Visitor: Allows adding extra behaviors to entire hierarchies of classes

1. Need to define a new operation on an entire class hierarchy
   1. E.g. make a document model printable to html/Markdown
2. Do not want to keep modifying every class in the hierarchy
3. Need access to the non-common aspects of classes in the hierarchy
4. Create an external component to handle rendering
   1. But avoid type checks
5. Visitor:A pattern where a component(visitor) is allowed to traverse the entire inheritance hierarchy. Implemented by propagating a single visit() method throughout the entire hierarchy.

Intrusive Visitor

1. You put the method you want to add to the abstract class so all the hierarchy needs to implement the method.
2. This breaks the open closed principle as well as single responsibility principle.

Reflective Visitor

1. We write an external class called ExternalPrinter for example.
2. This will be responsible for the printing method that we want to add.
3. We put the Expression abstract class with Stringbuilder in the arguments of the method.
4. Since we want to perform different operations for the type of expression we use reflection.
5. We get the class and case by case perform different operations.
6. This has a problem of being slow since it is using reflection.

Classic Visitor (Double Dispatch)

1. Tight coupling with this approach. (You have to implement all the methods of the interface)
2. We have a visitor interface that has methods for all expression types. So each class that wants to operate in these expressions needs to implement this interface and therefore override these methods.

Acyclic Visitor

1. Write multiple interfaces for each expression type.
2. Gives more flexibility for which methods you want to implement compared to classic visitors.

Summary

1. Propagate an accept(Visitor v) method throughout the entire hierarchy
2. Create a visitor with visit(Foo), visit(Bar),... for each element in the hierarchy
3. Each accept() simply calls visitor.visit(this)
4. Acyclic visitor allows greater flexibility at a cost to performance