```
// NO VISITOR

Group scene = new Group
{
    Name = "Da Scene",
    Children =
    {
        new Sphere {Name = "Sphere One", Radius = 3},
        new Group
        {
            Name = "Child Group",
            Children =
            {
                new Cuboid {Name = "Just a cube", Width = 2, Height = 3, Depth = 4},
                new Sphere {Name = "Sphere Two", Radius = 3.1415},
            }
        },
        new Cuboid {Name = "Another Cuboid", Width = 6, Height = 7, Depth = 8},
    }
};

scene.Render();
```

```
public abstract class GraphOb

{
    public string Name;
    public abstract void Render();
}

public class Sphere : GraphOb
{
    public double Radius;

    public override void Render()
    {
        Console.WriteLine($"I am a sphere with radius {Radius}");
    }
}

public class Cuboid : GraphOb
{
    public double Width;
    public double Height;
    public double Depth;

    public override void Render()
    {
        Console.WriteLine($"Hi I am {Name}, the cuboid with size {Width}, {Height}, {Depth}");
    }
}

public class Group : GraphOb
{
    public List<GraphOb> Children = new List<GraphOb>();

    public override void Render()
    {
         Console.WriteLine($"I am the group named {Name}. I have {Children.Count} children.");
         TraverseChildren();
    }

    public void TraverseChildren()
    {
        foreach(GraphOb child in Children)
        {
            child.Render();
        }
    }
}
```

```
// VISITOR With Reflection DURING VISITING

Group scene = new Group
{
    Name = "Da Scene",
    Children =
    {
        new Sphere {Name = "Sphere One", Radius = 3},
        new Group
        {
            Name = "Child Group",
            Children =
            {
                new Cuboid {Name = "Just a cube", Width = 2, Height = 3, Depth = 4},
                new Sphere {Name = "Sphere Two", Radius = 3.1415},
            }
        },
        new Cuboid {Name = "Another Cuboid", Width = 6, Height = 7, Depth = 8},
    }
};

RenderVisitor v = new RenderVisitor();
v.Visit(scene);



public interface Visitor

{
    public void Visit(Sphere s);
    public void Visit(Cuboid c);
    public void Visit(Group g);
}

public class RenderVisitor : Visitor
{
    public void Visit(Cuboid c)
    {
        Console.WriteLine($"Hi I am {c.Name}, the cuboid with size {c.Width}, {c.Height},
{c.Depth}");
    }
    public void Visit(Sphere s)
    {
        Console.WriteLine($"I am a sphere with radius {s.Radius}");
    }
    public void Visit(Group g)
    {
         Console.WriteLine($"I am the group named {g.Name}. I have {g.Children.Count}
children.");
         g.TraverseChildren(this);
    }
}
```

```
public class GraphOb
{
    public string Name;
}

public class Sphere : GraphOb
{
    public double Radius;
}

public class Cuboid : GraphOb
{
    public double Width;
    public double Height;
    public double Depth;
}


public class Group : GraphOb
{
    public List<GraphOb> Children = new List<GraphOb>();

    public void TraverseChildren(Visitor v)
    {
        foreach(GraphOb child in Children)
        {
            switch (child)
            {
                case Sphere s:
                    v.Visit(s);
                    break;
                case Cuboid c:
                    v.Visit(c);
                    break;
                case Group g:
                    v.Visit(g);
                    break;
            }
            // v.Visit(child);
        }
    }
}
```

```
// CLASSICAL VISITOR PATTERN

Group scene = new Group
{
    Name = "Da Scene",
    Children =
    {
        new Sphere {Name = "Sphere One", Radius = 3},
        new Group
        {
            Name = "Child Group",
            Children =
            {
                new Cuboid {Name = "Just a cube", Width = 2, Height = 3, Depth = 4},
                new Sphere {Name = "Sphere Two", Radius = 3.1415},
            }
        },
        new Cuboid {Name = "Another Cuboid", Width = 6, Height = 7, Depth = 8},
    }
};

RenderVisitor v = new RenderVisitor();
v.Visit(scene);


public interface Visitor
{
    public void Visit(Sphere s);
    public void Visit(Cuboid c);
    public void Visit(Group g);
}

public class RenderVisitor : Visitor
{
    public void Visit(Cuboid c)
    {
        Console.WriteLine($"Hi I am {c.Name}, the cuboid with size {c.Width}, {c.Height},
{c.Depth}");
    }
    public void Visit(Sphere s)
    {
        Console.WriteLine($"I am a sphere with radius {s.Radius}");
    }
    public void Visit(Group g)
    {
         Console.WriteLine($"I am the group named {g.Name}. I have {g.Children.Count}
children.");
         g.TraverseChildren(this);
    }
}
```

```
public abstract class GraphOb
{
    public string Name;
    public abstract void Accept(Visitor v);
}

public class Sphere : GraphOb
{
    public double Radius;

    public override void Accept(Visitor v)
    {
        v.Visit(this);
    }
}

public class Cuboid : GraphOb
{
    public double Width;
    public double Height;
    public double Depth;

    public override void Accept(Visitor v)
    {
        v.Visit(this);
    }
}

public class Group : GraphOb
{
    public List<GraphOb> Children = new List<GraphOb>();

    public override void Accept(Visitor v)
    {
        v.Visit(this);
    }

    public void TraverseChildren(Visitor v)
    {
        foreach(GraphOb child in Children)
        {
            child.Accept(v);
            // v.Visit(child);
        }
    }
}
```

```
// VISITOR WITH REFLECTION UP-FRONT
using System.Reflection;

Group scene = new Group
{
    Name = "Da Scene",
    Children =
    {
        new Sphere {Name = "Sphere One", Radius = 3},
        new Group
        {
            Name = "Child Group",
            Children =
            {
                new Cuboid {Name = "Just a cube", Width = 2, Height = 3, Depth = 4},
                new Sphere {Name = "Sphere Two", Radius = 3.1415},
            }
        },
        new Cuboid {Name = "Another Cuboid", Width = 6, Height = 7, Depth = 8},
    }
};

RenderVisitor v = new RenderVisitor();
v.Visit(scene);
```

```
[AttributeUsage(AttributeTargets.Method)]
public class VisitorAttribute : Attribute
{
}

public abstract class Visitor
{
    public delegate void VisitMethod(GraphOb ob);
    public Dictionary<Type, VisitMethod>? _dispMap;
    public Dictionary<Type, VisitMethod> DispMap
    {
        get
        {
            if (_dispMap == null)
            {
                _dispMap = new Dictionary<Type, VisitMethod>();
                foreach (MethodInfo mi in GetType().GetMethods())
                {
                    if (mi.GetCustomAttribute<VisitorAttribute>() == null)
                      continue;

                    ParameterInfo[] pi = mi.GetParameters();
                    if (pi == null || pi.Length != 1)
                      continue;

                    Type graphObType = pi[0].ParameterType;
                    if (!(typeof(GraphOb)).IsAssignableFrom(graphObType))
                        continue;

                    _dispMap[graphObType] = ob => mi.Invoke(this, new []{ob});
                }
            }
            return _dispMap;
        }
    }
    public void Visit(GraphOb ob)
    {
        DispMap[ob.GetType()](ob);
    }
}
```

```
public class RenderVisitor : Visitor
{
    [Visitor]
    public void Render(Cuboid c)
    {
        Console.WriteLine($"Hi I am {c.Name}, the cuboid with size {c.Width}, {c.Height},
{c.Depth}");
    }
    [Visitor]
    public void Render(Sphere s)
    {
        Console.WriteLine($"I am a sphere with radius {s.Radius}");
    }
    [Visitor]
    public void Render(Group g)
    {
         Console.WriteLine($"I am the group named {g.Name}. I have {g.Children.Count}
children.");
         g.TraverseChildren(this);
    }
}
public abstract class GraphOb
{
    public string Name;
}

public class Sphere : GraphOb
{
    public double Radius;
}

public class Cuboid : GraphOb
{
    public double Width;
    public double Height;
    public double Depth;
}

public class Group : GraphOb
{
    public List<GraphOb> Children = new List<GraphOb>();

    public void TraverseChildren(Visitor v)
    {
        foreach(GraphOb child in Children)
        {
            v.Visit(child);
        }
    }
}
```