TRAVERSING AST'S

- Once AST is built, rest of translator will want to traverse the tree to perform various operations on it:
  - Print the tree
  - Perform semantic analysis
  - Interpreter will evaluate AST
  - Compilers will generate code from AST
- Approaches:

Approach	Explanation	Example
Procedural	Recursive function traverses tree	Evaluate(ast)
Pure OOP	Each AST class has its own method for each kind of operation	ast.Evaluate()
Visitor OOP	Visitors are objects which traverse ASTs; ASTs accept visits from visitors.	Evaluate is a visitor

VISITOR APPROACH

- The AST node classes do not need different methods for each kind of visitation.
- Instead, each AST node simply contains a method to accept the visit of any visitor.
- Visitor knows what to do when it visits each type of AST node.
- Example: AST nodes will not have an evaluate method. Instead, the visitor will know how to evaluate each type of AST node.

JJTREE VISITOR SUPPORT

- Assuming that
  - the jjtree file is called HL.jjt
  - In HL.jjt, `VISITOR=true;`

- SimpleNode and all the AST classes include the following method:

```
/** Accept the visitor. **/
public Object jjtAccept(HLVisitor visitor, Object data) {
  return visitor.visit(this, data);
}
```

- A new interface HLVisitor.java is created:

```
public interface HLVisitor
{
  public Object visit(SimpleNode node, Object data);
  public Object visit(ASTEOFReached node, Object data);
  public Object visit(ASTbody node, Object data);
  public Object visit(ASTclause node, Object data);
  // etc…
}
```

WORKING WITH JJTREE VISITORS

- To write a visitor, define a new class that implements HLVisitor & write code for each of the methods defined in the interface. E.g.

```
public class Eval implements HLVisitor
```

- To use a visitor, instantiate the class and ask the AST to accept the instantiation's visit.

```
private static HL parser;
SimpleNode tree;
Eval evaluator = new Eval();
tree = parser.start();
System.out.println(tree.jjtAccept(evaluator,null));
```