자료구조: 2022년 1학기 [강의]

Tree Traversal Call Back

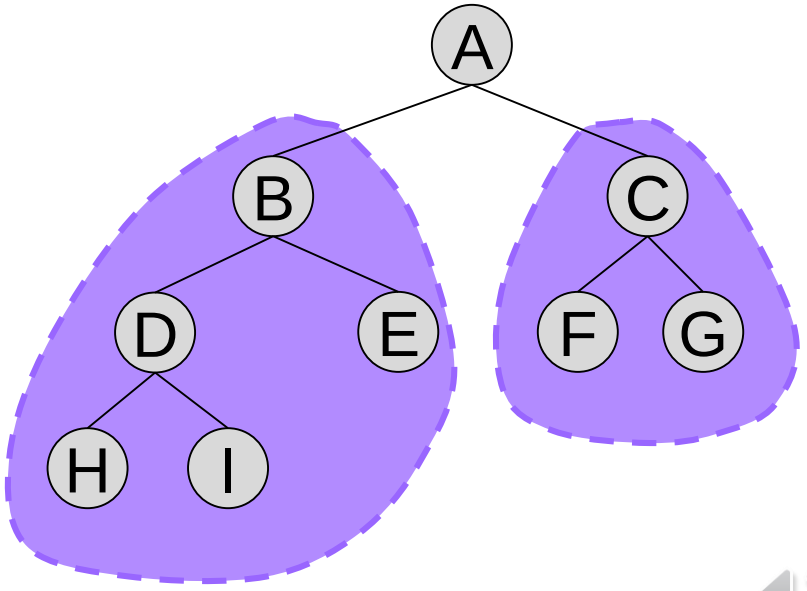
강지훈
jhkang@cnu.ac.kr
충남대학교 컴퓨터융합학부

이진 트리 탐색
(Binary Tree Traversal)

이진트리 탐색 (Traversal)

- 체계적으로 모든 노드를 방문 (visit)
  - 중위 탐색 (Inorder traversal)
  - 전위 탐색 (Preorder traversal)
  - 후위 탐색 (Postorder traversal)

- 기준은?
  - 루트

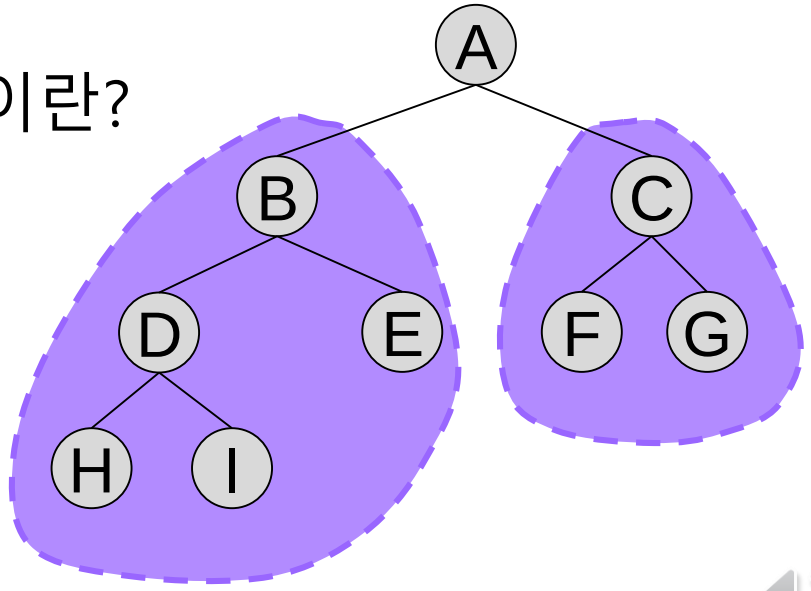


이진트리 탐색: 중위 탐색

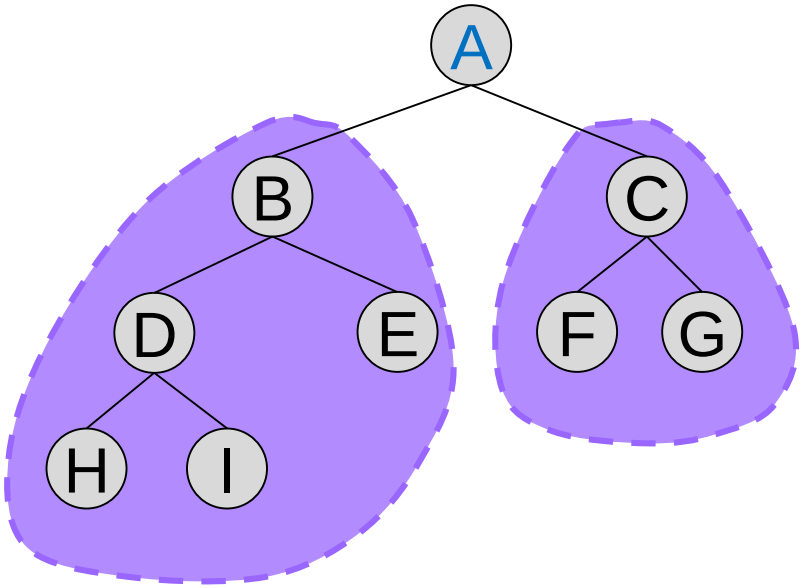
- 체계적으로 모든 노드를 방문 (visit)
  - 중위 탐색 (Inorder traversal)
  - 전위 탐색 (Preorder traversal)
  - 후위 탐색 (Postorder traversal)

- 기준은?
  - 루트

- 중위탐색 (Inorder Traversal) 이란?
  - 왼쪽 부트리를 (재귀적으로) 중위 탐색하여 모든 노드를 방문
  - 루트를 방문
  - 오른쪽 부트리를 (재귀적으로) 중위 탐색하여 모든 노드를 방문

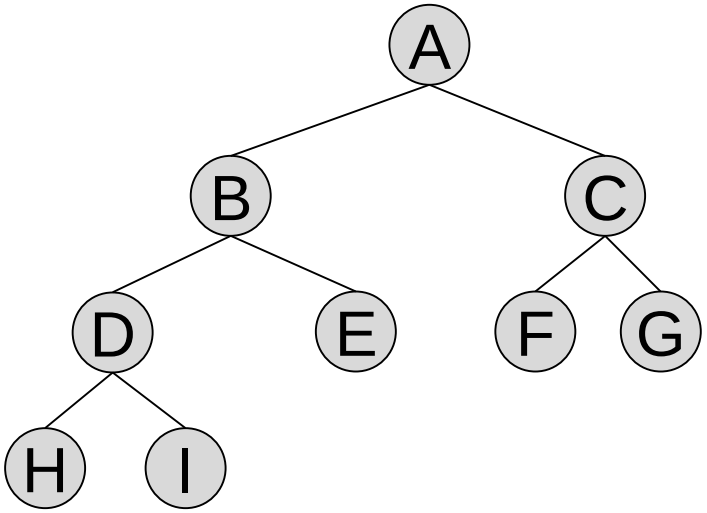


이진트리 탐색 순서에서 루트의 위치는?

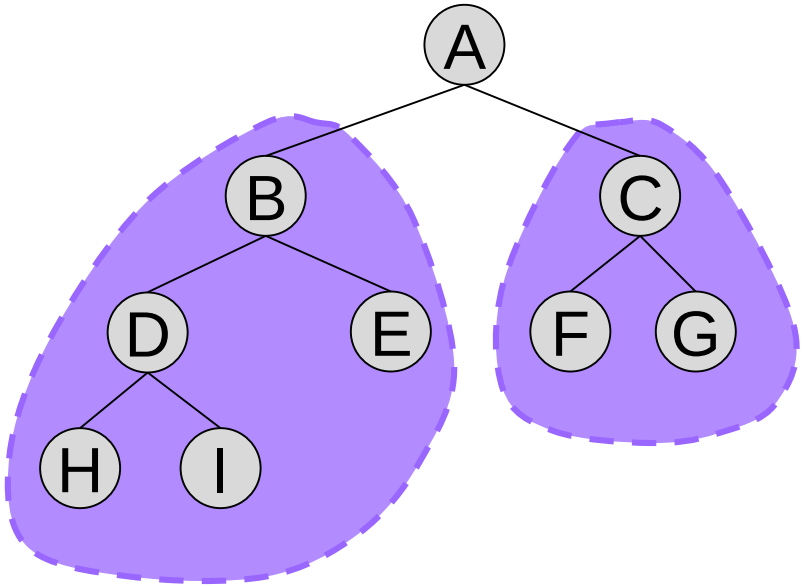


- Inorder : (((H)–D–(I))–B–(E))–A–((F)–C–(G))
  H – D – I – B – E – A – F – C – G
- Preorder : A – B – D – H – I – E – C – F – G
- Postorder : H – I – D – E – B – F – G – C – A

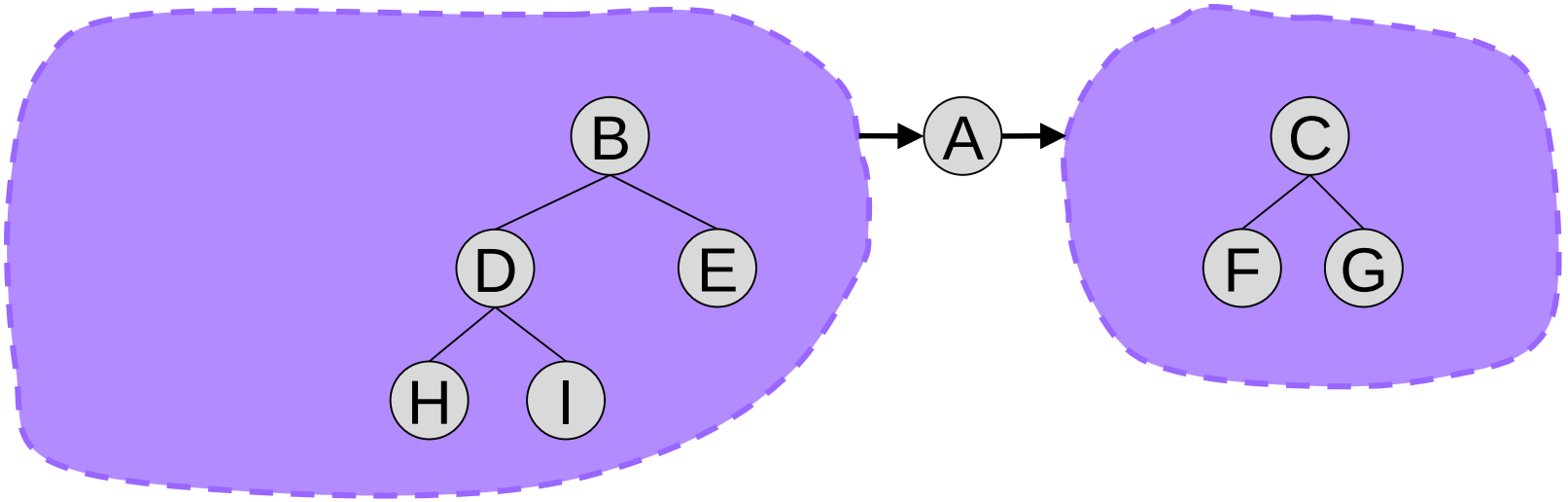
예제: 중위 탐색 (Inorder) [0]



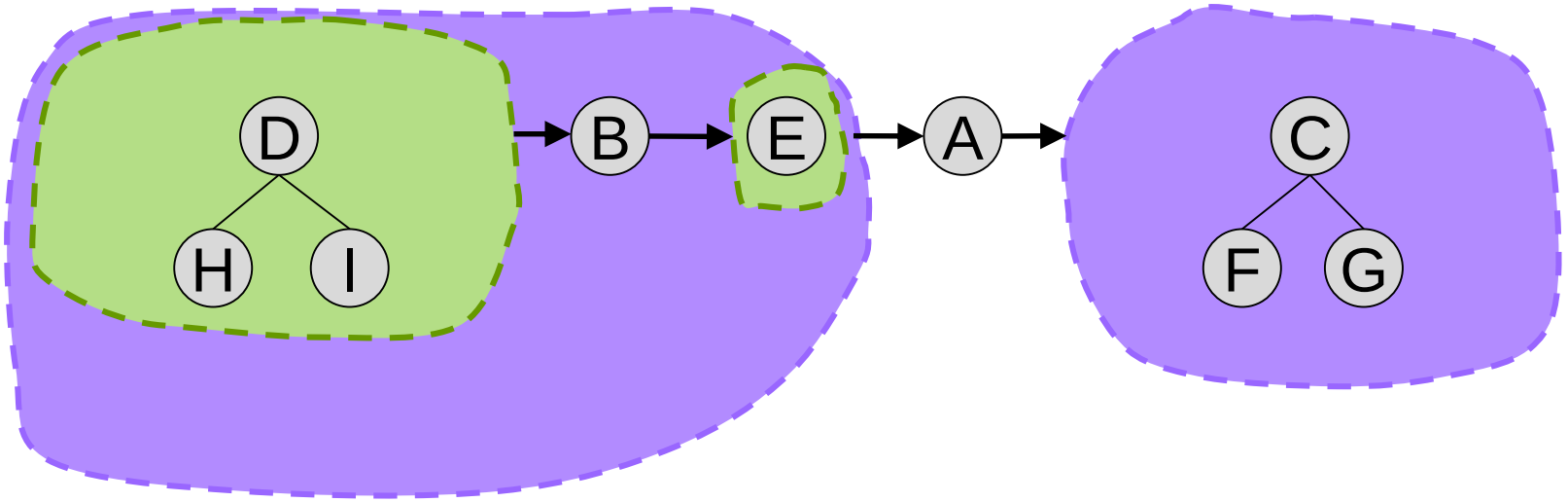
예제: 중위 탐색 (Inorder) [1]



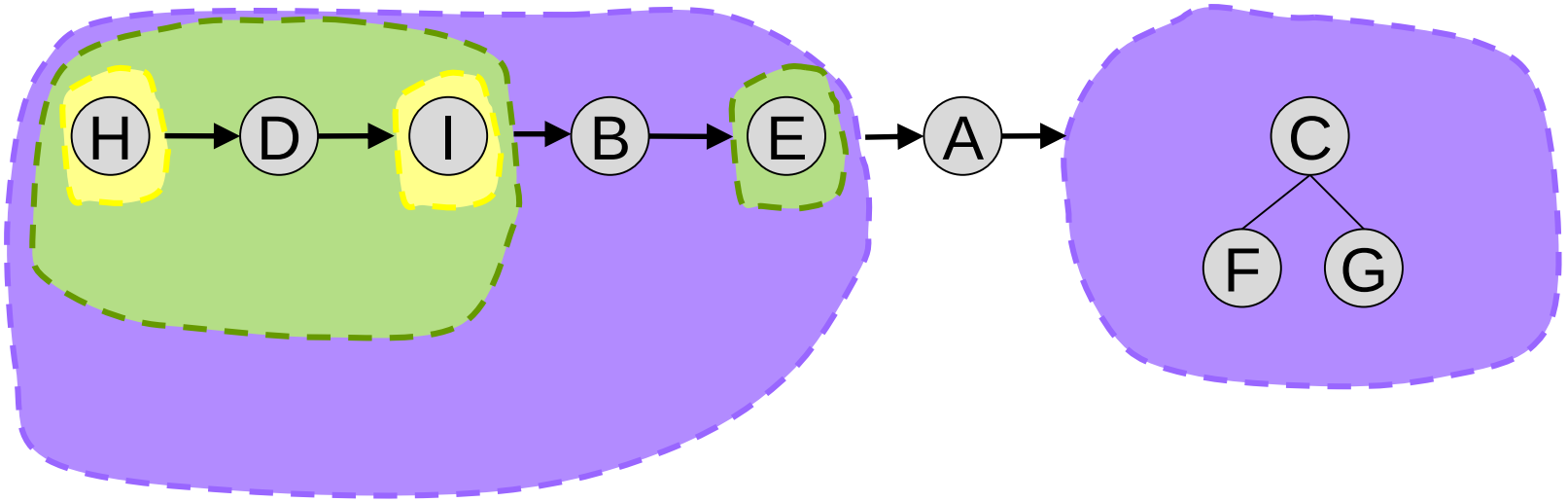
예제: 중위 탐색 (Inorder) [2]



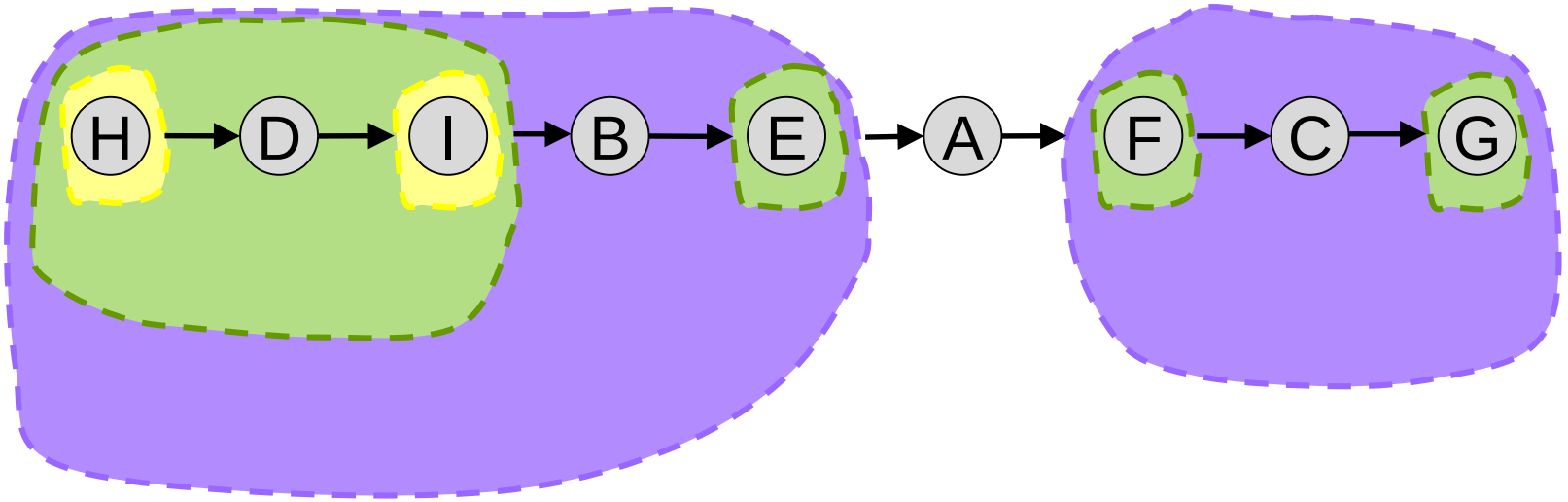
예제: 중위 탐색 (Inorder) [3]



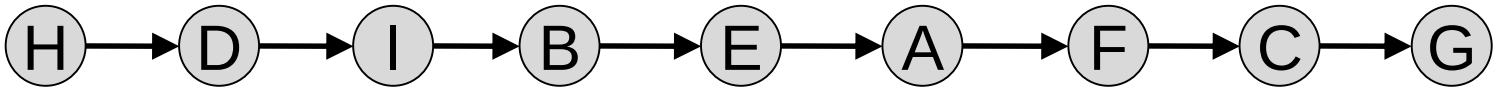
예제: 중위 탐색 (Inorder) [4]

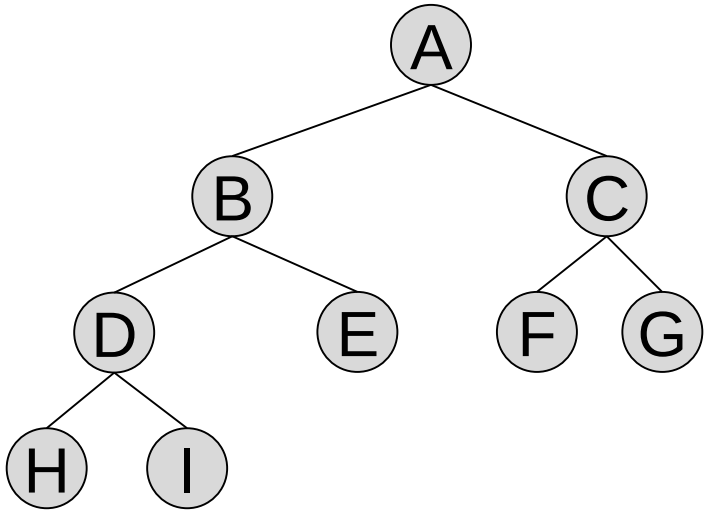


예제: 중위 탐색 (Inorder) [5]

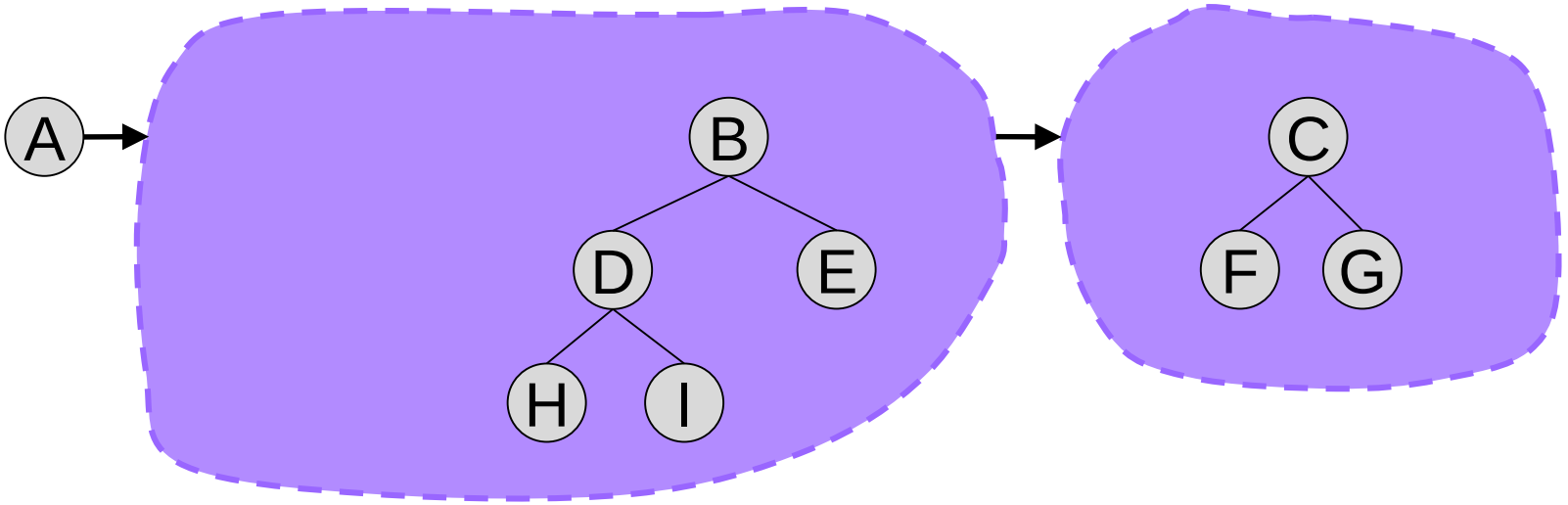


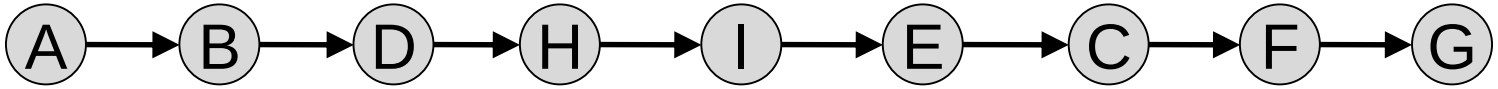
예제: 중위 탐색 (Inorder) [6]



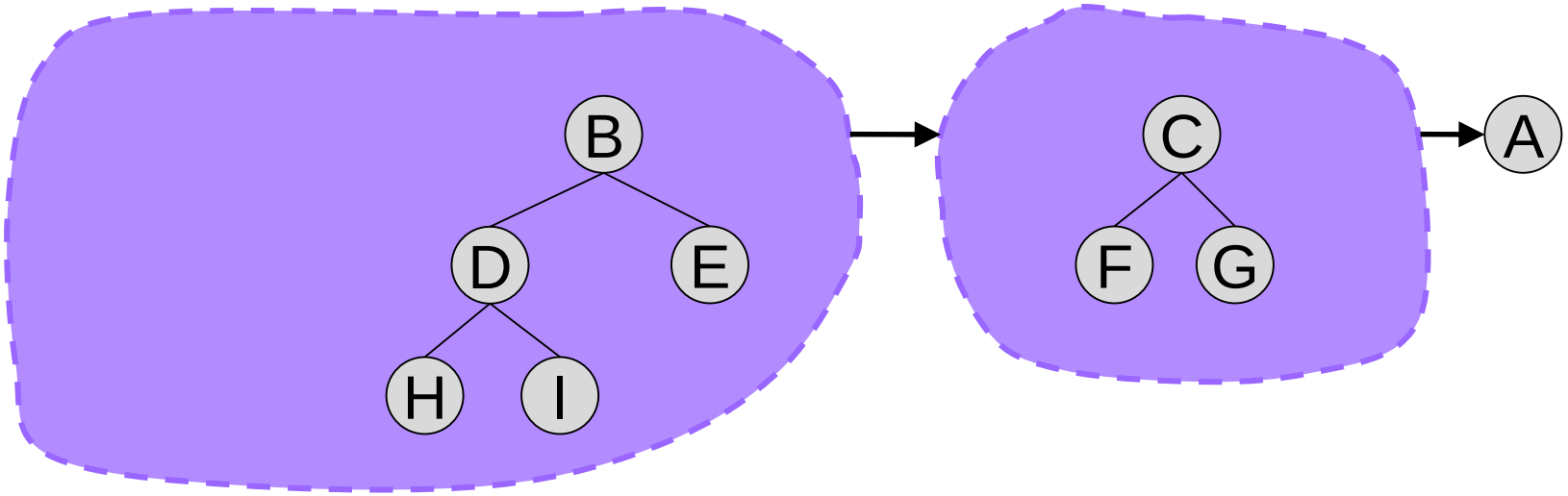


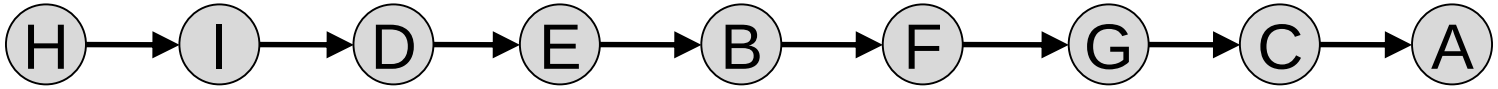
예제: 전위 탐색 (Preorder)





예제: 후위 탐색 (Postorder)





탐색 알고리즘

중위 탐색 (Inorder traversal)

```
private void inOrderRecursively (BinaryNode<T> aRoot)
{
    if ( aRoot != NULL ) {
        this.inOrderRecursively (aRoot.leftChild()) ;
        this.visit (aRoot.element()) ;
        this.inOrderRecursively (aRoot.rightChild()) ;
    }
}
```

- 탐색은 재귀적으로(recursively) 실행된다.
- 그러므로, 구현 코드에 보이지는 않지만 스택이 사용되고 있다.

BinaryTree<T>: inOrder() 의 구현

```
public void  inOrder ()
{
    this.inOrderRecursively (this.root()) ;
}
```

BinaryTree<T>: preOrder() 의 구현

```
public void  preOrder ()
{
    this.preOrderRecursively (this.root()) ;
}

private void  preOrderRecursively (BinaryNode<T> aRoot)
{
    if ( aRoot != NULL ) {
        this.visit (aRoot.element()) ;
        this.preOrderRecursively (aRoot.leftChild()) ;
        this.preOrderRecursively (aRoot.rightChild()) ;
    }
}
```

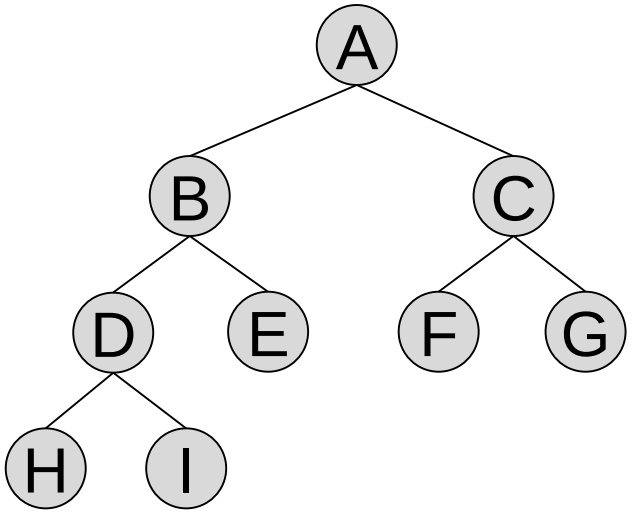
BinaryTree<T>: postOrder() 의 구현

```
public void  postOrder ()
{
    this.postOrderRecursively (this.root()) ;
}

private void  postOrderRecursively (BinaryNode<T> aRoot)
{
    if ( aRoot != NULL ) {
        this.postOrderRecursively (aRoot.leftChild()) ;
        this.postOrderRecursively (aRoot.rightChild()) ;
        this.visit (aRoot.element()) ;
    }
}
```

Level-order Traversal: 레벨 순으로 탐색

```
Initially, add root to queue ;
While ( queue is not empty ) {
    X = remove node from queue ;
    visit node X ;
    add all children of node X to queue ;
}
```



Queue	Visit
→ A →	A
→ C → B →	B
→ E → D → C →	C
→ G → F → E → D →	D
→ I → H → G → F → E →	E
→ I → H → G → F →	F
→ I → H → G →	G
→ I → H →	H
→ I →	I
→ →	(End of Traversal)

BinaryTree<T>: 공개함수 levelOrder() 의 구현

```
public void  levelOrder()
{
    Queue<BinaryNode<T>>  nodeQueue= new Queue<T>() ;
    BinaryNode<T>  visitingNode ;
    nodeQueue.add (this.root()) ;
    while ( ! nodeQueue.isEmpty() ) {
        visitingNode = nodeQueue.remove() ;
        this.visit (visitingNode.element()) ;
        if ( visitingNode.hasLeftChild() )  {
            nodeQueue.add (visitingNode.leftChild()) ;
        }
        if ( visitingNode.hasRightChild() )  {
            nodeQueue.add (visitingNode.rightChild()) ;
        }
    }
}
```

"Call Back"

Binary Tree 내부에 있는 visit 는 누가 정의하나?

- 예: 중위 탐색 (Inorder traversal)

```
public class  BinaryTree<T> {
    ......

    private void  visit (T anElement)
    {
        ..... // 여기를 누가 정의하나?
    }

    private void  inOrderRecursively (BinaryNode<T> aRoot)
    {
        if ( aRoot != NULL ) {
            this.inOrderRecursively (aRoot.leftChild()) ;
            this.visit (aRoot.element()) ;
            this.inOrderRecursively (aRoot.rightChild()) ;
        }
    }
    ......
}
```

Binary Tree 내부에 있는 visit 는 누가 정의하나?

- 이진트리의 탐색은 구현에 의존적이다.
  - 그러므로, 탐색 행위 자체는 이진 트리 내부에 구현되어야 할 일이다.

- 그러나,
  특정 노드를 방문했을 때 해야 할 일은, 이진트리의 사용자가 정의할 수 있어야 한다.
  즉, 이진 트리를 사용하는 사람의 용도에 맞게 표현되어야 한다.
  그렇다는 것은, 방문했을 때 해야 할 일을 이진 트리를 구현하는 class 내부에 정의할 수는 없다.
  - 이진 트리 class 내부에 구현하는 방법으로 해결하는 것은, Model-View-Controller 의 개념을 깨뜨리게 될 위험성이 높다.
    - 방문 시에 해야 할 일이 출력할 일이라면.....
  - 또한 고정시켜 버린 방문(visit) 행위는, "DictionaryByBinarySearchTree" 객체를 사용하는 또 다른 사용자가 자신의 목적에 맞는 용도로 사용할 수 없게 만든다. 아니면 각 사용자는 자신 만의 방문 행위를 포함하는 자신 만의 목적에 맞는 "DictionaryByBinarySearchTree" class 를 정의하여 사용해야 할 것이다.

사용자가 "visit" 기능을 정의할 수 있게 하려면?

- Binary Tree 객체를 사용하는 사용자는 자신의 용도에 맞게 visit 를 정의하고 싶다.
  - visit 에서 해야 할 일은 응용에 따라 달라질 수 밖에 없으므로, 그 정의를 사용자가 하는 것이 당연하다.

- 무엇이 문제인가?
  - inOrder() 는 Class "BinaryTree" 의 public method.
  - visit 는 Binary Tree 내부에 정의되며, inOrder() 안에서 사용한다.
    - 엄밀히는, inOrder() 가 call 하는 inOrderRecursively() 안에서.

- 그러므로, inOrder() 를 call 하는 쪽 즉 Binary Tree 객체의 사용자는, Class "BinaryTree" 안에 private 하게 정의되고 구현되는 (그래서 캡슐화 되는) visit 의 실행 내용을 임의로 변경할 수는 없다!

위임 (Delegate)

- "BinaryTree" inorder() 탐색 중에 visit 에서 실제로 해야 할 일을 사용자의 클래스에 정의하고 구현한다.
  - 사용자가 AppController 객체라면, Class "AppController" 안에.

- 다만, 이 내용을 Binary Tree 객체로 하여금 어떻게 알게 할 수 있을까?

- **위임 (delegate)** 의 방법을 사용한다:
  - Binary Tree 객체는 visit() 의 "정의" 와 "실행" 을 사용자에게 위임 한다.
  - 사용자는 Binary Tree 객체에게 visit 의 "정의" 와 "실행" 을 위임 (delegate) 받는다.

- 구체적인 방법은?

위임 (Delegate) 의 구체적인 방법은?

- 위임에 의해 실행되어야 할 함수는 사용자와 Binary Tree 객체 사이에 미리 약속하여 정의해 놓기로 한다.
  - 위임 용도의 class 를 미리 정의해 놓을 수 있다.
  - 또는 class 대신 interface 로 정의해 놓을 수 있다. (주로 사용)
- Binary Tree 객체는 visit 의 행위를 위임할 사용자를, 사용자로부터 통보 받기로 한다.
- 사용자는 자신이 위임 (delegate) 받아 처리할 수 있음을 Binary Tree 객체에게 알려준다.
  - 즉, 사용자는 inOrder() 를 실행하기 전에, 자신이 사용할 Binary Tree 객체에게 스스로 자신이 누구인지를 미리 알려준다.
- 사용자는 자신이 누구인지를 아는 Binary Tree 객체에게 inOrder() 를 실행하게 한다.
- Binary Tree 객체는 visit 를 위임 받아 처리할 위임자가 누구인지를 알고 있으므로, inorder() 를 실행하는 동안에, visit 가 필요하면 위임자로 하여금 visit 를 실행하게 한다.

Call Back

- 이러한 위임의 방식의 해결책을 **"Call Back"** 이라 하는 이유는?
  - AppController 객체는 Binary Tree 객체의 사용자이다.
  - AppController 객체는 Binary Tree 객체의 inorder() 를 call 함으로써, Binary Tree 가 inorder() 를 실행하게 한다.
    - 사용자 쪽에서 Binary Tree 객체 쪽으로 call 이 발생하였다.
  - Binary Tree 객체는 inorder를 실행하는 동안에 visit 의 행위가 필요할 때 마다, 위임자 즉 사용자에게 visit 를 실행하도록 요청한다.
  - 이 요청으로, Binary Tree 객체 쪽에서 다시 거꾸로 사용자 쪽으로 call 이 발생하였다.
- 피사용자 쪽에서 사용자쪽으로 거꾸로 call 을 하는 바로 이 행위가 **"call Back"** 이다.
  - 피사용자 객체 (Binary Tree 객체) 가 사용자 객체 (AppController 객체) 에게 일을 시키는 것은 일반적인 경우가 아닌, 특수한 상황이다.
  - 그러나 이러한 상황은 발생할 수 밖에 없으며, 우리는 위임의 방식으로 해결하고 있다.

"Call Back" 의 구현

문제의 요약 [1]

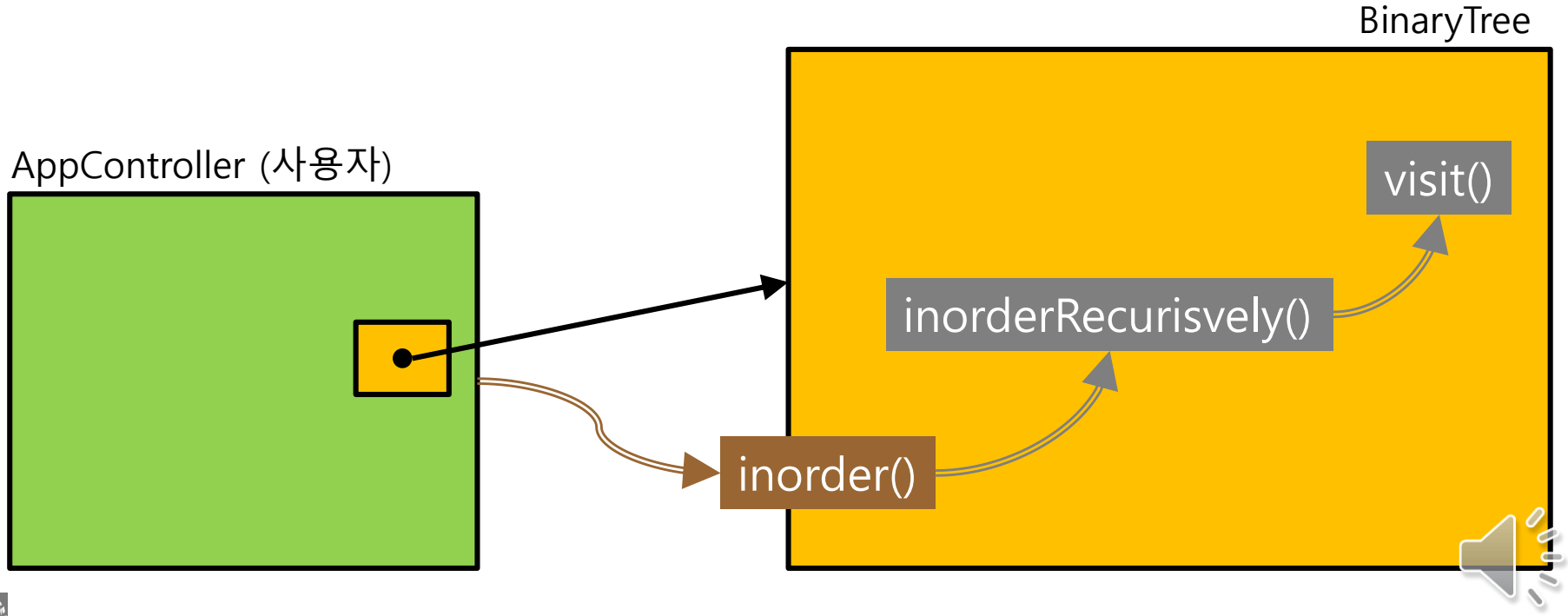
- 주어진 상황:
  - 우리가 사용할 이진트리를 구현하는 class는 "BinaryTree" 이다.
  - 이진트리 탐색은 inorder traverse (중위 탐색) 을 실행한다:

```
public class  BinaryTree<...>
{
      ......

      private void  inorderRecursively (BinaryNode<...> aRoot) {
            if ( aRoot != null ) {
                  this.inorderRecursively (aRoot.left()) ;
                  this.visit (aRoot.element()) ;
                  this.inorderRecursively (aRoot.right()) ;
            }
      }
      public void  inOrder() {
            this.inorderRecursively (this.root());
      }
} // End of class "BinaryTree"
```

문제의 요약 [2]

- "visit()"는 class "BinaryTree"의 private method.
- AppController로서는 "visit()" 의 내용을 자신의 목적에 맞게 구현할 수 있을까?



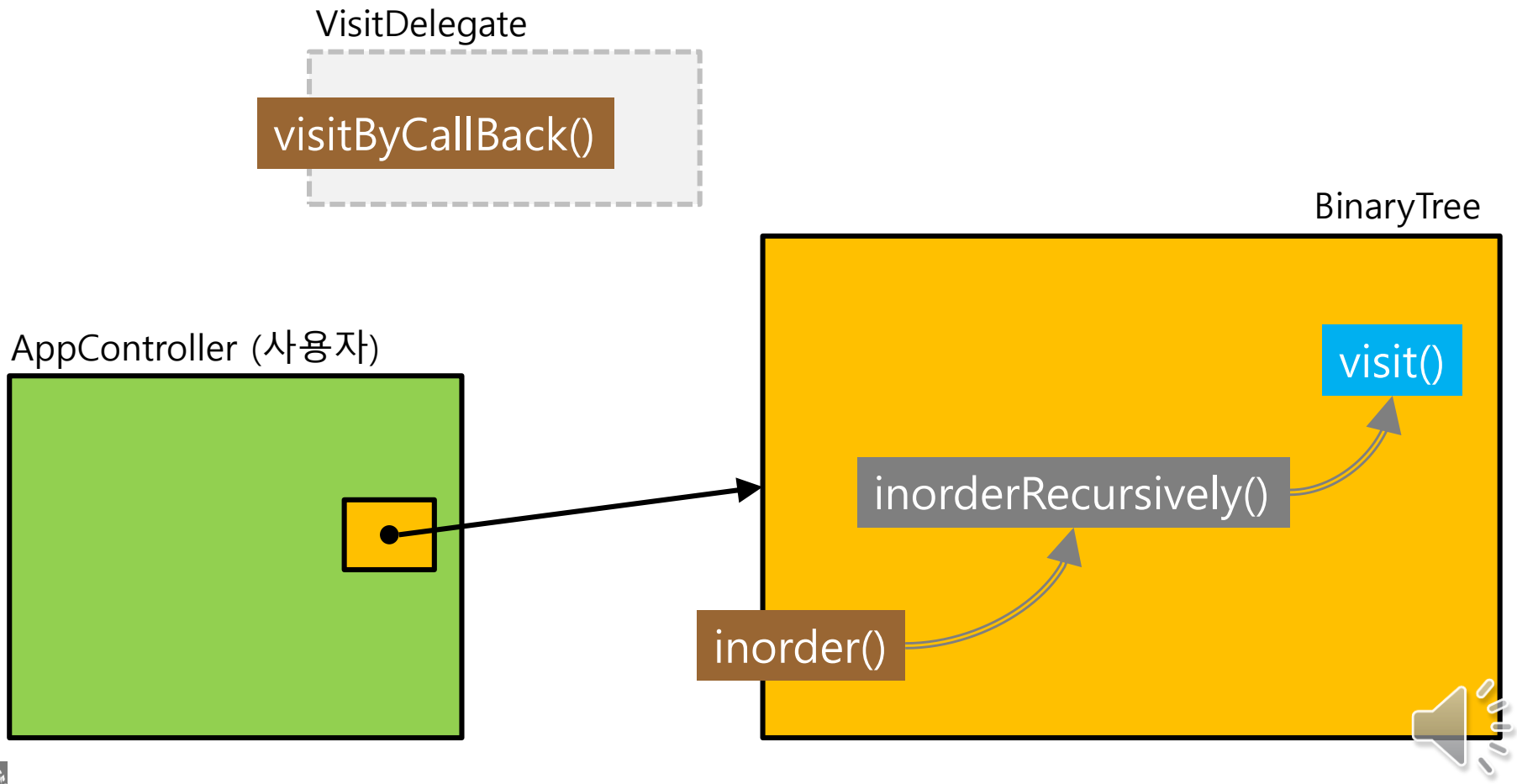
문제의 요약 [3]

- 문제점:
  - "inorderRecursively()" 내부에서 call 하는 함수 "visit()" 는, 형식적인 관점에서 자신의 class "BinaryTree" 안에 정의하는 것이 보통의 방법일 것이다.
  - 그러나 실제적인 관점에서는 함수 "visit()"의 내용은, class "BinaryTree" 객체의 사용자가 자신의 목적에 맞게 정의할 수 밖에 없다.
  - 또한 사용자 마다 사용 목적이 다를 수 있으므로 "visit()" 의 내용이 달라질 것이다.
  - 그렇다면, inorder traverse 의 사용자에게, 자신의 원하는 바를 class "BinaryTree" 내부의 함수 visit() 를 직접 구현할 수 있게 허락해야 할까? 허락하면 어떤 문제가 있을까? 허락하지 않는다면 해결책은?

- 해결책:
  - "visit()"의 구현은 사용자가, 실행은 "BinaryTree" 객체가!!
  - 어떻게? ⇒ Call Back 을 사용하여!

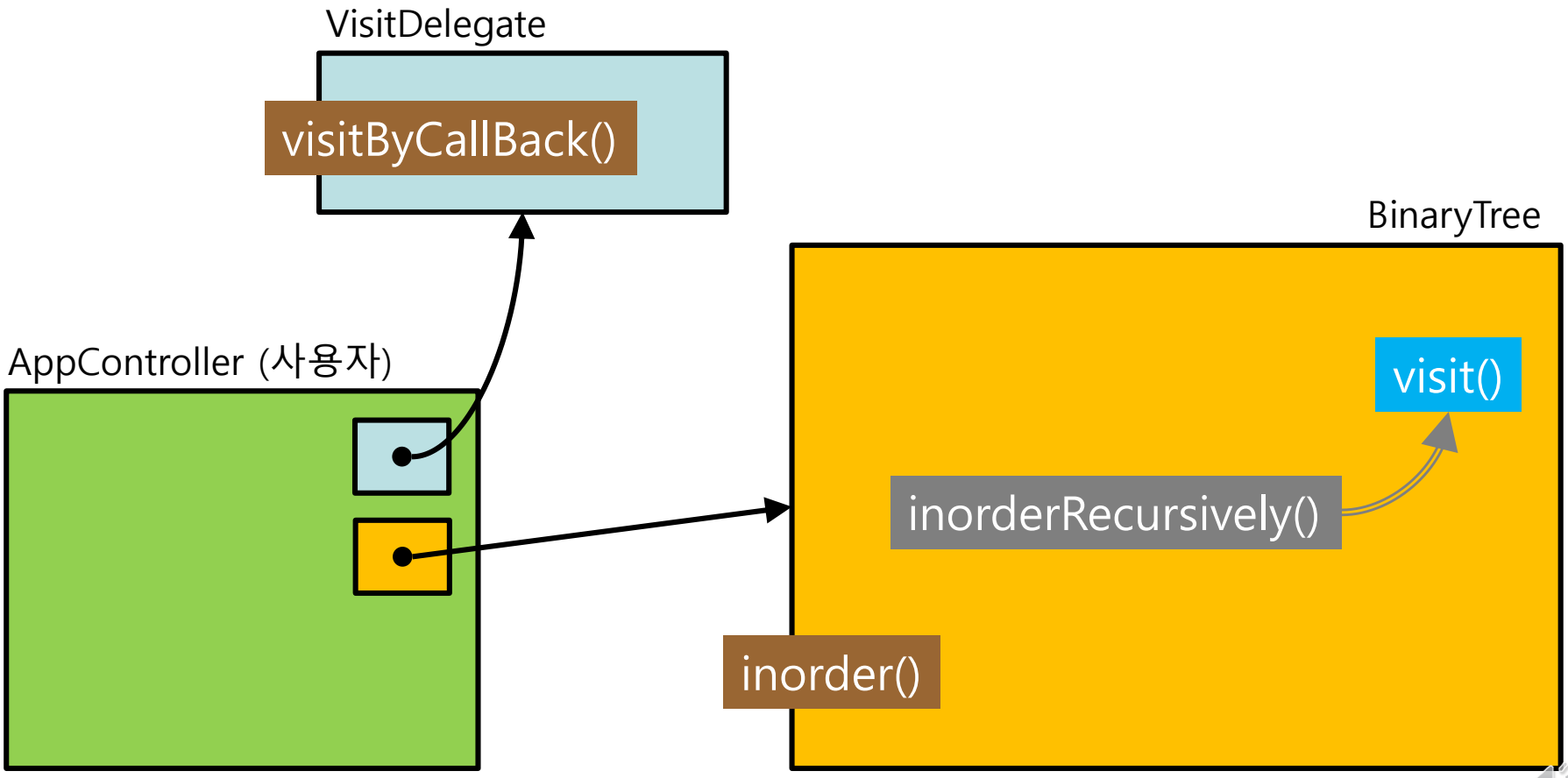
해결책은? [1]

- "BinaryTree" 는 "VisitDelegate"를 Java Interface 로 정의하여 사용자에게 제공한다. 즉 함수 "visitByCallBack()" 의 사용법만 정의하게 된다. 결국 "visitByCallBack()" 이 "visit()"의 역할을 하게 된다.



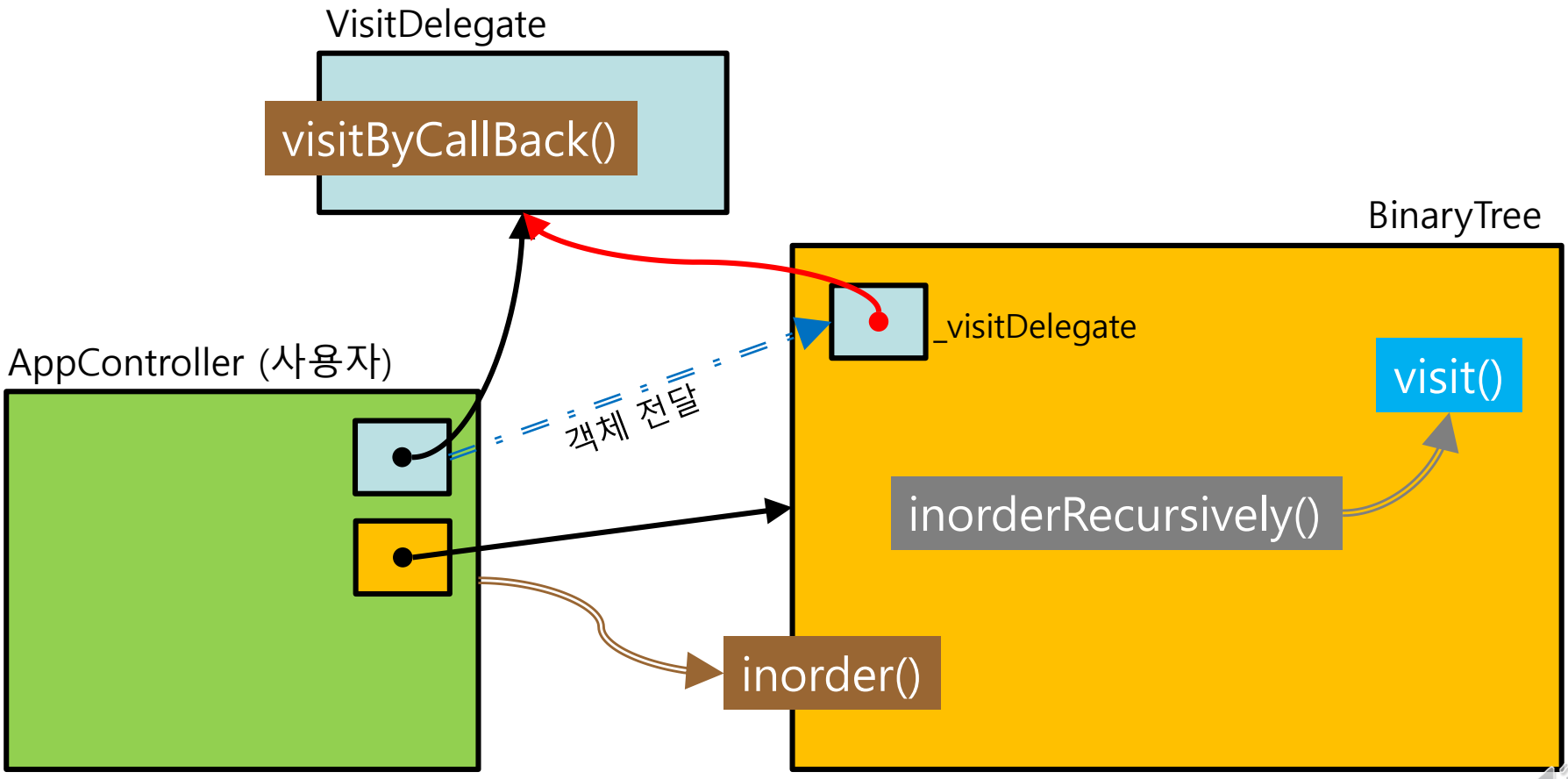
해결책은? [2]

- 사용자는 class "VisitDelegate" 를 정의 해 놓는다.
- 사용하는 시점에 사용자는 "VisitDelegate" 객체를 생성 한다.



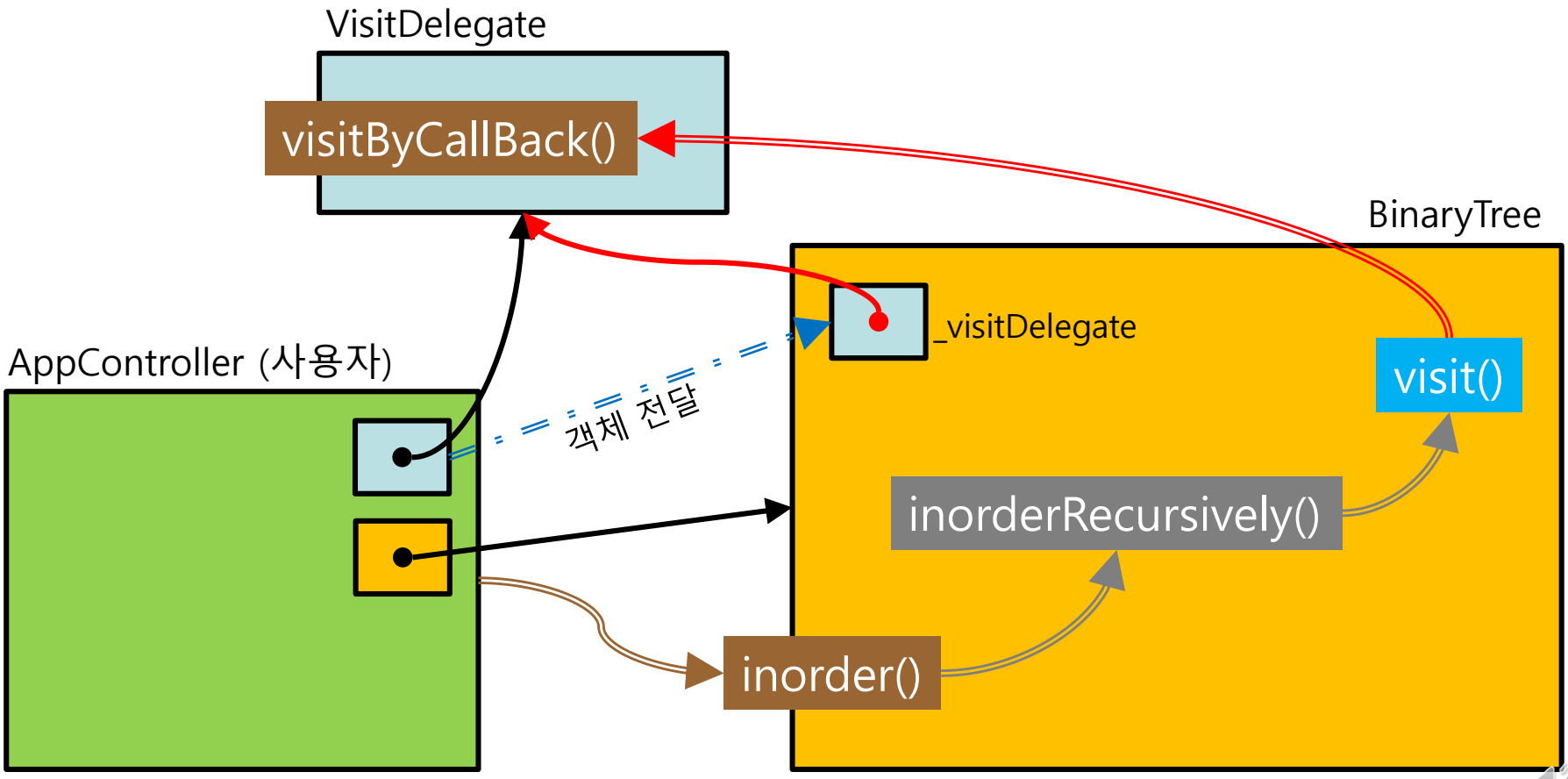
해결책은? [3]

- 사용자는 "VisitDelegate" 객체를 "BinaryTree" 객체로 전달한다.
- 사용자는 "BinaryTree" 객체에게 "inorder()"를 실행하게 한다.



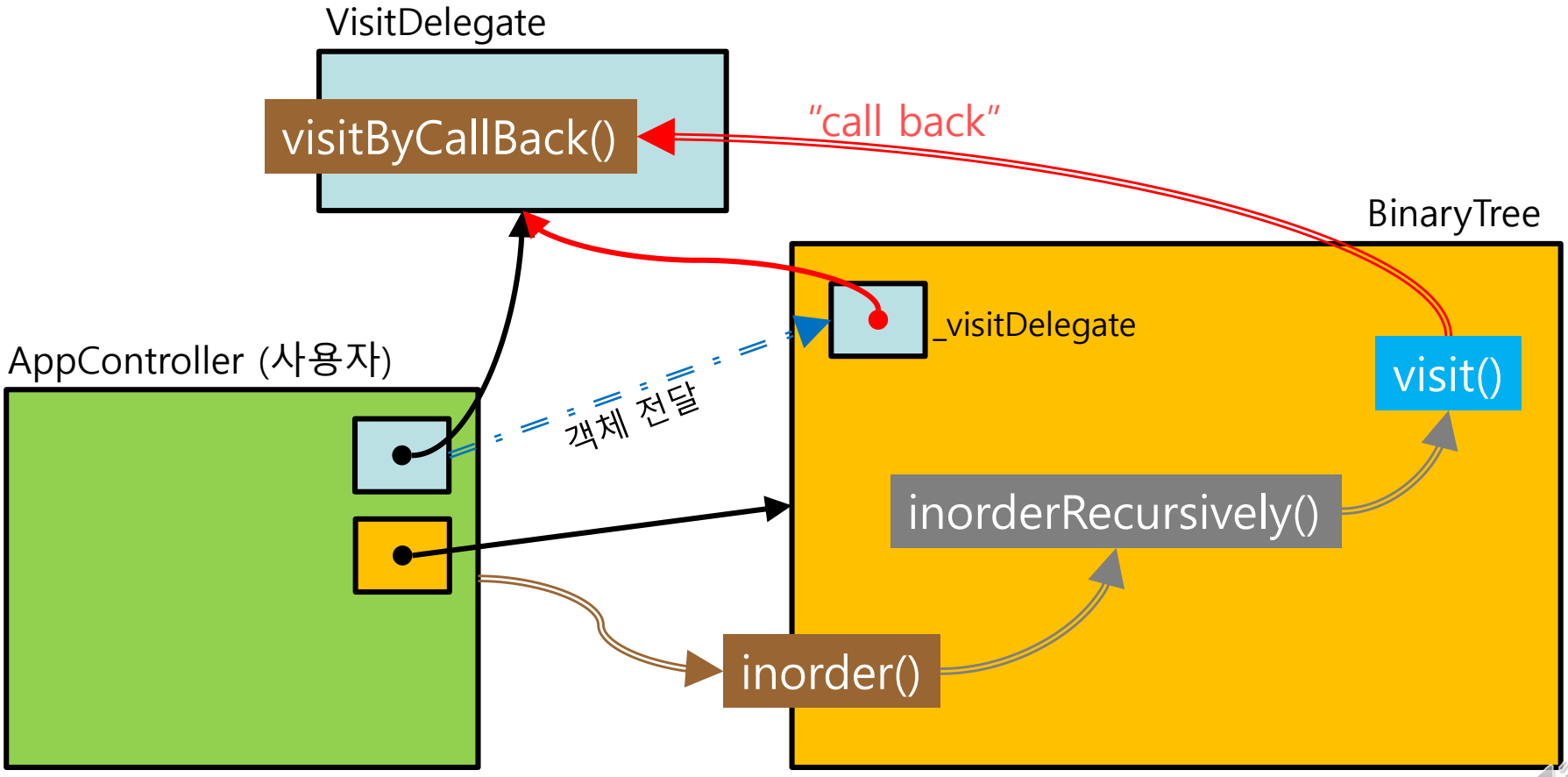
해결책은? [4]

- "BinaryTree" 객체는 전달받은 "VisitDelegate" 객체에게 "visitByCallBack()"을 실행하게 한다.
- 이로써, <visit 의 행위는 사용자가 정의하고, 실행은 "BinaryTree"가 한다> 는 우리의 목적이 달성될 수 있다.



Call Back?

- AppController 객체는 VisitDelegate 객체를 생성하여 소유하고 있다. 그리고 사전(BinaryTree) 객체에게 "inorder()" 라는 일을 시키려고 call 하고 있다. 사전 객체는 자신의 사용자인 AppController 객체가 소유한, 그리고 지금 자신에게 전달한 VisitDelegate 객체에게 일을 시킨다. 이 행위는 자신을 call 한 객체 쪽으로 되돌아 call 을 하는, 즉 call back 하는 셈이 된다.



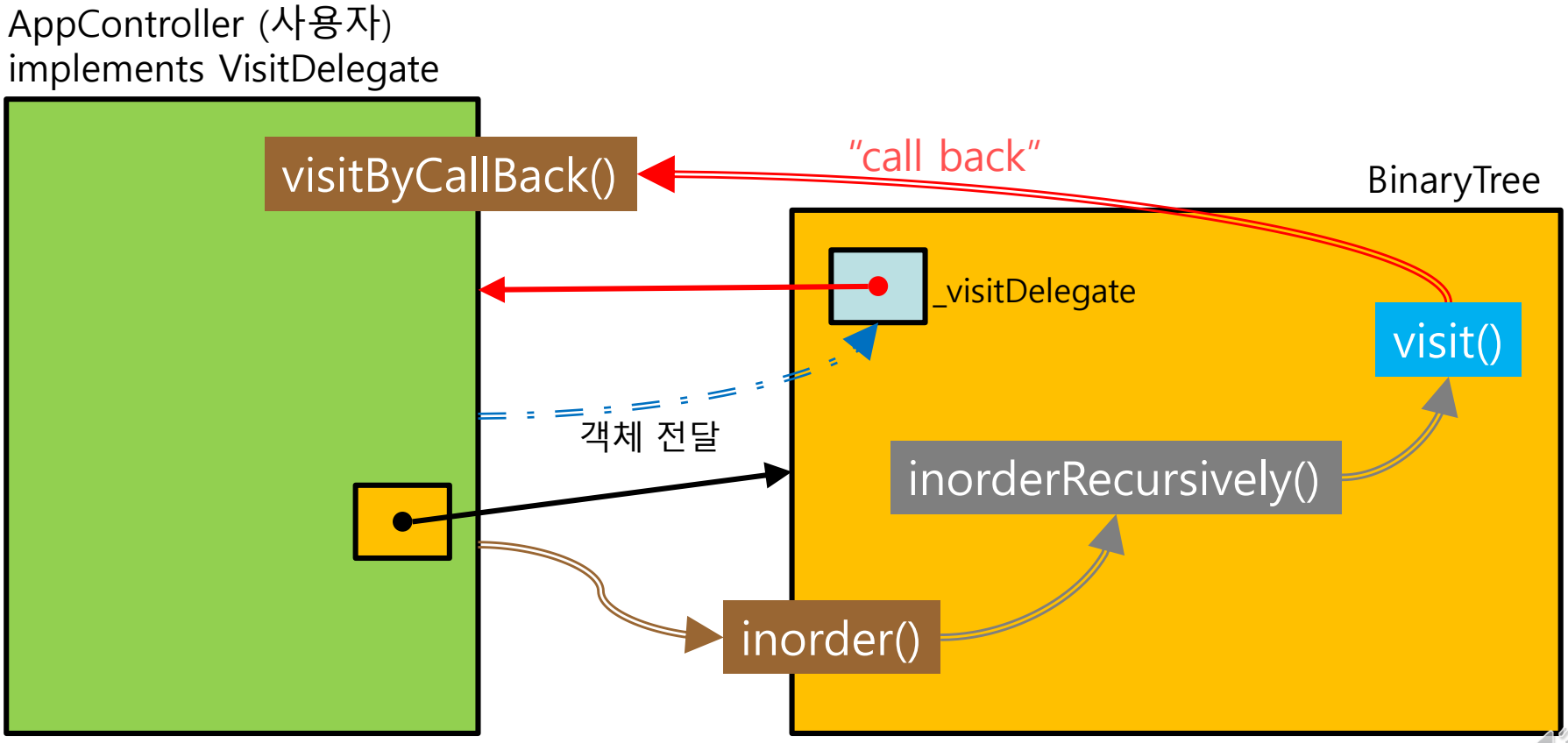
VisitDelegate 는 class?

- "VisitDelegate" 는 class 로 선언해도 된다.
- 그런데, "visitByCallBack()" method 는:
  - 사용자 (AppController) 쪽에서 구현해야 할 method 이다.
  - 사용자 쪽에 구현할 수 만 있으면 되므로, delegate 객체로 사용할 별도의 class 를 정의하기 보다, 사용자 (AppController) 객체 자신을 delegate 객체로 사용할 수 있다.

- 그렇다면, "VisitDelegate" 는 interface 로 선언하자!
  - 그리고, class "AppController" 가 interface "VisitDelegate" 를 구현하게 한다.
  - "BinaryTree" 객체에게는, call back 의 delegate 는 "AppController" 객체 임을 알린다.

Call Back?

- AppController 객체는 VisitDelegate 객체를 생성하여 소유하고 있다. 그리고 사전(BinaryTree) 객체에게 "inorder()" 라는 일을 시키려고 call 하고 있다. 사전 객체는 자신의 사용자인 AppController 객체가 소유한, 그리고 지금 자신에게 전달한 VisitDelegate 객체에게 일을 시킨다. 이 행위는 자신을 call 한 객체 쪽으로 되돌아 call 을 하는, 즉 call back 하는 셈이 된다.



Call Back 함수는 Interface 로 정의하자

```
public interface  VisitDelegate<T>
{
    public void  visitByCallBack (T anObject) ;
}
```

- 사용자와 "BinaryTree"가 서로 알고 있어야 할 함수를 Interface로 정의한다.
  - 구현은 사용자 클래스에서
  - 사용 (call back) 은 위임하는 클래스 (BinaryTree) 에서
- Call Back 함수
  - 사용자 class 에 정의되어 있지만, 위임하는 객체가 call 하게 되는 함수
  - 여기서는 "visitByCallBack()" 이 call back 함수이다.

사용자 쪽의 선언

- Call Back을 위한 Interface는 당연히 사용자 클래스에서 구현

```
public class AppController implements VisitDelegate<T>
{
    ……
    BinaryTree<T> _binaryTree = new BinaryTree<T>() ;
    ……
    @Override
    public void  visitByCallback (T anObject) {
        // TODO Auto-generated method stub
        …… // visit 에서 실제로 할 일
    }

    public ……  // inorder treversal 을 실행하는 함수
    {
        ……
        // "BinaryTree" 객체에게 사용자가 누구인지를 알려준다.
        this.binaryTree().setVisitDelegate (this) ;
        this.binaryTree().inorder() ; // "BinaryTree" 객체에게 탐색을 하게 한다.
        ……
    }
}
```

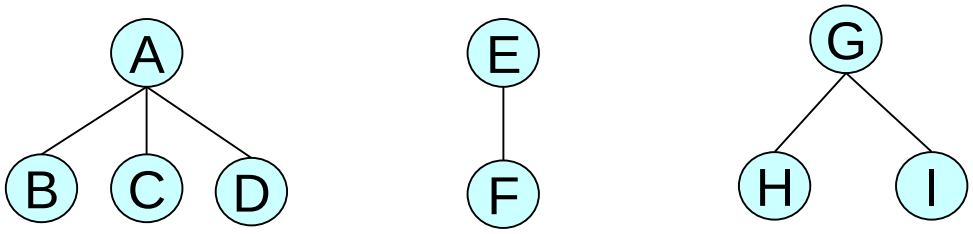
피사용자인 "BinaryTree" 쪽은?

```
public class  BinaryTree<T>
{
    private VisitDelegate<T>  _visitDelegate ;
        // VisitDelegate 를 구현한 임의의 Class 의 객체를 의미한다.
    ……
    public void  setVisitDelegate (VisitDelegate<T> newVisitDelegate)
    {   // Setter for _visitDelegate
        this._visitDelegate = newVisitDelegate ;
    }
    public VisitDelegate<T>  visitDelegate() {
        return  this._visitDelegate ;
    }
    ……
    private void  visit (T anElement)
    {
        this.visitDelegate().visitByCallBack (anElement) ;
    }
    ……
}
```

숲 (Forest)

숲 (Forest)

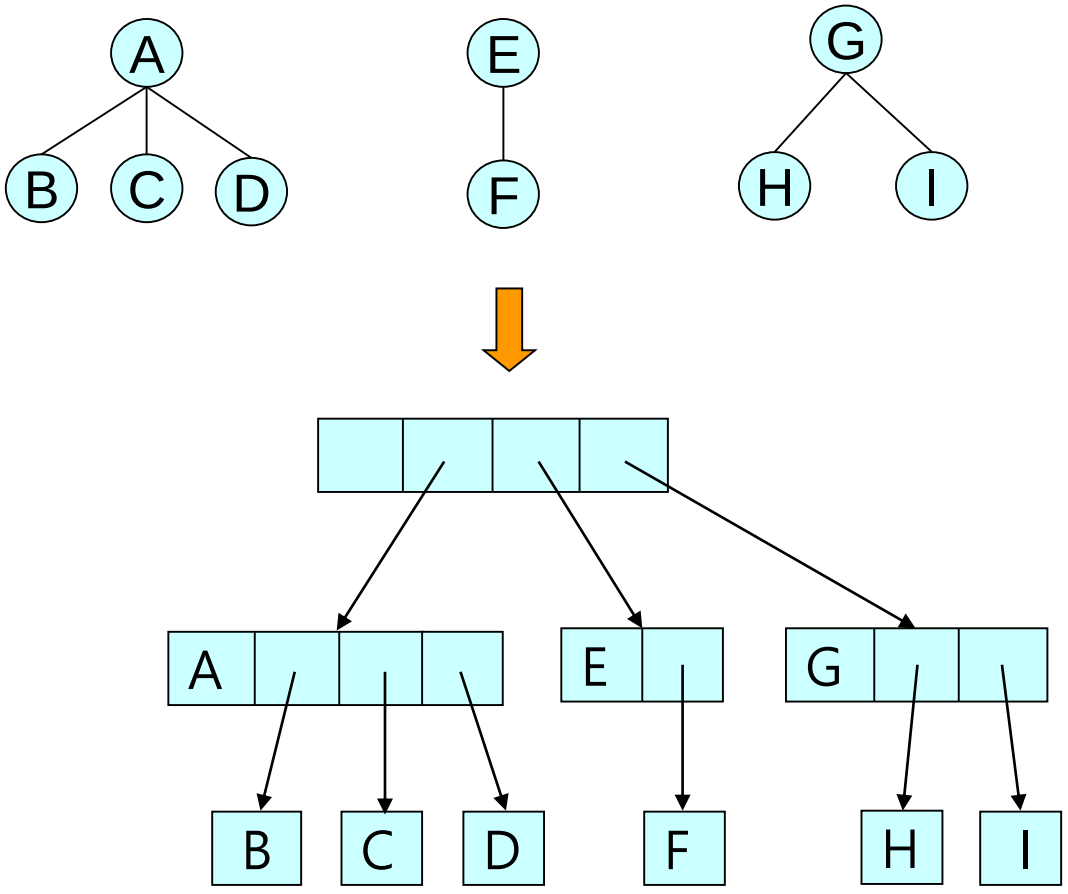
- 숲 (forest): 0 개 이상의 서로 원소가 겹치지 않는 트리들의 집합.



- 트리의 일반화
  - Empty Tree? ⇒ Empty Forest!

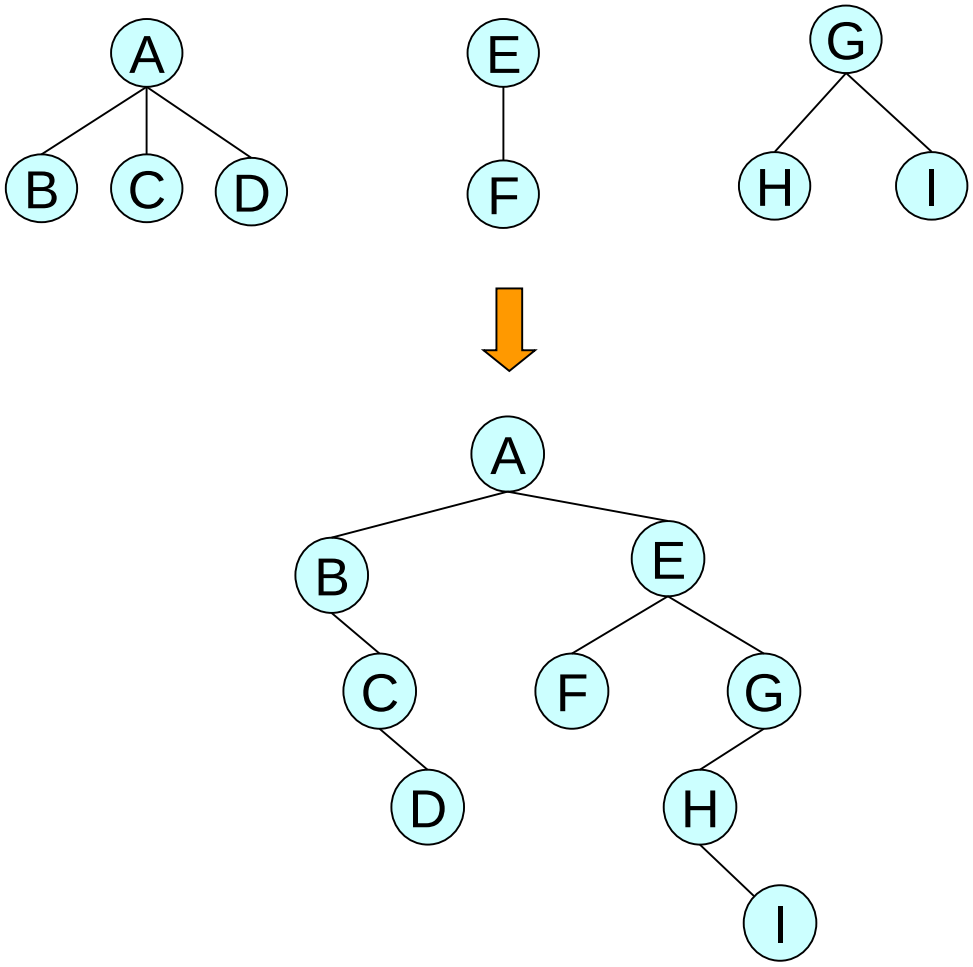
숲 (Forest) 의 표현 [1]

- 트리의 리스트: 가변길이 노드 사용



숲 (Forest) 의 표현 [2]

- 숲을 이진트리로 변환:
  - Left-Child-Right-Sibling, 그리고 시계방향으로 45도 회전



숲의 탐색 (Traversal)

- 전위 (preorder) 탐색
- 중위 (inorder) 탐색
- 후위 (postorder) 탐색

숲의 전위 탐색

- F = { T1, T2, …, Tn }
- Forest Preorder (F)

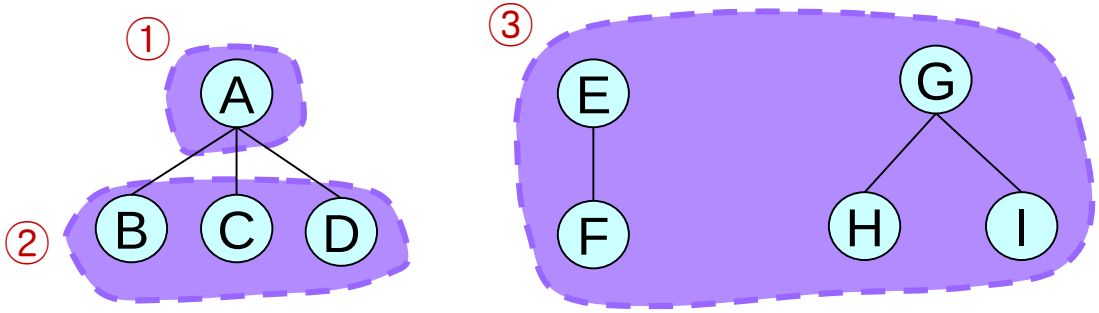
  If (F 가 공집합이 아니라면) {

  T1 의 루트를 방문 ;

  T1 의 부트리들을 Forest Preorder 로 탐색 ;

  F 의 나머지 트리들 { T2 , … , Tn } 을 Forest Preorder 로 탐색 ;

  }



- 예:
  - 숲: A – B – C – D – E – F – G – H – I
  - 이진 트리: A – B – C – D – E – F – G – H – I

숲의 중위 탐색

- F = { T1, T2, …, Tn }
- Forest Inorder (F)

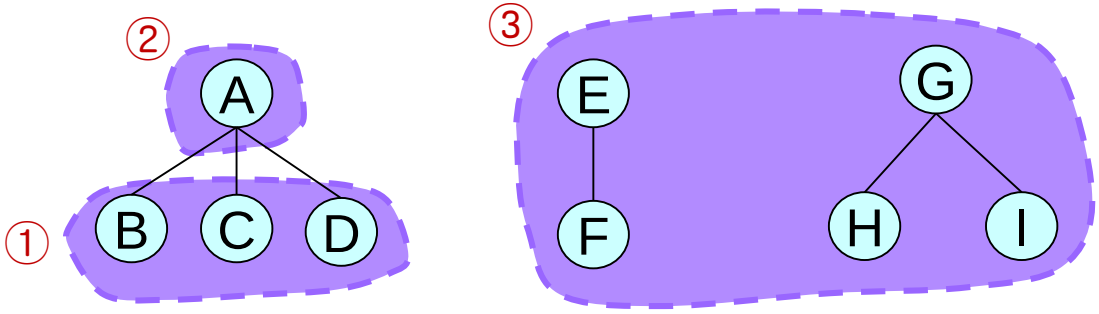
  If (F 가 공집합이 아니라면) {

  T1 의 부트리들을 Forest Inorder 로 탐색 ;

  T1 의 루트를 방문 ;

  F 의 나머지 트리들 { T2 , … , Tn } 을 Forest Inorder 로 탐색 ;

  }



- 예:
  - 숲: B – C – D – A – F – E – H – I – G
  - 이진 트리: B – C – D – A – F – E – H – I – G

숲의 후위 탐색

- F = { T1, T2, …, Tn }
- Forest Postorder (F)

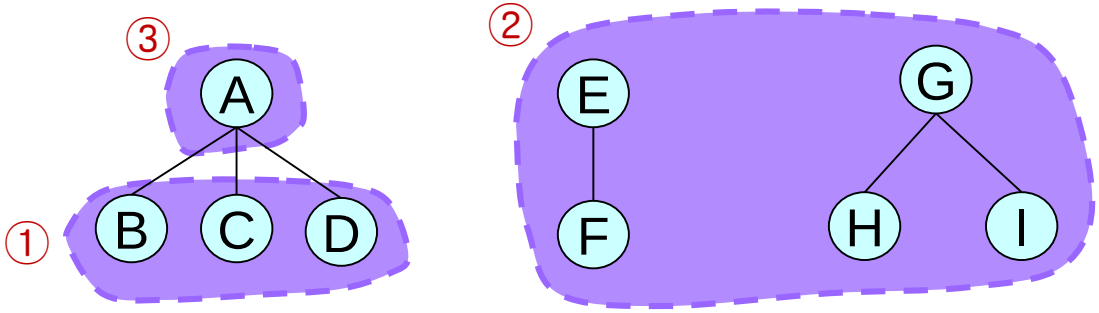
  If (F 가 공집합이 아니라면) {

  T1 의 부트리들을 Forest Postorder 로 탐색 ;

  F 의 나머지 트리들 { T2 , … , Tn } 을 Forest Postorder 로 탐색 ;

  T1 의 루트를 방문 ;

  }



- 예:
  - 숲: D – C – B – F – I – H – G – E – A
  - 이진 트리: D – C – B – F – I – H – G – E – A

End of

"Tree 2"