Visitor pattern

- It's a design pattern specifically designed to traverse a tree
- It's an **essential** feature in all compilers in order to travser AST

How to use

- Create a class that extends `CoolTreeVisitor`
- If we have an object of type `CoolTreeNode` then we can use it's `accept
- Override any visit method of `CoolTreeVisior` to intercept AST node type(<CoolTreeVisitor>)` method
    - This gives us a way to intercept the node of specific type
    - Do not forget to call `accept(this)` on the argument otherwise traverse will stop for the subsequent nodes

Example AST Visitor

```
class TestTreeVisitor(var program: Program) extends CoolTreeVisitor() {
  def run() = {
    program.accept(this)
    print(sb)
  }

  override def visit_program(node: Cprogram, classes: Classes) = {
    classes.accept(this)
  }

  def visit_formals(formals: Formals) = {
    val e = formals.elements()
    if (e.hasNext()) {
      while (e.hasNext()) {
        e.next().accept(this)
      }
    }
  }

  override def visit_formal(node: Cformal, name: Symbol, of_type: Symbol) = {
    // TODO: this is a leaf node, we can not go further down
  }

  override def visit_string_lit(node: Cstring_lit, token: Symbol) = {
    // TODO: this is a leaf node, we can not go further down
  }

  override def visit_nil(node: Cnil) = {
    // TODO: this is a leaf node, we can not go further down
  }

  override def visit_unit(node: Cunit) = {
    // TODO: this is a leaf node, we can not go further down
  }
}
```

Use our Visitor

```
object Main {
  def main(args: Array[String]) = {
    var parser = new CoolTreeParser();
    parser.start()

    var program = parser.parse_Program(0);
    parser.resolve_attributes()

    new TestTreeVisitor(program).run()
  }
}
```

Lab assignmnet

In this lab we will write a program to dump a formatted (or pretty printed) a Cool program given it's AST