Lista de Linguagens de Programação – 18

Nome: _____ Matrícula: _____

1. Considere a classe `Staff`, implementada em Python e mostrada logo abaixo:

```
class Staff:
  payroll = {}
  def getSalary(self, name):
    if self.payroll.has_key(name):
      return self.payroll[name]
    else:
      return 0.0
  def addEmp(self, name, salary):
    self.payroll[name] = salary
  def raiseSalary(self, name, salary):
    self.payroll[name] = self.payroll[name] + salary
```

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

   (b) Crie uma classe de exceções `NonExistentEmployee` para tratar a situação excepcional 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, chamada `NegativeSalary`, para lidar com tentativas de criar empregados com salarário negativo. Instâncias desta classe devem possuir dois atributos, `name` e `salary`, que representam, respectivamente, o nome do empregado e o salário que deram origem à exceção.

(e) Modifique o método `addEmp` para disparar uma exceção do tipo `NegativeSalary`, caso seja feita a tentativa de adicionar um empregado cujo salário seja negativo.

(f) Modifique o método `raiseSalary` para tratar tanto erros de empregado não existente quanto erros de salário negativo.

(g) Considere o método `readEmployees` abaixo. 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 deveria ter uma nova chance de informar um nome. O mesmo vale para o salário negativo.

```
def readEmployees(s):
  name = raw_input("Please, enter a name (Press RETURN to finish) ")
  while name != '':
    salary = float(raw_input("Please, enter the salary: "))
    s.raiseSalary(name, salary)
    name = raw_input("Please, enter a name (Press RETURN to finish) ")
```

2. Cada uma das questões a seguir diz respeito ao programa abaixo:

```
class Wrapper<E> {
  private E o;
  Wraper() {this.o = null;}
  Wrapper(E o) {this.o = o;}
  E get() {return o;}
}
public class Test {
  public static void main(String a[]) {
    Wrapper<String> w = new Wrapper<String>();
    System.out.println(w.get().toString());
  }
}
```

(a) Que tipo de erro será causado pelo programa acima?

(b) Defina uma classe de exceção para representar este erro.

(c) Modifique o método `get` para disparar a exceção criada anteriormente.

(d) Modifique o método `main` para tratar esta exceção.

3. Ao contrário de Python, a linguagem Java provê exceções verificáveis estaticamente. Estas exceções precisam ser tratadas explicitamente pelo desenvolvedor. Por exemplo, o método Java abaixo não compilaria caso removêssemos as cláusulas de tratamento de erro:

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
public static void fileReader(final String fileName) {
  try {
    Scanner s = new Scanner(new File(fileName));
    // ... read the file here ...
  } catch (FileNotFoundException e) {
    e.printStackTrace();
  }
}
```

   (a) Quais exceções são verificadas estaticamente? Isto é, como podemos declarar exceções assim?

   (b) Cite uma vantagem desta abordagem adotada por Java

   (c) Existem exceções, como por exemplo `ClassCastException`, que não são verificadas estaticamente. Por que Java provê exceções assim?

   (d) Java é a única linguagem muito popular que adota exceções verificáveis estaticamente. Obviamente existem desvantagens nesta abordagem, pois linguagens posteriores à Java não a seguiram. Cite uma desvantagem das exceções verificadas estaticamente.

4. Alguns problemas são mais fáceis em linguagens que provêem aos programadores estruturas de dados como dicionários. Python é uma destas linguagens. Em Python dicionários são *built-in* na linguagem, isto é, estas estruturas possuem uma sintaxe própria, não sendo implementadas em uma biblioteca, como acontece em Java (`java.util.Map`). Veja este exemplo de uso:

```
>>> a = {"a": [1,2], "b":[4,5,6]}
>>> a = {"Brasil": [1,2], "Burkina_Faso":[4,5,6]}
>>> a["Brasil"]
[1, 2]
>>> a["Brasil"] = [7,5,4,3]
>>> a["Brasil"]
[7, 5, 4, 3]
```

O problema 62 do Projeto Euler pode ser resolvido de forma eficiente (e muito elegante), usando dicionários. Trata-se de encontrar a primeira sequência de cinco números que são permutações dos mesmos dígitos, e que são cubos perfeitos. Por exemplo, os três números $\{41063625, 56623104, 66430125\}$ são todos cubos perfeitos dos números $\{345, 384, 405\}$, e são todos permutações dos mesmos dígitos `[0, 1, 2, 3, 4, 5, 6, 6]`. Porém, esta sequência possui somente três elementos. O problema pede uma com cinco dígitos. É claro que não precisamos parar por aí:

```
>>> Please, enter another integer: 32

[11237467249565803L, 11648763955224703L, 13250642619347875L, 13742426579056183L,
 19347254507362816L, 21103745854269736L, 22685375419136704L, 24103758793614625L,
 25101768364473592L, 25864743621190375L, 31206189625447537L, 32475498673016125L,
 32721634685517049L, 34726063458791251L, 35916748405321672L, 37741681329526504L,
 43643207961178552L, 43912627356578401L, 45417137892036625L, 47663155012973248L,
 48696301552432717L, 56103591473276248L, 59422183570463176L, 63048593772416512L,
 63281267415509473L, 64152761380957432L, 68754211423765309L, 75673011395826424L,
 83274261515694703L, 86345613279405271L, 90371357248465216L, 94780145162353672L]]

Time to find the answer: 6.243766 secs (cpu clock 2.26GHz)
```

Implemente um programa, em Python, que resolva o problema 62 do Projeto Euler. Você não precisa usar dicionários, mas honre o nome da nossa universidade com uma solução eficiente (e elegante).