compiler一个特点就是可以验证 identifier（id）//variable 是否被正确使用

例如variable的type是否是对的

function 的return type

object符合其对应的class declaration使用

在manifestly-typed languages显式类型语言中，每一个identifier在声明的时候都会被静态绑定一个type

class与data structure这种是user-defined data type

variable declaration(内置的type，以及class/data structure member//object)

function declaration

//意思就是每个identifier不管是普通的object变量名字，或是class声明的时候class名字，还是function名字，在声明的时候，都要指定对应type

其他language不会显示的静态绑定type

例如动态语言

他们可以在compile-type进行有限的分析

我们的project是显示语言

identification的declaration是在tree被遍历的时候进行的 //assignment2 的AST part

选择1：parse tree:是在处理parse phase阶段，使用syntax-directed translation的产物

选择2：abstract syntax tree: abstract syntax tree时parse phase遍历时生成的中间表示

处理declaration是用aggregating聚合与处理语义信息semantic information——这些语义信息出现在tree的leave处，并被整合，迁移到tree的中间Node

迁移/聚合/处理 是通过tree被遍历时的语义动作semantic action所激活的

我们要进行语义动作当对应的Node被traverse到的时候

如果是对一个中间表示（例如AST）进行processing,那么之后的processing可以在successive phases进行，因为successive phase中的每一个phase都负责了一个特定的analysis/translation process

概念

symbol table:符号表

scope:作用域

entry：条目 例如姓名 手机号 邮箱 就是一个联络簿上的一个条目

identifier：标识符

refer: 引用

通常来说，language/program描述的每个作用域scope都需要一个自己的symbol table

global scope包含所有在任何scope之外声明的identifier标识符的项

有些scope（例如class scope或者function scope）内也允许声明identifier，例如local variable或者inner class/inner function(这个project 没有)

有些scope没有具体名字，例如一个for loop,也允许定义自己的sub-scope子scope以及对应的sub-symbol table

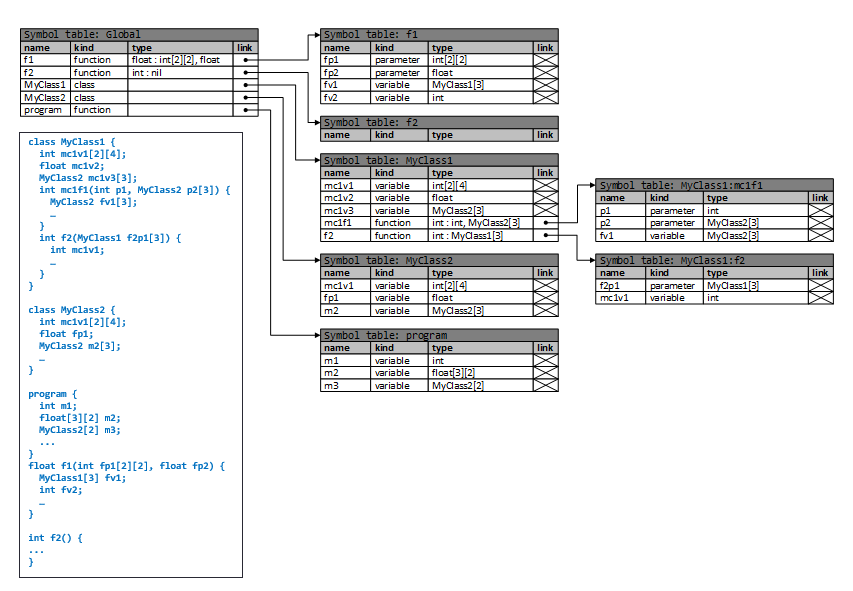
CreateNewTable:当一个新scope被创建的时候，我们也要创建一个对应的empty symbol table。同时这个symbol table的entry也要被记录在higher-level的symbol table中

insert:在一个scope内的时候，每一次有declaration，一个symbol table record被创建并插入

Search:当一个scope内引用一个identifier的时候，compiler需要检查他之前是否被declare过，有可能1.他在当前scope被declare，或者2.one of higher-level scopes被declare过。因此我们的search method需要能够沿着table hierarchy向上搜索。如果信息隐藏描述符information hiding descriptors是语言的一部分，必须考虑（我们的language没这么复杂）

Print:打出我们的symbol tabe，以及他的sub-table

Delete:有些scope内declare的identifier不应该在他们的scope之外被引用，，因此他们的symbol table必须被删除。而有些scope（例如class可以在他们的scope之外被引用）需要在运行过程一直到最后都保持symbol table



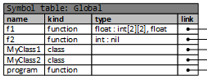
例子

分析：

首先我们一定要创造一个global symbol table:

任意scope之外的declaration都应该记录在symbol table中//就我的理解来说，更像一个目录

例如这里MyClass1, MyClass2,PROGRAM, float f1 ,int f2()，只要有identifier声明（不一定是variable，function名字也是identifier）且不在任意括号内，就应该加入global



而一个table中的一个entry有四个元素，

名字//identifier，

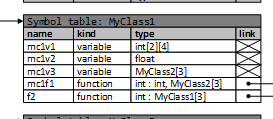
kind//绑定的类型， function,class,parameter(function或class的参数), variable(变量)

type:对kind具体区分，

例如对function来说，他的type就取决于return，以及参数（这两个要素相同，就代表是type相同的function），

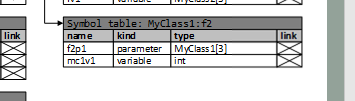
对于parameter或variable来说， type就是具体的 例如int[2], float这种

link，链接，这毕竟只是一个identifier的declaration，如果这个identifier的declaration会创建新的symbol table，我们就要创建对应link记录



注意这里的f2与外面global的f2不同

他是MyClass1的scope内declare的子function，因此有sub scope,sub table



Symbol table generation:

一共有两种方法：

1.使用syntax-directed translation

2.让parser生成abstract syntax tree 中间表示，然后安装一个tree traversal phase，遍历树阶段，来遍历AST，生成一个symbol table//我们使用这种

1.使用syntax-direccted translation的

在所有的Decl的时候加入对应的createTable / create entry

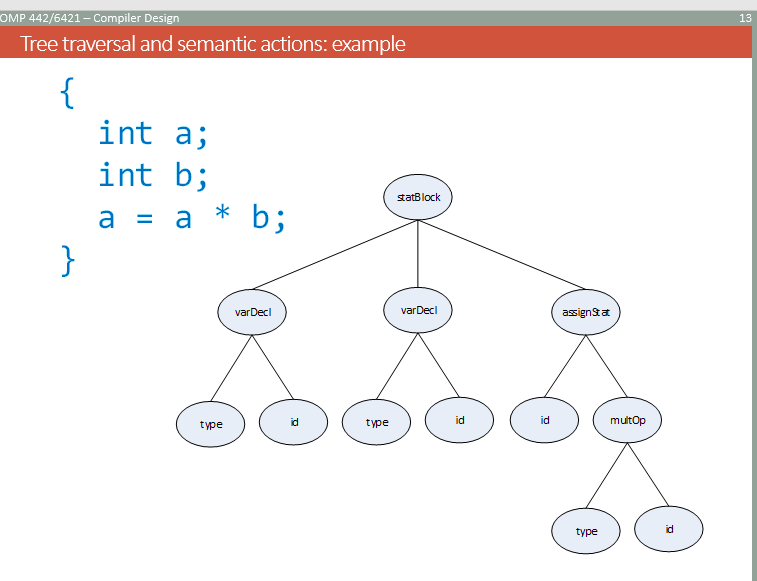


必须进行对应的属性迁移，因为创造需要对应的信息

缺陷：

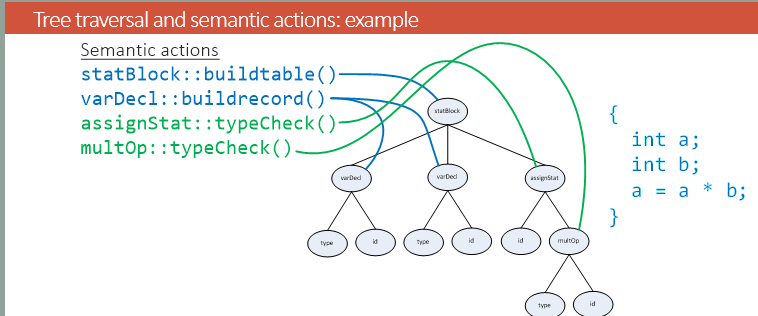
大量的semantic action造成了潜在风险

使用Visitor pattern来进行abstract syntax tree 遍历， 通过对abstract syntax tree遍历生成symbol table



例如这个abstract syntax tree

在遍历树的时候加入语义动作，两种情况：node第一次被reach，或者node的subtree被fully traversed



//进行的statBlock，buildtable, varDecl,build record，

每个node可能有0个或多个semantic actions，你可以吧语义动作也作为一种node

然而有些语义动作需要其他语义动作的先一步发生

因此语义动作必须以Phase来归类，并且映射到特定节点

Visitor design pattern是可以完成以上所有目的的一种结构

Visitor pattern:描述

动机：

当我们需要许多新operation并且object已经由许多不同类组成,每次进行新operation都加一个新subclass是不灵活的

将这些operation分配给不同的class会导致整个系统难以理解，保持，更改

通常来说，新的operations会形成groups——与operation类别有关

把所有的operation都加到同一个class里会导致low cohesion(凝聚力)

通常来说，只应该触发特定group内的operation

目的：  
创造一个机制，当新的operation被插入现有class中时，最小的改变现有class

把有关联的operation分类，提高cohesion与模块化

一次只能进行/差入某一特定group的operation

参与了visitor pattern的class/object //与下面网课结合着看

一共分为五类

Element: interface, 就是visitable

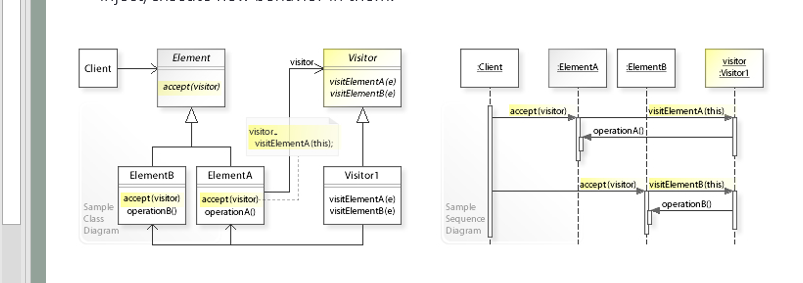
ConcreteElement: 就是具体的class，需要accept所需要的visitor(取决于这个visitor的behavior)来进行对应的operation()//例如改变node的某个private值

page15

Visitor:就是interface,记录了所有不同的concreteElement call visitor的function

ConcreteVisitor：具体的 能inject的behavior group

Client: 一段使用一组concreteElements的代码，并且想要把新behavior插入这个group（使用accept.visitor）//可以忽略



Client具体代码，让ElementA.accept(visitor)，然后visitor根据call他的是ElemenA,对他进行operationA

**一些概念：**

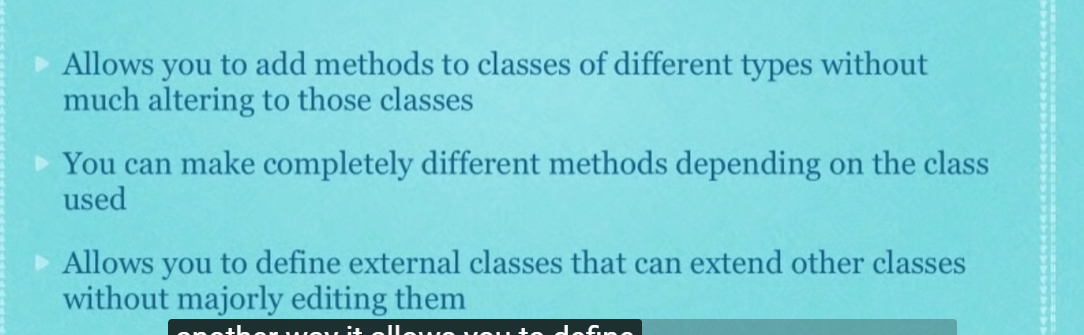
Behavior Group:

不同的Visitor代表着不同的Behavior，而group象征着不同node组成的group，



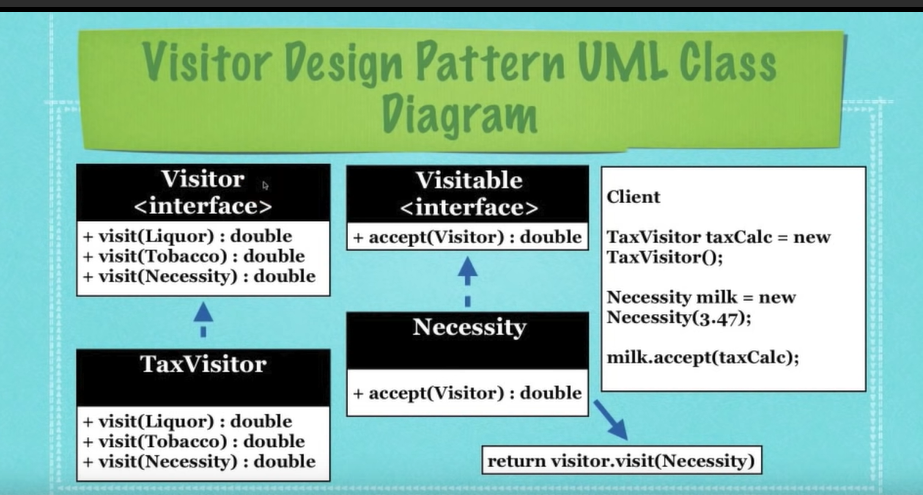
因此TCVISITOR就是代表着typechecking这个behavior，里面记录了各个element对应的visit，

网课：



这实际上是一种design pattern，允许你不大改class的情况下添加不同的method，在不编辑原class的情况下，extend them

以交税为例



Visitor Interface是一个抽象类 （.h）

假设你买一个东西的第一瞬间是没有交任何税的，然后我们的目的就是用TaxVisitor给他交税，而不同的物体税率不同

TaxVisitor会visit我们待检测的物体，并且告诉你对应的税率 //方法是visit(某一visitable )

然后创造我们的物体是visitable，待visit的，方法是accept,他会接受一个visitor来检查他，然后visitor告诉他怎么交税， accept的参数是visitor

会return visitor.visit(当前物体)

然后我们先创造一个taxCalc // visitor

然后对于具体object作为visitable实体创造

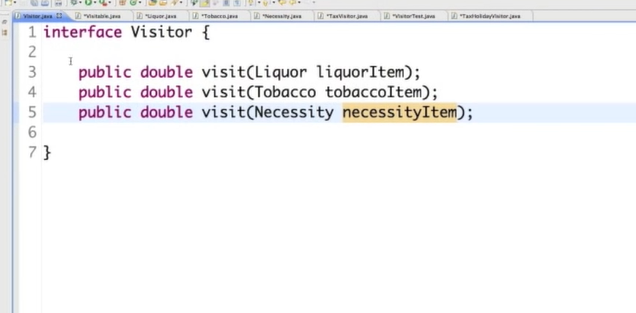
然后milk.accept(visitor) 得到return 的double也就是我们的税

总结：

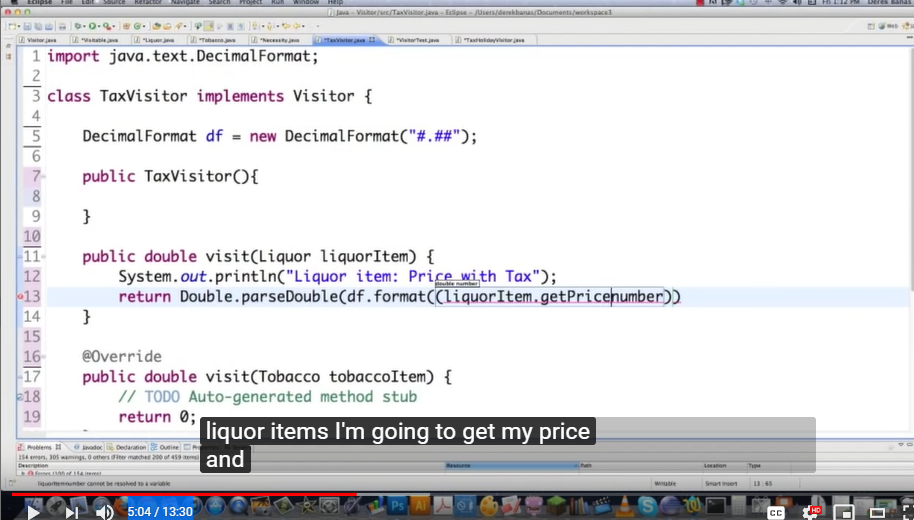
具体的实体visitable，它的功能是accept(visitor)然后让得到visitor.visit（这个实体）return的数

visitor更像一个工具，他能根据实体不同类型，return不同的东西

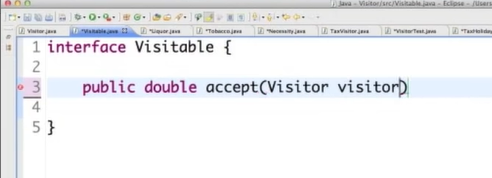
具体coding



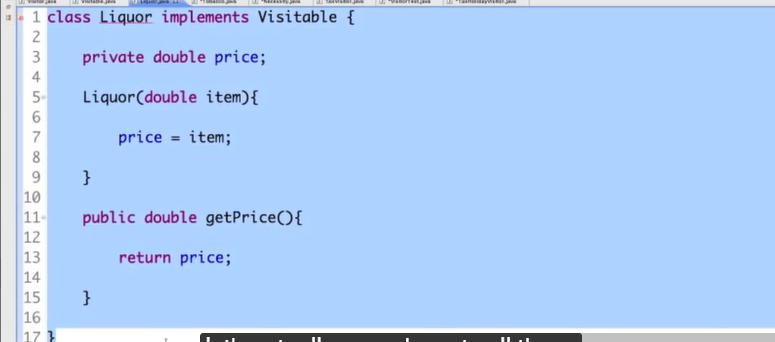
首先创造visitor (interface，理解成.h文件)



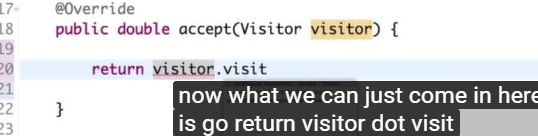
然后在concrete visitor中，根据visitable物体的accept(visitor)，会把这个物体传递给visitor，然后激活对应的visit function



visitable就是一个interface，更简单，就告诉你所有要被visit的东西是accept一个visitor



不同具体实体都要implement visitble这个特性，



然后我们给他们加上override accept

//Neccsity并不是大类，milk是一种Neccessity生活品, 而不是tabacco烟草或者酒精liqour

这样我们就完成了分类，例如水milk牙膏啥的用一种税率， 伏特加白酒啥的用一种税率

我们项目中Visitor的具体实施

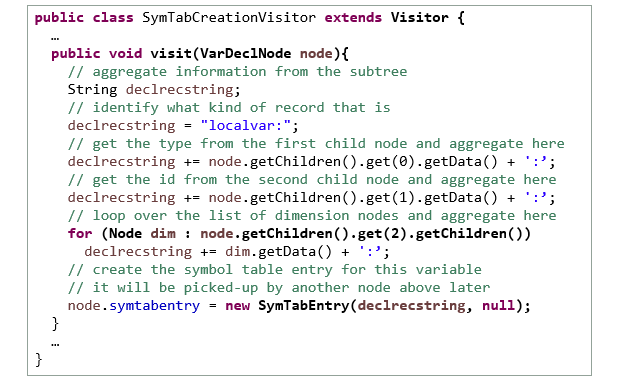
对于不支持Multiple dispatch//多分派，只支持single dispatch//单一分派的程序语言来说，visitor pattern是必要的。在这种情况下，假设有两个object一个是Element,一个是Visitor

Visitor class里的唯一function就是visit，通过参数的不同，注入相对应的behavior

Visitor class是Interface

为了提供一个默认的行为，它可以是一个常规class然后每个method是空的//然后继承就行了，但最好还是abstract

Concrete Visitor继承自Visitor class， 并且 会真正实装这些visit() methods， 每一个都能将behavior group的一部分操作operate到object structure中。 behavior group的state 被ConcreteVisitor local 保持



例如对VarDeclNode的visit

我们·visit他是想从子树聚合信息

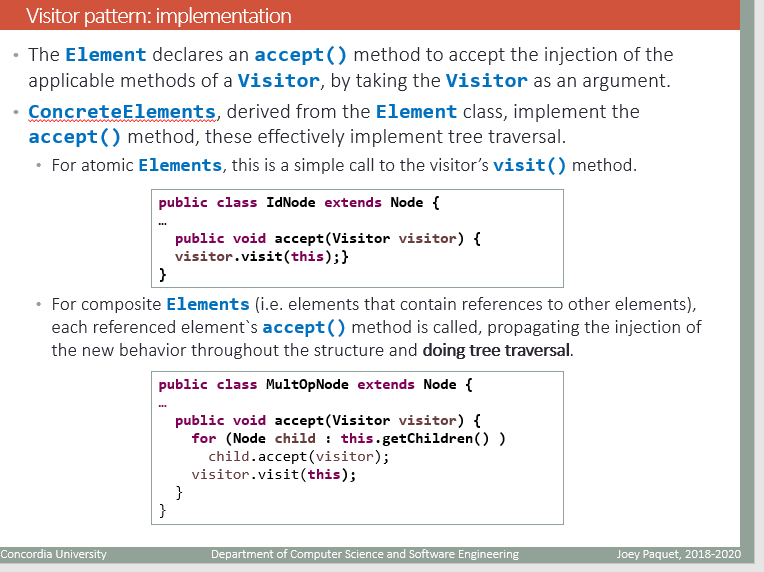
聚合的信息是declrecstring

Localvar: 左边get 第一子child//type, 然后get2,得到id，聚合到 declrecstring

然后用for loop得到全部data

再对这个node的symtable entry赋值，，让他等于NEW SYMTABLeNTRY

但这个过程中declrecstring只是一个暂时的temp local 变量



而Element会有一个accept()method既每一个node都有一个accept()，允许visitor来注入

对于atomic Element，只要单纯的call visit就好

但是对于复合型elements

我们要先让他的每个child accept visitor（需要被引用的element,例如乘号），然后visitor.visit this

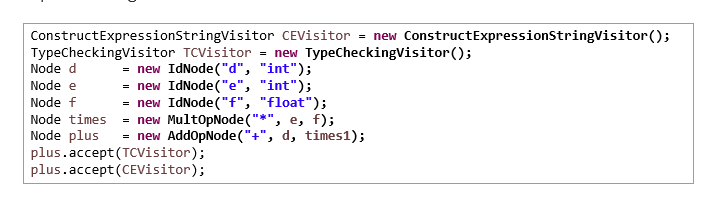
然后具体的代码中

Client(具体代码)会创造一个Element object，

然后初始化一个CEVisitor

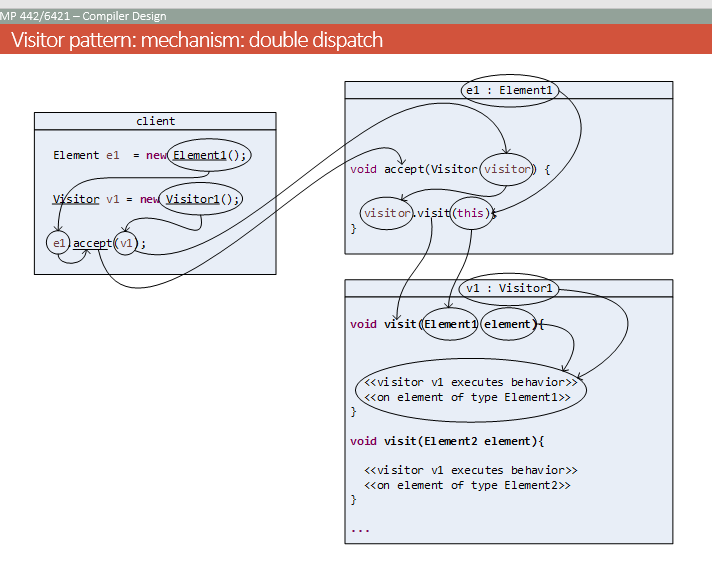
初始化一个TCVisitor//不同的visitor代表着不同的behavior,他们会对object structure(某个元素)进行inject(修改) //这个behavior能非常宽泛，目的只是单纯的在不对class做出大改变的情况下，对他进行操作

然后Element会用accept使用对应的visitor//对应他要的behavior



当accept被使用时，他会把自己作为参数让Vsitor.visit(this)，

然后好长一段话，总而言之Vsit() 会的选择基于dynamic type of the element与dynamic type of visitor，完全实现了double dispatch mechanism机制



这样，我们可以在Visitor里implement behavior，并且应用到Element上，随着我们遍历这个graph—这个graph有各式各样的element组成

我们可以通过使用不同的ConcreteVisitor来实现不同的behavior。

在Compiler中，Elements是AST node，

不同的ConcreteVisitor可以被设计成实装不同的semantic analysis与translation phases例如

Construction of symbol table建造SYMBOL TABLE

typeChecking, 检查type

translation to executable code， 转换成能运行的code

various optimization phases不同优化阶段

使用Visitor pattern,各个phase都可以被有效区分开，并且降低compiler的复杂性，提升维护性

Ast visitor来创造Symbol Table

每一个scope都应该有他自己的symbol table

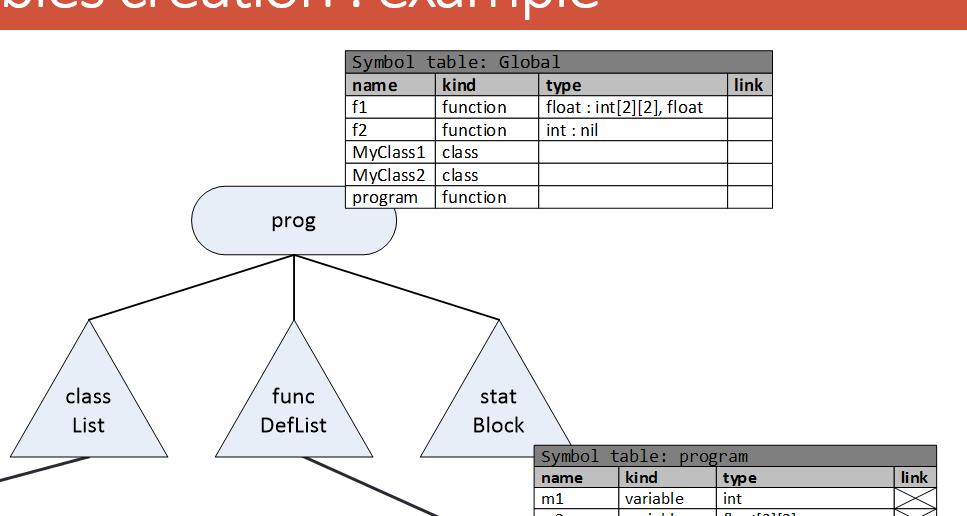
Global Class, Function Body: 三大scope

member function, free function ,main function都是function body

当我们visit 这些node type的时候

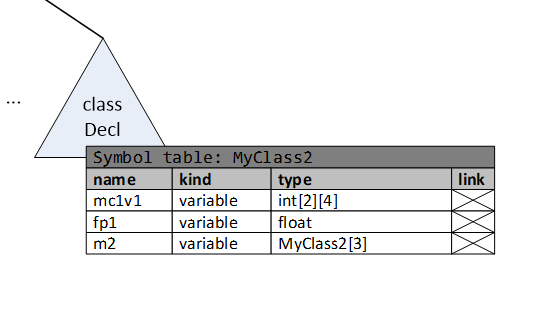
创建一个当地的symbol table，他是这个node的一个data member

聚合来自其子树的id声明的符号表记录



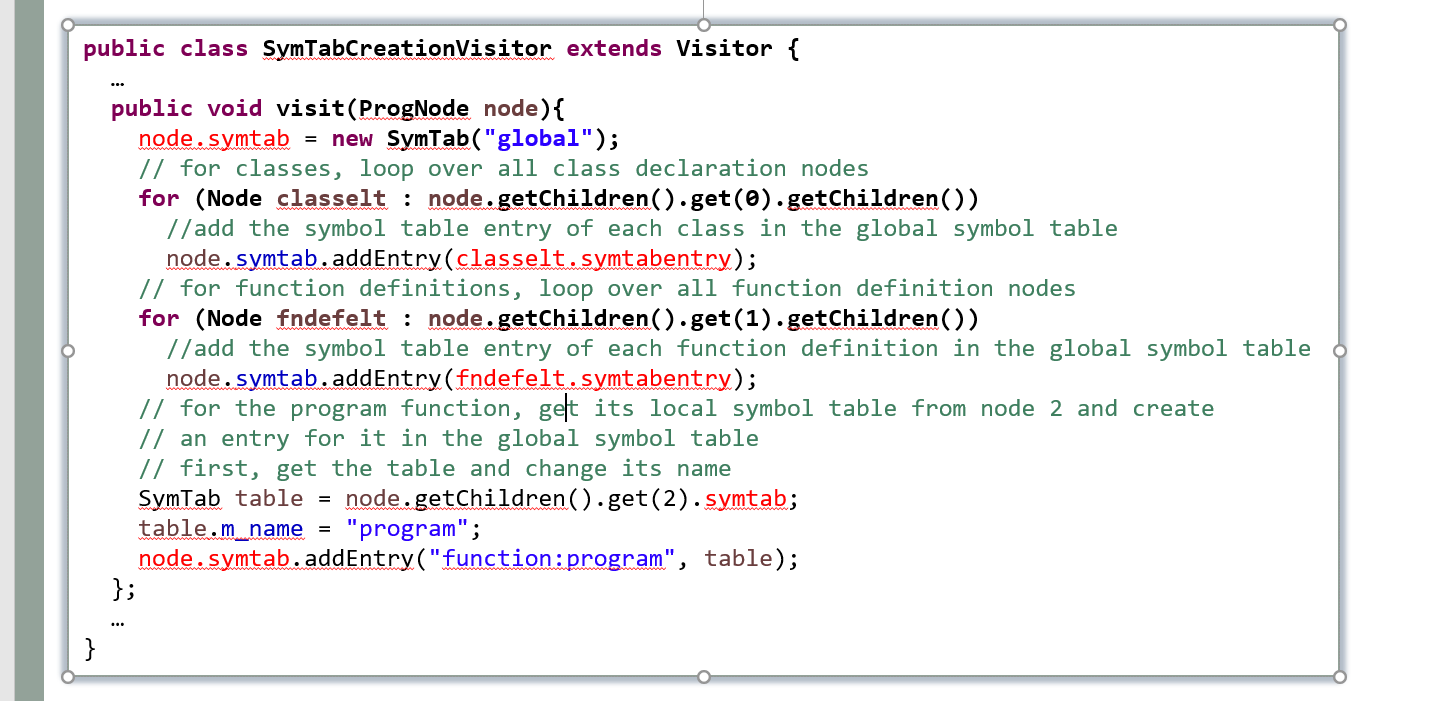
遇到prog node创造 Global table

global table聚合来自子树 Class List, Func DefList, 的id声明符号表记录（其实就是对应Entry）



ClassDecl创造SymbolTable

….



例如创造SymbolTableVisitor

Visitor不同的node

例如Visit到ProgNode，

创造新的SYMBol table global

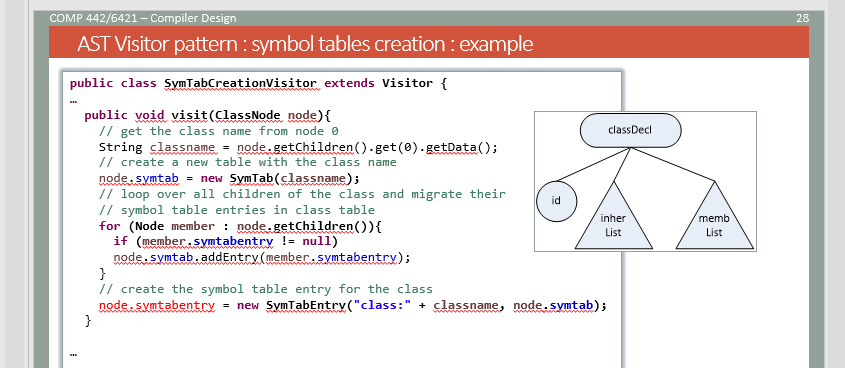
然后Node所有Classlist，并且getChildren， addEntry

**//Entry本身有没有pointer完全取决于你怎么创建Node，与symbol table本身无关，SymbolTable只负责两件事，遇到这个Node，创建新SymbolTable，然后把子Entry加入Table**

对于FUNCTIONList，得到所有children，add Entry

对于Program Function，(stat block) ,得到stat block的SymbolTable,复制成Temp,改变名字，然后add到Prog里

或者是ClassNode



他的Name是第一个children的Data，

然后创建SymbolTable，名字就是ClassName

然后搞到他所有的memberList

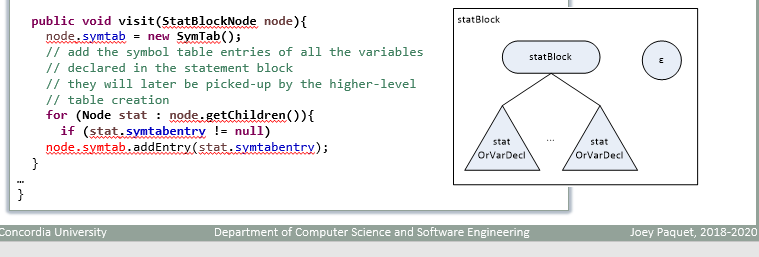
如果Member的SymtabEntry不是Null（有Entry）

addEntry

最后设定自己这个Node的SymbleTableEntry

第一个参数是名字，第二个参数是链接向这个symboltable

//prog不用设定自己的SymbleTableEntry因为没人要用它



在每一个Scope中，都有可能有Identifier被Delare

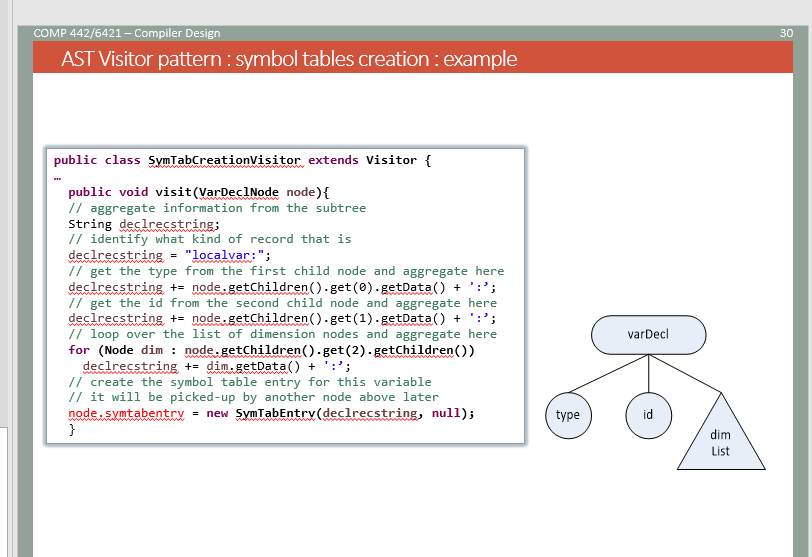
例如Class中，有data/function members

在function body中：有Local Variable,Parameters

当我们遇到这类Node

创建对应Symbol table record//也就是创造对应Entry,记录在Table中

然后父类就可以pick他当他被traverse到的时候



VarDecl的例子

才创造它的Entry

得到第一个Children

是type,加入Entry String上

得到第二个Children

是ID，加入Entry String

Dimm List,把dimension node都加入

因为是variable，不需要自己的symboltable，所以是Null

Semantic Checking

被Declare多次的Identifier：在插入Symbol table Records的时候检查//就是不能用一个变量，由两个Class叫一个名字就是ERROR

Undeclared identifier

在使用某一identifier，他并不在当前SCOPE或者他的higher-level scope

对于Function call

Function id 不存在于当前scope(或Higher-level scope)

参数的数量与这个Function描述的不一致

参数的Type与Function 描述的不一致

对于Variable 三种情况狂

Variable id不存在于当前scope(or higher-level scope)

使用的维度数与声明的维度数不匹配。 //array

id使用.operator，然后他的class type并没有描述对应的//不能存在的方法或变量

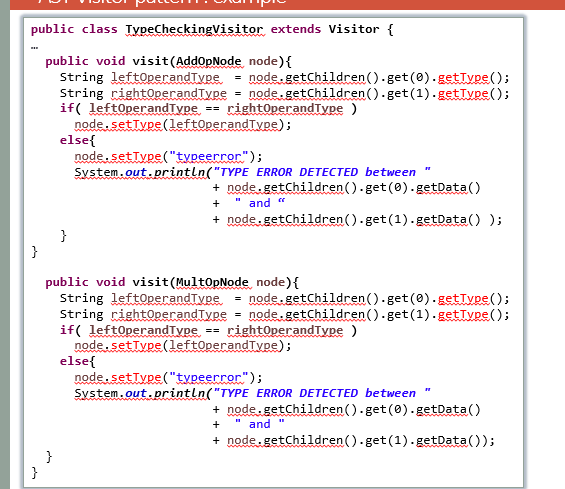
Type Checking

例如Operator的operand type不对//加号两边不是int float

参数传递与function要求的参数type不对

赋值=左右两边type不匹配

return 语句与function return type 不匹配



你需要多种ConcreteVVisitor

例如SymbtabCreationVisitor

TypeCheckingVisitor

ConstructAssignmentAndExpressionStringVisitor//这个project不用

当你装visitor的时候，语义信息必须被存储