

MC 302A – Primeira Prova 06/09/2011

Considere a classe abaixo, usada para representar árvores binárias:

```
public class Node {  
  
    Node left;  
    Node right;  
    String info;  
  
    public Node(String info, Node left, Node right){  
        this.left = left;  
        this.right = right;  
        this.info = info;  
    }  
}
```

1. Inclua nessa classe o método altura() que calcula a *altura* a altura da árvore que tem por raiz o nó ao qual o método é aplicado. A *altura* de uma árvore é definida como zero se a árvore for vazia ou 1 + altura da maior sub-árvore. (valor 1.5)

```
public class Node {  
  
    ...  
  
    public int altura(){  
        int h1 = 0, h2 = 0;  
        if(left != null) h1 = left.altura();  
        if(right != null) h2 = right.altura();  
        if(h1 > h2) return 1+h1;  
        return 1+h2;  
    }  
}
```

2. Inclua nessa classe o método clone() que devolve uma cópia da árvore cuja raiz é o objeto ao qual se aplica o método. A cópia deve ter a mesma estrutura que a árvore original, sem nenhum nó em comum. (valor 1.5)

```
public class Node {  
  
    ...  
    public Node clone(){  
        Node n1 = null, n2 = null;  
        if(left != null) n1 = left.clone();  
        if(right != null) n2 = right.clone();  
        return new Node(info,n1,n2);  
    }  
}
```

3. Inclua nessa classe o método static boolean isomorfias(Node n1, Node n2) que verifica se duas árvores passadas como parâmetro são isomorfas. Duas árvores são ditas isomorfas se ambas são vazias ou se os conteúdos das raízes são iguais e as sub-árvores esquerdas e direitas são isomorfas. (valor 2.0)

```
public class Node { ...  
    static boolean isomorfias(Node n1, Node n2){  
        if(n1 == null) return n2 == null;  
        if(n2 == null) return false;  
        // n1 != null && n2 != null  
        if(n1.info == null) {  
            if(n2.info != null) return false;  
        } else if(! n1.info.equals(n2.info)) return false;  
        return (isomorfias(n1.left,n2.left) && isomorfias(n1.right,n2.right));  
    }  
}
```

4. Crie a classe Node2, derivada de Node, que inclui o campo *father*, usado para referenciar o ‘nó pai’ de um nó. O construtor da nova classe deve usar o construtor da classe mãe e preencher o campo **father** de forma adequada. A classe Node2 deve também implementar a interface **Hierarquia**, mostrada a seguir (valor 2.0):

```
public interface Hierarquia{

    // Verifica se este nó descende do nó passado como parâmetro
    public boolean descende(Node n);

    // Verifica se o nó passado como parâmetro pertence a esta árvore.
    public Boolean pertence(Node n);

}
```

```
public class Node2 extends Node implements Hierarquia {

    Node2 father;

    public Node2(String info, Node2 left, Node2 right){
        super(info,left,right);
        if(left != null) left.father = this;
        if(right != null) right.father = this;
    }

    @Override
    public boolean descende(Node n) {
        if(this.father == null) return false;
        if(this.father == n) return true;
        return this.father.descende(n);
    }

    @Override
    public boolean pertence(Node n) {
        if(n == null) return false;
        return ((Node2)n).descende(this);
    }

    /**
     * Versão 2 - busca por n através do percurso em pré-ordem.
     */
    public boolean pertence_2(Node n){
        if(n == null) return false;
        if(n == this) return true;
        if((left != null) && ((Node2)left).pertence_2(n)) return true;
        if((right != null) && ((Node2)right).pertence_2(n)) return true;
        return false;
    }

}
```

O método abaixo foi escrito na tentativa de trocar dois objetos:

```
public static void troca(Node n1, Node n2){
    Node t = n1;
    n1 = n2;
    n2 = t;
}
```

5. explique porque esse método não faz o que é esperado. (valor 1.5)

R. : O esperado é que após a troca, n1 se comporte exatamente como o 'antigo n2' e n2 se comporte exatamente como o 'antigo n1'. Como em Java a única forma de passagem de parâmetros é por valor (como em C, antes da chamada ao método o valor dos *parâmetros da chamada* é atribuído aos respectivos *parâmetros da definição* da função), qualquer alteração no valor dos *parâmetros da definição* da função é uma alteração local.

6. proponha um método que use uma forma alternativa de se obter o efeito desejado. (valor 1.5)

R. : Como o que se espera com a troca é a 'troca de comportamento', uma saída alternativa é trocar os atributos de n1 e n2, conforme mostrado abaixo:

```
public static void troca(Node n1, Node n2){
    // supondo que n1 e n2 são diferentes de null
    String tInfo = n1.info;
    Node tLeft = n1.left;
    Node tRight = n2.right;
    n1.info = n2.info; n2.info = tInfo;
    n1.left = n2.left; n2.left = tLeft;
    n1.Right = n2.right; n2.right = tRight;
}
```