# Composite Pattern

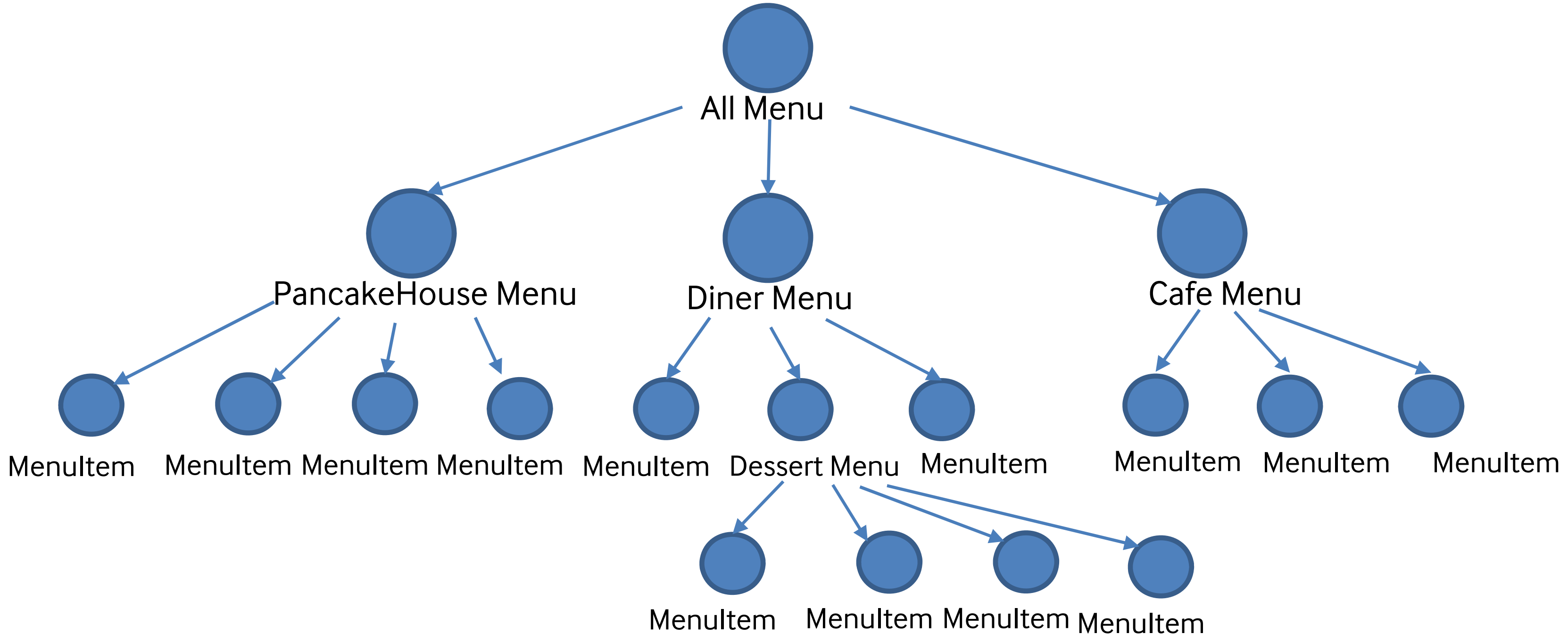
# Contents

- Tree Structure
- Structure of Composite Pattern
- Combining with Iterator pattern

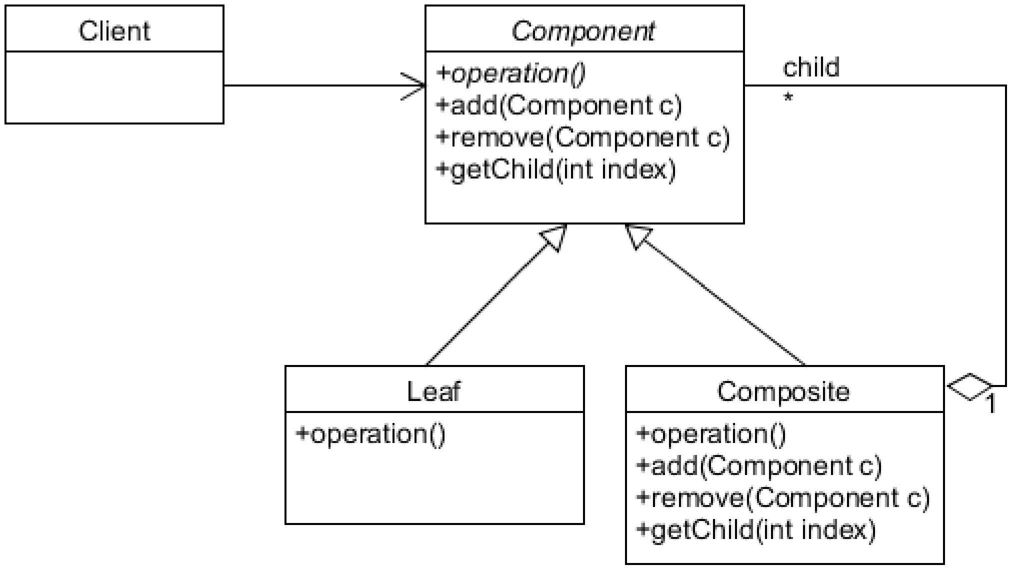
# Composite Pattern

- Purpose
  - Facilitates the creation of object hierarchies where each object can be treated independently or as a set of nested objects through the same interface.
- Use When
  - Hierarchical representations of objects are needed.
  - Objects and compositions of objects should be treated uniformly
    - This is called recursive composition.

# Composite and Leaf



# Composite Pattern Diagram



# Component Class

```
public class MenuComponent {
    public void add(MenuComponent menuComponent) {
        throw new UnsupportedOperationException();
    }
    public void remove(MenuComponent menuComponent) {
        throw new UnsupportedOperationException();
    }
    public void getChild(int i) {
        throw new UnsupportedOperationException();
    }
    public String getName() {
        throw new UnsupportedOperationException();
    }
    public String getDescription() {
        throw new UnsupportedOperationException();
    }
    public double getPrice() {
        throw new UnsupportedOperationException();
    }
    public double isVegetarian() {
        throw new UnsupportedOperationException();
    }
    public double print() {
        throw new UnsupportedOperationException();
    }
}
```

# Leaf Class

```java
public class MenuItem extends MenuComponent {
    String name;
    String description;
    boolean vegetarian;
    double price;
    public MenuItem(String name, String description, boolean vegetarian,
        double price) {
                   ...
    }
    public String getName() {
        return name;
    }
    public String getDescription() {
        return description;
    }
    public double getPrice() {
        return price;
    }
    public boolean isVegetarian() {
        return vegetarian;
    }
    public void print() {
        System.out.print("  " + getName());
        if (this.isVegetarian()) System.out.println("(v)");
             System.out.println(", " + getPrice());
        System.out.println("   --" + getDescription());
    }
}
```

# Composite Class

```
public class Menu extends MenuComponent {
    ArraryList menuComponents = new ArrayList();
    String name;
    String description;
    public Menu(String name, String description) {
        this.name = name;
        this.description = description;
    }
    public void add(MenuComponent menuComponent) {
        menuComponents.add(menuComponent);
    }
    public void remove(MenuComponent menuComponent) {
        menuComponents.remove(menuComponent);
    }
    public MenuComponent getChild(int i) {
        return (MenuComponent)menuComponents.get(i);
    }
    public String getName() { return name; }
    public String getDescription () { return description; }
    public void print() {
        Iterator iterator = menuComponents.iterator();
        while (iterator.hasNext()) {
            MenuComponent menuComponent = (MenuComponent)iterator.next();
            menuComponent.print();
        }
    }
}
```

# Printing All Menus

```
public class Waitress {
    MenuComponent allMenus;

    public Waitress(MenuCompoent allMenus) {
         this.allMenus = allMenus;
    }

    public void printMenu() {
         allMenus.print();
    }
}
```

# Extending Iterator to support Composite Traversal

```
public class Menu extends MenuComponent {
   Iterator iterator = null;
   // other code here doesn't change
   public Iterator createIterator() {
         if (iterator == null)
             iterator = new CompositeIterator(menuComponents.iterator());
         return iterator;
   }
}

public class MenuItem extends MenuComponent {
    // other code here doesn't change
    public Iterator createIterator() {
          return new NullIterator();
    }
}
```

Null Object

# Iterator for Composite

```
public class CompositeIterator extends Iterator {
    Stack stack = new Stack();
    public CompositeIterator(Iterator iterator) {
        stack.push(iterator);
    }
    public Object next() {
        if (hasNext()) {
            Iterator iterator = (Iterator)stack.peek();
            MenuComponent component = (MenuComponent)iterator.next();
            if (component instance of Menu)
                stack.push(component.createIterator());
            return component;
        }
        else return null;
    }
    public boolean hasNext() {
        if (stack.empty()) return false;
        Iterator iterator = (Iterator)stack.peek();
        if (!iterator.hasNext()) {
            stack.pop();
            return hasNext();
        } else return true;
    }
    public void remove() {
        throw new UnsupportedOperationException();
    }
}
```

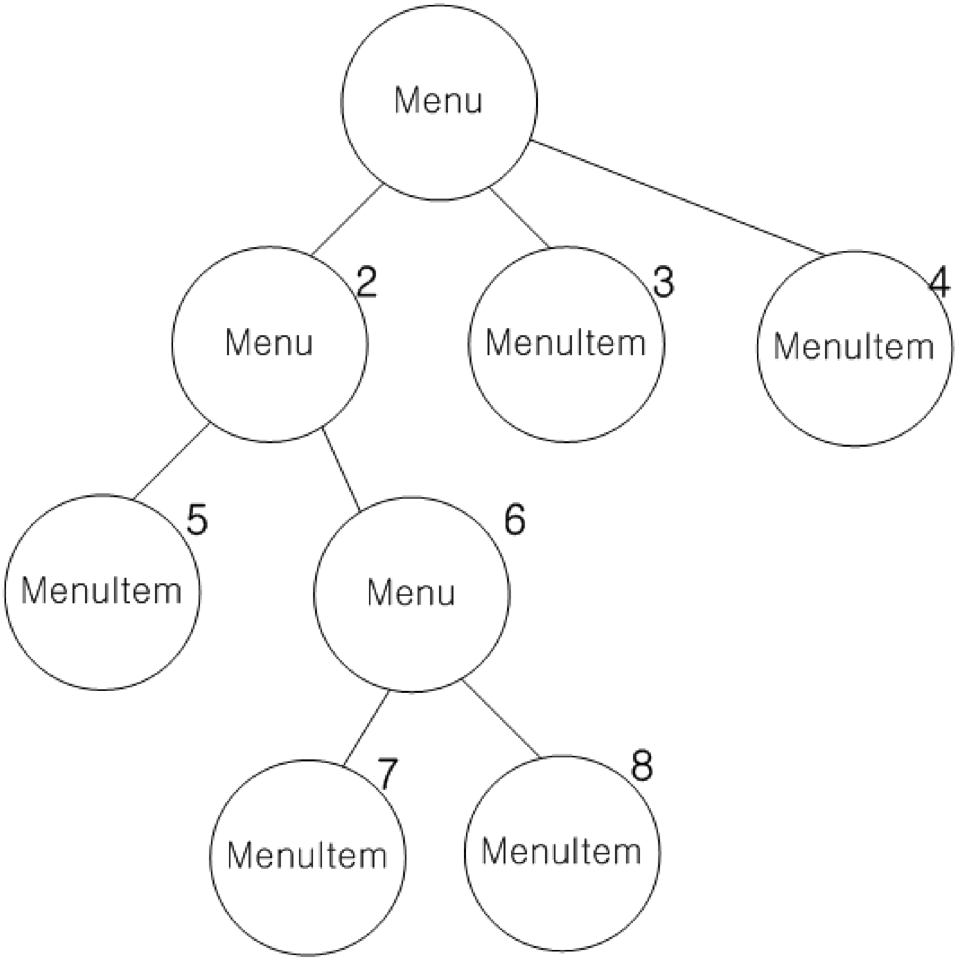
# Null Iterator

```
public class NullIterator implements Iterator {
    public Object next() {
          return null;
    }
    public boolean hasNext() {
          return false;
    }
    public void remove() {
          throw new UnsupportedOperationException();
    }
}
```

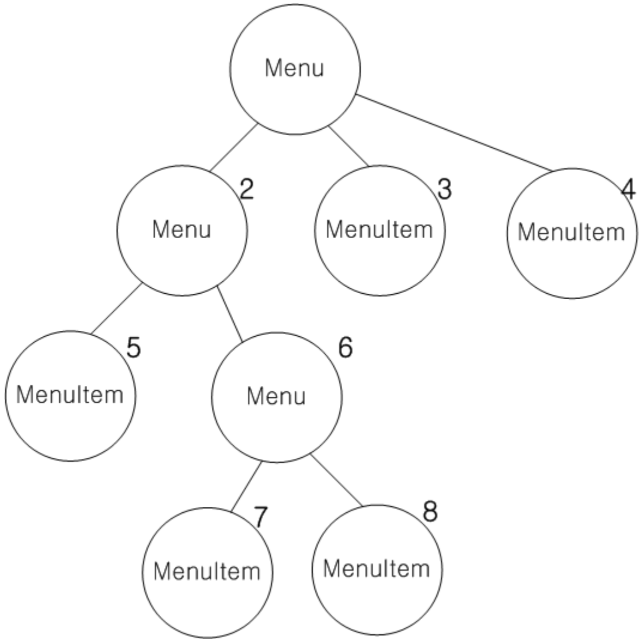
# Printing the Vegetarian Menu

```
public class Waitress {
    MenuComponent allMenus;
    public Watiress(MenuComponent allMenus) {
        this.allMenus = allMenus;
    }
    public void printMenu() {
        allMenus.print();
    }
    public void printVegetarianMenu() {
        Iterator iterator = allMenus.createIterator();
        System.out.println("\nVEGETARIAN MENU\n-------");
        while (iterator.hasNext()) {
            MenuComponent menuComponent = (MenuComponent)iterator.next();
            try {
                if (menuComponent.isVegetarian()) menuComponent.print();
            } catch (UnsupportedOperationException e) {}
        }
    }
}
```

# Simulate the Algorithm



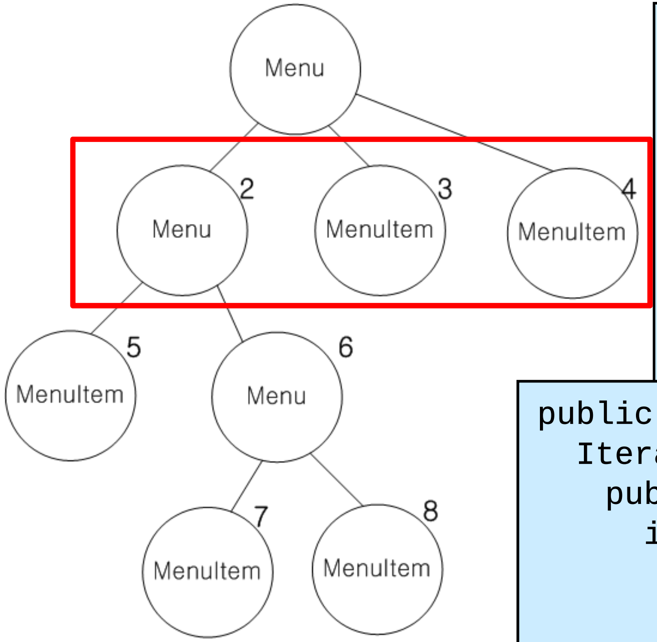
# Simulate the Algorithm



```
public class CompositeIterator extends Iterator {
  Stack stack = new Stack();
  public CompositeIterator(Iterator iterator) {
    stack.push(iterator);
  }
  public Object next() {
    if (hasNext()) {
        Iterator iterator = (Iterator)stack.peek();
        MenuComponent component=(MenuComponent)iterator.next();
        if (component instance of Menu)
            stack.push(component.createIterator());
        return component;
    }
    else return null;
  }
  public boolean hasNext() {
    if (stack.empty()) return false;
    Iterator iterator = (Iterator)stack.peek();
    if (!iterator.hasNext()) {
        stack.pop();
        return hasNext();
    } else return true;
  }
}
```

```
public void printVegetarianMenu() {
    Iterator iterator = allMenus.createIterator();
       ....
    }
}
```

# Simulate the Algorithm

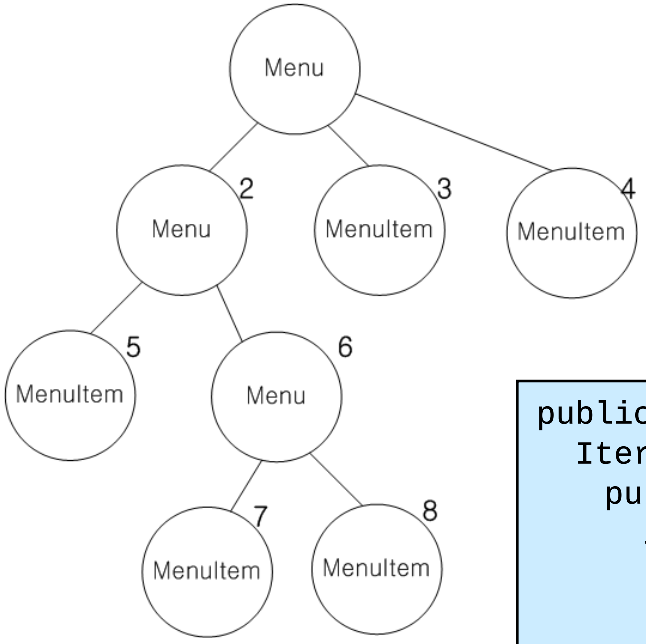


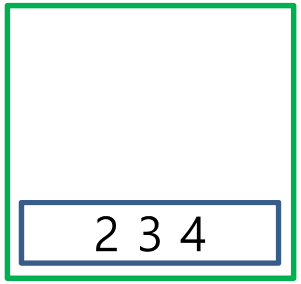
```
public class CompositeIterator extends Iterator {
  Stack stack = new Stack();
  public CompositeIterator(Iterator iterator) {
    stack.push(iterator);
  }
  public Object next() {
    if (hasNext()) {
        Iterator iterator = (Iterator)stack.peek();
        MenuComponent component=(MenuComponent)iterator.next();
                  enu)
                  eateIterator());



                  ;
                  tack.peek();
    if (!iterator.hasNext()) {
        stack.pop();
        return hasNext();
    } else return true;
  }
}
```

```
public class Menu extends MenuComponent {
  Iterator iterator = null;
   public Iterator createIterator() {
     if (iterator == null)
       iterator = new CompositeIterator
           (menuComponents.iterator());
     return iterator;
   }
}
```

```
public void printVegetarianMenu() {
    Iterator iterator = allMenus.createIterator();
       ....
    }
}
```

# Simulate the Algorithm



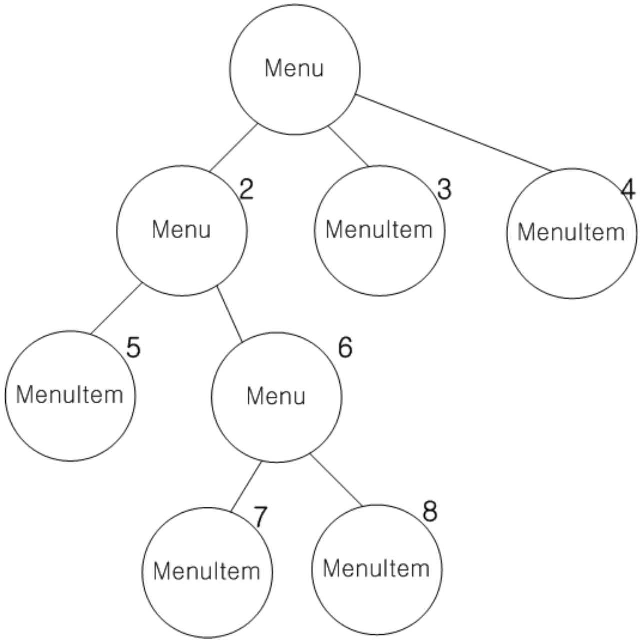


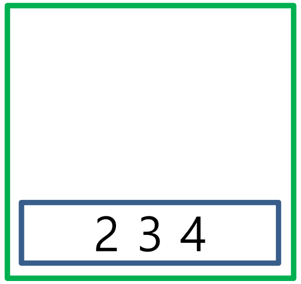
```
public class CompositeIterator extends Iterator {
  Stack stack = new Stack();
  public CompositeIterator(Iterator iterator) {
    stack.push(iterator);
  }
  public Object next() {
    if (hasNext()) {
        Iterator iterator = (Iterator)stack.peek();
        MenuComponent component=(MenuComponent)iterator.next();
                                         enu)
                                         eateIterator());
                                         ;
                                         tack.peek();
    if (!iterator.hasNext()) {
        stack.pop();
        return hasNext();
    } else return true;
  }
}
```

```
public class Menu extends MenuComponent {
  Iterator iterator = null;
    public Iterator createIterator() {
      if (iterator == null)
        iterator = new CompositeIterator
          (menuComponents.iterator());
      return iterator;
    }
}
```

```
public void printVegetarianMenu() {
    Iterator iterator = allMenus.createIterator();
        ....
    }
}
```

# Simulate the Algorithm

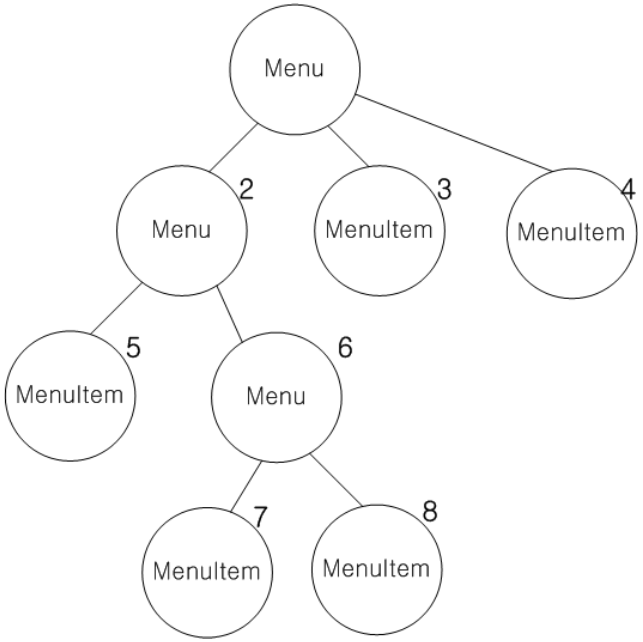


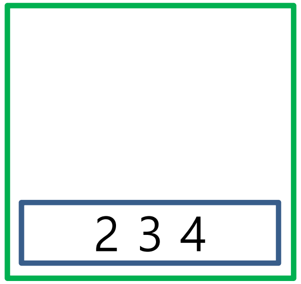


```
public Object next() {
    if (hasNext()) {
        Iterator iterator = (Iterator)stack.peek();
        MenuComponent component=(MenuComponent)iterator.next();
        if (component instance of Menu)
            stack.push(component.createIterator());
        return component;
    }
    else return null;
}
public boolean hasNext() {
    if (stack.empty()) return false;
    Iterator iterator = (Iterator)stack.peek();
    if (!iterator.hasNext()) {
        stack.pop();
        return hasNext();
    } else return true;
}
```

```
public void printVegetarianMenu() {
    Iterator iterator = allMenus.createIterator();
    System.out.println("\nVEGETARIAN MENU\n-------");
    while (iterator.hasNext()) {
        MenuComponent menuComponent =
            (MenuComponent) iterator.next();
        try {
            if (menuComponent.isVegetarian())
                menuComponent.print();
        } catch (UnsupportedOperationException e) {}
    }
}
```

# Simulate the Algorithm

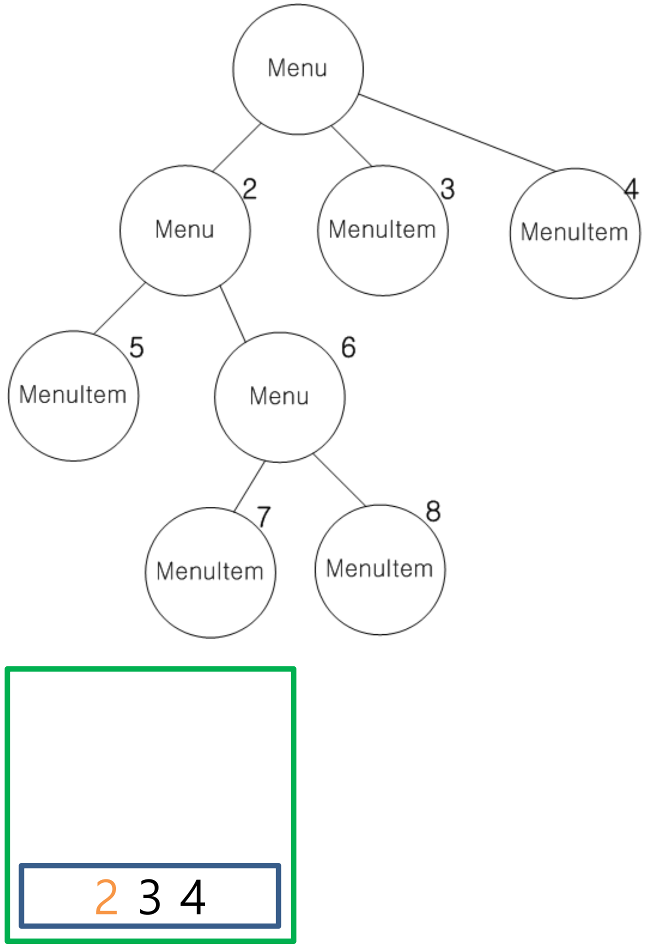




```
public Object next() {
    if (hasNext()) {
        Iterator iterator = (Iterator)stack.peek();
        MenuComponent component=(MenuComponent)iterator.next();
        if (component instance of Menu)
            stack.push(component.createIterator());
        return component;
    }
    else return null;
}
public boolean hasNext() {
    if (stack.empty()) return false;
    Iterator iterator = (Iterator)stack.peek();
    if (!iterator.hasNext()) {
        stack.pop();
        return hasNext();
    } else return true;
}
```

```
public void printVegetarianMenu() {
    Iterator iterator = allMenus.createIterator();
    System.out.println("\nVEGETARIAN MENU\n-------");
    while (iterator.hasNext()) {
        MenuComponent menuComponent =
            (MenuComponent) iterator.next();
        try {
            if (menuComponent.isVegetarian())
                menuComponent.print();
        } catch (UnsupportedOperationException e) {}
    }
}
```

# Simulate the Algorithm

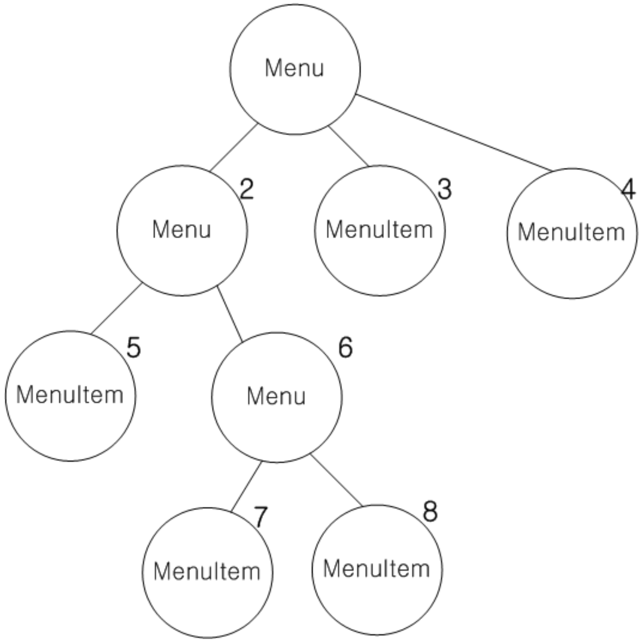


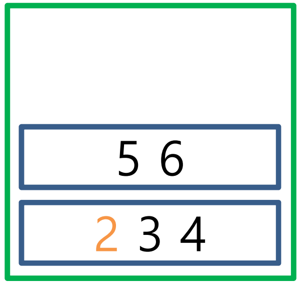
```
public Object next() {
    if (hasNext()) {
        Iterator iterator = (Iterator)stack.peek();
        MenuComponent component=(MenuComponent)iterator.next();
        if (component instance of Menu)
            stack.push(component.createIterator());
        return component;
    }
    else return null;
}
public boolean hasNext() {
    if (stack.empty()) return false;
    Iterator iterator = (Iterator)stack.peek();
    if (!iterator.hasNext()) {
        stack.pop();
        return hasNext();
    } else return true;
}
```

2

```
public void printVegetarianMenu() {
    Iterator iterator = allMenus.createIterator();
    System.out.println("\nVEGETARIAN MENU\n-------");
    while (iterator.hasNext()) {
        MenuComponent menuComponent =
            (MenuComponent) iterator.next();
        try {
            if (menuComponent.isVegetarian())
                menuComponent.print();
        } catch (UnsupportedOperationException e) {}
    }
}
```

# Simulate the Algorithm





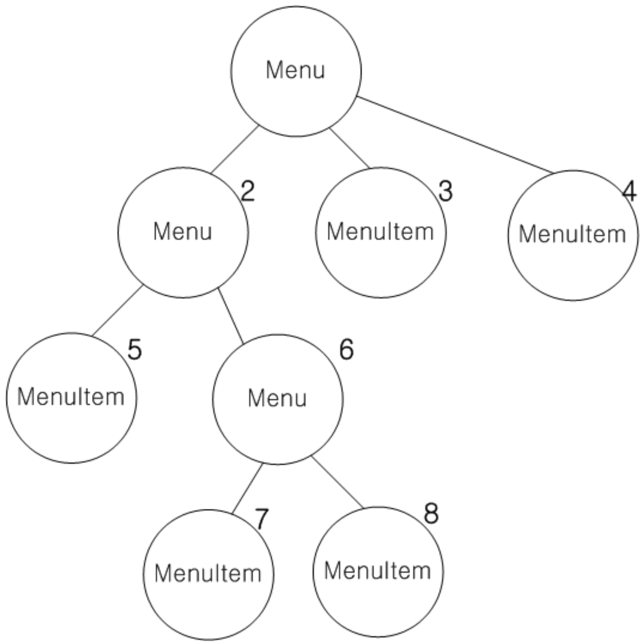
5 6

2 3 4

```
public Object next() {
    if (hasNext()) {
        Iterator iterator = (Iterator)stack.peek();
        MenuComponent component=(MenuComponent)iterator.next();
        if (component instance of Menu)
            stack.push(component.createIterator());
        return component;
    }
    else return null;
}
public boolean hasNext() {
    if (stack.empty()) return false;
    Iterator iterator = (Iterator)stack.peek();
    if (!iterator.hasNext()) {
        stack.pop();
        return hasNext();
    } else return true;
}
```

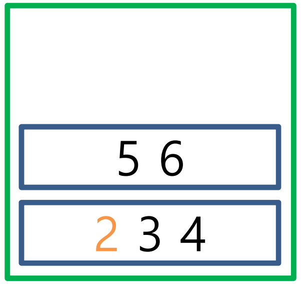
```
public void printVegetarianMenu() {
    Iterator iterator = allMenus.createIterator();
    System.out.println("\nVEGETARIAN MENU\n-------");
    while (iterator.hasNext()) {
        MenuComponent menuComponent =
            (MenuComponent) iterator.next();
        try {
            if (menuComponent.isVegetarian())
                menuComponent.print();
        } catch (UnsupportedOperationException e) {}
    }
}
```

# Simulate the Algorithm



```
public Object next() {
    if (hasNext()) {
        Iterator iterator = (Iterator)stack.peek();
        MenuComponent component=(MenuComponent)iterator.next();
        if (component instance of Menu)
            stack.push(component.createIterator());
        return component;
    }
    else return null;
}
public boolean hasNext() {
    if (stack.empty()) return false;
    Iterator iterator = (Iterator)stack.peek();
    if (!iterator.hasNext()) {
        stack.pop();
        return hasNext();
    } else return true;
}
```

2

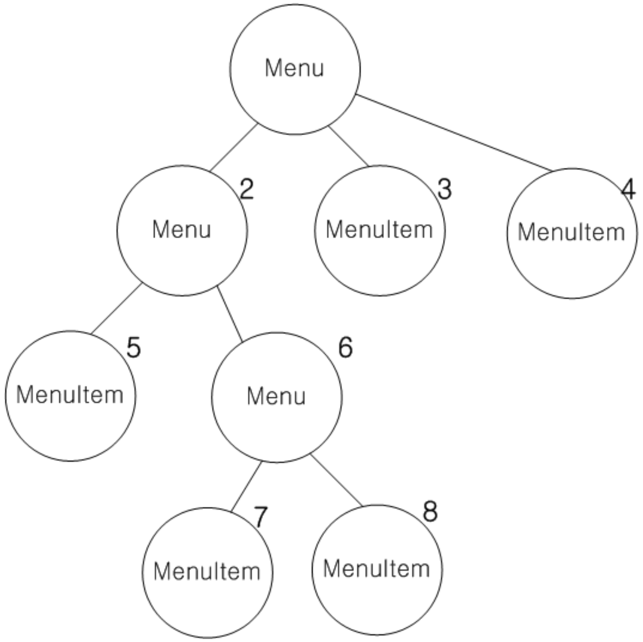


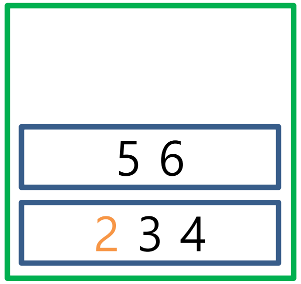
```
public void printVegetarianMenu() {
    Iterator iterator = allMenus.createIterator();
    System.out.println("\nVEGETARIAN MENU\n-------");
    while (iterator.hasNext()) {
        MenuComponent menuComponent =
            (MenuComponent) iterator.next();
        try {
            if (menuComponent.isVegetarian())
                menuComponent.print();
        } catch (UnsupportedOperationException e) {}
    }
}
```

2

visited: 2

# Simulate the Algorithm





5 6

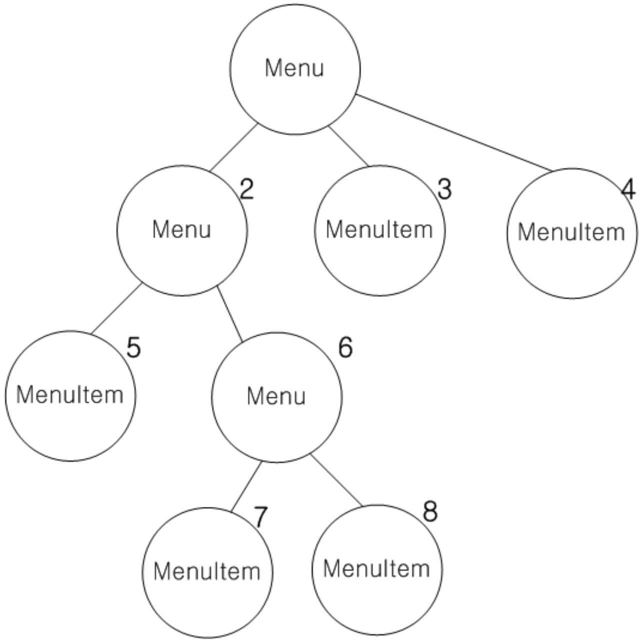
2 3 4

visited: 2

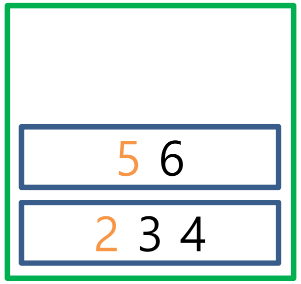
```
public Object next() {
    if (hasNext()) {
        Iterator iterator = (Iterator)stack.peek();
        MenuComponent component=(MenuComponent)iterator.next();
        if (component instance of Menu)
            stack.push(component.createIterator());
        return component;
    }
    else return null;
}
public boolean hasNext() {
    if (stack.empty()) return false;
    Iterator iterator = (Iterator)stack.peek();
    if (!iterator.hasNext()) {
        stack.pop();
        return hasNext();
    } else return true;
}
```

```
public void printVegetarianMenu() {
    Iterator iterator = allMenus.createIterator();
    System.out.println("\nVEGETARIAN MENU\n-------");
    while (iterator.hasNext()) {
        MenuComponent menuComponent =
            (MenuComponent) iterator.next();
        try {
            if (menuComponent.isVegetarian())
                menuComponent.print();
        } catch (UnsupportedOperationException e) {}
    }
}
```

# Simulate the Algorithm



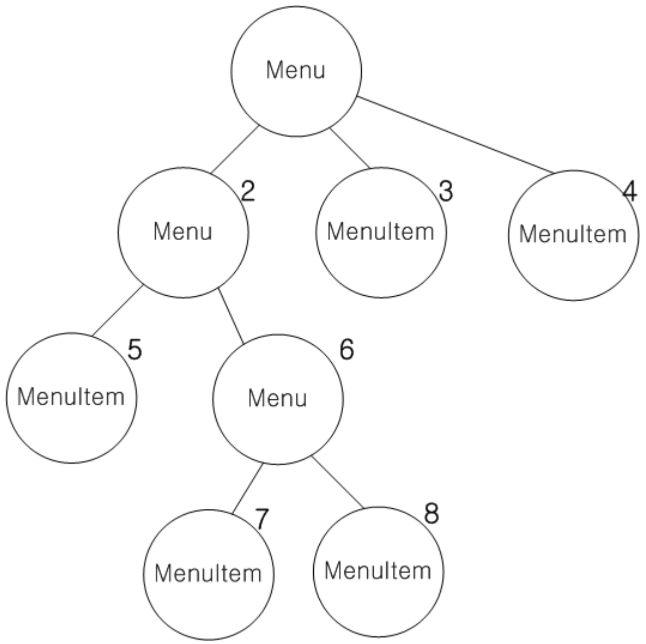
```
public Object next() {
    if (hasNext()) {
        Iterator iterator = (Iterator)stack.peek();      5
        MenuComponent component=(MenuComponent)iterator.next();
        if (component instance of Menu)
            stack.push(component.createIterator());
        return component;
    }
    else return null;
}
public boolean hasNext() {
    if (stack.empty()) return false;
    Iterator iterator = (Iterator)stack.peek();
    if (!iterator.hasNext()) {
        stack.pop();
        return hasNext();
    } else return true;
}
```

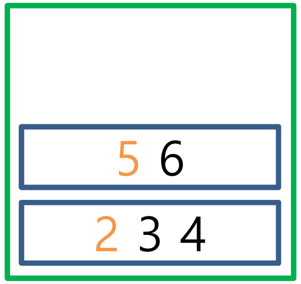


```
public void printVegetarianMenu() {
    Iterator iterator = allMenus.createIterator();
    System.out.println("\nVEGETARIAN MENU\n-------");
    while (iterator.hasNext()) {
        MenuComponent menuComponent =
            (MenuComponent) iterator.next();
        try {
            if (menuComponent.isVegetarian())
                menuComponent.print();
        } catch (UnsupportedOperationException e) {}
    }
}
```

visited: 2

# Simulate the Algorithm





5 6

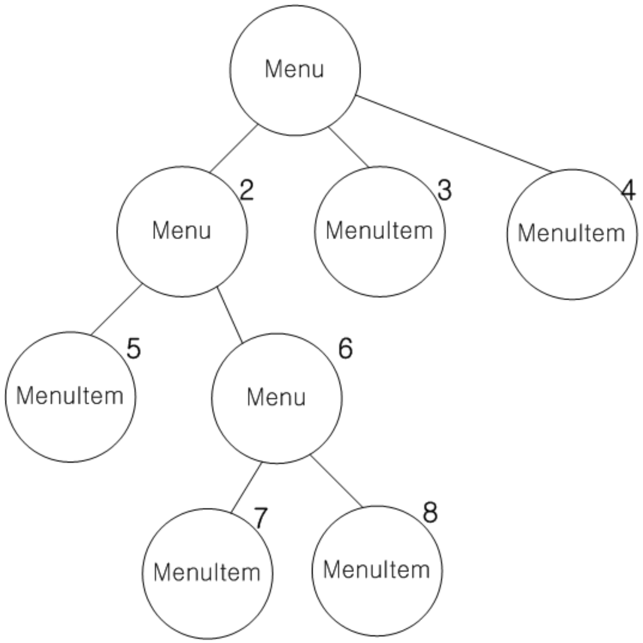
2 3 4

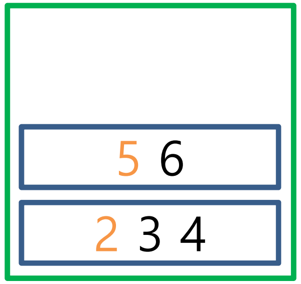
visited: 2

```
public Object next() {
    if (hasNext()) {
        Iterator iterator = (Iterator)stack.peek();   5
        MenuComponent component=(MenuComponent)iterator.next();
        if (component instance of Menu)
            stack.push(component.createIterator());
        return component;
    }
    else return null;                  5
}
public boolean hasNext() {
    if (stack.empty()) return false;
    Iterator iterator = (Iterator)stack.peek();
    if (!iterator.hasNext()) {
        stack.pop();
        return hasNext();
    } else return true;
}
```

```
public void printVegetarianMenu() {
    Iterator iterator = allMenus.createIterator();
    System.out.println("\nVEGETARIAN MENU\n-------");
    while (iterator.hasNext()) {
        MenuComponent menuComponent =
            (MenuComponent) iterator.next();
        try {
            if (menuComponent.isVegetarian())
                menuComponent.print();
        } catch (UnsupportedOperationException e) {}
    }
}
```

# Simulate the Algorithm



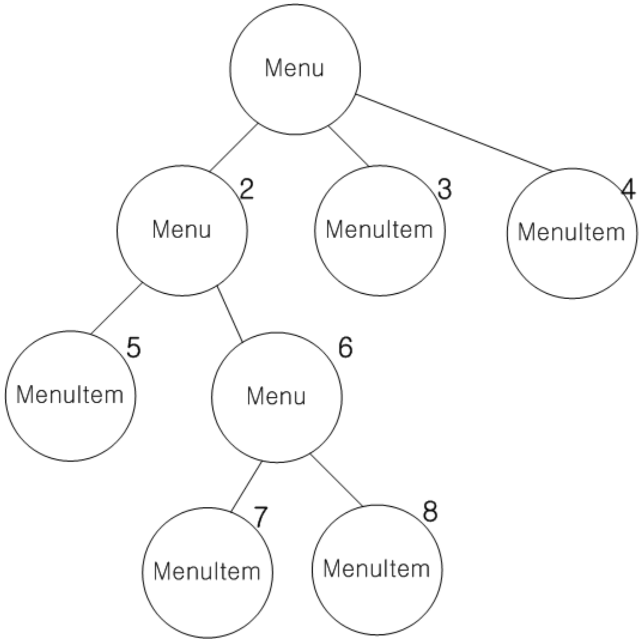


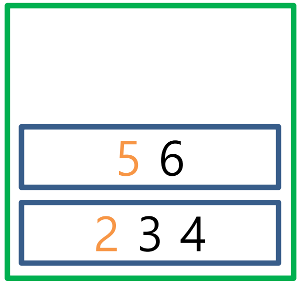
visited: 2, 5

```
public Object next() {
    if (hasNext()) {
        Iterator iterator = (Iterator)stack.peek();   5
        MenuComponent component=(MenuComponent)iterator.next();
        if (component instance of Menu)
            stack.push(component.createIterator());
        return component;
    }                                        5
    else return null;
}
public boolean hasNext() {
    if (stack.empty()) return false;
    Iterator iterator = (Iterator)stack.peek();
    if (!iterator.hasNext()) {
        stack.pop();
        return hasNext();
    } else return true;
}
```

```
public void printVegetarianMenu() {
    Iterator iterator = allMenus.createIterator();
    System.out.println("\nVEGETARIAN MENU\n-------");
    while (iterator.hasNext()) {
        MenuComponent menuComponent =
            (MenuComponent) iterator.next();  5
        try {
            if (menuComponent.isVegetarian())
                menuComponent.print();
        } catch (UnsupportedOperationException e) {}
    }
}
```

# Simulate the Algorithm



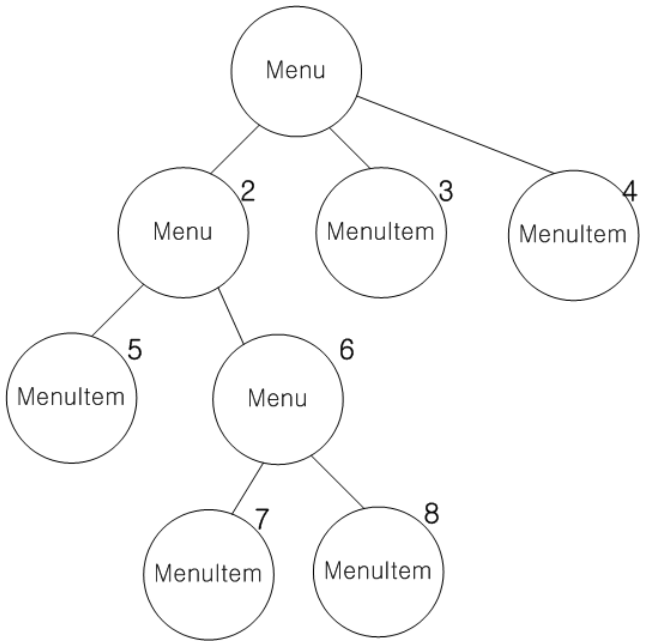


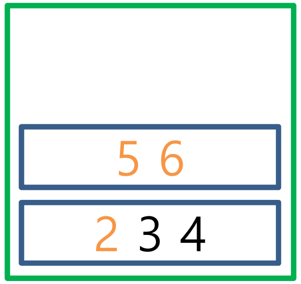
visited: 2, 5

```
public Object next() {
    if (hasNext()) {
        Iterator iterator = (Iterator)stack.peek();
        MenuComponent component=(MenuComponent)iterator.next();
        if (component instance of Menu)
            stack.push(component.createIterator());
        return component;
    }
    else return null;
}
public boolean hasNext() {
    if (stack.empty()) return false;
    Iterator iterator = (Iterator)stack.peek();
    if (!iterator.hasNext()) {
        stack.pop();
        return hasNext();
    } else return true;
}
```

```
public void printVegetarianMenu() {
    Iterator iterator = allMenus.createIterator();
    System.out.println("\nVEGETARIAN MENU\n-------");
    while (iterator.hasNext()) {
        MenuComponent menuComponent =
            (MenuComponent) iterator.next();
        try {
            if (menuComponent.isVegetarian())
                menuComponent.print();
        } catch (UnsupportedOperationException e) {}
    }
}
```

# Simulate the Algorithm





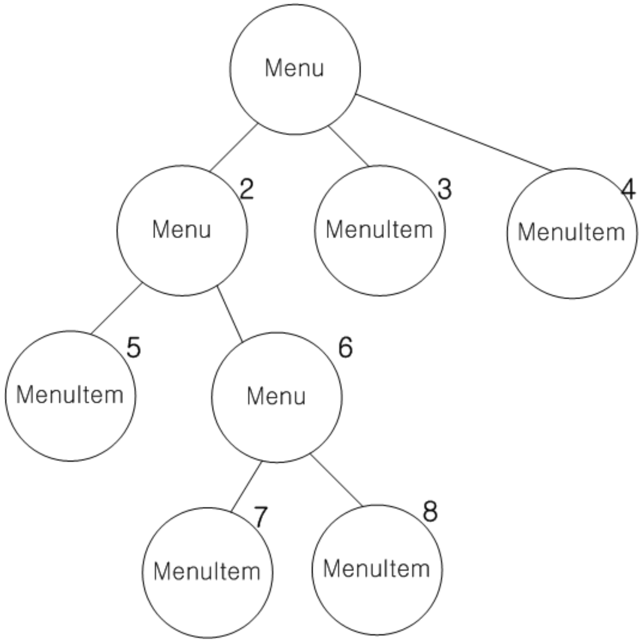
visited: 2, 5

```
public Object next() {
    if (hasNext()) {
        Iterator iterator = (Iterator)stack.peek();
        MenuComponent component=(MenuComponent)iterator.next();
        if (component instance of Menu)
            stack.push(component.createIterator());
        return component;
    }
    else return null;
}
public boolean hasNext() {
    if (stack.empty()) return false;
    Iterator iterator = (Iterator)stack.peek();
    if (!iterator.hasNext()) {
        stack.pop();
        return hasNext();
    } else return true;
}
```

6

```
public void printVegetarianMenu() {
    Iterator iterator = allMenus.createIterator();
    System.out.println("\nVEGETARIAN MENU\n-------");
    while (iterator.hasNext()) {
        MenuComponent menuComponent =
            (MenuComponent) iterator.next();
        try {
            if (menuComponent.isVegetarian())
                menuComponent.print();
        } catch (UnsupportedOperationException e) {}
    }
}
```

# Simulate the Algorithm



7 8

5 6

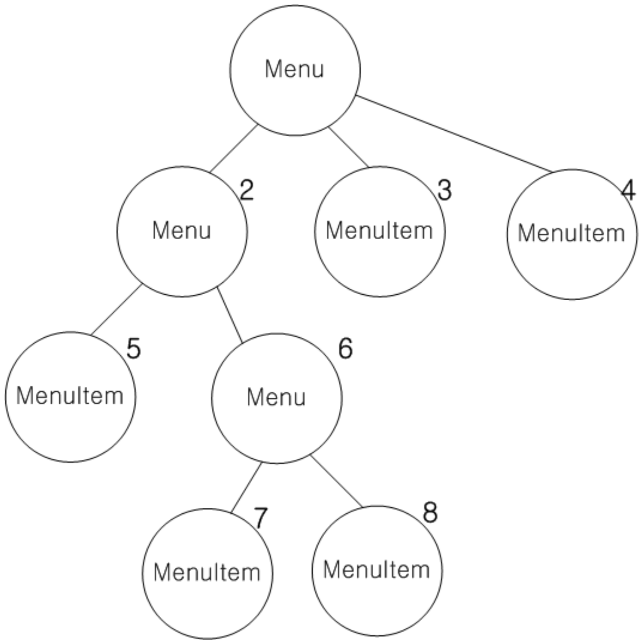
2 3 4

visited: 2, 5, 6

```
public Object next() {
    if (hasNext()) {
        Iterator iterator = (Iterator)stack.peek();   6
        MenuComponent component=(MenuComponent)iterator.next();
        if (component instance of Menu)
            stack.push(component.createIterator());
        return component;
    }                                                 6
    else return null;
}
public boolean hasNext() {
    if (stack.empty()) return false;
    Iterator iterator = (Iterator)stack.peek();
    if (!iterator.hasNext()) {
        stack.pop();
        return hasNext();
    } else return true;
}
```

```
public void printVegetarianMenu() {
    Iterator iterator = allMenus.createIterator();
    System.out.println("\nVEGETARIAN MENU\n-------");
    while (iterator.hasNext()) {
        MenuComponent menuComponent =
            (MenuComponent) iterator.next();   6
        try {
            if (menuComponent.isVegetarian())
                menuComponent.print();
        } catch (UnsupportedOperationException e) {}
    }
}
```

# Simulate the Algorithm



```
public Object next() {
    if (hasNext()) {
        Iterator iterator = (Iterator)stack.peek();   7
        MenuComponent component=(MenuComponent)iterator.next();
        if (component instance of Menu)
            stack.push(component.createIterator());
        return component;
    }
    else return null;                                 7
}
public boolean hasNext() {
    if (stack.empty()) return false;
    Iterator iterator = (Iterator)stack.peek();
    if (!iterator.hasNext()) {
        stack.pop();
        return hasNext();
    } else return true;
}
```

7 8

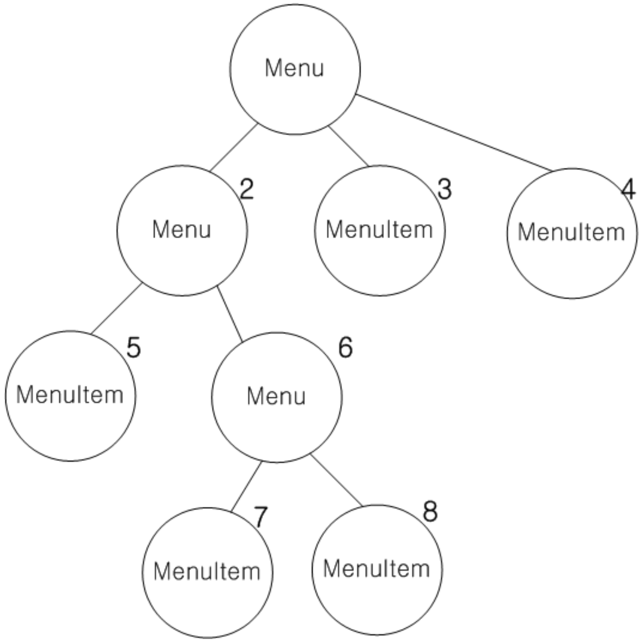
5 6

2 3 4

```
public void printVegetarianMenu() {
    Iterator iterator = allMenus.createIterator();
    System.out.println("\nVEGETARIAN MENU\n-------");
    while (iterator.hasNext()) {
        MenuComponent menuComponent =
            (MenuComponent) iterator.next();   7
        try {
            if (menuComponent.isVegetarian())
                menuComponent.print();
        } catch (UnsupportedOperationException e) {}
    }
}
```

visited: 2, 5, 6, 7

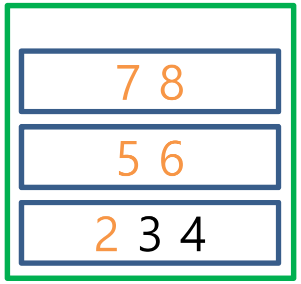
# Simulate the Algorithm



```
public Object next() {
    if (hasNext()) {
        Iterator iterator = (Iterator)stack.peek();
        MenuComponent component=(MenuComponent)iterator.next();
        if (component instance of Menu)
            stack.push(component.createIterator());
        return component;
    }
    else return null;
}
public boolean hasNext() {
    if (stack.empty()) return false;
    Iterator iterator = (Iterator)stack.peek();
    if (!iterator.hasNext()) {
        stack.pop();
        return hasNext();
    } else return true;
}
```

8

8



7 8

5 6

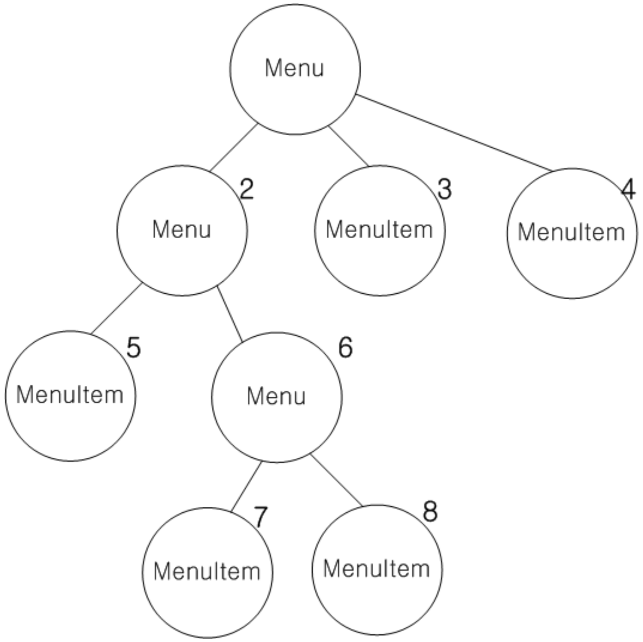
2 3 4

```
public void printVegetarianMenu() {
    Iterator iterator = allMenus.createIterator();
    System.out.println("\nVEGETARIAN MENU\n-------");
    while (iterator.hasNext()) {
        MenuComponent menuComponent =
            (MenuComponent) iterator.next();
        try {
            if (menuComponent.isVegetarian())
                menuComponent.print();
        } catch (UnsupportedOperationException e) {}
    }
}
```

8

visited: 2, 5, 6, 7, 8

# Simulate the Algorithm



7 8

5 6

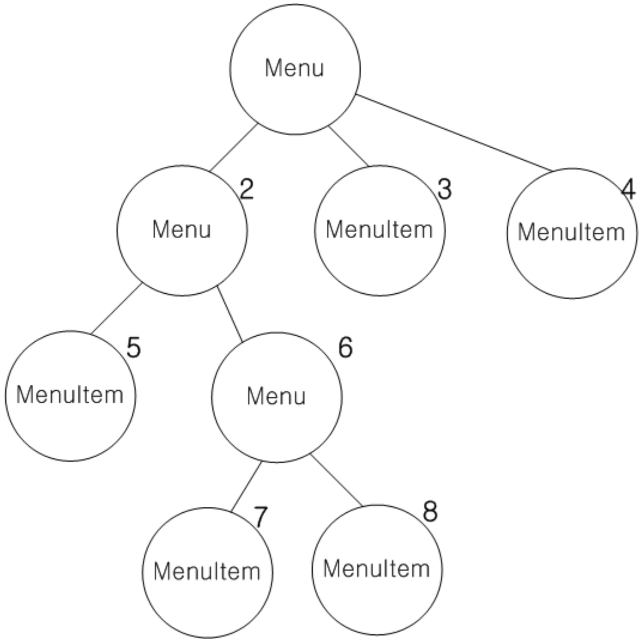
2 3 4

visited: 2, 5, 6, 7, 8

```
public Object next() {
    if (hasNext()) {
        Iterator iterator = (Iterator)stack.peek();
        MenuComponent component=(MenuComponent)iterator.next();
        if (component instance of Menu)
            stack.push(component.createIterator());
        return component;
    }
    else return null;
}
public boolean hasNext() {
    if (stack.empty()) return false;
    Iterator iterator = (Iterator)stack.peek();
    if (!iterator.hasNext()) {
        stack.pop();
        return hasNext();
    } else return true;
}
```

```
public void printVegetarianMenu() {
    Iterator iterator = allMenus.createIterator();
    System.out.println("\nVEGETARIAN MENU\n-------");
    while (iterator.hasNext()) {
        MenuComponent menuComponent =
            (MenuComponent) iterator.next();
        try {
            if (menuComponent.isVegetarian())
                menuComponent.print();
        } catch (UnsupportedOperationException e) {}
    }
}
```

# Simulate the Algorithm



7 8

5 6

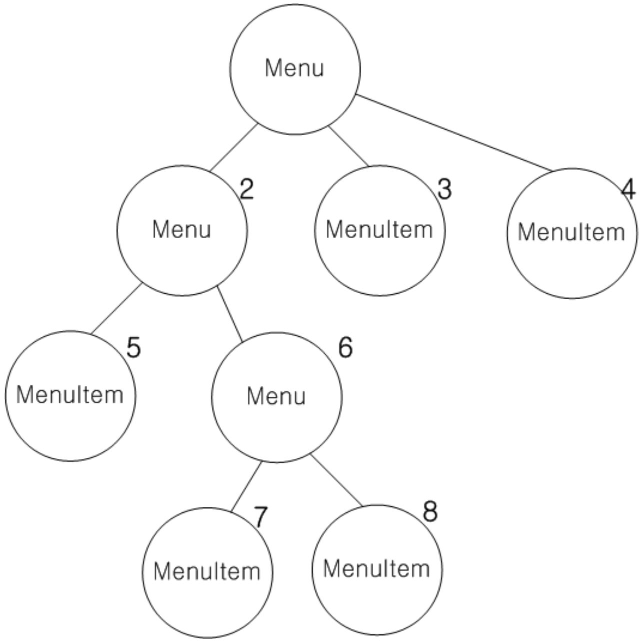
2 3 4

visited: 2, 5, 6, 7, 8

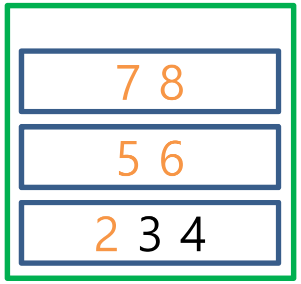
```
public Object next() {
    if (hasNext()) {
        Iterator iterator = (Iterator)stack.peek();
        MenuComponent component=(MenuComponent)iterator.next();
        if (component instance of Menu)
            stack.push(component.createIterator());
        return component;
    }
    else return null;
}
public boolean hasNext() {
    if (stack.empty()) return false;
    Iterator iterator = (Iterator)stack.peek();
    if (!iterator.hasNext()) {
        stack.pop();
        return hasNext();
    } else return true;
}
```

```
public void printVegetarianMenu() {
    Iterator iterator = allMenus.createIterator();
    System.out.println("\nVEGETARIAN MENU\n-------");
    while (iterator.hasNext()) {
        MenuComponent menuComponent =
            (MenuComponent) iterator.next();
        try {
            if (menuComponent.isVegetarian())
                menuComponent.print();
        } catch (UnsupportedOperationException e) {}
    }
}
```

# Simulate the Algorithm



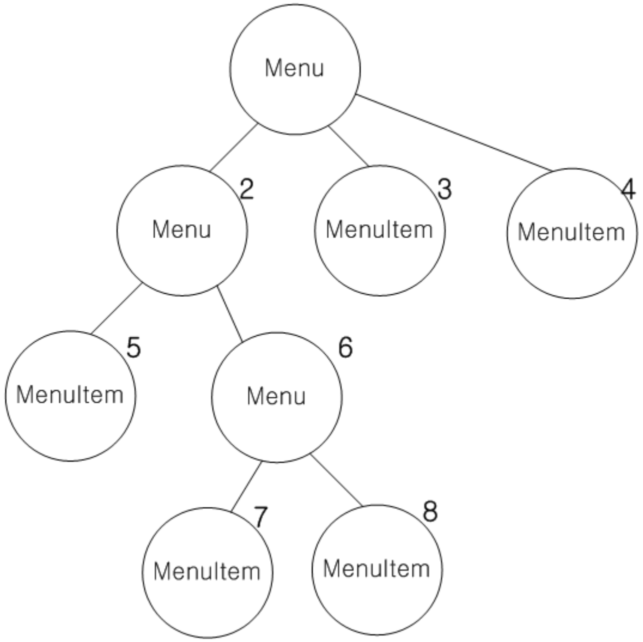
```
public Object next() {
    if (hasNext()) {
        Iterator iterator = (Iterator)stack.peek();
        MenuComponent component=(MenuComponent)iterator.next();
        if (component instance of Menu)
            stack.push(component.createIterator());
        return component;
    }
    else return null;
}
public boolean hasNext() {
    if (stack.empty()) return false;
    Iterator iterator = (Iterator)stack.peek();
    if (!iterator.hasNext()) {
        stack.pop();
        return hasNext();
    } else return true;
}
```

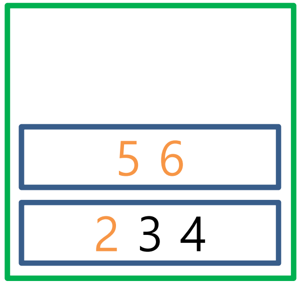


```
public void printVegetarianMenu() {
    Iterator iterator = allMenus.createIterator();
    System.out.println("\nVEGETARIAN MENU\n-------");
    while (iterator.hasNext()) {
        MenuComponent menuComponent =
            (MenuComponent) iterator.next();
        try {
            if (menuComponent.isVegetarian())
                menuComponent.print();
        } catch (UnsupportedOperationException e) {}
    }
}
```

visited: 2, 5, 6, 7, 8

# Simulate the Algorithm



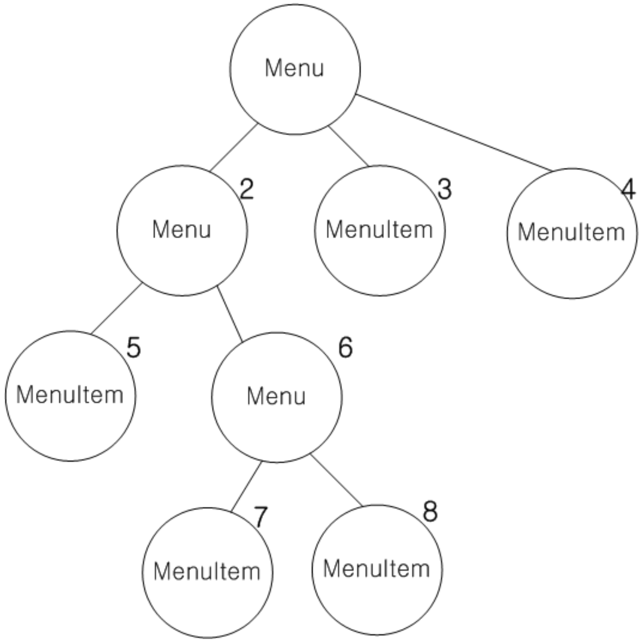


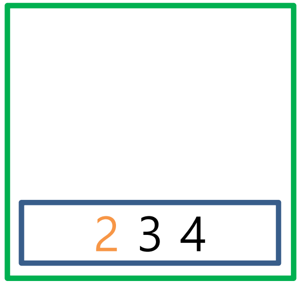
visited: 2, 5, 6, 7, 8

```
public Object next() {
    if (hasNext()) {
        Iterator iterator = (Iterator)stack.peek();
        MenuComponent component=(MenuComponent)iterator.next();
        if (component instance of Menu)
            stack.push(component.createIterator());
        return component;
    }
    else return null;
}
public boolean hasNext() {
    if (stack.empty()) return false;
    Iterator iterator = (Iterator)stack.peek();
    if (!iterator.hasNext()) {
        stack.pop();
        return hasNext();
    } else return true;
}
```

```
public void printVegetarianMenu() {
    Iterator iterator = allMenus.createIterator();
    System.out.println("\nVEGETARIAN MENU\n-------");
    while (iterator.hasNext()) {
        MenuComponent menuComponent =
            (MenuComponent) iterator.next();
        try {
            if (menuComponent.isVegetarian())
                menuComponent.print();
        } catch (UnsupportedOperationException e) {}
    }
}
```

# Simulate the Algorithm





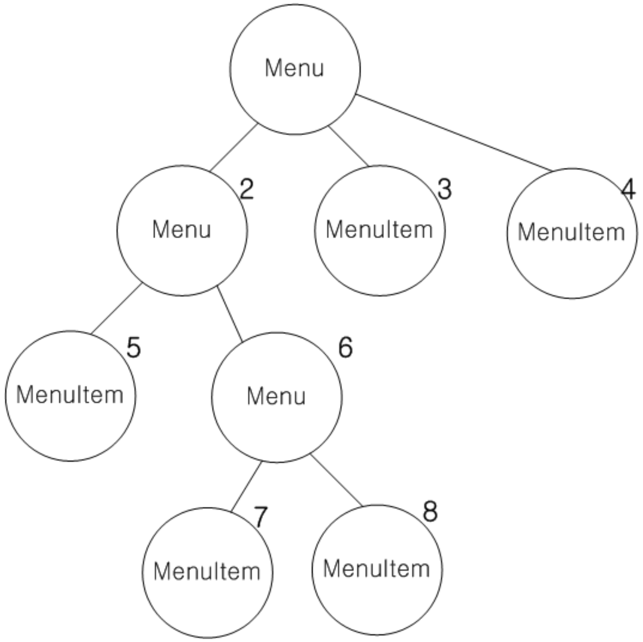
2 3 4

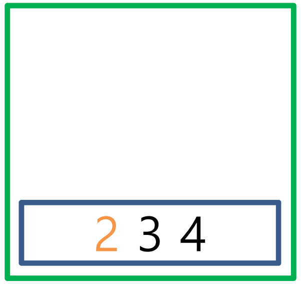
visited: 2, 5, 6, 7, 8

```
public Object next() {
    if (hasNext()) {
        Iterator iterator = (Iterator)stack.peek();
        MenuComponent component=(MenuComponent)iterator.next();
        if (component instance of Menu)
            stack.push(component.createIterator());
        return component;
    }
    else return null;
}
public boolean hasNext() {
    if (stack.empty()) return false;
    Iterator iterator = (Iterator)stack.peek();
    if (!iterator.hasNext()) {
        stack.pop();
        return hasNext();
    } else return true;
}
```

```
public void printVegetarianMenu() {
    Iterator iterator = allMenus.createIterator();
    System.out.println("\nVEGETARIAN MENU\n-------");
    while (iterator.hasNext()) {
        MenuComponent menuComponent =
            (MenuComponent) iterator.next();
        try {
            if (menuComponent.isVegetarian())
                menuComponent.print();
        } catch (UnsupportedOperationException e) {}
    }
}
```

# Simulate the Algorithm



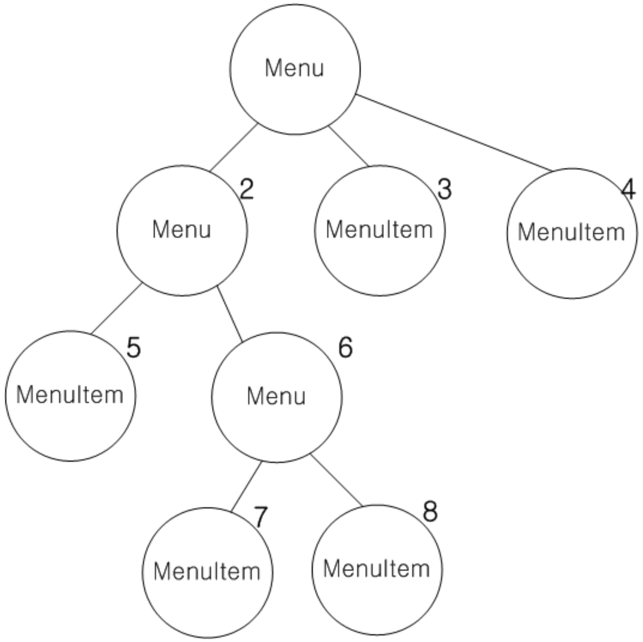


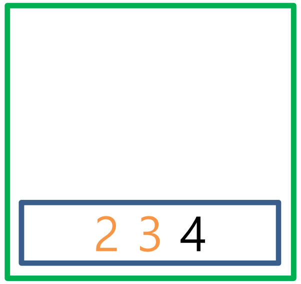
visited: 2, 5, 6, 7, 8

```
public Object next() {
    if (hasNext()) {
        Iterator iterator = (Iterator)stack.peek();
        MenuComponent component=(MenuComponent)iterator.next();
        if (component instance of Menu)
            stack.push(component.createIterator());
        return component;
    }
    else return null;
}
public boolean hasNext() {
    if (stack.empty()) return false;
    Iterator iterator = (Iterator)stack.peek();
    if (!iterator.hasNext()) {
        stack.pop();
        return hasNext();
    } else return true;
}
```

```
public void printVegetarianMenu() {
    Iterator iterator = allMenus.createIterator();
    System.out.println("\nVEGETARIAN MENU\n-------");
    while (iterator.hasNext()) {
        MenuComponent menuComponent =
            (MenuComponent) iterator.next();
        try {
            if (menuComponent.isVegetarian())
                menuComponent.print();
        } catch (UnsupportedOperationException e) {}
    }
}
```

# Simulate the Algorithm





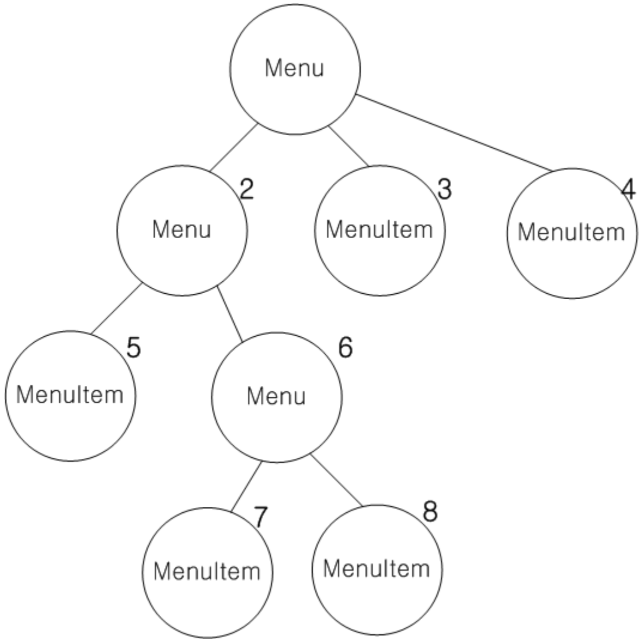
visited: 2, 5, 6, 7, 8

```
public Object next() {
    if (hasNext()) {
        Iterator iterator = (Iterator)stack.peek();
        MenuComponent component=(MenuComponent)iterator.next();
        if (component instance of Menu)
            stack.push(component.createIterator());
        return component;
    }
    else return null;
}
public boolean hasNext() {
    if (stack.empty()) return false;
    Iterator iterator = (Iterator)stack.peek();
    if (!iterator.hasNext()) {
        stack.pop();
        return hasNext();
    } else return true;
}
```

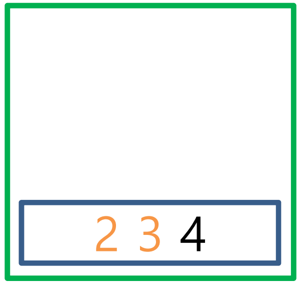
3

```
public void printVegetarianMenu() {
    Iterator iterator = allMenus.createIterator();
    System.out.println("\nVEGETARIAN MENU\n-------");
    while (iterator.hasNext()) {
        MenuComponent menuComponent =
            (MenuComponent) iterator.next();
        try {
            if (menuComponent.isVegetarian())
                menuComponent.print();
        } catch (UnsupportedOperationException e) {}
    }
}
```

# Simulate the Algorithm



```
public Object next() {
    if (hasNext()) {
        Iterator iterator = (Iterator)stack.peek();
        MenuComponent component=(MenuComponent)iterator.next();
        if (component instance of Menu)
            stack.push(component.createIterator());
        return component;
    }
    else return null;
}
public boolean hasNext() {
    if (stack.empty()) return false;
    Iterator iterator = (Iterator)stack.peek();
    if (!iterator.hasNext()) {
        stack.pop();
        return hasNext();
    } else return true;
}
```

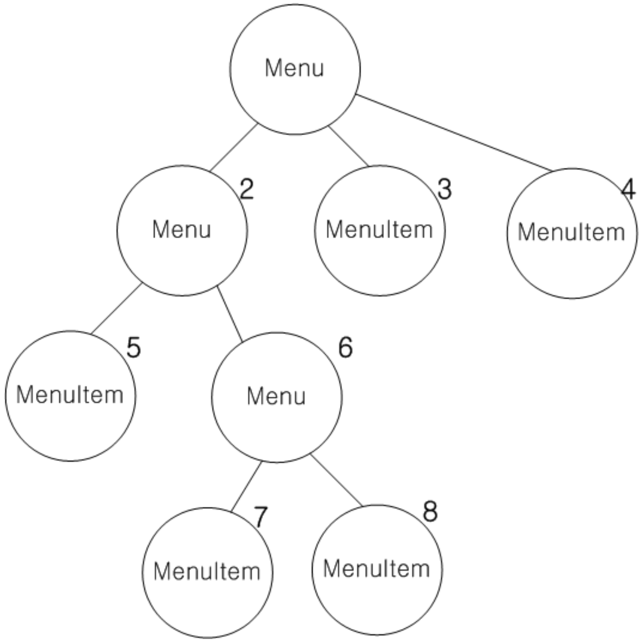


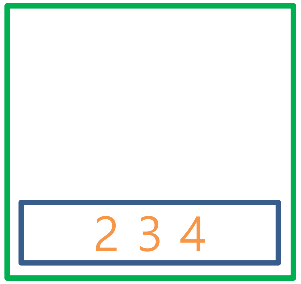
```
public void printVegetarianMenu() {
    Iterator iterator = allMenus.createIterator();
    System.out.println("\nVEGETARIAN MENU\n-------");
    while (iterator.hasNext()) {
        MenuComponent menuComponent =
            (MenuComponent) iterator.next();
        try {
            if (menuComponent.isVegetarian())
                menuComponent.print();
        } catch (UnsupportedOperationException e) {}
    }
}
```

3

visited: 2, 5, 6, 7, 8, 3

# Simulate the Algorithm





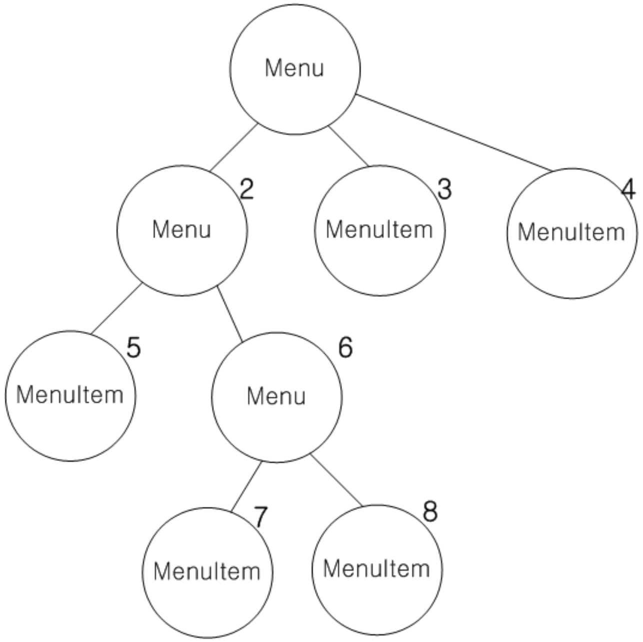
visited: 2, 5, 6, 7, 8, 3

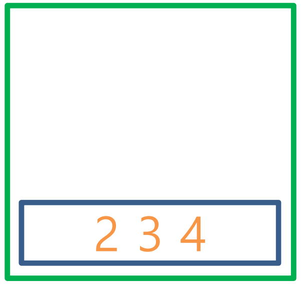
```
public Object next() {
    if (hasNext()) {
        Iterator iterator = (Iterator)stack.peek();
        MenuComponent component=(MenuComponent)iterator.next();
        if (component instance of Menu)
            stack.push(component.createIterator());
        return component;
    }
    else return null;
}
public boolean hasNext() {
    if (stack.empty()) return false;
    Iterator iterator = (Iterator)stack.peek();
    if (!iterator.hasNext()) {
        stack.pop();
        return hasNext();
    } else return true;
}
```

4

```
public void printVegetarianMenu() {
    Iterator iterator = allMenus.createIterator();
    System.out.println("\nVEGETARIAN MENU\n-------");
    while (iterator.hasNext()) {
        MenuComponent menuComponent =
            (MenuComponent) iterator.next();
        try {
            if (menuComponent.isVegetarian())
                menuComponent.print();
        } catch (UnsupportedOperationException e) {}
    }
}
```

# Simulate the Algorithm





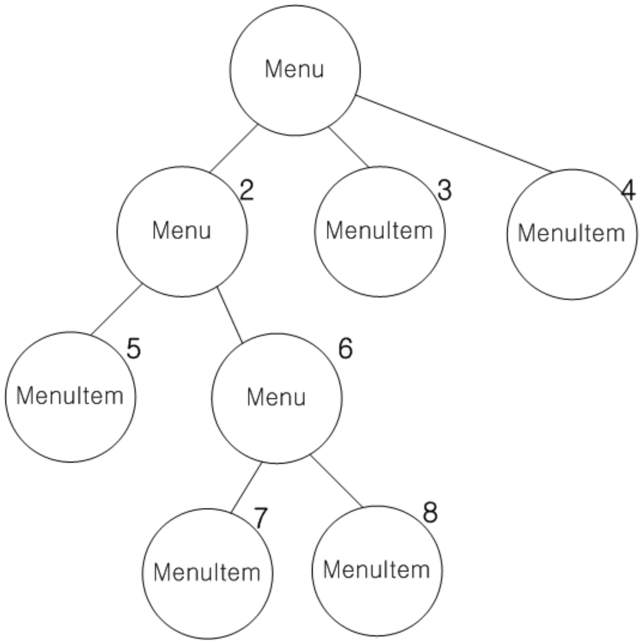
visited: 2, 5, 6, 7, 8, 3, 4

```
public Object next() {
    if (hasNext()) {
        Iterator iterator = (Iterator)stack.peek();
        MenuComponent component=(MenuComponent)iterator.next();
        if (component instance of Menu)
            stack.push(component.createIterator());
        return component;
    }
    else return null;
}
public boolean hasNext() {
    if (stack.empty()) return false;
    Iterator iterator = (Iterator)stack.peek();
    if (!iterator.hasNext()) {
        stack.pop();
        return hasNext();
    } else return true;
}
```

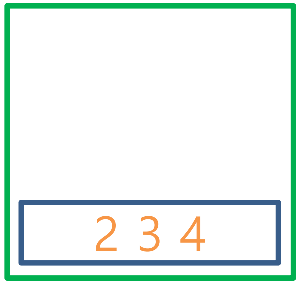
```
public void printVegetarianMenu() {
    Iterator iterator = allMenus.createIterator();
    System.out.println("\nVEGETARIAN MENU\n-------");
    while (iterator.hasNext()) {
        MenuComponent menuComponent =
            (MenuComponent) iterator.next();
        try {
            if (menuComponent.isVegetarian())
                menuComponent.print();
        } catch (UnsupportedOperationException e) {}
    }
}
```

4

# Simulate the Algorithm



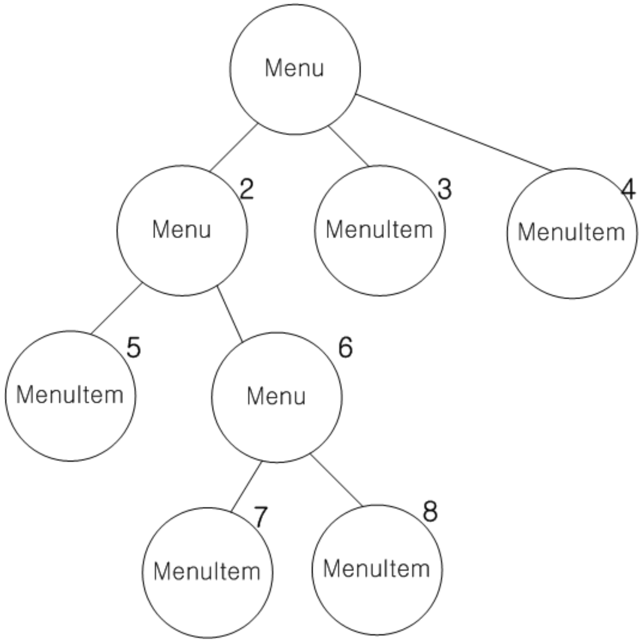
```
public Object next() {
    if (hasNext()) {
        Iterator iterator = (Iterator)stack.peek();
        MenuComponent component=(MenuComponent)iterator.next();
        if (component instance of Menu)
            stack.push(component.createIterator());
        return component;
    }
    else return null;
}
public boolean hasNext() {
    if (stack.empty()) return false;
    Iterator iterator = (Iterator)stack.peek();
    if (!iterator.hasNext()) {
        stack.pop();
        return hasNext();
    } else return true;
}
```



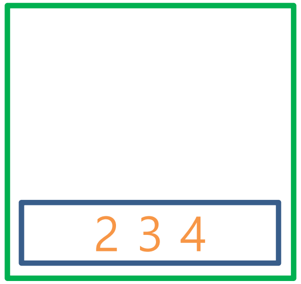
```
public void printVegetarianMenu() {
    Iterator iterator = allMenus.createIterator();
    System.out.println("\nVEGETARIAN MENU\n-------");
    while (iterator.hasNext()) {
        MenuComponent menuComponent =
            (MenuComponent) iterator.next();
        try {
            if (menuComponent.isVegetarian())
                menuComponent.print();
        } catch (UnsupportedOperationException e) {}
    }
}
```

visited: 2, 5, 6, 7, 8, 3, 4

# Simulate the Algorithm



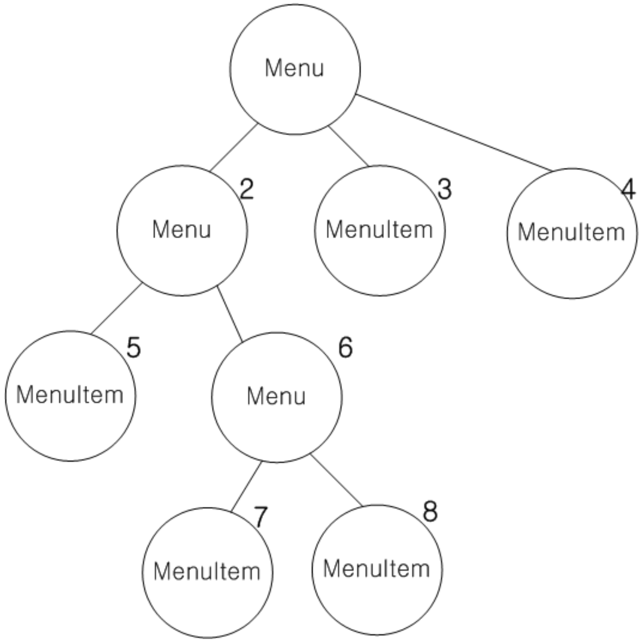
```
public Object next() {
    if (hasNext()) {
        Iterator iterator = (Iterator)stack.peek();
        MenuComponent component=(MenuComponent)iterator.next();
        if (component instance of Menu)
            stack.push(component.createIterator());
        return component;
    }
    else return null;
}
public boolean hasNext() {
    if (stack.empty()) return false;
    Iterator iterator = (Iterator)stack.peek();
    if (!iterator.hasNext()) {
        stack.pop();
        return hasNext();
    } else return true;
}
```



```
public void printVegetarianMenu() {
    Iterator iterator = allMenus.createIterator();
    System.out.println("\nVEGETARIAN MENU\n-------");
    while (iterator.hasNext()) {
        MenuComponent menuComponent =
            (MenuComponent) iterator.next();
        try {
            if (menuComponent.isVegetarian())
                menuComponent.print();
        } catch (UnsupportedOperationException e) {}
    }
}
```

visited: 2, 5, 6, 7, 8, 3, 4

# Simulate the Algorithm

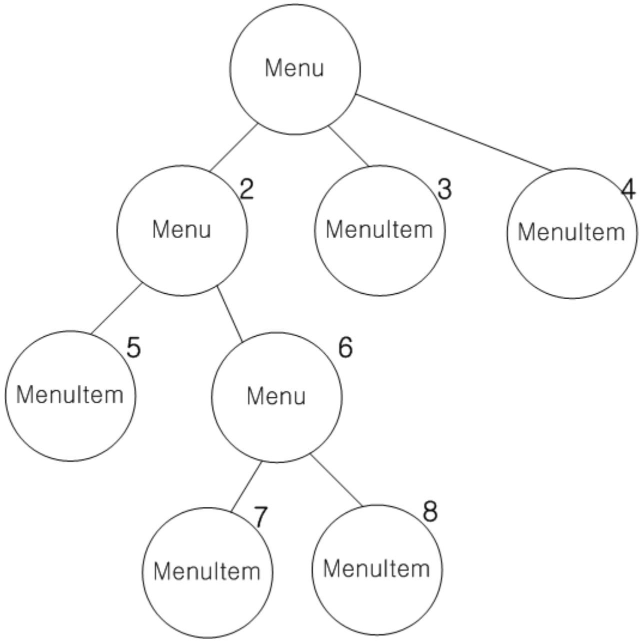


```
public Object next() {
    if (hasNext()) {
        Iterator iterator = (Iterator)stack.peek();
        MenuComponent component=(MenuComponent)iterator.next();
        if (component instance of Menu)
            stack.push(component.createIterator());
        return component;
    }
    else return null;
}
public boolean hasNext() {
    if (stack.empty()) return false;
    Iterator iterator = (Iterator)stack.peek();
    if (!iterator.hasNext()) {
        stack.pop();
        return hasNext();
    } else return true;
}
```

```
public void printVegetarianMenu() {
    Iterator iterator = allMenus.createIterator();
    System.out.println("\nVEGETARIAN MENU\n-------");
    while (iterator.hasNext()) {
        MenuComponent menuComponent =
            (MenuComponent) iterator.next();
        try {
            if (menuComponent.isVegetarian())
                menuComponent.print();
        } catch (UnsupportedOperationException e) {}
    }
}
```

visited: 2, 5, 6, 7, 8, 3, 4

# Simulate the Algorithm



```
public Object next() {
    if (hasNext()) {
        Iterator iterator = (Iterator)stack.peek();
        MenuComponent component=(MenuComponent)iterator.next();
        if (component instance of Menu)
            stack.push(component.createIterator());
        return component;
    }
    else return null;
}
public boolean hasNext() {
    if (stack.empty()) return false;
    Iterator iterator = (Iterator)stack.peek();
    if (!iterator.hasNext()) {
        stack.pop();
        return hasNext();
    } else return true;
}
```

```
public void printVegetarianMenu() {
    Iterator iterator = allMenus.createIterator();
    System.out.println("\nVEGETARIAN MENU\n-------");
    while (iterator.hasNext()) {
        MenuComponent menuComponent =
            (MenuComponent) iterator.next();
        try {
            if (menuComponent.isVegetarian())
                menuComponent.print();
        } catch (UnsupportedOperationException e) {}
    }
}
```

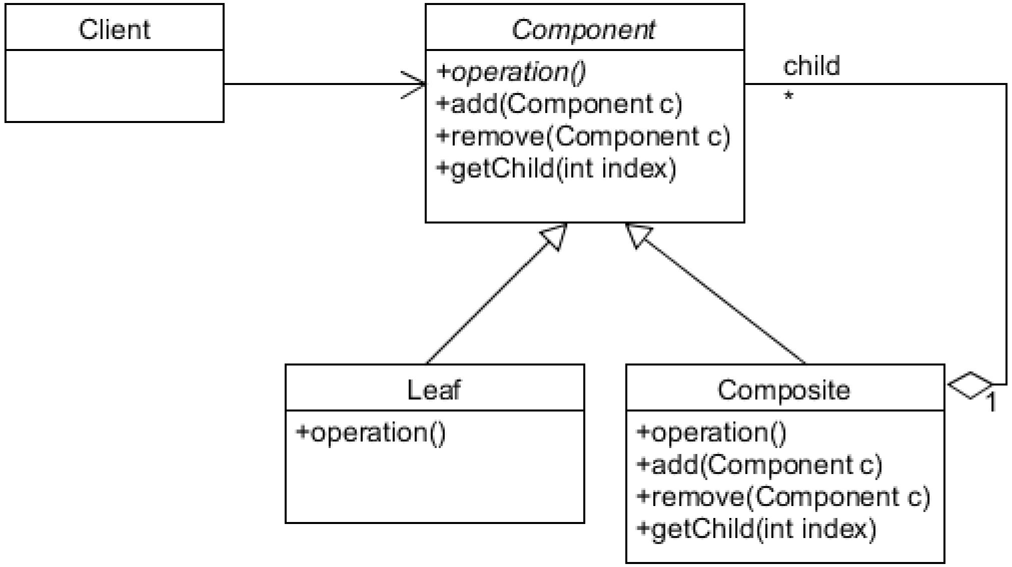
visited: 2, 5, 6, 7, 8, 3, 4

# Things to Consider

- A composite object stores the information about its contained components, i.e., its children. Should each component maintain a reference to its parent component?
  - It depends on applications. Having these references supports the Chain of Responsibility pattern

- Where do we need to declare the methods such as add(), remove(), and getChild() for managing children?
  - Implementation for transparency or safety

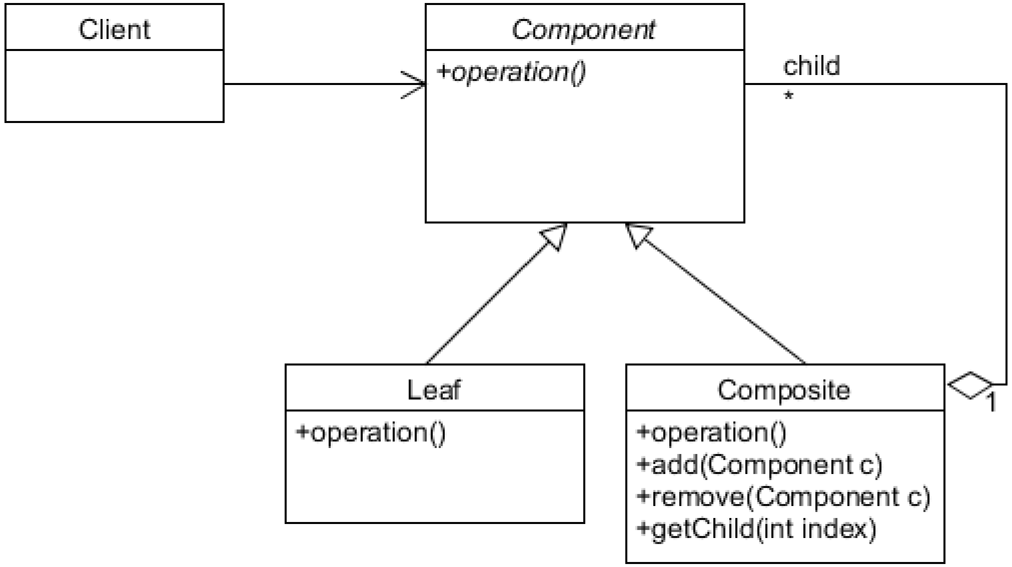
# Implementation for Transparency



- Implementation in the Component class:

Gives transparency, since all components can be treated the same. However, it is not safe. Clients can try to do meaningless things to leaf components at run-time.

# Implementation for Safety



- Implementation in the Composite class:

Gives safety, since any attempt to perform a child operation on a leaf component will be caught at compile-time. However, we lose transparency because now leaf and composite components have different interfaces.

# Related patterns

- Composite VS Decorator
  - Both have similar structure diagrams, reflecting the fact that both rely on recursive composition to organize an open-ended number of objects
  - Decorator is designed to let you add responsibilities to objects without subclassing
  - Composite's focus is not on embellishment but on representation
  - These intents are distinct but complementary. Consequently, Composite and Decorator are often used in concert

- Iterator
  - Provide a way to access the elements of an aggregate object (=typically uses composite pattern) sequentially without exposing its underlying representation

# Summary

- Composite Pattern
  - Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly