Lista de Exercícios 11
Tratamento de Exceções

Exercício 01) Java possui dois tipos de tratamentos de exceções. Um sistema rigoroso que obriga que sejam descritos na assinatura dos métodos todas as exceções do tipo `Throwable`, com exceção das exceções do tipo `RuntimeException`, que podem ser lançadas por ele através da cláusula `throws`. Ainda um outro sistema para erros gerados em tempo de execução (`RuntimeException`), que não exigem tal restrição de forma que exceções desse tipo possam ser lançadas livremente. Discuta as vantagens e desvantagens de cada abordagem.

Exercício 02) Considere a classe Staff implementada em Java a seguir.

```
01: import java.util.*;
02: class Staff {
03:   private Map<String,Double> payroll = new HashMap<String,Double>();
04:   public double getSalary(String name) {
05:     if (payroll.containsKey(name)) {
06:       return payroll.get(name);
07:     } else {
08:       return 0.0;
09:     }
10:   }
11:   public void addEmployee(String name, double salary) {
12:     payroll.put(name, salary);
13:   }
14:   public void raiseSalary(String name, double increment) {
15:     double newSalary = payroll.get(name) + increment;
16:     payroll.put(name, newSalary);
17:   }
18: }
```

a) O método `getSalary` utiliza o valor especial `0.0` como o salário de um empregado inexistente. Qual a desvantagem desta forma de tratamento?

b) Crie a classe `NonExistentEmployee` como uma exceção verificada para tratar a situação de uma busca sobre um empregado inexistente.

c) Modifique o método `getSalary` para disparar uma exceção do tipo `NonExistentEmployee` caso o nome solicitado não possua uma entrada no banco de dados.

d) Agora, crie uma outra classe de exceção, desta vez não verificada, chamada `NegativeSalary` para lidar com tentativas de criar empregados com salários negativos. Instâncias desta classe possuem dois atributos extras: `name` e `salary` que representam, respectivamente, o nome do empregado e o salário resultante que deu origem à exceção.

e) Considere o código a seguir com a função `updateEmployees`. Reescreva este método para que ele implemente o tratamento de erro. Cuide para que sua nova implementação não termine o programa logo que uma exceção for disparada. Isto é, tendo inserido um nome inválido, o usuário poderá ter uma nova chance de informar um outro nome. O mesmo vale para o salário negativo.

```java
public void updateEmployees(Staff s) {
  Scanner input = new Scanner(System.in);
  for (;;) {
    System.out.print("Entre com o nome: ");
    String name = input.nextLine().trim();
    if (name.isEmpty())
      break;

    System.out.print("Entre com a promoção salarial: ");
    double increment = input.nextDouble();
    s.raiseSalary(name, increment);
    input.nextLine(); // clear buffer.
  }
  input.close();
}
```

Exercício 03) Considere os seguintes códigos em Java que recebem um valor pela linha de comando (variável `x`). Mostre o que será impresso caso seja executado com as seguintes entradas `0`, `1`, `2`, `3` e `4`, uma de cada vez.

a)

```java
class Level1Exception extends Exception {}
class Level2Exception extends Level1Exception {}
class Level3Exception extends Level2Exception {}
class Test {
  public static void main(String args[]) {
    int a, b, c, d, f, g, x;
    a = b = c = d = f = g = 0;
    x = Integer.parseInt(args[0]);
    try {
      try {
        switch (x) {
          case 0:
            throw new Level1Exception();
          case 1:
           throw new Level2Exception();
          case 2:
            throw new Level3Exception();
          case 3:
           throw new Exception();
        }
        a++;
      } catch (Level2Exception e) {
        b++;
      } finally {
        c++;
      }
    } catch (Level1Exception e) {
      d++;
    } catch (Exception e) {
      f++;
    } finally {
      g++;
    }
    System.out.print(a + "," + b + "," + c +
        "," + d + "," + f + "," + g);
  }
}
```

b)

```java
import java.io.EOFException;
import java.io.IOException;

class Test {
  public static void main(String[] args) {
    int x = Integer.parseInt(args[0]);
    try {
      primeiro(x);
      System.out.println("Após primeiro");
    } catch (IllegalArgumentException e) {
      System.out.println("trata primeiro");
    }
    System.out.println("saiu primeiro");
  }
  static void primeiro(int x)
      throws IllegalArgumentException {
    try {
      segundo(x);
      System.out.println("depois segundo");
    } catch (IOException e) {
      System.out.println("trata segundo");
    }
    System.out.println("saiu segundo");
  }
  static void segundo(int x)
        throws IllegalArgumentException,
               IOException {
    try {
      switch (x) {
        case 0:
          throw new IllegalArgumentException();
        case 1:
          throw new IOException();
        case 2:
          throw new EOFException();
      }
      System.out.println("depois switch");
    } catch (EOFException e) {
      System.out.println("trata terceiro");
    }
    System.out.println("saiu terceiro");
  }
}
```

Exercício 04) Considere o seguinte esqueleto de programa em C++ a seguir. Indique, para cada possível exceção disparada, a linha onde ela será tratada.

```
01: class A {
02:   private:
03:     int j;
04:     float g;
05:     B b;
06:   public:
07:     void f2() {
08:       // ...
09:       try {
10:         // ...
11:         try {
12:           // ...
13:           b.f1();
14:           // ...
15:           if (/* ... */)
16:             throw j;
17:           // ...
18:           if (/* ... */)
19:             throw g;
20:           // ...
21:         } catch(int) {
22:           // ...
23:         }
24:         //...
25:       } catch (float) {
26:         // ...
27:       }
28:       // ...
29:     }
30: };
```

```
31: class B {
32:   private:
33:     int k;
34:     float f;
35:   public:
36:     void f1() {
37:       // ...
38:       try {
39:         // ...
40:         if (/* ... */)
41:           throw k;
42:         // ...
43:         if (/* ... */)
44:           throw f;
45:         // ...
46:       } catch (float) {
47:         // ...
48:       }
49:       // ...
50:     }
51: };
52:
53: void main() {
54:   A a;
55:   // ...
56:   a.f2();
57:   // ...
58: }
```

Exercício 05) Considere o seguinte trecho de código em Java.

```
01: public static void salvar(String arquivo, String dados) {
02:   try {
03:     OutputStream out = new FileOutputStream(arquivo);
04:     out.write(dados.getBytes());
05:     out.close();
06:   } catch (Exception e) {
07:     e.printStackTrace();
08:   }
09: }
```

O código acima pode gerar exceções em três pontos: ao abrir o arquivo (linha 3), ao escrever os dados (linha 4) ou ao fechar o arquivo (linha 5). Com base nisso responda.

a) Qual o problema deste código? Quando ele pode acontecer?
b) Reescreva o código, sem `AutoCloseable`, para corrigir o problema.
c) Agora reescreva o código usando o novo formato com `AutoCloseable`.

Exercício 06) Considere o problema de conversão de temperaturas entre Celsius e Fahrenheit em Java.

a) Crie duas exceções, `TemperatureException` que herda de `Exception` e `FahrenheitException` que herda de `TemperatureException`.

b) Crie uma classe utilitária `Utils` com um método estático `toCelsius()` para converter temperaturas em graus Fahrenheit para graus Celsius, ambas representadas por um número em ponto-flutuante (`double`). Porém, caso o valor a ser convertido seja menor que zero absoluto (-459,67°F) deve-se lançar a exceção `FahrenheitException`. A fórmula para conversão é dada a seguir.

$$C = \frac{5(F - 32)}{9}$$

c) Considere o trecho de código a seguir. Caso o código não possua problema, mostre o resultado de sua execução, caso contrário, mostre o porquê do problema.

```
01: try {
02:   double c = Utils.toCelsius(-500.0);
03:   System.out.println(c);
04: } catch (TemperatureException e) {
05:   System.out.println("Erro 1");
06: } catch (FahrenheitException e) {
07:   System.out.println("Erro 2");
08: }
```