.mes) / 12;

    }

}
```

Solução 2 - Código com classe nativa java.time.LocalDate

programa\Data2.java

```
package programa;

import java.time.LocalDate;
import java.time.Period;

public class Data2 {

    private LocalDate data;

    public Data2(final int dia, final int mes, final int ano) {
        this.data = LocalDate.of(ano, mes, dia);
    }

    public float calculoIdade() {
        if (data == null) {
            return 0;
        }
        Period periodo = Period.between(this.data, LocalDate.now());
        return (float) periodo.getYears() + periodo.getMonths() / (float)
12;
    }

}
```

Teste

Main.java

```
import programa.Data;
import programa.Data2;

public class Main {

    public static void main(String[] args) {
        int[][] dados = { { 5, 7, 1977 }, { 5, 2, 1977 }, { 30, 12, 2008 },
{ 7, 2, 2014 }, { 13, 13, 2014 },
                { 65, 12, 1977 } };

        int d, m, a;
        for (int n = 0; n < dados.length; n++) {
            d = dados[n][0];
            m = dados[n][1];
            a = dados[n][2];
            System.out.printf("\nTestando data #%d: %d/%d/%d\n", n + 1, d,
m, a);
            try {
                System.out.println("Data:");
                Data d1 = new Data(d, m, a);
                System.out.printf("Idade: %f\n", d1.calculoIdade());
            } catch (Exception exc) {
                System.out.printf("ERRO: %s\n\n", exc.getMessage());
            }
            try {
                System.out.println("Data2:");
                Data2 d2 = new Data2(d, m, a);
                System.out.printf("Idade: %f\n", d2.calculoIdade());
            } catch (Exception exc) {
                System.out.printf("ERRO: %s\n\n", exc.getMessage());
            }

        }
    }
}
```

Saída do programa de testes

```
Testando data #1: 5/7/1977
Data:
Idade: 42,833332
Data2:
Idade: 42,833332

Testando data #2: 5/2/1977
Data:
Idade: 43,250000
Data2:
Idade: 43,250000
```

```
Testando data #3: 30/12/2008
Data:
Idade: 11,416667
Data2:
Idade: 11,333333

Testando data #4: 7/2/2014
Data:
Idade: 6,250000
Data2:
Idade: 6,250000

Testando data #5: 13/13/2014
Data:
ERRO: Mês deve ser entre 1 e 12: 13

Data2:
ERRO: Invalid value for MonthOfYear (valid values 1 - 12): 13


Testando data #6: 65/12/1977
Data:
ERRO: Dia deve ser entre 1 e 31: 65

Data2:
ERRO: Invalid value for DayOfMonth (valid values 1 - 28/31): 65
```

Observações:

A classe Data2 é mais eficiente, pois já tem sua validação nativa. Além disso, podemos verificar que houve uma diferença no cálculo de idade (#3), pois a classe *java.time.Period* tem uma resolução de cálculo em milissegundos, enquanto na classe Data criada para este projeto, a resolução está limitada em meses.

Efetuei alteração no comportamento das classes, para que os problemas com validação dos dados gerassem exceção. A justificativa é que o princípio SOLID, mais especificamente o "Single Responsability" nos diz que uma classe deve fazer bem uma coisa.

No nosso caso, essa responsabilidade única seria armazenar uma data e retornar uma idade.

Nossa classe não deve ser responsável por imprimir mensagens. Portanto, nos casos de erro, uma exceção é elevada e deve ser tratada por quem chamou a classe.

Por último, vemos a diferença no tamanho do código produzido. Enquanto nossa classe Data tem 50 linhas de código, nossa classe Data2 tem 22 e é mais eficiente.