Lista de Linguagens de Programação – 17

Nome: _____ Matrícula: _____

1. Costuma-se dizer que Python é uma linguagem orientada por objetos, porém, a orientação por objetos, enquanto uma filosofia de desenvolvimento de *software*, é muito mais uma característica do programa, que da linguagem de programação. Considere, por exemplo, o programa abaixo. Este programa, embora escrito em Python, não segue qualquer princípio de codificação orientado por objetos.

```
class Node:
  def __init__(self):
    self.data = ''
    self.link = 0
class Stack:
  def __init__(self):
    self.top = Node()
def add(s, data):
  n = Node()
  n.data = data
  n.link = s.top
  s.top = n
def hasMore(s):
  return s.top.link != 0
def remove(s):
  n = s.top
  s.top = n.link
  return n.data
def test():
  s = Stack()
  add(s, "AA")
  add(s, 0)
  while (hasMore(s)):
    print remove(s)
```

   (a) Por que o programa foge às características da programação orientada por objetos?

   (b) Reescreva o programa, a fim de torná-lo mais "orientado por objetos".

2. Considere o programa abaixo, e responda o que acontecerá em cada linha numerada. As opções possíveis são: (i) Algo será impresso. Neste caso, escreva o que será impresso. (ii) Um erro será produzido em tempo de execução.

```
class Animal:
  atrb_animal = 0
  def __init__(self, name):
    self.name = name
  def __str__(self):
    return self.name + " is an animal"
  def eat(self):
    print self.name + ", which is an animal, is eating."
class Mammal(Animal):
  def __str__(self):
    return self.name + " is a mammal"
  def suckMilk(self):
    print self.name + ", which is a mammal, is sucking milk."
class Dog(Mammal):
  def __str__(self):
    return self.name + " is a dog"
  def bark(self):
    print self.name + " is barking rather loudly."
  def eat(self):
    print self.name + " barks when it eats."
    self.bark
def test():
  a1 = Animal("Tigrinho")
  a2 = Mammal("Oncinha")
  a3 = Dog("Mameluco")
  print a1              # 1
  print a2              # 2
  print a3              # 3
  a1.eat()              # 4
  a2.suckMilk()         # 5
  a2.eat()              # 6
  a3.bark()             # 7
  a3.suckMilk()         # 8
  a3.eat()              # 9
  a1.bark()             # 10
  a1 = a3
  a1.bark()             # 11
```

3. Este exercício é similar ao anterior, porém usaremos Java, em vez de Python. Considere o programa abaixo, e responda o que acontecerá em cada linha numerada. As quatro opções possíveis são: (i) Algo será impresso. Neste caso, escreva o que será impresso. (ii) Um erro será produzido em tempo de compilação. (iii) Um erro será produzido em tempo de execução. (iv) Uma atribuição acontece, e nada será impresso. Note que agora temos a possibilidade de receber avisos de erro em tempo de compilação, o que não acontecia em Python!

```
class Animal {
  public void eat() { System.out.println(this + " is eating"); }
  public String toString () { return "Animal"; }
}
class Mammal extends Animal {
  public void suckMilk() { System.out.println(this + " is sucking"); }
  public String toString () { return "Mammal"; }
}
class Dog extends Mammal {
  public void bark() { System.out.println(this + " is barking"); }
  public String toString () { return "Dog"; }
}
public class Zoo {
  public static void main (String args[]) {
    Animal a1 = new Animal();
    Animal a2 = new Mammal();
    Animal a3 = new Dog();
    System.out.println(a1);                          // 1
    System.out.println(a2);                          // 2
    System.out.println(a3);                          // 3
    a1.eat();                                        // 4
    a2.suckMilk();                                   // 5
    a2.eat();                                        // 6
    Dog d1 = args.length > 1 ? a3 : new Dog();       // 7
    Mammal m1 = d1;                                  // 8
    d1.bark();                                       // 9
    m1.suckMilk();                                   // 10
    d1.suckMilk();                                   // 11
    Dog d2 = (Dog)a3;                                // 12
    a3.bark();                                       // 13
    d2.bark();                                       // 14
    Dog d3 = (Dog)a2;                                // 15
  }
}
```

4. Considere o programa Java abaixo, e diga o que será impresso pelo método `main`.

```
public class Avatar {
  public void buy(Knife k) {
    System.out.println("Avatar bought a knife");
  }
  public void buy(Sword s) {
    System.out.println("Avatar bought a sword");
  }
}

public class Knife {
  public void isBoughtBy(Avatar a) {
    a.buy(this);
  }
}

public class Sword extends Knife {
  public void isBoughtBy(Avatar a) {
    a.buy(this);
  }
}

public class SpiderAv extends Avatar {
  public void buy(Knife k) {
    System.out.println("Spider bought a knife");
  }
  public void buy(Sword s) {
    System.out.println("Spider bought a sword");
  }
  public static void main(String args[]) {
    Avatar a1 = new Avatar();
    Avatar a2 = new SpiderAv();
    SpiderAv sa = new SpiderAv();
    Knife ks = new Sword();
    a1.buy(ks);
    a2.buy(ks);
    sa.buy(ks);
    ks.isBoughtBy(a1);
    ks.isBoughtBy(a2);
    ks.isBoughtBy(sa);
  }
}
```

5. Em Java podemos usar o tipo estático dos parâmetros passados para um método para decidir qual método chamar, em tempo de compilação. Veja, por exemplo, a classe `Avatar`, usada no exercício anterior:

```
public class Avatar {
  public void buy(Knife k) { System.out.println("Avatar bought a knife"); }
  public void buy(Sword s) { System.out.println("Avatar bought a sword"); }
  public static void main(String args[]) {
    Avatar a = new Avatar();
    Knife k = new Knife();
    Sword s = new Sword();
    a.buy(k);
    a.buy(s);
  }
}
```

   (a) Por que este tipo de estratégia não é possível em Python?

   (b) Como o programa acima poderia ser escrito em Python? Na verdade, não é possível implementar um programa idêntico, porém você pode tentar *emular* a funcionalidade daquele programa.

   (c) (**Questão opcional**) Existe um padrão de projetos chamado *visitor*. Este padrão lança mão da capacidade que Java – e outras linguagens estaticamente tipadas – têm de permitir que a correta implementação de um método seja chamada com base no tipo estático dos parâmetros passados para aquele método. Faça uma breve pesquisa sobre este padrão e imagine alternativas para implementá-lo em uma linguagem como Python.

6. Considere o programa abaixo:

```
public interface TreeNode {}

public class Leaf implements TreeNode {
  final int value;
  public Leaf(int value) {
    this.value = value;
  }
}

public class Branch implements TreeNode {
  final TreeNode l, r;
  final int value;
  public Branch(int value, TreeNode l, TreeNode r) {
    this.value = value;
    this.l = l;
    this.r = r;
  }
  public static int computeValue(TreeNode t) {
    if (t instanceof Leaf) {
      return ((Leaf) t).value;
    } else {
      int v1 = computeValue(((Branch)t).l);
      int v2 = computeValue(((Branch)t).r);
      return v1 + v2;
    }
  }
}
```

O projeto deste programa contraria alguns princípios básicos da orientação por objetos. Re-implemente este programa para que ele passe a ser mais "orientado por objetos".

7. Considere as duas classes abaixo, implementadas na linguagem Java:

```
class Mammal extends Animal {
  public void eat() {
    System.out.println("Eating. Yummy, yummy!!!");
  }
}
public class Animal {
  public static void main(String args[]) {
    Animal a = new Animal();
    Animal m = new Mammal();
    // Chamada 1:
    a.eat();

    // Chamada 2:
    m.eat();
  }
}
```

Existe alguma diferença, ainda que mínima, em termos de eficiência, entre a primeira chamada de método (`Chamada 1`) e a segunda (`Chamada 2`)? Em caso afirmativo, diga qual chamada é mais eficiente, e explique o por quê. Em caso negativo, justifique a sua resposta.

8. Este exercício é similar ao anterior, porém usaremos Python em vez de Java. Assim, considere o programa abaixo:

```
class Animal:
  atrb_animal = 0
  def __init__(self, name):
    self.name = name
  def eat(self):
    print "Eating. Yummy, yummy!!!"

class Mammal(Animal):
  pass

def test():
  a1 = Animal("Tigrinho")
  a2 = Mammal("Oncinha")
  # Chamada 1:
  a1.eat()
  # Chamada 2:
  a2.eat()
```

Existe alguma diferença, ainda que mínima, em termos de eficiência, entre a primeira chamada de método (`Chamada 1`) e a segunda (`Chamada 2`)? Em caso afirmativo, diga qual chamada é mais eficiente, e explique o por quê. Em caso negativo, justifique a sua resposta.