Part-3: Java Tree structure interview and coding questions

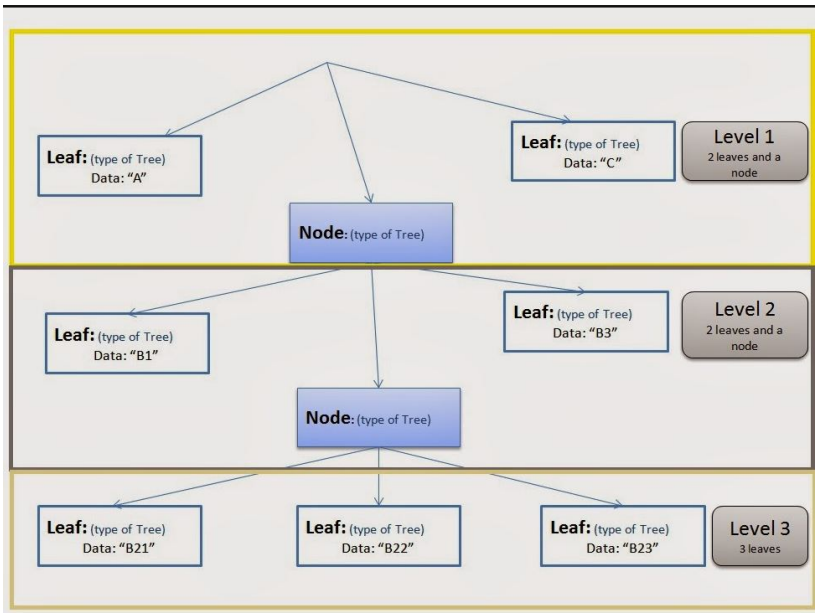
Posted on December 17, 2014 by Arulkumaran Kumaraswamipillai — No Comments ↓

This is an extension to Java Tree structure interview and coding questions — Part 2, and adds **functional programming** and **recursion**.

600+ Full Stack Java/JEE Interview Q&As ♥Free ♦FAQs

open all | close all

- Ice Breaker Interview
- Core Java Interview C
  - Java Overview (4)
  - Data types (6)
  - constructors-metho
  - Reserved Key Wor
  - Classes (3)
  - Objects (8)
  - OOP (10)
  - GC (2)
  - Generics (5)
  - FP (8)
  - IO (7)
  - Multithreading (12)
  - Algorithms (5)
  - Annotations (2)
  - Collection and Data
    - ♦ Find the first n
    - ♦ Java Collectior
    - ♥ Java Iterable V
    - ♥♦ HashMap & H



Tree Data structure

Step 1: The ***Tree*** interface with *get( )* method that returns either a *Triple* tree or *Leaf* data.

```
1  package com.mycompany.flatten;
2
3  public interface Tree<T>
4  {
5      abstract Either<T, Triple<Tree<T>>> get();
6  }
7
```

Step 2: The ***Leaf*** that implements the ***Tree*** interface.

```
1   package com.mycompany.flatten;
2
3   public class Leaf<T> implements Tree<T>
4   {
5
6       private final T data;
7
8       public static <T> Tree<T> leaf(T value)
9       {
10          return new Leaf<T>(value);
11      }
12
13      public Leaf(T t)
14      {
15          this.data = t;
```


16 Technical Key Areas

```
16      }
17
18      public T getData()
19      {
20          return data;
21      }
22
23      @SuppressWarnings("unchecked")
24      public Either<T, Triple<Tree<T>>> get()
25      {
26          return Either.left(data);
27      }
28
29      @Override
30      public String toString()
31      {
32          return "Leaf [data=" + data + "]";
33      }
34  }
35
```

Step 3: The ***Node*** with **Triple** tree that implements the ***Tree*** interface.

```
1   package com.mycompany.flatten;
2
3   public class Node<T> implements Tree<T>
4   {
5
6       private final Triple<Tree<T>> branches;
7
8       public static <T> Tree<T> tree(T left, T mid
9       {
10          return new Node<T>(Leaf.leaf(left), Leaf
11      }
12
13      public Node(Tree<T> left, Tree<T> middle, Tr
14      {
15          this.branches = new Triple<Tree<T>>(left
16      }
17
18      public Either<T, Triple<Tree<T>>> get()
19      {
20          return Either.right(branches);
21      }
22
23      public Triple<Tree<T>> getBranches()
24      {
25          return branches;
26      }
27
28      @Override
29      public String toString()
30      {
31          return "Node {branches=" + branches + "}
32      }
33
34  }
35
36
```

⊞ Low Latency (7)
⊞ Memory Managemen
⊞ Performance (13)
⊞ QoS (8)
⊞ Scalability (4)
⊞ SDLC (6)
⊞ Security (13)
⊞ Transaction Managen

Step 4: The **Triple** class used by the **Node**.

```
1  package com.mycompany.flatten;
2
3  /**
4   * A type that stores three values of the same t
5   */
6  public class Triple<T>
7  {
8
9      private final T left, middle, right;
10
11     public Triple(T l, T m, T r)
12     {
13         this.left = l;
14         this.middle = m;
15         this.right = r;
16     }
17
18     public T left()
19     {
20         return left;
21     }
22
23     public T middle()
24     {
25         return middle;
26     }
27
28     public T right()
29     {
30         return right;
31     }
32
33     @Override
34     public String toString()
35     {
36         return "Triple [l=" + left + ", m=" + mi
37     }
38
39 }
40
```

Step 5: As you can see that the Node and Leaf are using the class Either to handle Node and Leaf differently. The Either stores left or right values but not both. The Leaf uses the left and the Node uses the right. You can pass in a Function to be executed for the leaf and node.

```
1  package com.mycompany.flatten;
2
3  /**
4   * X type which stores one of either of two type
5   */
6  public class Either<X, Y>
7  {
```

```
 8      private final X x;
 9      private final Y y;
10
11      private Either(X x, Y y)
12      {
13          this.x = x;
14          this.y = y;
15      }
16
17      /**
18       * Constructs x left-type Either
19       */
20      public static <X> Either left(X x)
21      {
22          if (x == null)
23              throw new IllegalArgumentException()
24          return new Either(x, null);
25      }
26
27      /**
28       * Constructs x right-type Either
29       */
30      public static <Y> Either right(Y y)
31      {
32          if (y == null)
33              throw new IllegalArgumentException()
34          return new Either(null, y);
35      }
36
37      /**
38       * Applies function f to the contained value
39       * returns the result.
40       */
41      public void ifLeft(Function<X> f)
42      {
43          if (!this.isLeft())
44          {
45              throw new IllegalStateException();
46          }
47
48          f.apply(x);
49
50      }
51
52      /**
53       * Applies function f to the contained value
54       * returns the result.
55       */
56      public void ifRight(Function<Y> f)
57      {
58          if (this.isLeft())
59          {
60              throw new IllegalStateException();
61          }
62
63          f.apply(y);
64
65      }
66
67      /**
68       * @return true if this is x left, false if
69       */
70      public boolean isLeft()
71      {
72          return y == null;
73      }
```

```
74
75      @Override
76      public String toString()
77      {
78          return "Either [x=" + x + ", y=" + y + "
79      }
80
81  }
82
```

Step 6: Define the **Function** interface

```
1  package com.mycompany.flatten;
2
3  public interface Function<P>
4  {
5
6      void apply(P p);
7  }
8
```

Step 7: Define two different implementations for the **Function**. *LeafPrint* and *NodePrint* for printing *Leaf* and *Node* respectively.

```
1  package com.mycompany.flatten;
2
3  public class LeafPrint<P> implements Function<P>
4  {
5
6      public void apply(P p)
7      {
8          System.out.println("--> Leaf:" + p);
9      }
10
11 }
12
```

```
1  package com.mycompany.flatten;
2
3  public class NodePrint<P> implements Function<P>
4  {
5
6      public void apply(P p)
7      {
8          Triple t = (Triple<P>) p;
9          System.out.println("left ==>  " + t.left
10     }
11
12 }
13
```

Step 8: The ***FlattenTree*** interface that works on the *Tree*.

```
1 package com.mycompany.flatten;
2
3 public interface FlattenTree<T>
4 {
5     void flatten(Tree<T> tree);
6 }
7
```

Step 9: Implementation of *FlattenTree* interface *RecursiveFlattenTree.*

```
1  package com.mycompany.flatten;
2
3  public class RecursiveFlattenTree<T> implements
4  {
5
6      public void flatten(Tree<T> tree)
7      {
8          if (tree == null)
9          {
10             return;
11         }
12
13         Either<T, Triple<Tree<T>>> either = tree
14
15         if (either.isLeft())
16         {
17             either.ifLeft(new LeafPrint<T>());
18         }
19
20         else
21         {
22             either.ifRight(new NodePrint<Triple<
23             Triple<Tree<T>> trippleTree = ((Node
24             flatten(trippleTree.left());   // re
25             flatten(trippleTree.middle()); // re
26             flatten(trippleTree.right());  // re
27         }
28
29     }
30 }
31
```

Step 10: Finally, the ***SpecialTreeTest*** test class with main method.

```
1  package com.mycompany.flatten;
2
3  public class SpecialTreeTest
4  {
5
6      public static void main(String[] args)
7      {
8
9          Tree<String> leafB21 = Leaf.leaf("B21");
10         Tree<String> leafB22 = Leaf.leaf("B22");
11         Tree<String> leafB23 = Leaf.leaf("B23");
12
13         //takes all 3 args as "Leaf<String>" and
14         Tree<Tree<String>> level3 = Node.tree(le
15
16         Tree<String> leafB1 = Leaf.leaf("B1");
17         Tree<String> leafB3 = Leaf.leaf("B3");
18
19         //takes  3 args as "Leaf<String>", "Tree
20         Tree<Tree<String>> level2 = new Node(lea
21
22         Tree<Tree<String>> level1 = new Node(Lea
23
24         //System.out.println(level1); //level1 i
25
26         FlattenTree<Tree<String>> flatTree = new
27         flatTree.flatten(level1);
28
29     }
30
31 }
32
```

The ouput

```
1  left ==>  Leaf [data=A] || middle ==> Node {bran
2  --> Leaf:A
3  left ==>  Leaf [data=B1] || middle ==> Node {bra
4  --> Leaf:B1
5  left ==>  Leaf [data=Leaf [data=B21]] || middle
6  --> Leaf:Leaf [data=B21]
7  --> Leaf:Leaf [data=B22]
8  --> Leaf:Leaf [data=B23]
9  --> Leaf:B3
10 --> Leaf:C
11
```

Popular Posts

♦ 11 Spring boot interview questions & answers

823 views

♦ Q11-Q23: Top 50+ Core on Java OOP Interview Questions & Answers

765 views

18 Java scenarios based interview Questions and Answers

399 views

001A: ♦ 7+ Java integration styles & patterns interview questions & answers

388 views

01b: ♦ 13 Spring basics Q8 – Q13 interview questions & answers

295 views

♦ 7 Java debugging interview questions & answers

293 views

01: ♦ 15 Ice breaker questions asked 90% of the time in Java job interviews with hints

285 views

♦ 10 ERD (Entity-Relationship Diagrams) Interview Questions and Answers

279 views

♦ Q24-Q36: Top 50+ Core on Java classes, interfaces and generics interview questions & answers

239 views

001B: ♦ Java architecture & design concepts interview questions & answers

201 views

Bio | Latest Posts

Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

About Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

‹ Part 2: Java Tree structure interview questions and coding questions

Top 5 Core Java Exceptions and best practices ›

Posted in Coding, Collection and Data structures, member-paid, Tree structure

Leave a Reply

Logged in as geethika. Log out?

Comment

Post Comment

Empowers you to open more doors, and fast-track

Technical Know Hows

☀ Java generics in no time ☀ Top 6 tips to transforming your thinking from OOP to FP ☀ How does a HashMap internally work? What is a hashing function?
☀ 10+ Java String class interview Q&As ☀ Java auto un/boxing benefits & caveats ☀ Top 11 slacknesses that can come back and bite you as an experienced Java developer or architect

Non-Technical Know Hows

☀ 6 Aspects that can motivate you to fast-track your career & go places ☀ Are you reinventing yourself as a Java developer? ☀ 8 tips to safeguard your Java career against offshoring ☀ My top 5 career mistakes

Prepare to succeed

☀ Turn readers of your Java CV go from “Blah blah” to “Wow”? ☀ How to prepare for Java job interviews? ☀ 16 Technical Key Areas ☀ How to choose from multiple Java job offers?

Select Category

© Disclaimer

 ↑