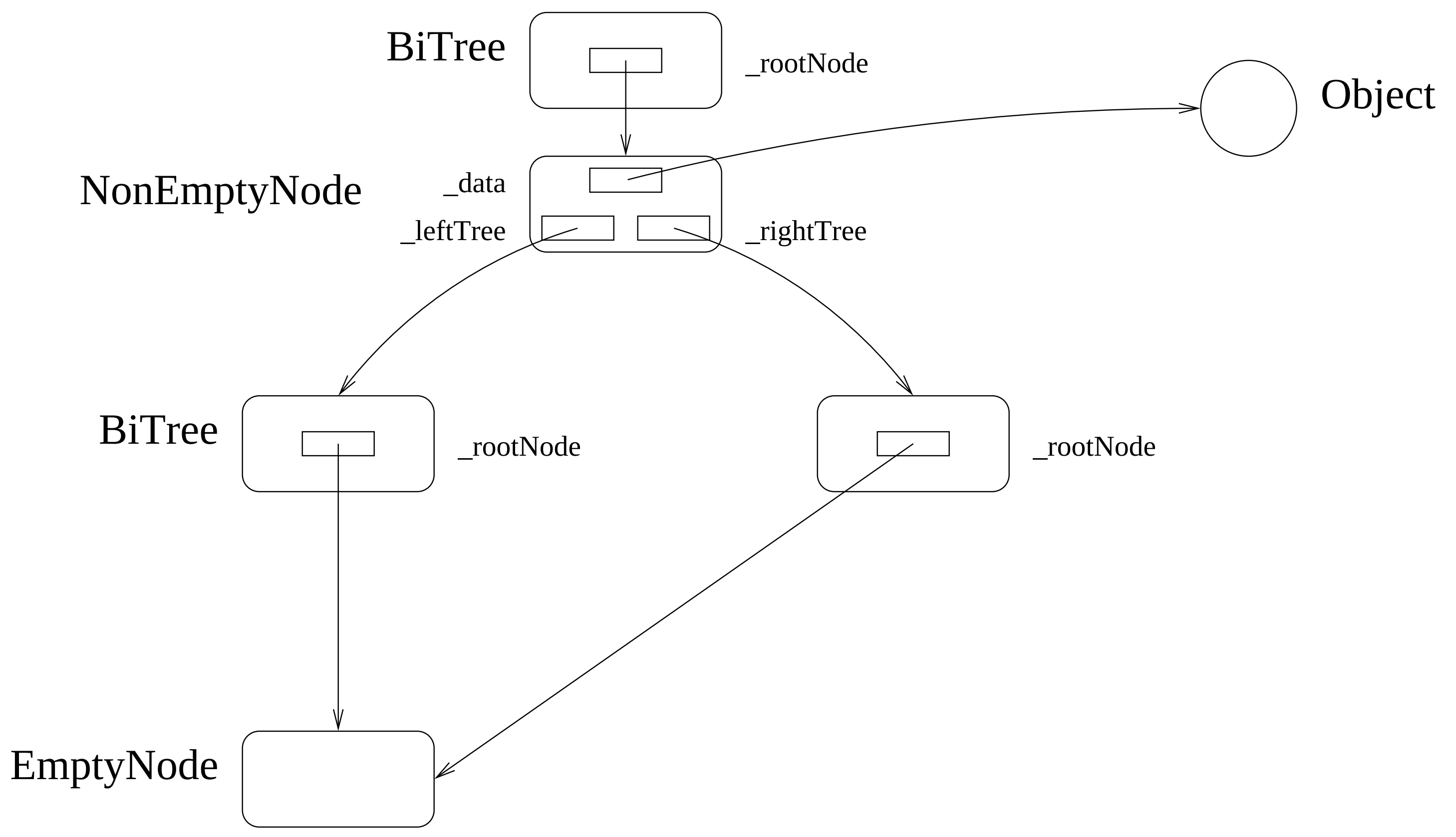
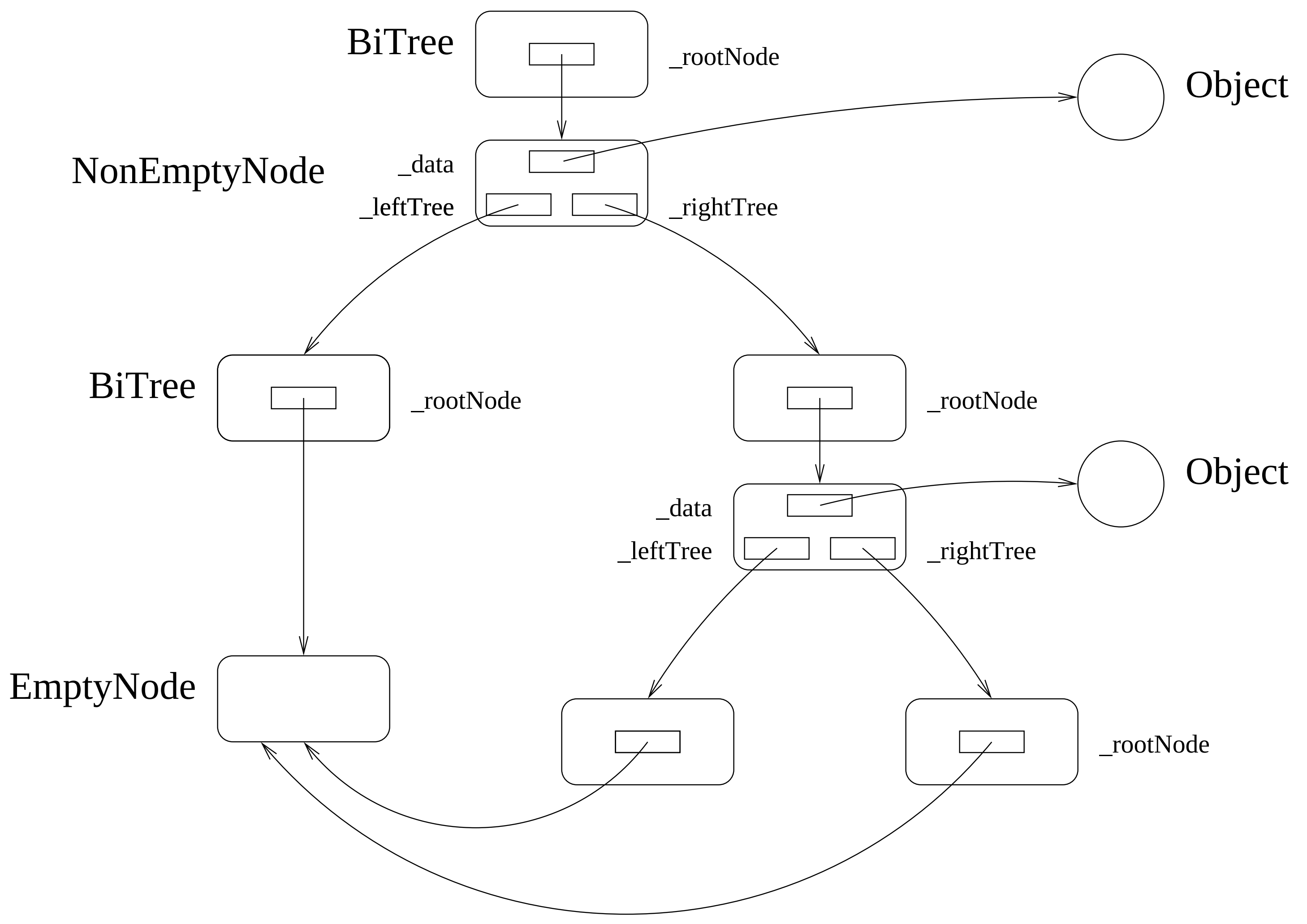
Binary Trees



Binary Trees



Binary Trees: A Visitor

```
public class VerticalPrinter implements IAlgo
{
  . . .
  public Object emptyCase(BiTree host, Object input)
  {
    System.out.print("[]");
    return null;
  }

  public Object nonEmptyCase(BiTree host, Object input)
  {
    return host.execute(VerticalPrintHelper.Singleton,
                        new Integer(0));
  }
```

Binary Trees: A Visitor

```
public class VerticalPrintHelper implements IAlgo
{
  . . .
  public Object emptyCase(BiTree host, Object input)
  {
    int level = ((Integer)input).intValue();

    printSpaces(level);
    System.out.print("[]");

    return null;
  }
```

Binary Trees: A Visitor

```
public Object nonEmptyCase(BiTree host, Object input)
{
  int level = ((Integer)input).intValue();

  printSpaces(level);
  System.out.println(host.getRootData());

  level++;

  host.getLeftSubTree().execute(this, new Integer(level));
  System.out.println();

  host.getRightSubTree().execute(this, new Integer(level));

  return null;
}
```

Binary Trees: A Visitor

```
private void printSpaces(int n)
{
  for (int i = 0; i < n; i++)
    System.out.print(’ ’);
}
```

Binary Trees

- The following program creates and prints a simple binary tree.

```
import binaryTree.*;
import binaryTree.visitor.*;

class Test {
  public static void main(String args[])
  {
    BiTree tree = new BiTree();

    tree.insertRoot("I'm the root!");
    tree.getLeftSubTree().insertRoot("I'm the left child!");
    tree.getRightSubTree().insertRoot("I'm the right child!");
    tree.execute(binaryTree.visitor.VerticalPrinter.Singleton,
                 null);
  }
}
```

Binary Trees

- The printout looks like:

```
I'm the root!
 I'm the left child!
  []
  []
 I'm the right child!
  []
  []
```